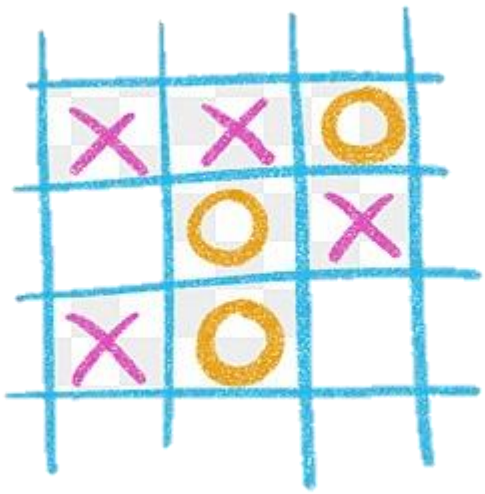
PROJECT REPORT

Project Name: Tic-Tac-Toe with Artificial Intelligence



S M Asiful Islam Saky

Contents

ABSTRACT

Tic Tac Toe is a classic paper-and-pencil game played on a grid of 3x3 squares. The idea of the game is to be the first to get three of your marks (either 'X' or 'O') in a row, either horizontally, vertically, or diagonally to win the game. In this report we will try to discover about the game from when it started, the objectives of the game, what artificial intelligence is and how to apply the game using artificial intelligence, what motivate us to do this game using artificial intelligence, why do we care about the game system, what type of algorithm we use, the description of the project, appendix code implementation and overall information about the being developed using artificial intelligence.

INTRODUCTION

Tic-tac-toe also known as nought and crosses is a paper and pencil game for two players, who take turns marking the spaces in a 3x 3 grid traditionally. The player who succeeds in placing three of their marks in a horizontal, vertical or diagonal row wins the game. It is a zero-sum of perfect information game. This means that it is deterministic, with fully observable environments in which two agents act alternately and the utility values at the end of the game are always equal and opposite. Because of the simplicity of tic-tac-toe, it is often used as pedagogical tool in artificial intelligence to deal with searching of game trees. The optimal move for this game can be gained by using mini max algorithm, where the opposition between the utility functions makes the situation adversarial, hence requiring adversarial search supported by mini ma algorithm with alpha beta pruning concept in artificial intelligence.

This game was first introduced at ancient time, however there is no evidence who invented it and which year. Some people think this game was invented at Ancient Egypt, and then Roman Empire called this game "Terni Lapilli". The grid drawing for the game had been found chalked all over the ancient city's ruins. Terni Lapilli was resurfaced in England with the name "Nought and Crosses" in 1864. This resurfaced version is the modern of Tic-Tac-Toe game that people know until this present day. In 1952, Alexander S. Douglas for the EDSAC computer at University of Cambridge developed a computerized Tic-Tac- Toe game called "OXO". This was the first video game of Tic-Tac-Toe and it has Al inside, therefore human could play against the computer opponent.

OBJECTIVES

1. To develop Artificial intelligence-based tic-tac-toe game for human Vs AI by implementing mini max algorithm with adversarial search concept.
2. To analyse the complexity of mini max algorithm through 4x4 tic tac toe game.
3. To study and implement alpha-beta pruning concept for improved speed of searching the optimal choice in tic-tac-toe game.
4. To study optimizing methods for alpha-beta pruning using heuristic evaluation function.

What is Artificial Intelligence?

In very simple terms, artificial intelligence is when you enable the computer to think like how a human would: predicting the weather for the next day, recognizing animals, considering which move to play next, are some examples of Artificial Intelligence.

MOTIVATION

The Tic-tac-toe game serves as an excellent platform for exploring various concepts in game development, artificial intelligence, and user interface design. By building an advanced version of the game, we can:

- Demonstrate the application of AI algorithms, such as the mini max algorithm, in creating intelligent game-playing agents.
- Showcase the implementation of a graphical user interface (GUI) using Python libraries like Tkinter, enhancing user interaction and experience.
- Provide an opportunity for users to challenge themselves against a computer opponent, honing their strategic thinking and decision-making skills.

WHY DO WE CARE?

