TIC TAC TOE GAME

A PROJECT REPORT

Submitted by

R.Dharshini (9517202109014)

B.Sandhiya Devi (9517202109046)

S.T.Shri Subiksha (9517202109048)

in partial fulfillment for the award of the degree

of

Artificial Intelligence and Data Science

IN

Bachelor of Technology

MEPCO SCHLENK ENGINEERING COLLEGE,

SIVAKASI-626123

(An Autonomous Institution)

ANNA UNIVERSITY: CHENNAI 600 025

DECEMBER 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**MILK AND DAIRY MANAGEMENT SYSTEM**" is the bonafide work of "**R. Dharshini(21BAD008), B.Sandhiya Devi(21BAD022), S.T.Shri Subiksha(21BAD024)**"who carried out the project work under my supervision.

SIGNATURE

Dr. A.Shenbagarajan, M.E.,Ph.D

Assistant Professor (SG)

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

SIGNATURE

Dr.J.Angela Jennifa Sujana,M.E.,Ph.D

Associate Professor (SG) & Head

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**MILK AND DAIRY MANAGEMENT SYSTEM**" is the bonafide work of "**R. Dharshini(21BAD008), B.Sandhiya Devi(21BAD022), S.T.Shri Subiksha(21BAD024)**"who carried out the project work under my supervision.

SIGNATURE

Dr.P.Thendral,M.E.,Ph.D

Assistant Professor (SG)

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

SIGNATURE

Dr.J.Angela Jennifa Sujana,M.E.,Ph.D

Associate Professor (SG) & Head

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

ACKNOWLEDGEMENT

First and foremost we **praise and thank "The Almighty",** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Dr.A.Shenbagarajan.,M.E.,Ph.D.,** Assistant Professor(SG), **Dr.P.Thendral,M.E.,Ph.D., M.E(Ph.D)**, Assistant Professor (SG)Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Dr.A.Shenbagarajan.,M.E.,Ph.D.,** Assistant Professor(SG), **Dr.P.Thendral,M.E.,Ph.D,** Assistant Professor **(SG)** Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

TABLE OF CONTENTS

ABSTRACT

The project aims to develop a Tic Tac Toe game prediction system using artificial intelligence techniques. The system will utilize machine learning algorithms to train a model that can predict the optimal moves in the game based on the current board state. The trained model will be integrated into a user-friendly interface where players can play against the AI and receive recommendations for their next move. The project will involve several key steps. Firstly, a dataset will be generated by simulating numerous Tic Tac Toe games, capturing the board states and corresponding optimal moves. This dataset will be used for training the machine learning model. Various features will be extracted from the board state to represent the game's current state, including the positions of X's and O's, as well as any winning or blocking opportunities. Different machine learning algorithms, such as decision trees, neural networks, or reinforcement learning techniques, will be explored to determine the most effective approach for predicting the optimal moves. The trained model will be evaluated using cross-validation techniques and performance metrics such as accuracy and F1 score. OAI model. The player can choose to follow the AI's recommendation or make their own move. The success of the project will be measured by the accuracy of the AI's predictions and the overall gameplay experience. The system has the potential to enhance the gameplay by providing intelligent recommendations, challenging players, and promoting strategic thinking. Additionally, the project can be expanded to incorporate more advanced AI techniques and extend the game to larger board sizes or more complex variations.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The project aims to develop a Tic Tac Toe game prediction system using artificial intelligence techniques. The system will utilize machine learning algorithms to train a model that can predict the optimal moves in the game based on the current board state. The trained model will be integrated into a user-friendly interface where players can play against the AI and receive recommendations for their next move.

The project will involve several key steps. Firstly, a dataset will be generated by simulating numerous Tic Tac Toe games, capturing the board states and corresponding optimal moves. This dataset will be used for training the machine learning model. Various features will be extracted from the board state to represent the game's current state, including the positions of X's and O's, as well as any winning or blocking opportunities.

Different machine learning algorithms, such as decision trees, neural networks, or reinforcement learning techniques, will be explored to determine the most effective approach for predicting the optimal moves. The trained model will be evaluated using cross-validation techniques and performance metrics such as accuracy and F1 score.

Once the model is trained and validated, it will be integrated into a user interface that allows players to play against the AI. The interface will display the current board state and provide recommendations for the player's next move based on the predictions from the AI model. The player can choose to follow the AI's recommendation or make their own move.

The success of the project will be measured by the accuracy of the AI's predictions and the overall gameplay experience. The system has the potential to enhance the gameplay by providing intelligent recommendations, challenging players, and promoting strategic thinking. Additionally, the project can be expanded to incorporate more advanced AI techniques and extend the game to larger board sizes or more complex variations

1.2 OBJECTIVES

Optimal Move Prediction: Develop an AI model that can accurately predict the optimal move for a given board state. The AI should consider the current position of X's and O's and identify winning or blocking opportunities.

Strategic Decision-Making: Enable the AI to analyze the game state and make strategic decisions to maximize its chances of winning. This includes identifying patterns, recognizing winning or blocking moves, and adapting its gameplay based on the opponent's moves.

Intelligent Recommendations: Provide players with intelligent recommendations for their next move based on the AI's predictions. This feature can assist players in improving their gameplay and understanding strategic moves.

Learning and Adaptation: Explore techniques for the AI to learn and adapt over time. This could involve reinforcement learning algorithms that allow the AI to learn from its own gameplay experiences and improve its decision-making abilities.

User Experience Enhancement: Create a user-friendly interface where players can enjoy playing against the AI. The interface should provide clear visualizations of the game state, offer options for human vs. AI or human vs. human gameplay, and allow players to interact with the AI for move recommendations.

By achieving these objectives, the AI playing Tic Tac Toe will provide an engaging and challenging gaming experience for players of all skill levels. It will serve as a platform for strategic thinking, improve gameplay skills, and showcase the capabilities of AI in the context of a popular and accessible game

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

- AMD Ryzen 5 5625U Processor
- 16 GB RAM
- Mouse
- Keyboard

2.2 SOFTWARE REQUIREMENTS

- OS – Windows 11
- Python Interpreter
- Python IDLE

CHAPTER 3

FRONT END DESCRIPTION

3.1 Introduction

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a “batteries included” language due to its comprehensive standard library. Such standard libraries and modules in python are used to develop Graphical user interface in this project.

3.2 Graphical User Interface

A Graphical user interface is an interface through which a user interacts with electronic devices via visual indicator representations. Graphical user interfaces would become the standard of user-centered design in software application programming, providing users the capability to intuitively operate computers and other electronic devices through the direct manipulation of graphical icons such as buttons, scroll bars, windows, tabs, menus, cursor,.. etc. Graphical user interface can be designed using python language. Tkinter is the standard GUI library for python.

3.3.1 Tkinter module

The Tkinter module in python allows to build GUI applications in Python. Tkinter provides a variety of common GUI elements that can be used to build interfaces. These elements include buttons, menus, and various kinds of entry fields and display area. Tkinter provides an object-oriented interface to the Tk GUI toolkit. Other available GUI development platforms include wPython and JPython.

Fig 3.1: Tkinter module for creating GUI

Compared to other available options, Tkinter is lightweight and easy to use. Therefore, it is the go-to the choice for quickly building applications that can operate cross-platform and do not require a modern look.

Importing Tkinter and creating GUI in Python source code

```
from tkinter import *
import tkinter.font as f
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk,Image
```

Fig 3.2: Importing Tkinter modules in the source code

Creating a GUI application using Tkinter is an easy task. The steps to create GUI application are

- Import the Tkinter module.

- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application as desired by the user.
- Enter the main event loop to take action against each event triggered by the user.

```
root=Tk()
root.title("Home page")
root.minsize(640,400)
#root.state('zoomed')
bg_frame=Image.open(r"C:\Users\maha9\OneDrive\Documents\IV_semester\ai&da_minipr
photo=ImageTk.PhotoImage(bg_frame)
bg_panel=Label(root,image=photo)
bg_panel.image=photo
bg_panel.pack(fill='both',expand='yes')
gridframe=Frame(root,width=900,height=600)
gridframe.place(x=450,y=135)
B1=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B2=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B3=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B4=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B5=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B6=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B7=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B8=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
B9=Button(gridframe,text=" ",font=("Helvetica",20),height=3,width=6,bg="SystemBu
buttons=[B1,B2,B3,B4,B5,B6,B7,B8,B9]
board={B1:" ",B2:" ",B3:" ",
       B4:" ",B5:" ",B6:" ",
```

Fig 3.3: Window creation code

The above code would create a following window-

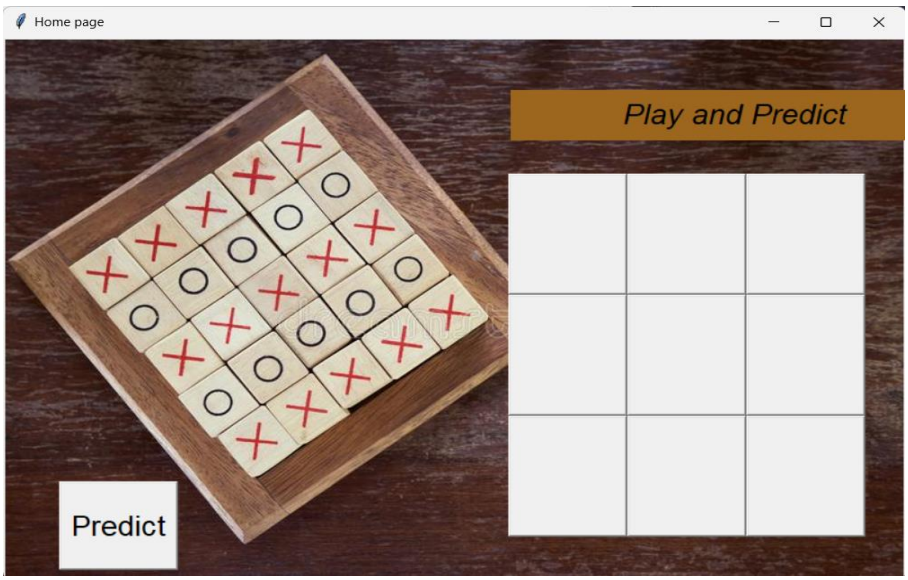

Fig 3.4: Game Interface to play

3.3.1.1 Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets. There are currently 19 types of widgets in Tkinter. The widgets are explained briefly in the below table.

S.No	Widget	Description
1	Button	The button widget is used to display buttons in the application.
2	Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in the application.
3	Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
5	Frame	The Frame widget is used as a container widget to organize other widgets.
6	Label	The Label widget is used to provide a single-line Caption for other widgets. It can also contain images.
7	Listbox	The Listbox widget is used to provide a list of options to a user.
8	Menubutton	The Menubutton widget is used to display menus in the application.
9	Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.

10	Message	The Message widget is used to display multiline text fields for accepting values from the user.
11	Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	Scale	The Scale widget is used to provide a slider widget.
13	Scrollbar	The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	Text	The Text widget is used to display text in multiple lines.
15	Toplevel	The Toplevel widget is used to provide a separate window container.
16	Spinbox	The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
17	PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	tkMessageBox	This module is used to display message boxes in your applications.

Table 3.1: Tkinter Widgets

Some of the above-mentioned widgets are used in the project to create a GUI application.

3.3.1.2 Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following manager classes:

- The pack() method
- The grid() method
- The place() method

The grid() method

The grid() method is one of the geometry manager organizes widgets in a table-like structure in the parent widget. This method is also used in the project to create good looking GUI.

```
heading=Label(bg_panel,text=txt
heading.place(x=450,y=50,width=
B1.grid(row=0,column=0)
B2.grid(row=0,column=1)
B3.grid(row=0,column=2)
B4.grid(row=1,column=0)
B5.grid(row=1,column=1)
B6.grid(row=1,column=2)
B7.grid(row=2,column=0)
B8.grid(row=2,column=1)
B9.grid(row=2,column=2)
```

Fig 3.5: Illustrating the use of grid () method

3.3 Retrieve Configuration Option from Widget

The cget() function is a method available in various widget libraries and frameworks that allows you to retrieve the value of a specific configuration option of a widget.

When working with user interface elements or widgets, we often need to access and manipulate their properties or settings. These properties can include attributes like the color, size, font, or any other configurable aspect of the widget. The cget() function simplifies the process of accessing these settings by providing a convenient way to retrieve the current value of a particular configuration option.

To use the cget() function, we typically pass the name of the configuration option as a parameter. The function then returns the current value associated with that option. This allows us to obtain the current state of the widget's properties dynamically and use the retrieved value for further processing or display purposes.

By utilizing cget(), we can easily obtain the values of various configuration options without resorting to manual inspection or searching through extensive documentation. This function streamlines the retrieval of widget properties, enabling efficient and concise code development.

```
##function to get input from grid##

def getboardinput():
    for i in buttons:
        mytext=i.cget('text')
        board[i]=mytext
```

Fig 3.6: Illustration of cget() function

CHAPTER 4

TIC TAC TOE

4.1 Introduction

Artificial Intelligence (AI) can be implemented in Tic-Tac-Toe to create a computer opponent that can play against a human player or another AI. The goal of implementing AI in Tic-Tac-Toe is to create a challenging and intelligent opponent that can make strategic decisions based on the current state of the game.

4.2 Game Description

Tic-Tac-Toe is a classic paper-and-pencil game played on a grid of 3x3 squares. The objective is to be the first player to form a line of three matching symbols (traditionally X or O) horizontally, vertically, or diagonally.

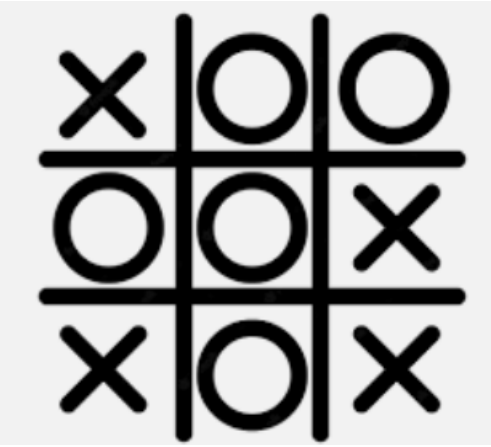

Fig 4.1 Tic Tac Toe Game

Here's a basic outline of how a typical Tic-Tac-Toe game is played:

- The game starts with an empty 3x3 grid.
- Two players take turns, with one player using X and the other using O.
- On their turn, a player selects an empty square on the grid and marks it with their symbol.

- The turn alternates between the players until a win condition is met or the grid is full.
- A win condition occurs when one player successfully forms a line of three matching symbols.
- If a win condition is met, the game ends, and the player who achieved the win is declared the winner.
- If the grid is full and no-win condition is met, the game ends in a draw.

Tic-Tac-Toe is a simple yet entertaining game that can be played on paper, a physical game board, or even implemented as a computer program.

CHAPTER 5

MINMAX ALGORITHM

5.1 Introduction

The Minimax algorithm is a popular approach to implementing AI in Tic-Tac-Toe. It allows the AI to make optimal moves by evaluating the possible outcomes of each move and selecting the move that maximizes its chances of winning while minimizing the opponent's chances.

By using the Minimax algorithm, the AI in Tic-Tac-Toe can evaluate all possible moves and select the best move based on the expected outcome. This allows the AI to make strategic decisions and provide a challenging opponent to play against.

5.2 Algorithm

- The algorithm considers the game as a tree structure, with each node representing a game state and the edges representing possible moves.
- The algorithm recursively explores the game tree by considering all possible moves and their resulting game states.
- At each level of the tree, the algorithm alternates between two players: the maximizing player and the minimizing player.
- The maximizing player aims to maximize its own score, while the minimizing player aims to minimize the maximizing player's score.
- Terminal states (game over states) are assigned scores based on the outcome: positive scores for a win, negative scores for a loss, and zero for a draw.
- The algorithm propagates these scores back up the tree, considering the best outcome for each player at each level.
- When the algorithm reaches the root of the tree, it selects the move that leads to the highest score for the maximizing player.

By evaluating all possible moves and their outcomes, the Minimax algorithm ensures that the maximizing player makes the best move possible, assuming the opponent also plays optimally. However, since Tic-Tac-Toe has a limited number of possible game states, it is possible to exhaustively search the entire game tree.

To optimize the algorithm's performance, techniques like alpha-beta pruning can be used. Alpha-beta pruning allows the algorithm to eliminate parts of the game tree that are guaranteed to be worse or irrelevant, reducing the number of nodes that need to be evaluated.

Overall, the Minimax algorithm provides a systematic approach for decision-making in games like Tic-Tac-Toe, allowing AI players to make strategic moves and play competitively against human players or other AI opponents

5.3 MinMax Algorithm in Tic Tac Toe

- The algorithm assumes that both players, the AI and the opponent, will play optimally and try to win the game.
- The game state is represented as a tree structure, where each node represents a possible state of the game, and the edges represent the moves that can be made to transition between states.
- The algorithm recursively explores the game tree by considering all possible moves from the current state.
- For each possible move, the algorithm evaluates the resulting game state by assigning a score to that move.
- If the current player is the AI, it selects the move with the highest score (maximizing player). If the current player is the opponent, it selects the move with the lowest score (minimizing player).
- The scores are assigned as follows:

- If the game is over and the AI wins, a positive score is assigned.
- If the game is over and the opponent wins, a negative score is assigned.
- If the game is over and it's a draw, a score of 0 is assigned.
- If the game is not over, the algorithm recursively continues exploring the game tree.
- The algorithm continues recursively until it reaches a terminal state (game over) and returns the score of that state.
- Finally, the AI selects the move with the highest score from the root node, which represents the optimal move to make.

