**OPEN ENDED**

**ARTIFICIAL INTELLIGENCE**

**[CSE401]**

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



**Submitted To: Submitted By:**

 Dr. S. K. Dubey Shefali Bansal Associate Professor A2305218263 CSE Department, ASET B. Tech (CSE)

7CSE4Y

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

NOIDA-201301

**OPEN-ENDED EXPERIMENT**

**Name:** SHEFALI BANSAL

**Enrollment No.:** A2305218263

**Subject:** Artificial Intelligence

**Section:** 7CSE4-Y

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Category**  **of Assignment** | **Code** | **Name of Experiment** | **Date of Allotment of experiment** | **Date of Evaluation** | **Max.**  **Marks** | **Marks obtained** | **Sign.**  **of Faculty** |
| **Design Based Open Ended experiment** | **PR (10)** | **Ultimate Tic-Tac- Toe**  This assignment will require the students to build an Ultimate Tic-Tac-Toe game, where a user can play against an AI agent. In this assignment students extend a Tic-Tac-Toe program to Ultimate Tic-Tac-Toe and implement a different search strategy. The Ultimate Tic-Tac-Toe game is a simple extension to the widely-known Tic-Tac-Toe game. Ultimate Tic-Tac-Toe is not known to many students yet is easy to understand, with non-trivial game strategy. | 22-10-21 | 29-10-21 | **10** |  |  |

**OPEN ENDED EXPERIMENT**

**DATE:** 22-10-21

**OBJECTIVE: Ultimate Tic-Tac-Toe**

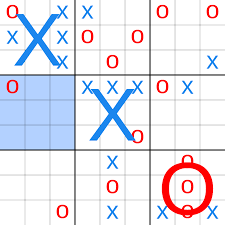
This assignment will require the students to build an Ultimate Tic-Tac-Toe game, where a user can play against an AI agent. In this assignment students extend a Tic-Tac-Toe program to Ultimate Tic-Tac-Toe and implement a different search strategy. The Ultimate Tic-Tac-Toe game is a simple extension to the widely-known Tic-Tac-Toe game. Ultimate Tic-Tac-Toe is not known to many students yet is easy to understand, with non-trivial game strategy.

**SOFTWARE USED:** Python

**THEORY:** Ultimate tic-tac-toe is a board game composed of nine [tic-tac-toe](https://en.wikipedia.org/wiki/Tic-tac-toe) boards arranged in a 3 × 3 grid, Players take turns playing in the smaller tic-tac-toe boards until one of them wins in the larger tic-tac-toe board. Compared to traditional tic-tac-toe, strategy in this game is conceptually more difficult and has proven more challenging for computers.

Rules:

1. Each small 3 × 3 tic-tac-toe board is referred to as a local board, and the larger 3 × 3 board is referred to as the global board.
2. The game starts with X playing wherever they want in any of the 81 empty spots. This move "sends" their opponent to its relative location. For example, if X played in the top right square of their local board, then O needs to play next in the local board at the top right of the global board. O can then play in any one of the nine available spots in that local board, each move sending X to a different local board.
3. If a move is played so that it is to win a local board by the rules of normal [tic-tac-toe](https://en.wikipedia.org/wiki/Tic-tac-toe), then the entire local board is marked as a victory for the player in the global board.
4. Once a local board is won by a player or it is filled completely, no more moves may be played in that board. If a player is sent to such a board, then that player may play in any other board.
5. Game play ends when either a player wins the global board or there are no legal moves remaining, in which case the game is a draw.



**CODE:**

from math import inf

from collections import Counter

import itertools

from time import time

def index(x, y):

    x -= 1

    y -= 1

    return ((x//3)\*27) + ((x % 3)\*3) + ((y//3)\*9) + (y % 3)

def box(x, y):

    return index(x, y) // 9

def next\_box(i):

    return i % 9

def indices\_of\_box(b):

    return list(range(b\*9, b\*9 + 9))

def print\_board(state):

    for row in range(1, 10):

        row\_str = [" "]

        for col in range(1, 10):

            row\_str += [state[index(row, col)]]

            if (col) % 3 == 0:

                row\_str += [" "]

        if (row-1) % 3 == 0:

            print(" "\*(len(row\_str)\*2-1))

        print(" ".join(row\_str))

def add\_piece(state, move, player):

    if not isinstance(move, int):

        move = index(move[0], move[1])

    return state[: move] + player + state[move+1:]

def update\_box\_won(state):

    temp\_box\_win = ["\_"] \* 9

    for b in range(9):

        idxs\_box = indices\_of\_box(b)

        box\_str = state[idxs\_box[0]: idxs\_box[-1]+1]

        temp\_box\_win[b] = check\_small\_box(box\_str)

    return temp\_box\_win

def check\_small\_box(box\_str):

    global possible\_goals

    for idxs in possible\_goals:

