SRM Institute of Science and Technology
College of Engineering & Technology | School of Computing
Department of Computing Technologies

18CSC305J Artificial Intelligence – Mini Project

TIC TAC TOE

-THERAPEUTIC GAME FOR REHABILITATION

1. RA2111027010039 JANVI TOPANI
2. RA2111027010040 AKASH RAHEJA
3. RA2111027010041 ARYAN CHAUDHARY
4. RA2111027010042 ANSH AGGARWAL

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SOCIETAL BENEFIT

- Educational Tool: Tic Tac Toe is an effective educational tool for teaching basic concepts like logic, strategy, and critical thinking to children.
- Social Interaction: Playing Tic Tac Toe encourages social interaction, teaching players how to take turns, follow rules, and engage in friendly competition.
- Problem-Solving Skills: The game requires players to think ahead and anticipate their opponent's moves, which helps develop problem-solving skills and strategic thinking.
- Stress Relief: Tic Tac Toe provides a form of stress relief and entertainment, making it a popular choice for relaxing and passing the time.

PROBLEM STATEMENT

- Develop an AI that can play Tic Tac Toe against a human player, making intelligent moves to win the game or force a draw.
- The AI should be able to evaluate the current game state, predict future moves, and select the best move based on a predefined strategy or algorithm.
- The AI should be implemented using a suitable algorithm, such as minimax with or without alpha-beta pruning, to ensure optimal or near-optimal gameplay.
- The AI's performance should be measured based on its ability to win against human players, its efficiency in making decisions, and its overall gameplay experience.

OBJECTIVE WITH TECHNICAL DEPTH

ALGORITHM SELECTION

- ➢Research and select a suitable algorithm for the AI, such as minimax with alpha-beta pruning, to efficiently search through the game tree and make optimal or near-optimal moves.

STATE REPRESENTATION

- ➢ Design a data structure to represent the game state, including the current board configuration, player positions, and possible moves, to facilitate efficient evaluation and decision-making.

OBJECTIVE WITH TECHNICAL DEPTH

MOVE EVALUATION FUNCTION

- ➢Develop a heuristic function to evaluate the desirability of a given game state for the AI, taking into account factors such as potential wins, blocking opponent's moves, and creating future winning opportunities.

GAME TREE TRAVERSAL

- ➢Implement the selected algorithm to traverse the game tree, evaluating possible moves and selecting the best move based on the current state and future projections, while minimizing the number of nodes explored.

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

OBJECTIVE WITH TECHNICAL DEPTH

OPTIMIZATION AND PERFORMANCE

- Optimize the AI's performance by implementing techniques such as memoization, iterative deepening, or other enhancements to reduce computation time and improve decision-making efficiency.

USER INTERFACE INTEGRATION

- Integrate the AI into a user-friendly interface that allows human players to interact with the AI, make moves, and receive feedback on the AI's decisions, enhancing the overall gaming experience.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

OBJECTIVE WITH TECHNICAL DEPTH

OPTIMIZATION AND PERFORMANCE

- ➢Optimize the AI's performance by implementing techniques such as memoization, iterative deepening, or other enhancements to reduce computation time and improve decision-making efficiency.

USER INTERFACE INTEGRATION

- ➢Integrate the AI into a user-friendly interface that allows human players to interact with the AI, make moves, and receive feedback on the AI's decisions, enhancing the overall gaming experience.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)

PROPOSED WORKFLOW

1. Requirements Gathering and Planning:

- ➢ - Define target audience and their rehabilitation needs.
- ➢ - Gather input from therapists and individuals with disabilities.
- ➢ - Establish key features based on requirements.

2. Design and Prototyping:

- ➢ - Design UI considering accessibility guidelines.
- ➢ - Create wireframes or prototypes for visualization.
- ➢ - Iterate on design based on feedback.

3. Development:

- ➢ - Implement game mechanics: adjustable board size, customizable symbols, voice commands, adaptive difficulty levels, haptic feedback.