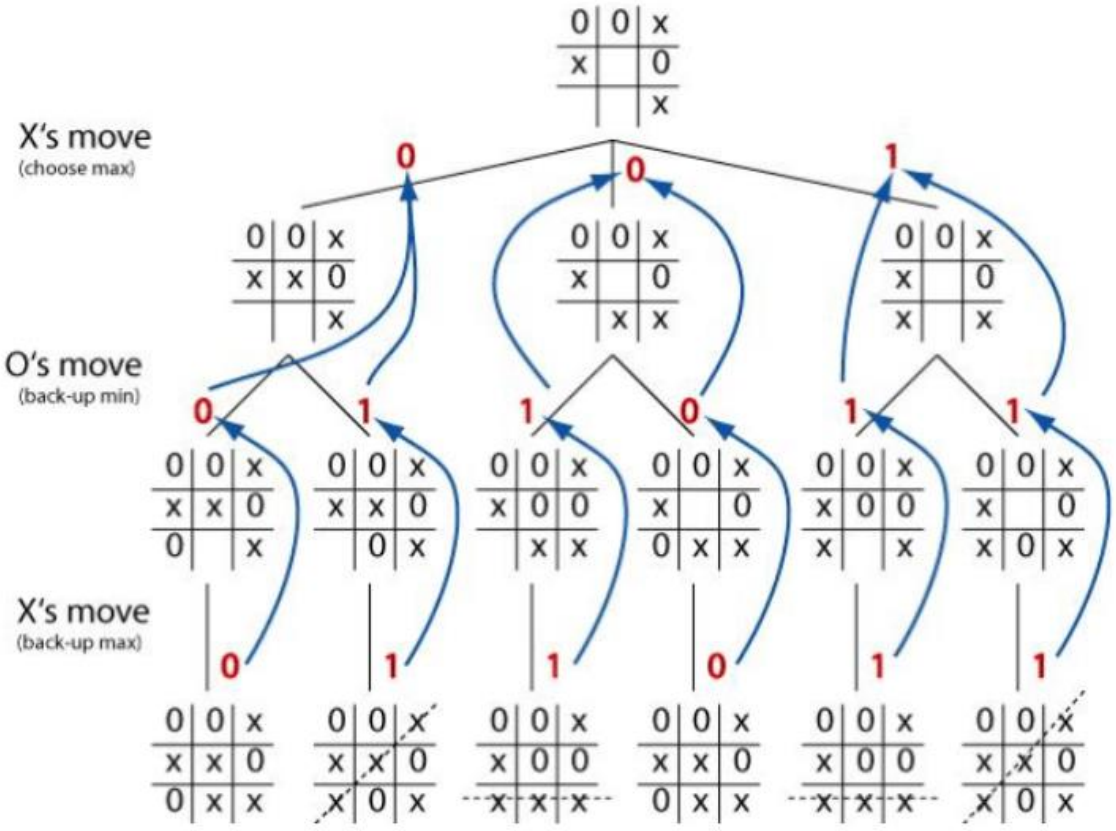

Fig 5.1 Illustrating MinMax algorithm implemented in Tic Tac Toe Game

5.4 Implementation Of MinMax Algorithm

```
        return True
    def compMove():
        bestScore = -800
        bestMove = 0
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = 'X'
                score = minimax(board, 0, False)
                board[key] = ' '
                if (score > bestScore):
                    bestScore = score
                    bestMove = key

        insertLetter('X', bestMove)
        return
```

Fig 5.2: Utility function where minmax algorithm is called recursively so the AI makes a move

The `compMove()` function is responsible for the AI player's move in the Tic-Tac-Toe game. Here's how it works:

- It initializes `bestScore` to a low value (-800) and `bestMove` to 0.
- It iterates through each key (board position) in the `board` dictionary.
- If the current position is empty (indicated by `' '`), it temporarily places an 'X' (AI's symbol) in that position using `board[key] = 'X'.
- It then calls the `minimax()` function to calculate the score for that move by passing the current `board`, `depth` (initially 0), and `isMaximizing` as `False`.
- After obtaining the score, the 'X' is removed from the position by assigning `' '` back to `board[key]`.
- If the calculated score is higher than the current `bestScore`, the `bestScore` and `bestMove` variables are updated.

- Once all possible moves are evaluated, the `compMove()` function calls `insertLetter()` to place the 'X' symbol in the chosen `bestMove` position on the board.

Finally, it returns from the function. In summary, the `compMove()` function iterates through each empty position on the board, temporarily places an 'X' in each position, evaluates the score for that move using the `minimax()` function, and selects the move with the highest score as the AI's optimal move.

```
def minimax(board, depth, isMaximizing):
    if (checkWhichMarkWon('X')):
        return 1
    elif (checkWhichMarkWon('O')):
        return -1
    elif (checkDraw()):
        return 0

    if (isMaximizing):
        bestScore = -800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = 'X'
                #key["text"]='X'
                score = minimax(board, depth + 1, False)
                board[key] = ' '
                if (score > bestScore):
                    bestScore = score
        return bestScore

    else:
        bestScore = 800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = 'O'
                score = minimax(board, depth + 1, True)
                board[key] = ' '
                if (score < bestScore):
                    bestScore = score
        return bestScore
```

Fig 5.3: Utility function that implements MinMax algorithm

The minimax() function is a recursive function that evaluates the game state and returns a score based on the outcome. Here's how it works:

- The function begins by checking for win conditions using the checkWhichMarkWon() function. If 'X' wins, it returns a score of 1. If 'O' wins, it returns a score of -1. If it's a draw, it returns a score of 0.

- If it's the maximizing player's turn (indicated by isMaximizing being True), the function initializes bestScore to a low value (-800).
- It then iterates through each key (board position) in the board dictionary.
- If the current position is empty (indicated by ' '), it temporarily places an 'X' in that position using board[key] = 'X'.
- The minimax() function is called recursively with the updated board, depth + 1, and False (indicating it's now the minimizing player's turn).
- After the recursive call, the 'X' is removed from the position by assigning ' ' back to board[key].
- If the returned score is greater than the current bestScore, bestScore is updated.
- Once all possible moves are evaluated, the function returns the bestScore.

- If it's the minimizing player's turn (indicated by isMaximizing being False), the process is similar, but this time it initializes bestScore to a high value (800) and updates it if a lower score is found.

In summary, the minimax() function uses recursion to evaluate the game state and returns the best score for the current player. It considers all possible moves, alternates between the maximizing and minimizing player, and updates the bestScore accordingly. The algorithm continues recursively until it reaches a terminal state (win, loss, or draw) and propagates the best score back up the call stack.

5.5 Advantages and Limitations

The Minimax algorithm offers several advantages in the game of Tic-Tac-Toe. It ensures optimal decision-making by guaranteeing that the AI player will make the best move possible, assuming the opponent also plays optimally. The algorithm strategically evaluates all possible moves and their outcomes, allowing the AI player to anticipate and counter the opponent's moves while aiming for winning combinations. Furthermore, the algorithm exhaustively searches the game tree, leaving no favourable move unexplored.

However, the Minimax algorithm has certain limitations. Its time complexity can be high, especially as the game progresses and the number of available moves decreases. Additionally, the algorithm assumes that both players make no mistakes and play optimally, which may not hold true in real-world scenarios. Moreover, the algorithm's scalability can be an issue for games with larger solution spaces. Furthermore, the algorithm lacks adaptability and does not learn from previous games, relying solely on the current game state for decision-making. Despite these limitations, the Minimax algorithm serves as a solid foundation for AI gameplay in Tic-Tac-Toe and has paved the way for further research and advancements in game-playing algorithms.

CHAPTER 6

DECISION TREE

6.1 Introduction

Prediction algorithms, also known as machine learning algorithms or models, are computational methods that are designed to make predictions or forecasts based on patterns and relationships found in data. These algorithms analyze input data and learn from it to make predictions about future or unseen instances. Prediction algorithms can be used for a wide range of tasks, such as classification, regression, clustering, and recommendation. They are commonly used in fields like finance, healthcare, marketing, weather forecasting, and many others where making accurate predictions is valuable. One of the prediction algorithm is Decision Tree. It is used in the project to find the win status at the current state.

6.2 Dataset Information

This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row").