        (x, y, z) = idxs

        if (box\_str[x] == box\_str[y] == box\_str[z]) and box\_str[x] != "\_":

            return box\_str[x]

    return "\_"

def possible\_moves(last\_move):

    global box\_won

    if not isinstance(last\_move, int):

        last\_move = index(last\_move[0], last\_move[1])

    box\_to\_play = next\_box(last\_move)

    idxs = indices\_of\_box(box\_to\_play)

    if box\_won[box\_to\_play] != "\_":

        pi\_2d = [indices\_of\_box(b) for b in range(9) if box\_won[b] == "\_"]

        possible\_indices = list(itertools.chain.from\_iterable(pi\_2d))

    else:

        possible\_indices = idxs

    return possible\_indices

def successors(state, player, last\_move):

    succ = []

    moves\_idx = []

    possible\_indexes = possible\_moves(last\_move)

    for idx in possible\_indexes:

        if state[idx] == "\_":

            moves\_idx.append(idx)

            succ.append(add\_piece(state, idx, player))

    return zip(succ, moves\_idx)

def print\_successors(state, player, last\_move):

    for st in successors(state, player, last\_move):

        print\_board(st[0])

def opponent(p):

    return "O" if p == "X" else "X"

def evaluate\_small\_box(box\_str, player):

    global possible\_goals

    score = 0

    three = Counter(player \* 3)

    two = Counter(player \* 2 + "\_")

    one = Counter(player \* 1 + "\_" \* 2)

    three\_opponent = Counter(opponent(player) \* 3)

    two\_opponent = Counter(opponent(player) \* 2 + "\_")

    one\_opponent = Counter(opponent(player) \* 1 + "\_" \* 2)

    for idxs in possible\_goals:

        (x, y, z) = idxs

        current = Counter([box\_str[x], box\_str[y], box\_str[z]])

        if current == three:

            score += 100

        elif current == two:

            score += 10

        elif current == one:

            score += 1

        elif current == three\_opponent:

            score -= 100

            return score

        elif current == two\_opponent:

            score -= 10

        elif current == one\_opponent:

            score -= 1

    return score

def evaluate(state, last\_move, player):

    global box\_won

    score = 0

    score += evaluate\_small\_box(box\_won, player) \* 200

    for b in range(9):

        idxs = indices\_of\_box(b)

        box\_str = state[idxs[0]: idxs[-1]+1]

        score += evaluate\_small\_box(box\_str, player)

    return score

def minimax(state, last\_move, player, depth, s\_time):

    succ = successors(state, player, last\_move)

    best\_move = (-inf, None)

    for s in succ:

        val = min\_turn(s[0], s[1], opponent(player), depth-1, s\_time,

                       -inf, inf)

        if val > best\_move[0]:

            best\_move = (val, s)

    return best\_move[1]

def min\_turn(state, last\_move, player, depth, s\_time, alpha, beta):

    global box\_won

    if depth <= 0 or check\_small\_box(box\_won) != ".":

        return evaluate(state, last\_move, opponent(player))

    succ = successors(state, player, last\_move)

    for s in succ:

        val = max\_turn(s[0], s[1], opponent(player), depth-1, s\_time,

                       alpha, beta)

        if val < beta:

            beta = val

        if alpha >= beta:

            break

    return beta

def max\_turn(state, last\_move, player, depth, s\_time, alpha, beta):

    global box\_won

    if depth <= 0 or check\_small\_box(box\_won) != "\_":

        return evaluate(state, last\_move, player)

    succ = successors(state, player, last\_move)

    for s in succ:

        val = min\_turn(s[0], s[1], opponent(player), depth-1, s\_time,

                       alpha, beta)

        if alpha < val:

            alpha = val

        if alpha >= beta:

            break

    return alpha

def valid\_input(state, move):

    global box\_won

    if not (0 < move[0] < 10 and 0 < move[1] < 10):

        return False

    if box\_won[box(move[0], move[1])] != "\_":

        return False

    if state[index(move[0], move[1])] != "\_":

        return False

    return True

def take\_input(state, bot\_move):

    print("\n")

    all\_open\_flag = False

    if bot\_move == -1 or len(possible\_moves(bot\_move)) > 9:

        all\_open\_flag = True

    if all\_open\_flag:

        print("Enter move --> Row[1-9] Col[1-9]")

    else:

        box\_dict = {0: "Upper Left [1-3][1-3]", 1: "Upper Center [1-3][4-6]", 2: "Upper Right [1-3][7-9]",