PROPOSED WORKFLOW

- - Integrate accessibility features.
- - Ensure compliance with standards.

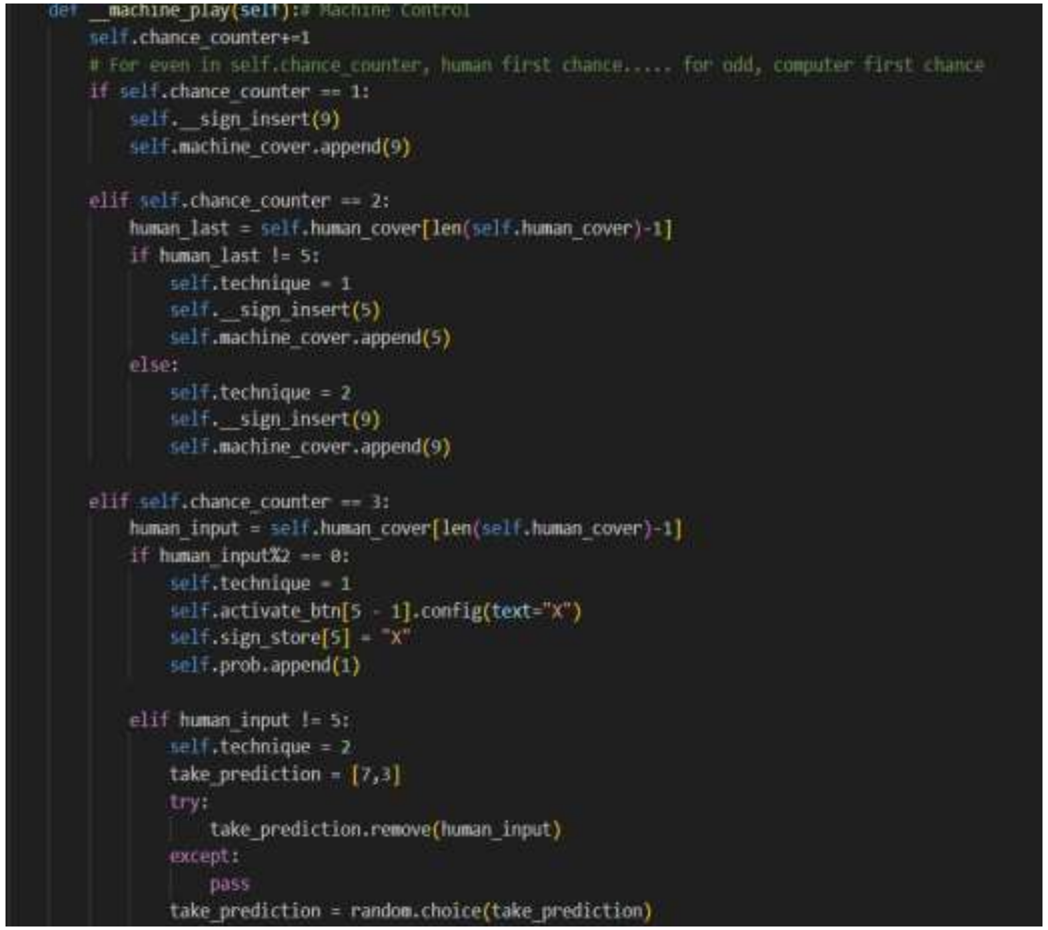
4. Testing:

- - Conduct usability testing with target audience.
- - Perform functional testing.
- - Test across different devices/platforms.