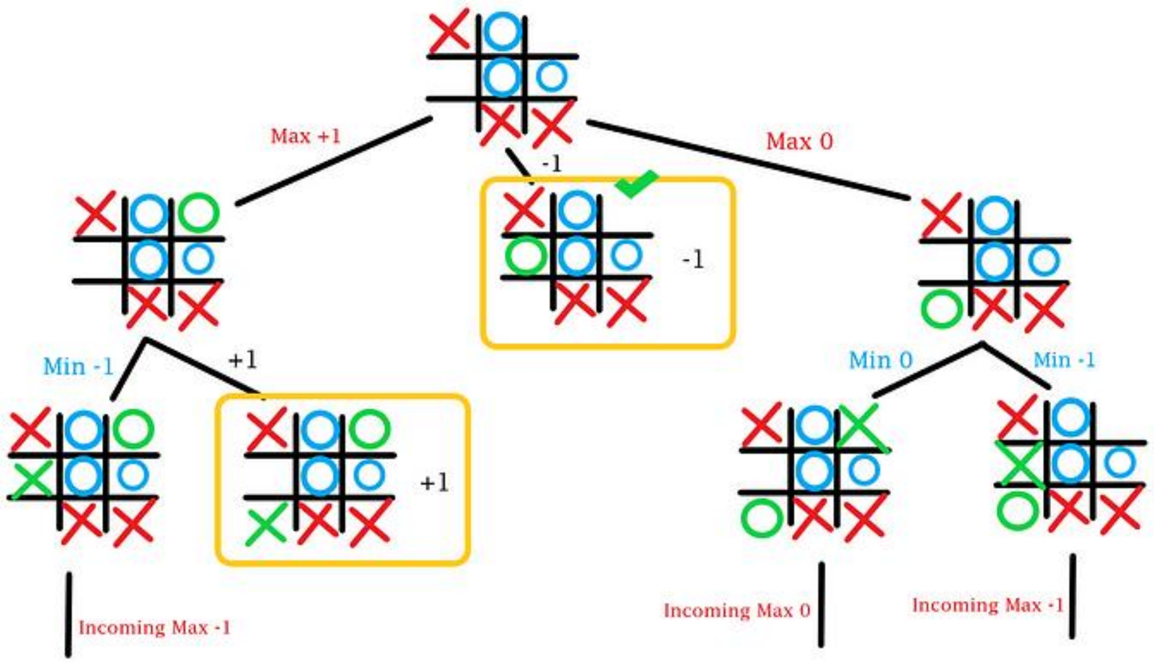
While Tic Tac Toe may not have significant real-world applications on its own, it serves as a stepping stone for understanding more complex games, such as chess or go. The knowledge gained from developing a Tic Tac Toe program can be extended to bud Al-based game playing agents or even be applied in the design of more sophisticated Al systems. Moreover, exploring the game helps in fostering logical thinking, problem solving skills, and algorithmic understandings.

ALGORITHM USED

Mini max is a recursive algorithm which is used to choose an optimal move for a player assuming that the opponent is also playing optimally. Its objective is to minimize the maximum loss. This algorithm is based on adversarial search technique. In a normal search problem, the optimal solution would be a sequence of actions leading to a goal state. Rather in adversarial search MAX finds the contingent strategy, which specifies the MAX': moves in the initial state, then MAX's moves in the states resulting from every possible response by MIN and continues till the termination condition comes alternately. Furthermore, given a choice, MAX prefers to move to state of maximum value whereas MIN prefers a state of minimum value.

Understanding the Minimax Algorithm

Consider the graph diagram below:



Say that we have a state that looks like the topmost (root) board, and it is the computer’s turn (Player O). The computer will aim to minimize the score as much as possible.

1. There are 3 moves available for the computer: top-right, middle-left and bottom-left.
2. After playing each of these moves, the computer will evaluate the possible moves of the user (Player X). The algorithm will recurse for the temporary updated state.
3. If the board is at a terminal state (orangish box), then it will return a specific value: 1 if X wins; -1 if O wins; and 0 if it’s a tie.
4. When the algorithm is evaluating the user’s (Player X’s) play, then it will maximize score — best move the user can play. When the algorithm is evaluating the computer’s (Player O’s) play, then it will minimize the score.

In the above example, at the root level, the 3 possible moves determine the score of a particular action. Playing top-right would result a +1 in the worst case. Playing middle-left would result a -1 in the worst case. Playing bottom-left would result a 0 in the worst case.

Since the computer is trying to minimize its score, it will choose to play the middle-left cell since it is the best play. In this case, the computer will win the game in the worst case. This is the Minimax Algorithm.