                    3: "Center Left [4-6][1-3]", 4: "Center [4-6][4-6]", 5: "Center Right [4-6][7-9]",

                    6: "Bottom Left [7-9][1-3]", 7: "Bottom Center [7-9][4-6]", 8: "Bottom Right [7-9]"}

        print("Shefali's Turn --> Place move 'X' in <"

              + box\_dict[next\_box(bot\_move)] + "> box")

    x = int(input("Row : "))

    if x == -1:

        raise SystemExit

    y = int(input("Col : "))

    print("")

    if bot\_move != -1 and index(x, y) not in possible\_moves(bot\_move):

        raise ValueError

    if not valid\_input(state, (x, y)):

        raise ValueError

    return (x, y)

def game(state="\_" \* 81, depth=20):

    global box\_won, possible\_goals

    possible\_goals = [(0, 4, 8), (2, 4, 6)]

    possible\_goals += [(i, i+3, i+6) for i in range(3)]

    possible\_goals += [(3\*i, 3\*i+1, 3\*i+2) for i in range(3)]

    box\_won = update\_box\_won(state)

    print\_board(state)

    bot\_move = -1

    while True:

        try:

            user\_move = take\_input(state, bot\_move)

        except ValueError:

            print("Invalid input or move not possible!")

            print\_board(state)

            continue

        except SystemError:

            print("Game Stopped!")

            break

        user\_state = add\_piece(state, user\_move, "X")

        print\_board(user\_state)

        box\_won = update\_box\_won(user\_state)

        game\_won = check\_small\_box(box\_won)

        if game\_won != "\_":

            state = user\_state

            break

        s\_time = time()

        bot\_state, bot\_move = minimax(user\_state, user\_move, "O", depth, s\_time)

        print("\nAI's move:\n")

        print\_board(bot\_state)

        state = bot\_state

        box\_won = update\_box\_won(bot\_state)

        game\_won = check\_small\_box(box\_won)

        if game\_won != "\_":

            break

    if game\_won == "X":

        print("\n\nSHEFALI WINS. yayyy !!")

    elif game\_won == "Y":

        print("\n\nAI WINS. Shefali, you lost.")

    else:

        print("\n\nIt's a Tie.")

    return state

if \_\_name\_\_ == "\_\_main\_\_":

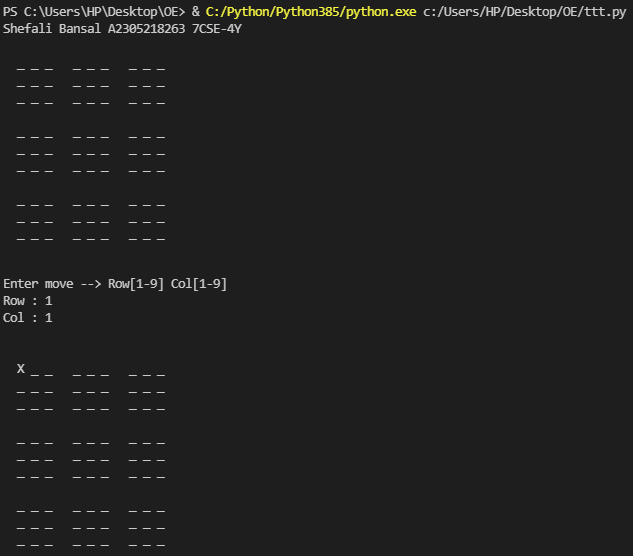
    print("Shefali Bansal A2305218263 7CSE-4Y")

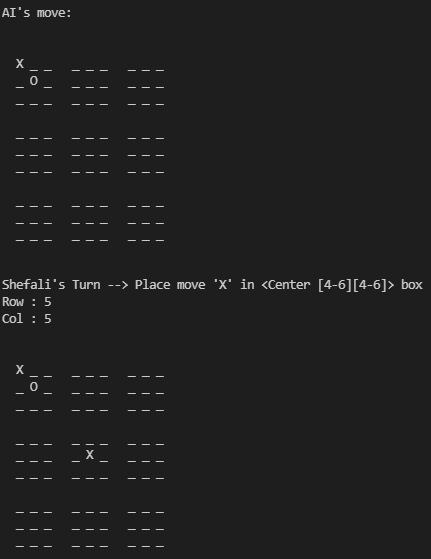
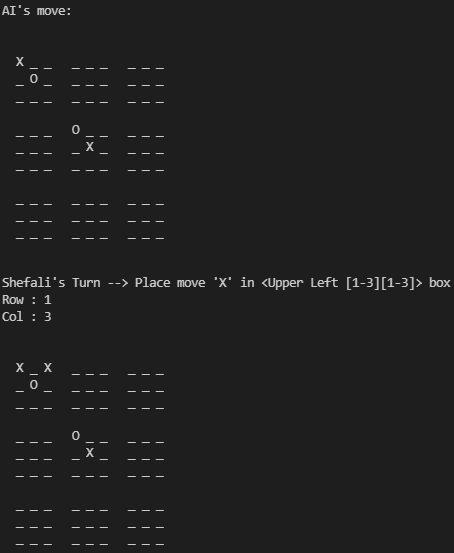
    INITIAL\_STATE = "\_" \* 81