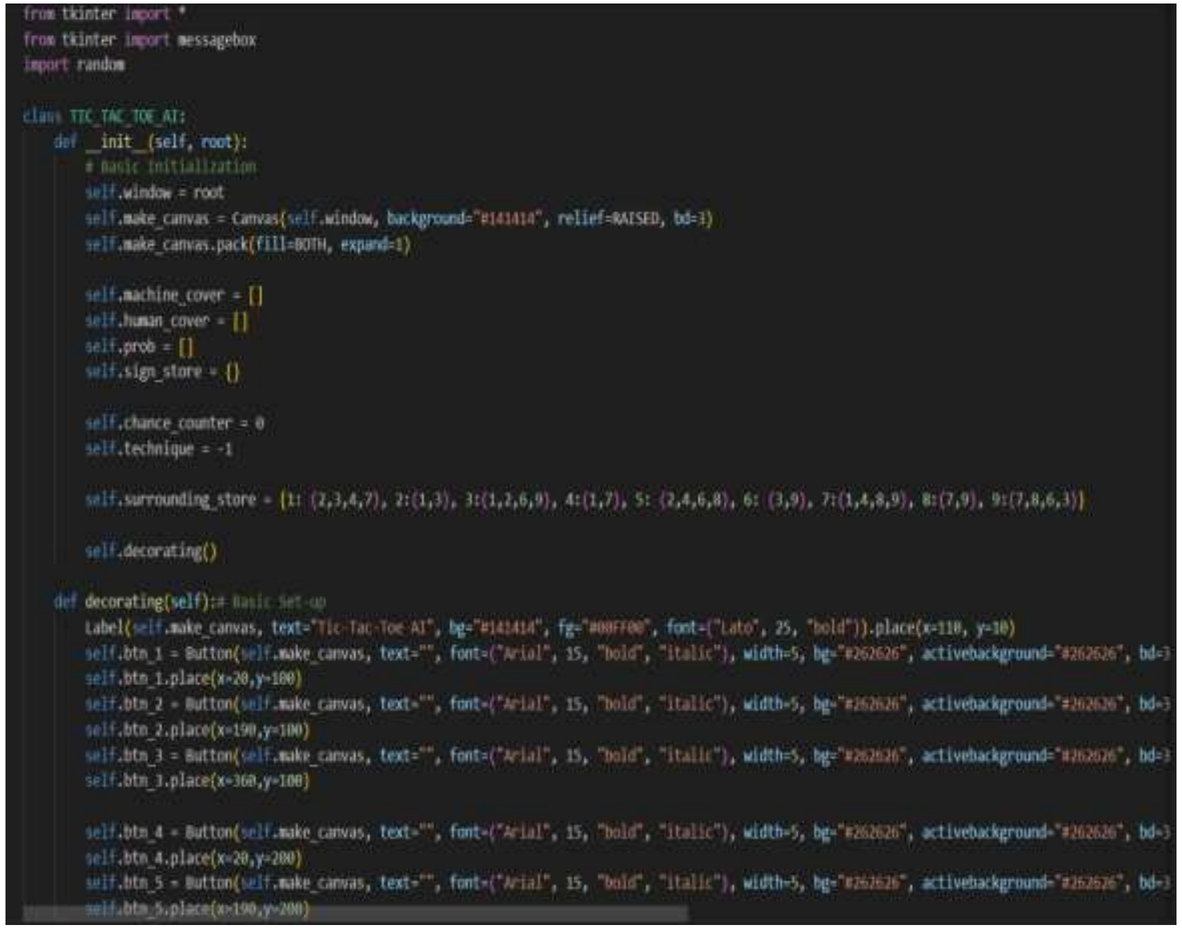
5. Iteration and Deployment:

- - Gather feedback and iterate on design.
- - Prepare for deployment.
- - Provide ongoing support and updates.