Project Description

Beyond the readings and references mentioned above, the project will involve the following key steps:

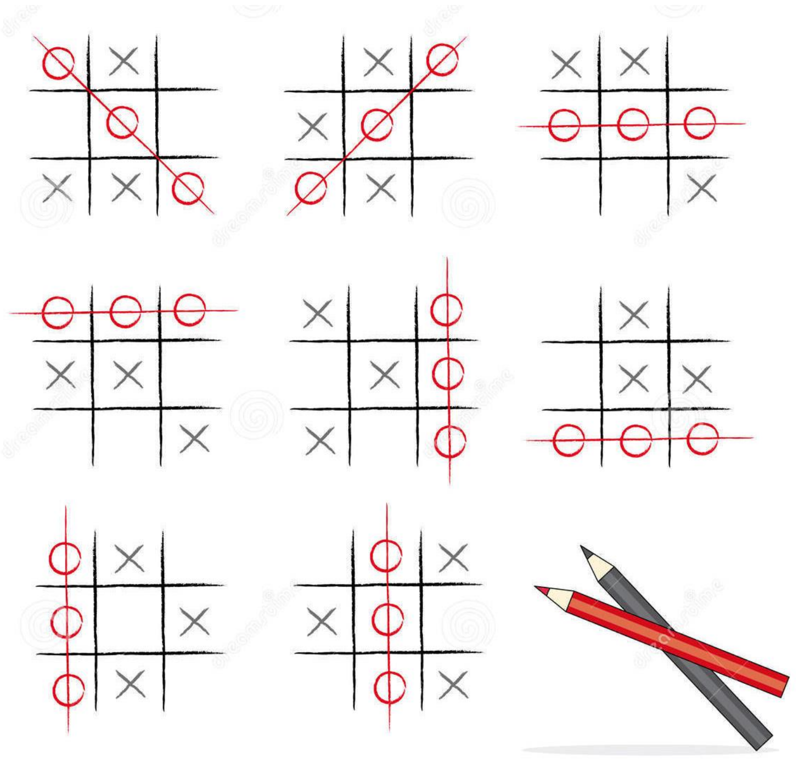
- Design and implement the graphical user interface (GUI) using the Tkinter library to create a visually appealing game board and buttons for user interaction.
- Develop the game logic, including handling player moves, validating moves, and checking for a winning condition or a draw. -Implement an AI opponent using the minimax algorithm, which will make intelligent moves by analysing the game state and selecting the best possible move.
- Enhance the user experience by incorporating additional features, such as a reset button, informative messages for game outcomes, and visual cues for the current player's turn
- Conduct thorough testing to ensure the game's functionality, including checking for correct AI moves, proper handling of user input, and accurate determination of game outcomes.
- Consider potential extensions or improvements, such as adding difficulty levels for the Al opponent, incorporating a scoring system, or enabling multiplayer functionality over a network.

By completing these tasks, the project aims to provide an engaging and enjoyable Tic-Tac-Toe experience for users combining Al techniques and a user-friendly interface.

How to play?

You can play this game following the steps:

1. Decide who will go first. You can choose to be either X or O.
2. It is your turn. Place your mark (X or O) in an empty square.
3. The AI will place its mark (X or O) in an empty square.
4. Look for three in a row horizontally, vertically or diagonally formed by your marks. If you find three in a row, you win.
5. End the game when either one of you has won or when all the squares are marked and none of you has won.
6. Congratulations! You win the game.



Summary

Tic-Tac-Toe game is a traditional game that still being played until present day. All algorithm that used in this game have the same purpose, to block the opponent’s way. Each developer has his/her own style of algorithm. However, in order to be more effective, the basic of algorithm should include all aspects based on Newell and Simon’s 1972 Tic-Tac-Toe.

Readings/References

To develop this project, the following readings and references were consulted-

- Play Tic Tac Toe with Artificial Inteligence Python BY Nishant Aanjaney Jalan: https://medium.com/codex/play-tic-tactoe-with-artifcial-intelligence-python-bf6725ed44f9
- A Mini-Project Report Tic-Tac-Toe" BY Nirusha Manandhar:
- A Project Report: https://ww.scribd.com/document/405589315/DSA-Report-docx#
- AI Analysis for Tic-Tac-Toe Game BY Ferdy Mulyadi: https://ww.researchgate.net/publication/335975566_AI

Appendix: Code Implementation

The code implementation for the Tie Tac Toe game can be found at the following link:

[*https://github.com/sakyAIU/Tic-Tac-Toe-with-Artificial-Intelligence*](https://github.com/sakyAIU/Tic-Tac-Toe-with-Artificial-Intelligence)

```
# TIC-TAC-TOE with Artificial Intelligence
from tkinter import *
from tkinter import messagebox
import random

class TIC_TAC_TOE_AI:
    def __init__(self, root):
        # Basic Initialization
        self.window = root
        self.make_canvas = Canvas(self.window, background="#141414", relief=RAISED, bd=3)
        self.make_canvas.pack(fill=BOTH, expand=1)

        self.machine_cover = []
        self.human_cover = []
        self.prob = []
        self.sign_store = {}

        self.chance_counter = 0
        self.technique = -1

        self.surrounding_store = {1: (2,3,4,7), 2:(1,3), 3:(1,2,6,9), 4:(1,7), 5: (2,4,6,8), 6: (3,9), 7:(1,4,8,9), 8:(7,9), 9:(7,8,6,3)}


        self.decorating()

    def decorating(self):# Basic Set-up
        Label(self.make_canvas, text="Tic-Tac-Toe AI", bg="#141414", fg="#00FF00", font=("Lato", 25,
"bold")).place(x=110, y=10)
```

```
        self.btn_1 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(1), state=DISABLED)
        self.btn_1.place(x=20,y=100)
        self.btn_2 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(2), state=DISABLED)
        self.btn_2.place(x=190,y=100)
        self.btn_3 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(3), state=DISABLED)
        self.btn_3.place(x=360,y=100)

        self.btn_4 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(4), state=DISABLED)
        self.btn_4.place(x=20,y=200)
        self.btn_5 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(5), state=DISABLED)
        self.btn_5.place(x=190,y=200)
        self.btn_6 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(6), state=DISABLED)
        self.btn_6.place(x=360,y=200)

        self.btn_7 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(7), state=DISABLED)
        self.btn_7.place(x=20,y=300)
        self.btn_8 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(8), state=DISABLED)
        self.btn_8.place(x=190,y=300)
        self.btn_9 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626",
activebackground="#262626", bd=3, command=lambda: self.__human_play(9), state=DISABLED)
        self.btn_9.place(x=360,y=300)

        self.activate_btn = [self.btn_1, self.btn_2, self.btn_3, self.btn_4, self.btn_5, self.btn_6, self.btn_7, self.btn_8, self.btn_9]
        self.machine_first_control = Button(self.make_canvas, text="Machine vs Human", font=("Arial", 15, "bold", "italic"), 
bg="#262626", activebackground="#262626", fg="#9d9dff", relief=RAISED, bd=3, command=lambda: 
self.control_give("machine_first"))
        self.machine_first_control.place(x=15, y=380)

        self.human_first_control = Button(self.make_canvas, text="Human vs Machine", font=("Arial", 15, "bold", "italic"), 
bg="#262626", activebackground="#262626", fg="#9d9dff", relief=RAISED, bd=3, command=self.control_give)
        self.human_first_control.place(x=240, y=380)

        self.reset_btn = Button(self.make_canvas, text="Reset", font=("Arial", 15, "bold", "italic"), bg="#262626", 
activebackground="#262626", disabledforeground="grey", fg="#9d9dff", relief=RAISED, bd=3, command=self.reset, 
state=DISABLED)
        self.reset_btn.place(x=190, y=440)

    def reset(self):# Reset the game
        self.machine_cover.clear()
        self.human_cover.clear()
        self.sign_store.clear()
        self.prob.clear()
        self.technique = -1
        self.chance_counter = 0
        for every in self.activate_btn:
            every.config(text="")
        self.machine_first_control['state'] = NORMAL
        self.human_first_control['state'] = NORMAL
        self.reset_btn['state'] = DISABLED

    def game_over_management(self):# After game over some works
        for every in self.activate_btn:
            every.config(state=DISABLED)
        self.reset_btn['state'] = NORMAL

    def control_give(self, indicator="human_first"):# Control give based on human first or computer first play
        self.machine_first_control.config(state=DISABLED, disabledforeground="grey")
```

```python
        self.human_first_control.config(state=DISABLED, disabledforeground="grey")
        self.reset_btn.config(state=DISABLED, disabledforeground="grey")
        for every in self.activate_btn:
            every.config(state=NORMAL)
        if indicator == "machine_first":
            self.__machine_play()

    def __sign_insert(self, btn_indicator, sign_is="X"):# Button sign Insert
        if sign_is == "X":
            self.activate_btn[btn_indicator - 1].config(text=sign_is, state=DISABLED, disabledforeground="#00FF00")
        else:
            self.activate_btn[btn_indicator - 1].config(text=sign_is, state=DISABLED, disabledforeground="red")
        self.sign_store[btn_indicator] = sign_is

    def __machine_play(self):# Machine Control
        self.chance_counter+=1
        # For even in self.chance_counter, human first chance..... for odd, computer first chance
        if self.chance_counter == 1:
            self.__sign_insert(9)
            self.machine_cover.append(9)

        elif self.chance_counter == 2:
            human_last = self.human_cover[len(self.human_cover)-1]
            if human_last != 5:
                self.technique = 1
                self.__sign_insert(5)
                self.machine_cover.append(5)
            else:
                self.technique = 2
                self.__sign_insert(9)
                self.machine_cover.append(9)

        elif self.chance_counter == 3:
            human_input = self.human_cover[len(self.human_cover)-1]
        if human_input%2 == 0:
            self.technique = 1
            self.activate_btn[5 - 1].config(text="X")
            self.sign_store[5] = "X"
            self.prob.append(1)

        elif human_input != 5:
            self.technique = 2
            take_prediction = [7,3]
            try:
                take_prediction.remove(human_input)
            except:
                pass
            take_prediction = random.choice(take_prediction)
            self.__sign_insert(take_prediction)
            self.prob.append(1)
            self.machine_cover.append(take_prediction)
        else:
            self.technique = 3
            self.__sign_insert(1)

    elif self.chance_counter == 4:
        human_first = self.human_cover[0]
        human_last = self.human_cover[1]
        opposite = {1:9, 2:8, 3:7, 4:6, 6:4, 7:3, 8:2, 9:1}

        if self.technique == 1:
            take_surr = list(self.surrounding_store[human_first])
            if human_last in take_surr:
                take_surr.remove(human_last)
                diff = human_last - human_first

                if diff == 6 or diff == -6:
                    if diff == 6:
```

```python
            place_it = human_first + 3
        elif diff == -6:
            place_it = human_first - 3
    elif diff == 2 or diff == -2:
        if diff == 2:
            place_it = human_first + 1
        else:
            place_it = human_first - 1
    elif diff == 1 or diff == -1:
        if diff == 1:
            if human_first-1 == 1 or human_first-1 == 7:
                place_it = human_first-1
            else:
                place_it = human_last+1
        else:
            if human_last-1 == 1 or human_last-1 == 7:
                place_it = human_last-1
            else:
                place_it = human_first+1
    elif diff == 3 or diff == -3:
        if diff == 3:
            if human_first-3 == 1 or human_first-3 == 3:
                place_it = human_first-3
            else:
                place_it = human_last+3
        else:
            if human_last-3 == 1 or human_last-3 == 3:
                place_it = human_last-3
            else:
                place_it = human_first+3


    self.__sign_insert(place_it)
    self.machine_cover.append(place_it)
    self.prob.append(opposite[place_it])
```

```
        self.surrounding_store[human_first] = tuple(take_surr)
    else:
        if 2 not in self.sign_store.keys():
            self.__sign_insert(2)
            self.machine_cover.append(2)
            if opposite[2] not in self.sign_store.keys():
                self.prob.append(opposite[2])
        else:
            temp = [4,6,8]
            take_total = self.human_cover+self.machine_cover
            for x in take_total:
                if x in temp:
                    temp.remove(x)
            take_choice = random.choice(temp)
            self.__sign_insert(take_choice)
            self.machine_cover.append(take_choice)
            self.prob.append(opposite[take_choice])

elif self.technique == 2:
    human_last = self.human_cover[len(self.human_cover)-1]
    if human_last == 1:
        take_place = 3
        self.prob.append(4)
        self.prob.append(6)
    else:
        take_place = opposite[human_last]
        diff = 9 - take_place
        if diff == 2:
            self.prob.append(9-1)
        elif diff == 6:
            self.prob.append(9-3)
        elif diff == 3:
            self.prob.append(9-6)
        else:
                self.prob.append(9-2)
        self.__sign_insert(take_place)
        self.machine_cover.append(take_place)

elif self.chance_counter == 5:
    human_input = self.human_cover[len(self.human_cover)-1]
    if self.technique == 1:
        if self.prob[0] != human_input:
            self.__sign_insert(self.prob[0])
            self.machine_line_match()
        else:
            if self.technique == 1:
                try:
                    try:
                        if self.sign_store[self.prob[0]+1] == "O":
                            pass
                    except:
                        if self.sign_store[self.prob[0]+1+6] == "O":
                            pass
                    value_take = self.prob[0]+2
                    self.prob.clear()
                    self.prob.append(6)
                    self.prob.append(7)
                except:
                    value_take = self.prob[0]+6
                    self.prob.clear()
                    self.prob.append(8)
                    self.prob.append(3)

                self.__sign_insert(value_take)
                self.machine_cover.append(value_take)

    elif self.technique == 2:
            if self.machine_cover[0] - self.machine_cover[1] == 6:
                    try:
                        if self.sign_store[self.machine_cover[1]+3] == "O":
                            self.prob.clear()
                            if 7 in self.sign_store.keys():
                                value_predict = 1
                                self.prob.append(2)
                            else:
                                value_predict = 7
                                self.prob.append(8)
                            self.prob.append(5)
                            self.__sign_insert(value_predict)
                            self.machine_cover.append(value_predict)
                    except:
                        self.__sign_insert(self.machine_cover[1]+3)
                        self.machine_line_match()
                else:
                    try:
                        if self.sign_store[self.machine_cover[1]+1] == "O":
                            self.prob.clear()
                            if 3 in self.sign_store.keys():
                                value_predict = 1
                                self.prob.append(4)
                            else:
                                value_predict = 3
                                self.prob.append(6)
                            self.prob.append(5)
                            self.__sign_insert(value_predict)
                            self.machine_cover.append(value_predict)
                    except:
                        self.__sign_insert(self.machine_cover[1]+1)
                        self.machine_cover.append(self.machine_cover[1]+1)
                        self.machine_line_match()
        else:
            if self.prob:
                    self.prob.clear()
                draw_occurance =  {2: 8, 8: 2, 4: 6, 6: 4}
                if human_input in draw_occurance.keys():
                    self.technique = 3.1
                    self.__sign_insert(draw_occurance[human_input])
                    self.machine_cover.append(draw_occurance[human_input])
                    next_prob = {8: 7, 4: 7, 2: 3, 6: 3}
                    self.prob.append(next_prob[draw_occurance[human_input]])
                else:
                    self.technique = 3.2
                    if human_input == 3:
                        self.__sign_insert(7)
                        self.machine_cover.append(7)
                        self.prob.append(8)
                        self.prob.append(4)
                    else:
                        self.__sign_insert(3)
                        self.machine_cover.append(3)
                        self.prob.append(2)
                        self.prob.append(6)

        elif self.chance_counter == 6:
            if self.human_line_match():
                opposite = {1:9, 2:8, 3:7, 4:6, 6:4, 7:3, 8:2, 9:1}
                human_last = self.human_cover[len(self.human_cover)-1]
                if self.technique == 1:
                    if self.prob and human_last != self.prob[0]:
                        self.__sign_insert(self.prob[0])
                        self.machine_cover.append(self.prob[0])
                        self.machine_line_match()

                    elif len(self.prob) == 0:
                        if human_last+3 == 7 or human_last+3 == 9:
                            take_place = human_last+3
```

```python
    elif human_last-3 == 1 or human_last-3 == 3:
        take_place = human_last-3
    elif human_last-3 == 4 or human_last-3 == 6:
        take_place = human_last-3
    elif human_last+3 == 4 or human_last+3 == 6:
        take_place = human_last+3

    self.__sign_insert(take_place)
    self.machine_cover.append(take_place)
    self.prob.append(opposite[take_place])

else:
    if self.prob:
        self.prob.clear()
    if human_last%2 == 0:
        if human_last == 8:
            if (human_last+1==3 or human_last+1==9) and human_last + 1 not in self.sign_store.keys():
                place_here = human_last + 1
            elif (human_last-1==1 or human_last-1==7) and human_last - 1 not in self.sign_store.keys():
                place_here = human_last - 1
            elif (human_last-3==1 or human_last-3==3) and human_last - 3 not in self.sign_store.keys():
                place_here = human_last - 3
            else:
                place_here = human_last + 3

            self.__sign_insert(place_here)
            self.machine_cover.append(place_here)
            temp_oppo = {7: 3, 3: 7, 1: 9, 9: 1}
            self.prob.append(temp_oppo[place_here])

        else:
            take_center_surr = list(self.surrounding_store[5])
            temp_store = self.human_cover+self.machine_cover
            for element in temp_store:
                    try:
                        take_center_surr.remove(element)
                    except:
                        pass

                if take_center_surr:
                    if (human_last+3==7 or human_last+3==9) or human_last+3 in self.sign_store.keys():
                        take_place = human_last-3
                    else:
                        take_place = random.choice(take_center_surr)
                        take_center_surr.remove(take_place)