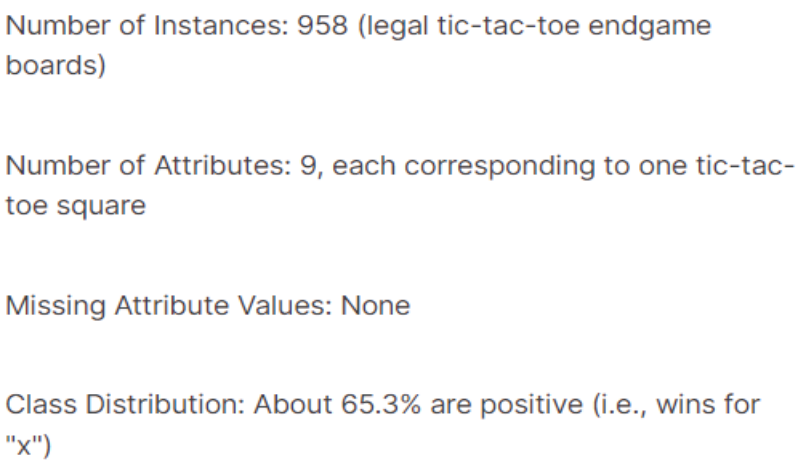

Fig 6.1: Attribute and Instance Information

Attribute Information

- V1=top-left-square:{x,o,b}
- V2=top-middle-square:{x,o,b}
- V3=top-right-square:{x,o,b}
- V4=middle-left-square:{x,o,b}
- V5=middle-middle-square:{x,o,b}
- V6=middle-right-square:{x,o,b}
- V7=bottom-left-square:{x,o,b}
- V8=bottom-middle-square:{x,o,b}
- V9 = bottom-right-square: {x,o,b}
- V10 = Class: {positive,negative}

6.3 Algorithm

A decision tree algorithm is a popular machine learning technique for solving classification and regression problems. Here's a high-level overview of the algorithm:

1.**Data Preparation**: Start with a dataset consisting of input features and corresponding labels. Make sure the data is cleaned and preprocessed appropriately.

2.**Feature Selection**: Determine the most relevant features for the problem at hand. Various methods like information gain, Gini index, or chi-square test can be used to evaluate the importance of different features.

3.**Tree Construction**:

→Select the best feature to split the dataset based on a predefined criterion (e.g., information gain or Gini index).

→Create a node for the selected feature.

→Divide the dataset into subsets based on the possible values of the selected feature.

→Recursively repeat the above steps for each subset until a stopping criterion is met (e.g., maximum depth is reached or a minimum number of samples remain in a node).

→If the stopping criterion is met, create a leaf node and assign the majority class (for classification) or the mean value (for regression) of the samples in that node.

→**Tree Pruning**: After the tree is constructed, it may suffer from overfitting. Pruning techniques like cost complexity pruning (or reduced error pruning) can be applied to simplify the tree and improve generalization.

→**Prediction**: To make predictions, traverse the decision tree from the root node based on the feature values of the input sample until a leaf node is reached. The label assigned to that leaf node is the predicted class (for classification) or the predicted value (for regression).

→**Evaluation**: Assess the performance of the decision tree algorithm using appropriate evaluation metrics such as accuracy, precision, recall, or mean squared error.

→**Optional Optimization Techniques**: Depending on the specific implementation, there are additional techniques that can enhance the decision tree algorithm, such as random feature selection (e.g., random forests), early stopping criteria, or handling missing values.

It's important to note that there are different variations of decision tree algorithms, such as ID3, C4.5, CART (Classification and Regression Trees), and Random Forests. Each algorithm may have slight differences in terms of the splitting criterion or pruning strategy, but the general principles remain similar.

Working of Decision Tree Algorithm

The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures (ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.
3. Start tree building by repeating this process recursively for each child until one of the conditions will match:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.

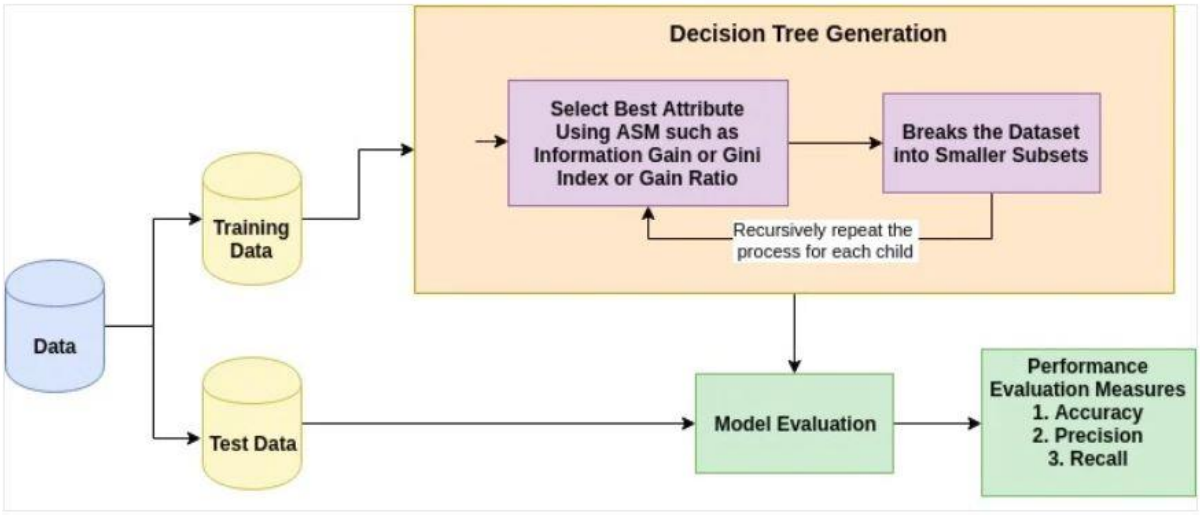

Fig 6.2 Working of Decision tree algorithm

6.4 Attribute Selection Measures

Attribute selection measure is a heuristic for selecting the splitting criterion that partitions data in the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature (or attribute) by explaining the given dataset. The best score attribute will be selected as a splitting attribute (**Source**). In the case of a continuous-valued attribute, split points for branches also need to define. The

most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain

Claude Shannon invented the concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy is referred to as the randomness or the impurity in a system. In information theory, it refers to the impurity in a group of examples. Information gain is the decrease in entropy. Information gain computes the difference between entropy before the split and average entropy after the split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain. Where Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\text{Info}_A(D)=\sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

Fig 6.3: Entropy calculation Formula

$$\text{Gain}(A)=\text{Info}(D)- \text{Info}_A(D)$$

Fig 6.4: Gain Calculation Formula

Where:

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- |Dj|/|D| acts as the weight of the jth partition.
- InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

6.5 Proposed Model

Implementing Decision Tree involves following steps:

1.Importing Required Libraries

```
: import pandas as pd
  from sklearn.tree import DecisionTreeClassifier
  from sklearn.model_selection import train_test_split
  from sklearn import metrics
  from sklearn.metrics import confusion_matrix
  from sklearn.metrics import accuracy_score
```

Fig 6.5 Imported Libraries

- Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.
- Scikit-Learn, also known as sklearn is a python library to implement machine learning models and statistical modelling. Through scikit-learn, we can implement various machine learning models for regression, classification, clustering, and statistical tools for analyzing these models.

2.Loading Data

First load the required dataset using pandas' read CSV function. Dataset is downloaded from Kaggle and the details of the dataset are mentioned above in 6.2.

```
col_name=["TL","TM","TR","ML","MM","MR","BL","BM","BR","goal"]
data_set=pd.read_csv(r"C:\Users\maha9\OneDrive\Documents\IV_semester\ai&da_miniproj\tic-tac-toe.csv", header=0, names=col_name)
print(data_set)
```

Fig 6.6 Loading the dataset

	TL	TM	TR	ML	MM	MR	BL	BM	BR	goal
0	2	2	2	2	1	1	2	1	1	1
1	2	2	2	2	1	1	1	2	1	1
2	2	2	2	2	1	1	1	1	2	1
3	2	2	2	2	1	1	1	0	0	1
4	2	2	2	2	1	1	0	1	0	1
..	..	..	..	..	..	..	..	..	..	...
953	1	2	2	2	1	1	1	2	2	0
954	1	2	1	2	2	1	2	1	2	0
955	1	2	1	2	1	2	2	1	2	0
956	1	2	1	1	2	2	2	1	2	0
957	1	1	2	2	2	1	1	2	2	0

[958 rows x 10 columns]

Fig 6.7 Dataset

3. Feature Selection

Here, you need to divide given columns into two types of variables dependent (or target variable) and independent variable (or feature variables).

```
cal_process_feature=["TL","TM","TR","ML","MM","MR","BL","BM","BR"]
target_feature=["goal"]
X=data_set[cal_process_feature]
Y=data_set[target_feature]
```

Fig 6.8 Feature Selection

4.Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy. This can be spilitted using train_test_split( ).

```
X_train,X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=100)
```

Fig 6.9 Splitting the dataset

5.Building Decision Tree Model

Creating a decision tree model using Scikit-learn. DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset. In case

that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes.

```
clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 42,
            max_depth = 3, min_samples_leaf = 5)
clf_entropy.fit(X_train, y_train)
```

Fig 6.10 Decision Tree Creation

6.6 Visualization

A decision tree visualization is used to illustrate how underlying data predicts a chosen target and highlights key insights about the decision tree. A decision rule predicts an outcome in the target field.

```
from sklearn import tree
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred=clf_entropy.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred=y_pred))
tree.plot_tree(clf_entropy)
plt.show()
cm=confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot();
```

Fig 6.11 Plotting decision tree using pyplot from matplotlib and tree from sklearn

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. plot_tree is an inbuilt function that automatically plots the decision tree created.