IMPLEMENTATION

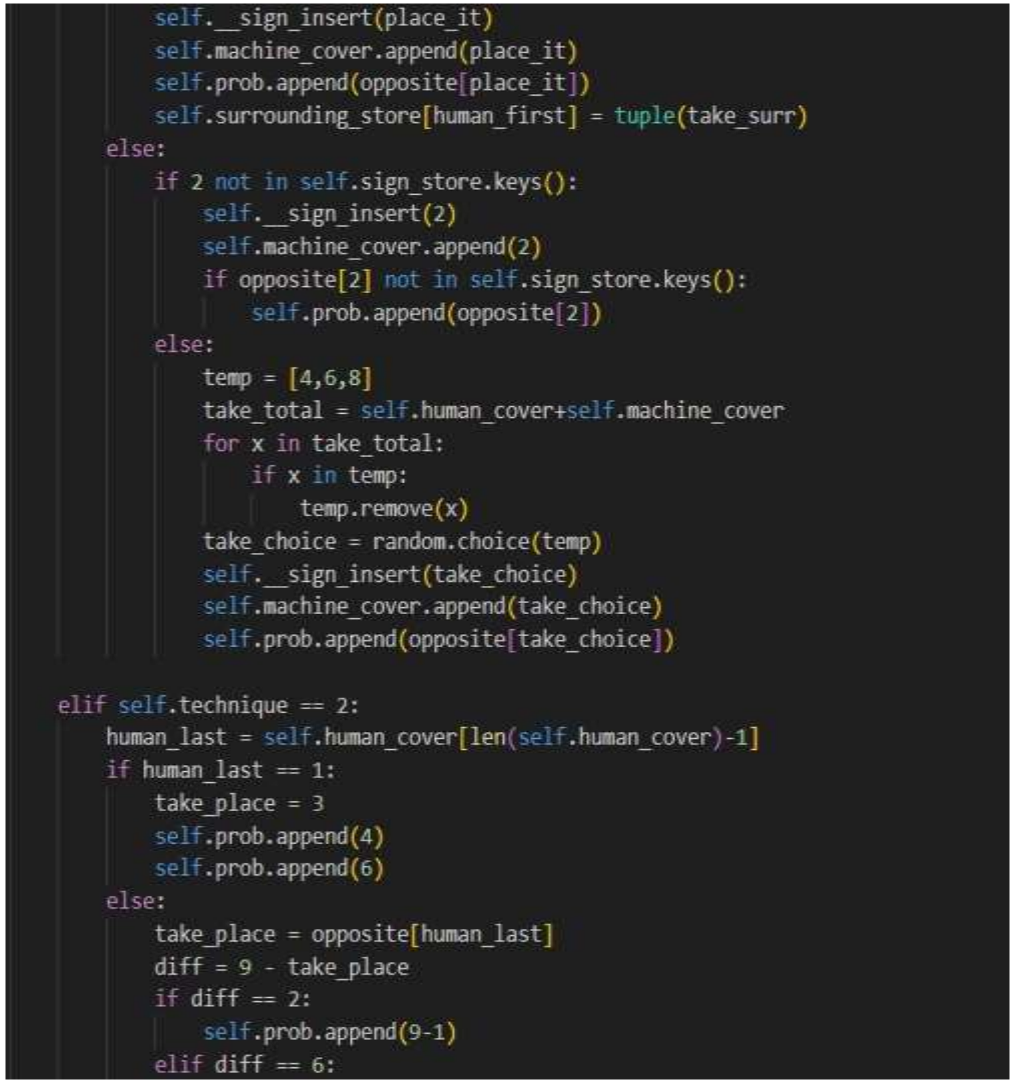


```
def __machine_play(self):# Machine Control
    self.chance_counter+=1
    # For even in self.chance_counter, human first chance..... for odd, computer first chance
    if self.chance_counter == 1:
        self.__sign_insert(9)
        self.machine_cover.append(9)

    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)
```