                    self.__sign_insert(take_place)
                    self.machine_cover.append(take_place)
                    self.surrounding_store[5] = tuple(take_center_surr)
                    self.prob.append(opposite[take_place])
                else:
                    for every in opposite.keys():
                        if every%2 != 0 and opposite[every] not in self.sign_store.keys():
                            self.__sign_insert(every)
                            self.machine_cover.append(every)
                            self.prob.append(opposite[every])
                            if (every+6 == 7 or every+6 == 9) and (every+6 not in self.sign_store.keys()):
                                self.prob.append(every+6)
                            elif (every-6 == 1 or every-6 == 3) and (every-6 not in self.sign_store.keys()):
                                self.prob.append(every-6)
                            elif (every-2 == 1 or every-2 == 7) and (every-2 not in self.sign_store.keys()):
                                self.prob.append(every-2)
                            else:
                                self.prob.append(every+2)
                            break
        else:
            take_surr = self.surrounding_store[human_last]
            for element in take_surr:
                if element in self.sign_store.keys():
                    pass
                else:
                    self.__sign_insert(element)
                    self.machine_cover.append(element)
                    if opposite[element] not in self.sign_store.keys():
                        self.prob.append(opposite[element])
                    break

else:
    if len(self.prob) == 2:
        if human_last in self.prob:

            if self.prob[1] != human_last:
                self.__sign_insert(self.prob[1])
                self.machine_cover.append(self.prob[1])
                self.machine_line_match()

            else:
                self.__sign_insert(self.prob[0])
                self.machine_cover.append(self.prob[0])
                self.prob.clear()
                self.prob.append(2)
        else:
            self.__sign_insert(self.prob[1])
            self.machine_cover.append(self.prob[1])
            self.machine_line_match()
    else:
        if human_last != self.prob[0]:
            self.__sign_insert(self.prob[0])
            self.machine_cover.append(self.prob[0])
            self.machine_line_match()
        else:
            self.__sign_insert(opposite[self.prob[0]])
            self.machine_cover.append(opposite[self.prob[0]])
```

```
elif self.chance_counter == 7:
    human_input = self.human_cover[len(self.human_cover)-1]
    if self.technique == 1:
        if self.prob[0] == human_input:
            self.__sign_insert(self.prob[1])
        else:
            self.__sign_insert(self.prob[0])
        self.machine_line_match()

    elif self.technique == 2:
        if human_input in self.prob:
            self.prob.remove(human_input)
        self.__sign_insert(self.prob[0])
        self.machine_line_match()
    else:
        if self.technique == 3.2:
            if human_input in self.prob:
                self.prob.remove(human_input)
            self.__sign_insert(self.prob[0])
            self.machine_line_match()
        else:
            if human_input in self.prob:
                self.prob.clear()
                machine_next_chance = {7: 3, 3: 7}
                self.__sign_insert(machine_next_chance[human_input])
                next_human_prob = {3: (2,6), 7: (4,8)}
                self.prob.append(next_human_prob[machine_next_chance[human_input]][0])
                self.prob.append(next_human_prob[machine_next_chance[human_input]][1])
            else:
                self.__sign_insert(self.prob[0])
                self.machine_line_match()

elif self.chance_counter == 8:
```

```python
if self.human_line_match():
    human_last = self.human_cover[len(self.human_cover)-1]
    opposite = {1:9, 2:8, 3:7, 4:6, 6:4, 7:3, 8:2, 9:1}

    if self.technique == 1:
        if self.prob and human_last not in self.prob:
            if self.prob[0] not in self.sign_store.keys():
                self.__sign_insert(self.prob[0])
                self.machine_cover.append(self.prob[0])
            else:
                temp=[1,2,3,4,5,6,7,8,9]
                temp_store = self.machine_cover + self.human_cover
                for x in temp_store:
                    if x in temp:
                        temp.remove(x)
                take_choice = random.choice(temp)
                self.__sign_insert(take_choice)
                self.machine_cover.append(take_choice)
            self.machine_line_match()

        elif len(self.prob) == 0:
            self.__sign_insert(human_last+2)
            self.machine_cover.append(human_last+2)

        else:
            if len(self.prob) == 2:
                if human_last in self.prob:
                    self.prob.remove(human_last)
                self.__sign_insert(self.prob[0])
                self.machine_cover.append(self.prob[0])
                self.machine_line_match()