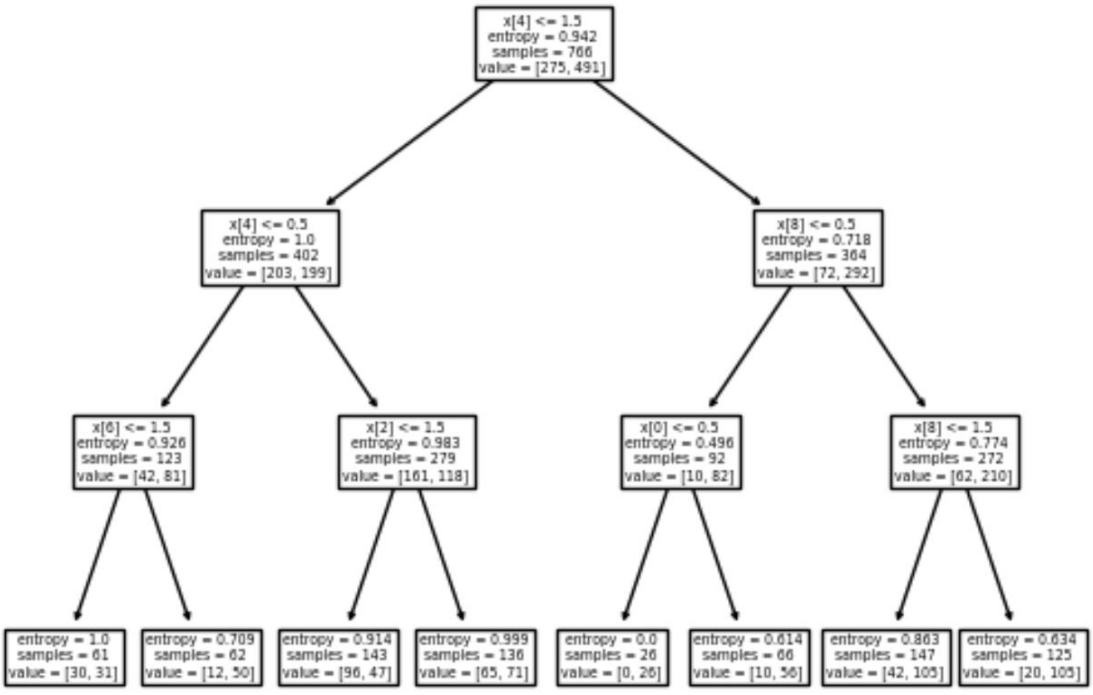

Fig 6.12 Decision Tree Plot

It is using a binary tree graph (each node has two children) to assign for each data sample a target value. The target values are presented in the tree leaves. To reach to the leaf, the sample is propagated through nodes, starting at the root node. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

6.7 Model Evaluation

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses. Model evaluation is important to assess the efficacy of a model during initial research phases, and it also plays a role in model monitoring.

The three main metrics used to evaluate a classification model are accuracy, precision, and recall.

sklearn.metrics Library

The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

Confusion Matrix

A confusion matrix is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm.

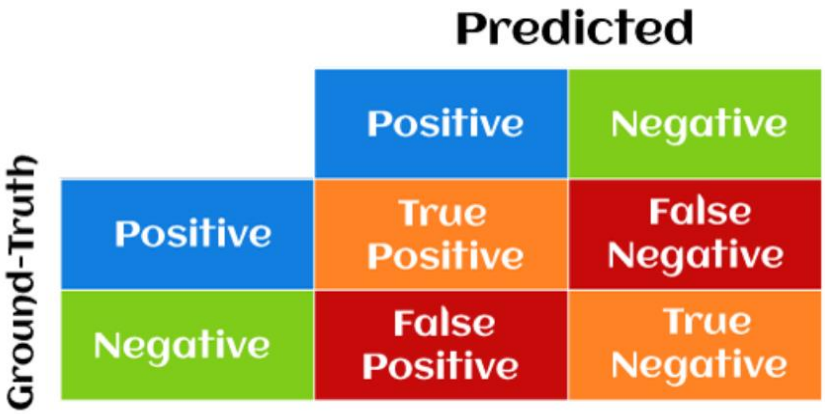

Fig 6.13: Confusion Matrix

This matrix consists of 4 main elements that show different metrics to count a number of correct and incorrect predictions. Each element has two words either as follows: True or False, Positive or Negative

If the predicted and truth labels match, then the prediction is said to be correct, but when the predicted and truth labels are mismatched, then the prediction is said to be incorrect. Further, positive and negative represents the predicted labels in the matrix.

There are four metrics combinations in the confusion matrix, which are as follows:

- True Positive: This combination tells us how many times a model correctly classifies a positive sample as Positive?
- False Negative: This combination tells us how many times a model incorrectly classifies a positive sample as Negative?
- False Positive: This combination tells us how many times a model incorrectly classifies a negative sample as Positive?
- True Negative: This combination tells us how many times a model correctly classifies a negative sample as Negative?

```
cm=confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot();
```

Fig 6.14: Python code to generate confusion matrix and visualize it

confusion_matrix is an inbuilt function that summarizes the predicted and actual values of a classification model and helps visualize it as matrix.

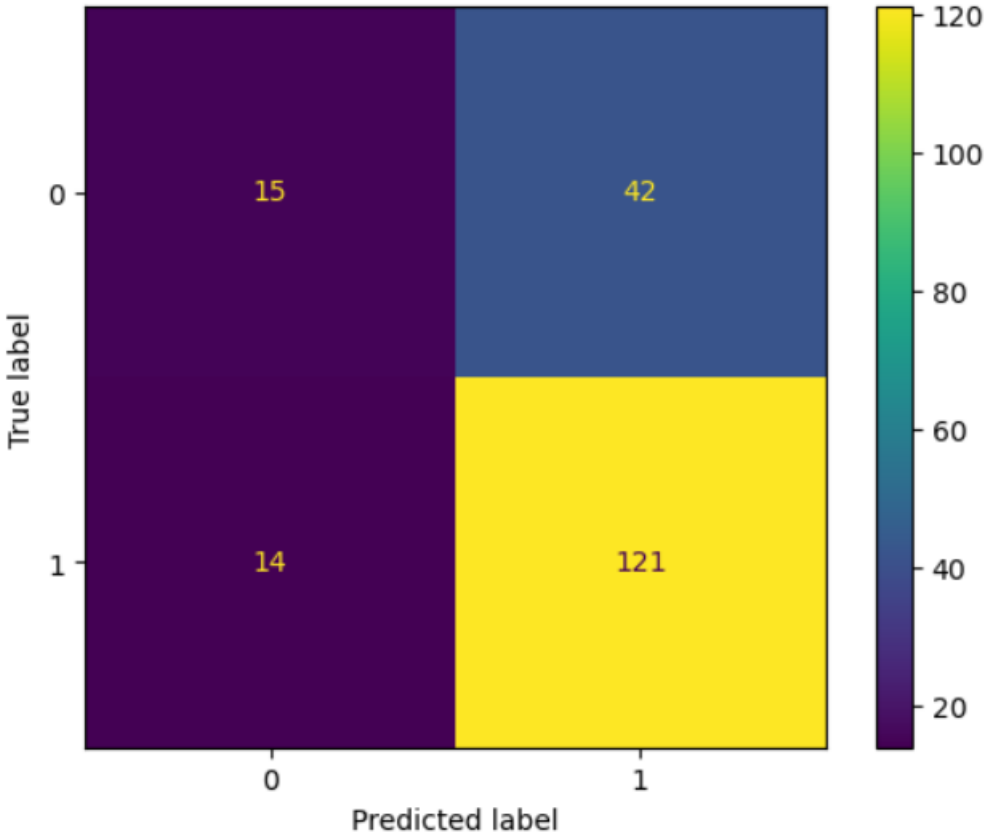

Fig 6.15 Visualization of confusion Matrix

Accuracy

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions. It's a way of assessing the performance of a model.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