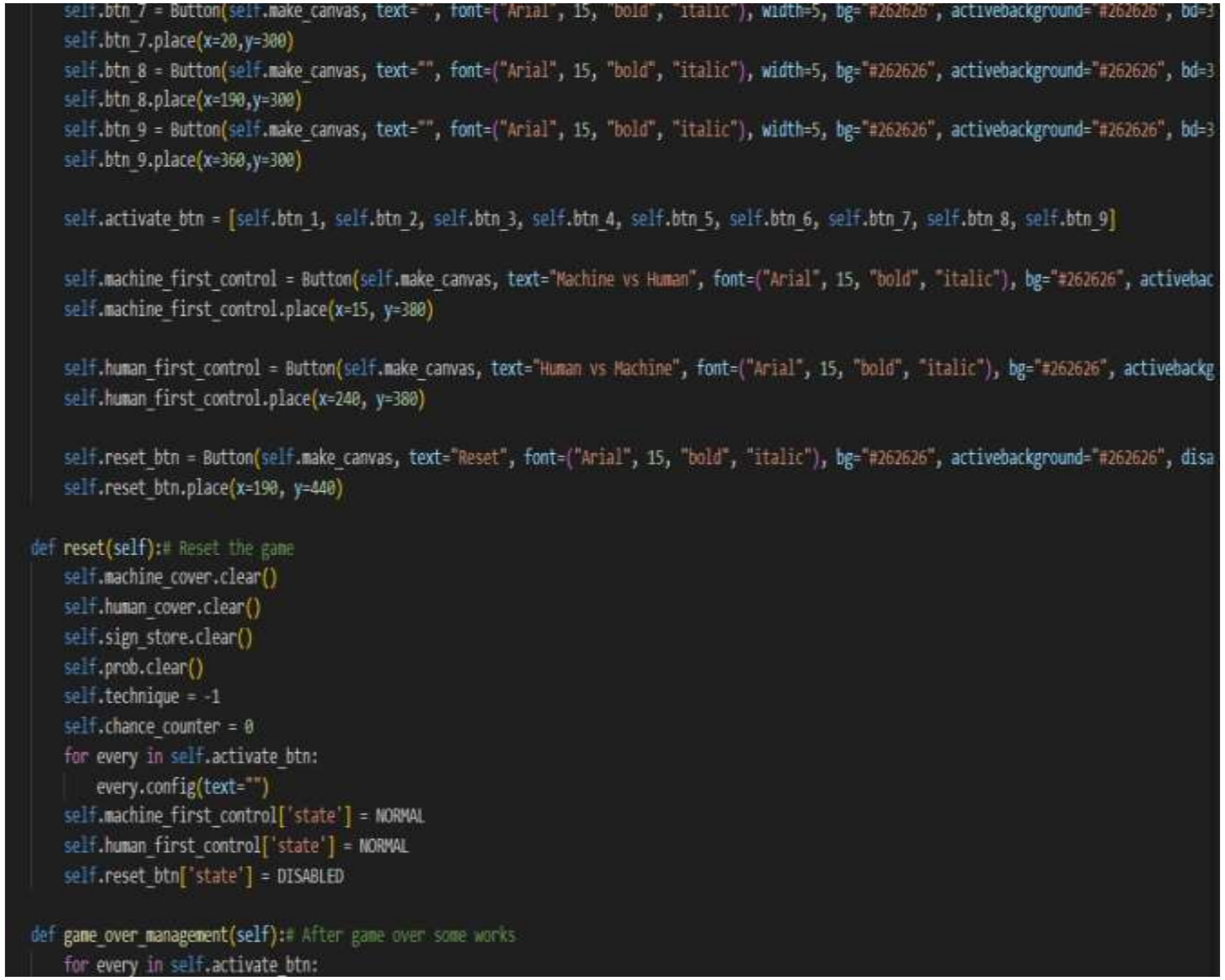


```
from tkinter import *
from tkinter import messagebox
import random

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
        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
        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
        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
        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
        self.btn_5.place(x=190,y=200)
```