            else:
                take_surr = self.surrounding_store[human_last]
```

```
                            for element in take_surr:
                                if element in self.sign_store.keys():
                                    pass
                                else:
                                    self.__sign_insert(element)
                                    self.machine_cover.append(element)
                                    break

                else:
                    if opposite[human_last] not in self.sign_store.keys():
                        self.__sign_insert(opposite[human_last])
                        self.machine_cover.append(opposite[human_last])
                    else:
                        temp_store = [1,2,3,4,5,6,7,8,9]
                        temp_total = self.machine_cover+self.human_cover
                        for element in temp_store:
                            if element in temp_total:
                                temp_store.remove(element)
                        take_choice = random.choice(temp_store)
                        self.__sign_insert(take_choice)
                        self.machine_cover.append(take_choice)

        elif self.chance_counter == 9:
            human_input = self.human_cover[len(self.human_cover)-1]
            if self.prob[0] in self.sign_store.keys() and self.prob[1] in self.sign_store.keys():
                self.prob.clear()
                opposite_detection = {2: 8, 8: 2, 6: 4, 4: 6}
                self.__sign_insert(opposite_detection[human_input])
                self.machine_line_match()
            else:

                if self.prob[0] in self.sign_store.keys():
                    self.__sign_insert(self.prob[1])
                else:
```

```
                self.__sign_insert(self.prob[0])
            self.machine_line_match()



    def __human_play(self, chance):# Human Control
        self.chance_counter+=1
        self.__sign_insert(chance, "O")
        self.human_cover.append(chance)
        if self.chance_counter == 9:
            self.human_line_match()
        else:
            self.__machine_play()


    def machine_line_match(self):
        found = 0
        if self.activate_btn[1-1]['text'] == self.activate_btn[2-1]['text'] == self.activate_btn[3-1]['text'] == "X":
            found=1
        elif self.activate_btn[4-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[6-1]['text'] == "X":
            found=1
        elif self.activate_btn[7-1]['text'] == self.activate_btn[8-1]['text'] == self.activate_btn[9-1]['text'] == "X":
            found=1
        elif self.activate_btn[1-1]['text'] == self.activate_btn[4-1]['text'] == self.activate_btn[7-1]['text'] == "X":
            found=1
        elif self.activate_btn[2-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[8-1]['text'] == "X":
            found=1
        elif self.activate_btn[3-1]['text'] == self.activate_btn[6-1]['text'] == self.activate_btn[9-1]['text'] == "X":
            found=1
        elif self.activate_btn[1-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[9-1]['text'] == "X":
            found=1
        elif self.activate_btn[3-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[7-1]['text'] == "X":
            found=1
        if found == 1:
            messagebox.showinfo("Game Over", "Computer is winner")
            self.game_over_management()
    elif self.chance_counter == 9:
        messagebox.showinfo("Game Over", "Game draw")
        self.game_over_management()

def human_line_match(self):
    found = 0
    if self.activate_btn[1-1]['text'] == self.activate_btn[2-1]['text'] == self.activate_btn[3-1]['text'] == "O":
        found=1
    elif self.activate_btn[4-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[6-1]['text'] == "O":
        found=1
    elif self.activate_btn[7-1]['text'] == self.activate_btn[8-1]['text'] == self.activate_btn[9-1]['text'] == "O":
        found=1
    elif self.activate_btn[1-1]['text'] == self.activate_btn[4-1]['text'] == self.activate_btn[7-1]['text'] == "O":
        found=1
    elif self.activate_btn[2-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[8-1]['text'] == "O":
        found=1
    elif self.activate_btn[3-1]['text'] == self.activate_btn[6-1]['text'] == self.activate_btn[9-1]['text'] == "O":
        found=1
    elif self.activate_btn[1-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[9-1]['text'] == "O":
        found=1
    elif self.activate_btn[3-1]['text'] == self.activate_btn[5-1]['text'] == self.activate_btn[7-1]['text'] == "O":
        found=1
    if found == 1:
        messagebox.showinfo("Game Over", "You are winner")
        self.game_over_management()
        return 0
    elif self.chance_counter == 9:
        messagebox.showinfo("Game Over", "Game draw")
        self.game_over_management()
        return 0
    else:
        return 1
```

```
if __name__ == "__main__":
    window = Tk()
    window.title("AI Tic-Tac-Toe")
    window.config(bg="#141414")
    window.geometry("450x500")
    window.maxsize(450,500)
    window.minsize(450,500)
    TIC_TAC_TOE_AI(window)
    window.mainloop()
```