Fig 6.16 Accuracy Formula

```
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred=clf_entropy.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred=y_pred))
```

Fig 6.17 Python code to calculate accuracy of the trained model

```
Train data accuracy: 0.7049608355091384
Test data accuracy: 0.7083333333333334
```

Fig 6.18: Accuracy of the model trained

Recall

Recall, also known as sensitivity, is calculated by dividing the number of positive samples classified correctly as positive by the total number of positive samples. It measures a model's ability to detect positives—the higher its recall, the more positives are detected. The higher the recall, the more positive samples detected.

$$\text{Recall} = \frac{TP}{TP+FN}$$

Fig 6.19: Recall Formula

```python
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```

```
Recall: 0.8962962962962963
```

Fig 6.20: Recall calculation code and corresponding value

Precision

Precision is one indicator of a machine learning model's performance – the quality of a positive prediction made by the model. Precision refers to the number of true positives divided by the total number of positive predictions (i.e., the number of true positives plus the number of false positives).

$$Precision = \frac{TP}{(TP + FP)}$$

Fig 6.21: Precision Formula

```
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Precision:", precision)
```

```
Precision: 0.7423312883435583
```

Fig 6.22: Precision calculation code and corresponding value

F1-Score

F1-Score is also an evaluation parameter calculated using both precision and recall. The formula to calculate the F1-score is as follows

$$F_1\text{−score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Fig 6.23: F1-score Formula

```
: from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("F1 Score:", f1)
```

```
F1 Score: 0.8120805369127516
```

Fig 6.24: F1-score calculation code and corresponding value

CHAPTER 7

IMPLEMENTATION

7.1 Block Diagram

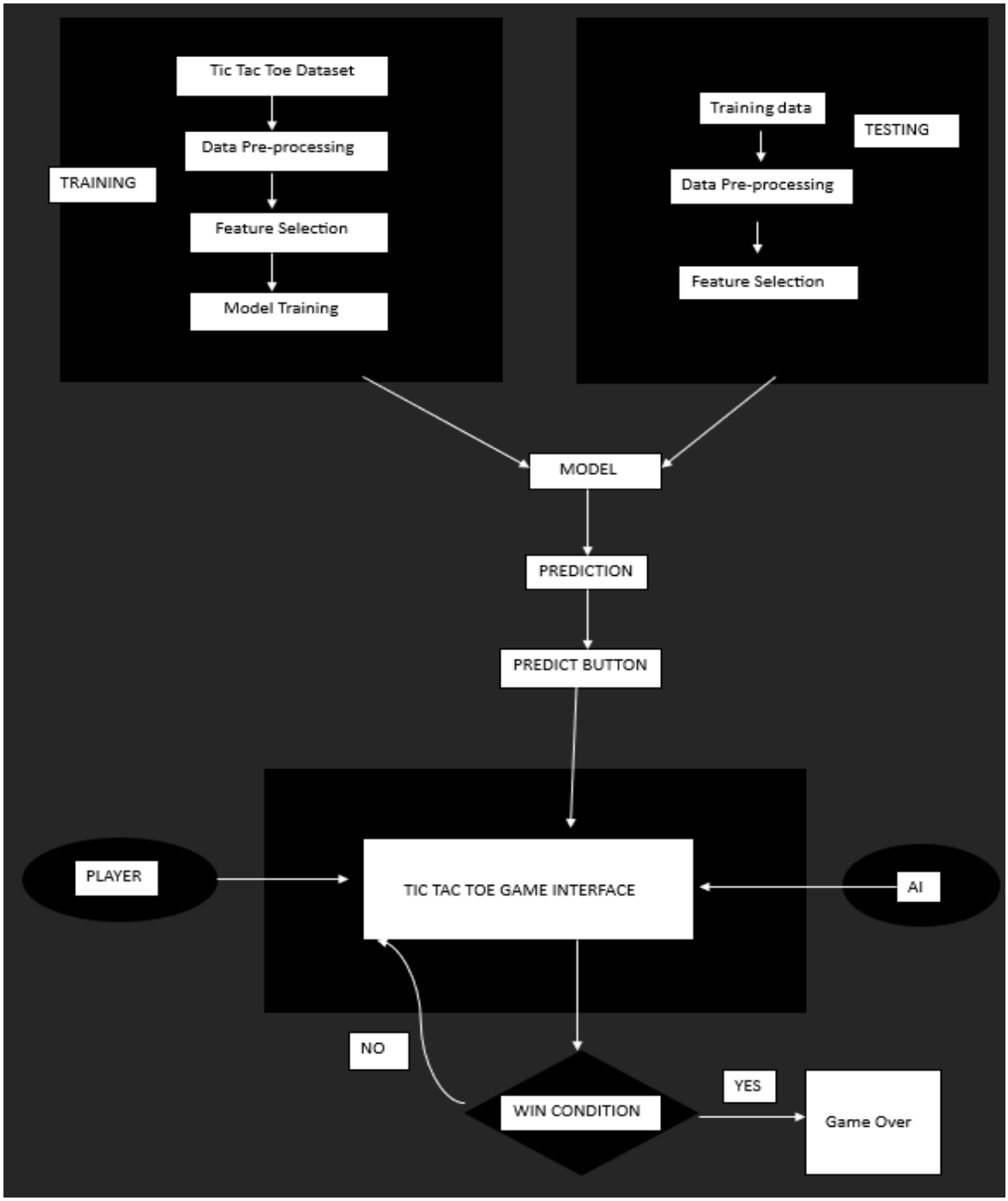

Fig 7.1: Block diagram of the project

The block diagram depicts that the decision tree model is created and trained. At first the Tic Tac Toe dataset is downloaded from the Kaggle website

and preprocessed then split into train and test dataset. Then the trained dataset is used to train the model. Features are selected to train the model. Following it the test dataset is used to predict the accuracy of the model. Then this created as utility function and called on clicking the predict button in the user interface.

A Tic Tac Toe Game interface is created for the user to play and AI makes the opposite move following the minmax algorithm once the player made the move. Correspondingly the winning conditions are checked according to the game constraints then once user or AI wins the game gets over otherwise the game will be draw.

7.2 Source code

```
from tkinter import *
import tkinter.font as f
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk,Image
import pandas as pd
import numpy as np
clicked=True
count=0
buttons=[]
board={}
from deci import main1
def call_decision_tree():
    clf_object1=main1()
    return clf_object1
def prediction(X_test, clf_object):
    # Predicton on test with entrophy
    y_pred = clf_object.predict(X_test)
    if y_pred==1:
        messagebox.showinfo("TIC TAC TOE","Max (AI) player has more
chances to win, play harder")
    else:
        messagebox.showinfo("TIC  TAC  TOE","Min  (User)  player  has
more chances to win, Try")

    return y_pred

##function to get input from grid##

def getboardinput():
    for i in buttons:
        mytext=i.cget('text')
        board[i]=mytext

##function to convert dic data into dataframe to predict the current
input from game##

def dic_list():
    getboardinput()
    ip=[]
    for i in board:
        ip.append(board[i])
    model_ip=[]
    model_ip2=[]
    for i in ip:
        if i=='X':
            model_ip2.append(1)
```

```
        elif i=='O':
            model_ip2.append(0)
        else:
            model_ip2.append(2)
    model_ip.append(model_ip2)

model_ip1=pd.DataFrame(model_ip,columns=["TL","TM","TR","ML","MM","
MR","BL","BM","BR"])
    return model_ip1

## function to call the model ##
def give_input():
    clf_object=call_decision_tree()
    X_test=dic_list()
    prediction(X_test,clf_object)