IMPLEMENTATION

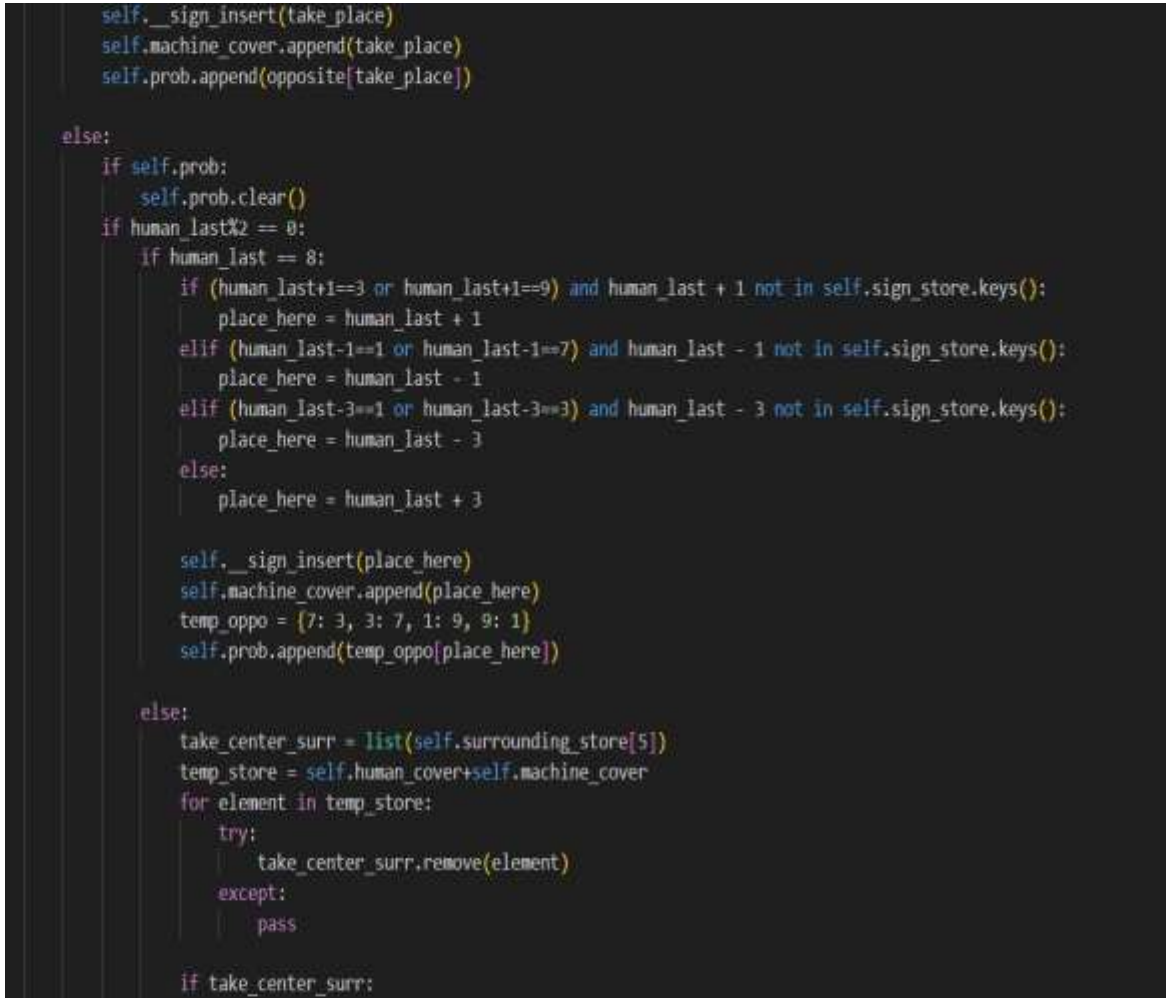


```
        self.__sign_insert(place_it)
        self.machine_cover.append(place_it)
        self.prob.append(opposite[place_it])
        self.surrounding_store[human_first] = tuple(take_surr)
    else:
        if 2 not in self.sign_store.keys():
            self.__sign_insert(2)
            self.machine_cover.append(2)
            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:
```

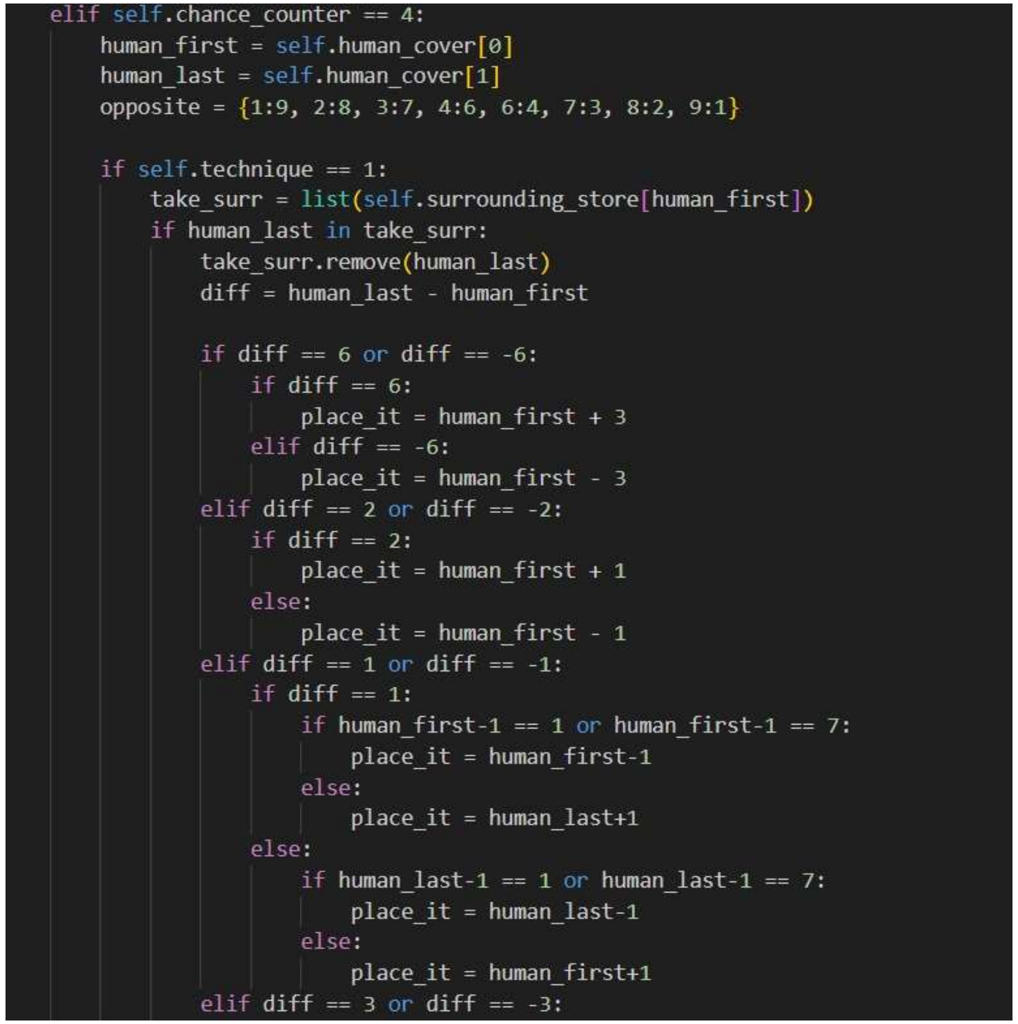


```
    self.btn_7 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626", activebackground="#262626", bd=3
    self.btn_7.place(x=20,y=300)
    self.btn_8 = Button(self.make_canvas, text="", font=("Arial", 15, "bold", "italic"), width=5, bg="#262626", activebackground="#262626", bd=3
    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
    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", activebac
    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", activebackg
    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", disa
    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:
```