## checking for free space ##

def spaceIsFree(position):
    if board[position] == ' ':
        return True
    else:
        return False
## getting the current button press ##
def whichButton(button_press):
     return button_press
## fixing the position of the 'X' OR 'O' ##

def insertLetter(letter, position):
    if spaceIsFree(position):
        board[position] = letter
        if(letter=='X'):
            position["text"]='X'
        else:
            position["text"]='O'
        if (checkDraw()):
            messagebox.showinfo("Win status","Draw")
            return
        if checkForWin():
            if letter == 'X':
                messagebox.showinfo("Win status","AI wins!")
                return
            else:
                messagebox.showinfo("Win status","Player wins!")
                return
        return

    else:
        return
```

```
## Wining conditions ##
def checkForWin():
    if (board[B1] == board[B2] and board[B1] == board[B3] and
board[B1] != ' '):
        return True
    elif (board[B4] == board[B5] and board[B4] == board[B6] and
board[B4] != ' '):
        return True
    elif (board[B7] == board[B8] and board[B7] == board[B9] and
board[B7] != ' '):
        return True
    elif (board[B1] == board[B4] and board[B1] == board[B7] and
board[B1] != ' '):
        return True
    elif (board[B2] == board[B5] and board[B2] == board[B8] and
board[B2] != ' '):
        return True
    elif (board[B3] == board[B6] and board[B3] == board[B9] and
board[B3] != ' '):
        return True
    elif (board[B1] == board[B5] and board[B1] == board[B9] and
board[B1] != ' '):
        return True
    elif (board[B7] == board[B5] and board[B7] == board[B3] and
board[B7] != ' '):
        return True
    else:
        return False
## function to find which mark won ##

def checkWhichMarkWon(mark):
    if board[B1] == board[B2] and board[B1] == board[B3] and
board[B1] == mark:
        return True
    elif (board[B4] == board[B5] and board[B4] == board[B6] and
board[B4] == mark):
        return True
    elif (board[B7] == board[B8] and board[B7] == board[B9] and
board[B7] == mark):
        return True
    elif (board[B1] == board[B4] and board[B1] == board[B7] and
board[B1] == mark):
        return True
    elif (board[B2] == board[B5] and board[B2] == board[B8] and
board[B2] == mark):
        return True
    elif (board[B3] == board[B6] and board[B3] == board[B9] and
board[B3] == mark):
        return True
```

```
    elif (board[B1] == board[B5] and board[B1] == board[B9] and
board[B1] == mark):
        return True
    elif (board[B7] == board[B5] and board[B7] == board[B3] and
board[B7] == mark):
        return True
    else:
        return False
## function to check draw state ##
def checkDraw():
    for key in board.keys():
        if (board[key] == ' '):
            return False
    return True
## function for ai turn to play ##
def compMove():
    bestScore = -800
    bestMove = 0
    for key in board.keys():
        if (board[key] == ' '):
            board[key] = 'X'
            score = minimax(board, 0, False)
            board[key] = ' '
            if (score > bestScore):
                bestScore = score
                bestMove = key

    insertLetter('X', bestMove)
    return

## minmax algorithm function ##

def minimax(board, depth, isMaximizing):
    if (checkWhichMarkWon('X')):
        return 1
    elif (checkWhichMarkWon('O')):
        return -1
    elif (checkDraw()):
        return 0

    if (isMaximizing):
        bestScore = -800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = 'X'
                #key["text"]='X'
                score = minimax(board, depth + 1, False)
                board[key] = ' '
                if (score > bestScore):
```

```
                    bestScore = score
        return bestScore

    else:
        bestScore = 800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = 'O'
                score = minimax(board, depth + 1, True)
                board[key] = ' '
                if (score < bestScore):
                    bestScore = score
        return bestScore



## button click function ##

def b_click(b):
    global clicked,count
    if clicked==True:
        position = b
        #print(position)
        player='O'
        insertLetter(player, position)

        clicked=True
        count+=1
        compMove()
        getboardinput()
        #give_input()
    else:
        messagebox.showerror("TIC  TAC  TOE","HEY!  That  box  has
already been selected\nPick Another Box")

## main window ##

root=Tk()
root.title("Home page")
root.minsize(640,400)
bg_frame=Image.open(r"C:\Users\maha9\OneDrive\Documents\IV_semester
\ai&da_miniproj\bg.jpg")
photo=ImageTk.PhotoImage(bg_frame)
bg_panel=Label(root,image=photo)
bg_panel.image=photo
bg_panel.pack(fill='both',expand='yes')
gridframe=Frame(root,width=900,height=600)
gridframe.place(x=450,y=135)
B1=Button(gridframe,text="
```

```
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B1)])
B2=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B2)])
B3=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B3)])
B4=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B4)])
B5=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B5)])
B6=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B6)])
B7=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B7)])
B8=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B8)])
B9=Button(gridframe,text="
",font=("Helvetica",20),height=3,width=6,bg="SystemButtonFace",comm
and=lambda: [b_click(B9)])
buttons=[B1,B2,B3,B4,B5,B6,B7,B8,B9]
board={B1:" ",B2:" ",B3:" ",
       B4:" ",B5:" ",B6:" ",
       B7:" ",B8:" ",B9:" "}
predict_button=Button(root,text="Predict",font=("Helvetica",20),hei
ght=2,width=6,bg="SystemButtonFace",command=lambda:give_input())
predict_button.place(x=50,y=440)
txt='Play and Predict'
heading=Label(bg_panel,text=txt,bg='#9C661F',font=('Helvetica',20,'
italic'))
heading.place(x=450,y=50,width=400,height=50)
B1.grid(row=0,column=0)
B2.grid(row=0,column=1)
B3.grid(row=0,column=2)
B4.grid(row=1,column=0)
B5.grid(row=1,column=1)
B6.grid(row=1,column=2)
B7.grid(row=2,column=0)
B8.grid(row=2,column=1)
B9.grid(row=2,column=2)
```

```
root.mainloop()
```

##deci.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
def getdata_set():
  #column names
  col_name=["TL","TM","TR","ML","MM","MR","BL","BM","BR","goal"]
  #reading file

data_set=pd.read_csv(r"C:\Users\maha9\OneDrive\Documents\IV_semeste
r\ai&da_miniproj\tic-tac-toe.csv", header=0, names=col_name)
  #print(data_set.head(10))
  return data_set
def spiliting_dataset(data_set):

cal_process_feature=["TL","TM","TR","ML","MM","MR","BL","BM","BR"]
  target_feature=["goal"]
  X=data_set[cal_process_feature]
  Y=data_set[target_feature]
  return X,Y
def split_for_model_training(X,Y):
  X_train,X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=100)
  return X_train, X_test, y_train
def train_using_entropy(X_train, X_test, y_train):
    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 42,
            max_depth = 3, min_samples_leaf = 5)
    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy
def main1():
    data_set=getdata_set()
    X,Y=spiliting_dataset(data_set)
    X_train, X_test, y_train=split_for_model_training(X,Y)
    clf_object=train_using_entropy(X_train, X_test, y_train)
    return clf_object
```