IMPLEMENTATION

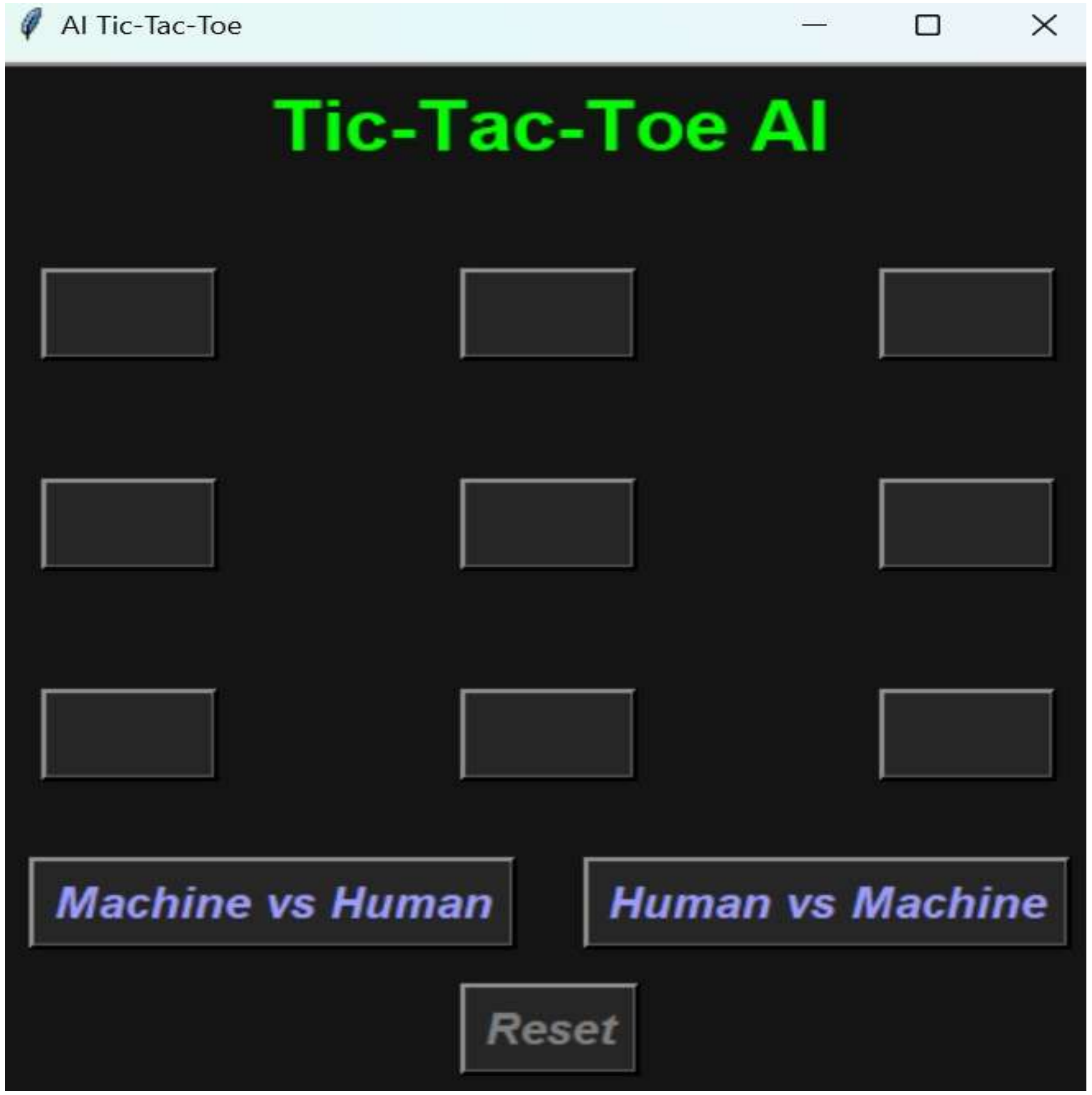


```
        self.__sign_insert(take_place)
        self.machine_cover.append(take_place)
        self.prob.append(opposite[take_place])

    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:
```



```
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:
                    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:
```

IMPLEMENTATION



SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
*(Deemed to be **University** u/s 3 of UGC Act, 1956)*

THANK YOU