CHAPTER 8

RESULTS

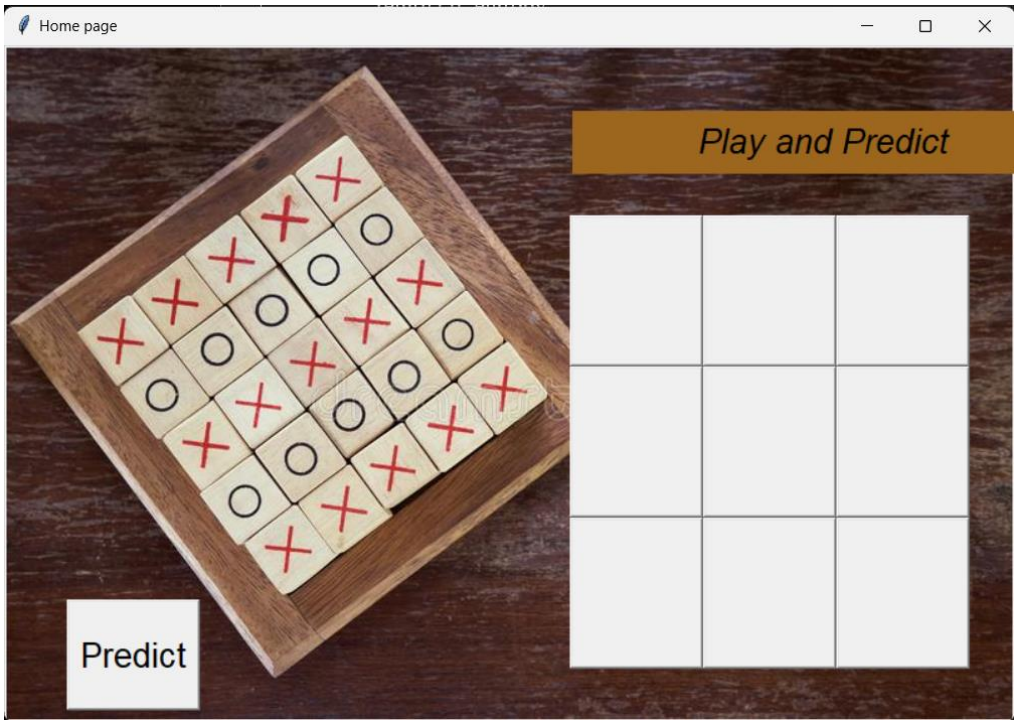

Fig 8.1 Tic Tac Toe game interface

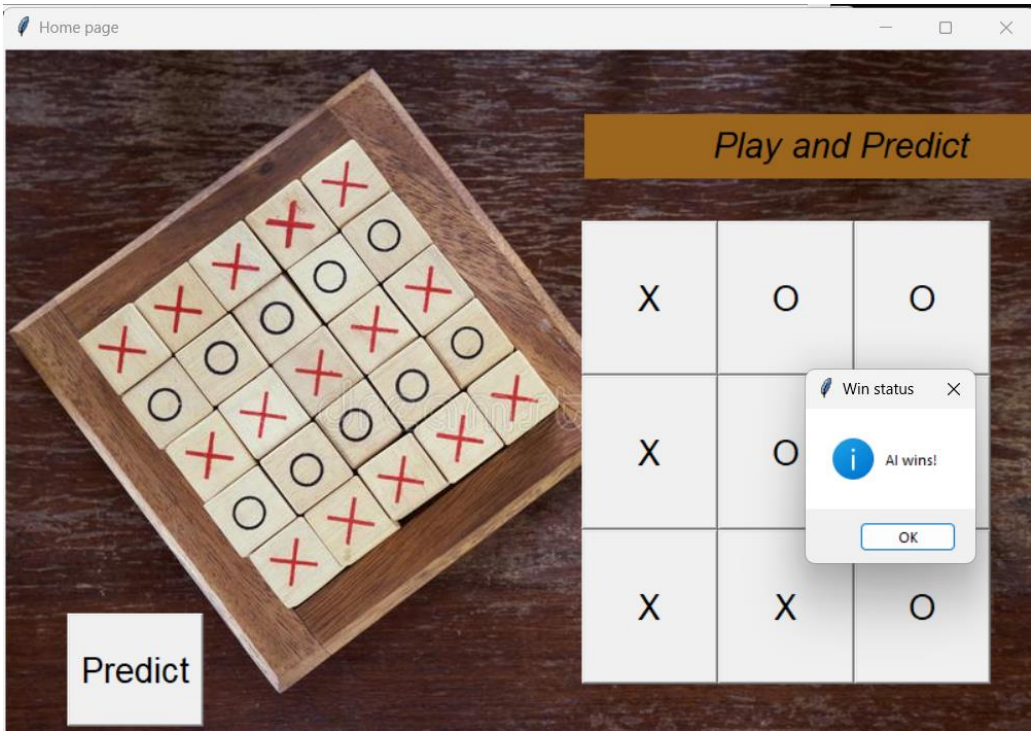

Fig 8.2 AI wins the game

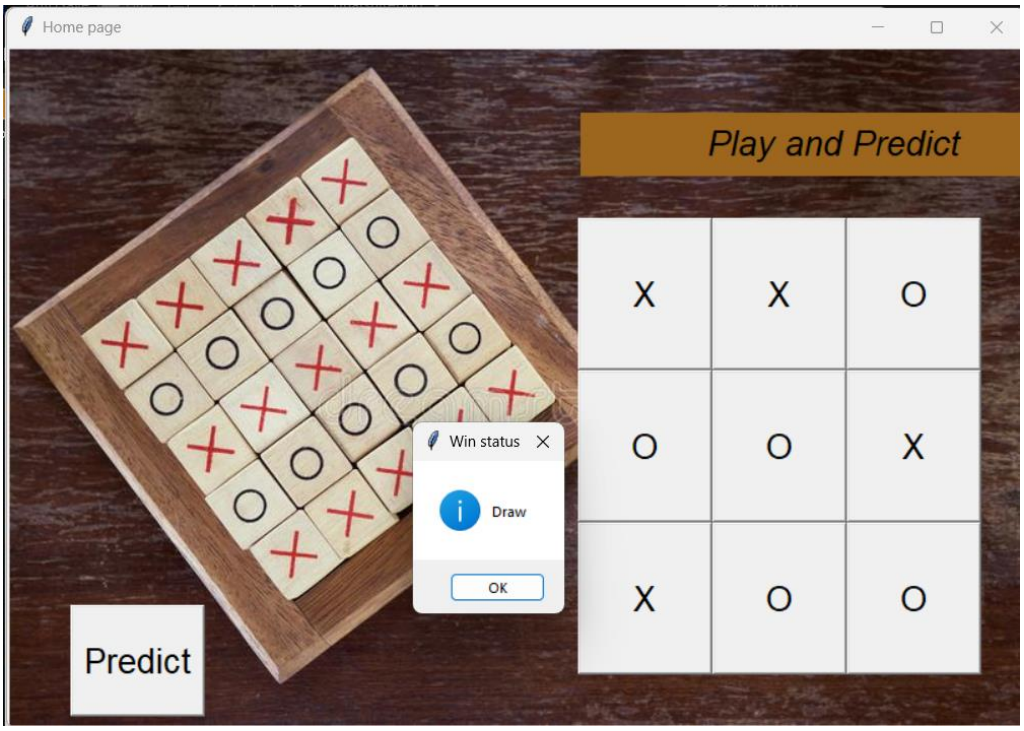

Fig 8.3 Game Draw

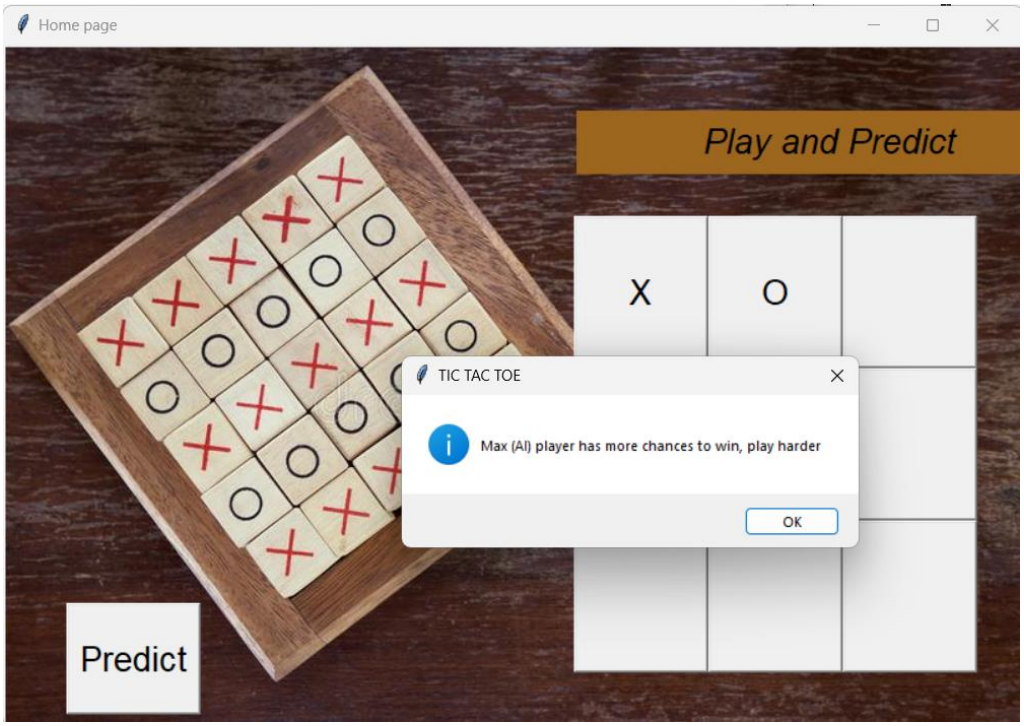

Fig 8.4 Predicting the winner with the current state of the game

CHAPTER 9

CONCLUSION

The implementation of the Minimax algorithm in the Tic-Tac-Toe project offers a powerful approach to creating an AI player that can make optimal moves and predict the outcome of the game at the current state. The Minimax algorithm ensures strategic gameplay by considering all possible moves and their subsequent outcomes, allowing the AI player to anticipate the opponent's moves and aim for winning combinations. By recursively evaluating the game state and utilizing the Minimax algorithm, the AI player can make informed decisions that maximize its chances of winning or minimize the opponent's chances. The algorithm takes into account both immediate wins or losses and future possibilities, leading to intelligent and strategic gameplay. The prediction capability of the algorithm allows the AI player to assess the game state and predict the final outcome, whether it is a win, loss, or draw. This provides valuable insights into the game's progress and aids in decision-making. However, it is important to note that the Minimax algorithm has certain limitations, such as its high time complexity and assumption of optimal play from both players. These limitations may affect its scalability to larger game scenarios and real-world variations. Overall, the integration of the Minimax algorithm into the Tic-Tac-Toe project enhances the gameplay experience by providing an intelligent and challenging opponent. The project demonstrates the power of decision-making algorithms in game environments and highlights the potential for further exploration and enhancements in AI gameplay

REFERENCES

- https://docs.python.org/3/library/tk.html
- https://www.datacamp.com/tutorial/decision-tree-classification-python
- https://www.javatpoint.com/mini-max-algorithm-in-ai