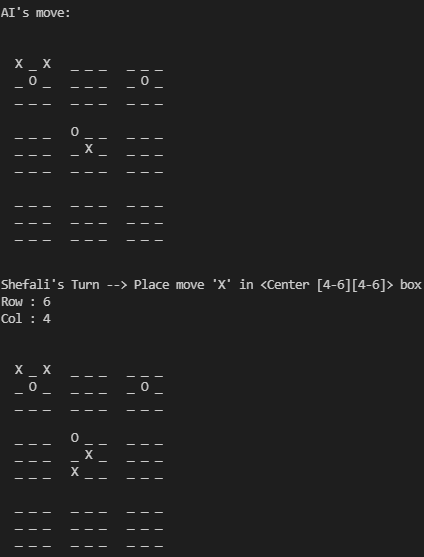
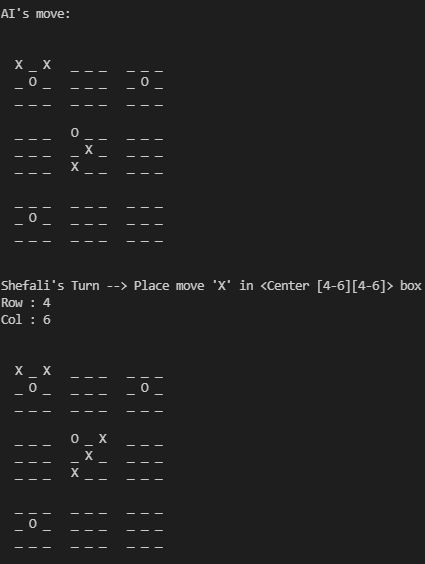
    final\_state = game(INITIAL\_STATE, depth=5)

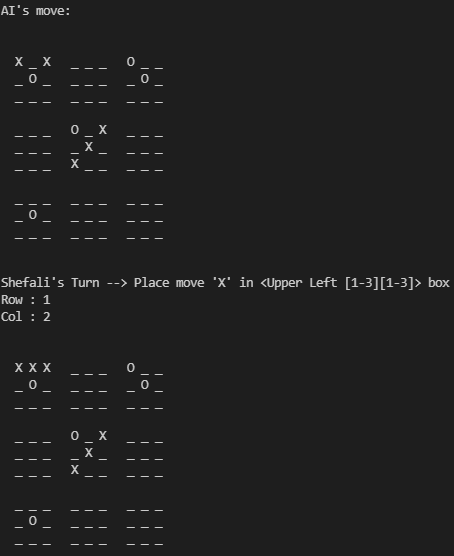
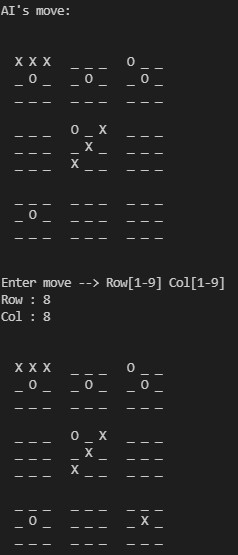
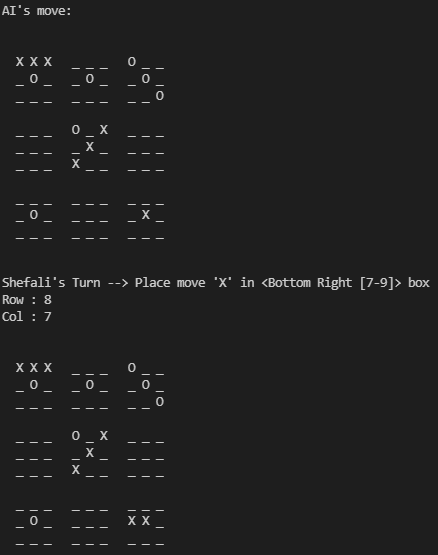
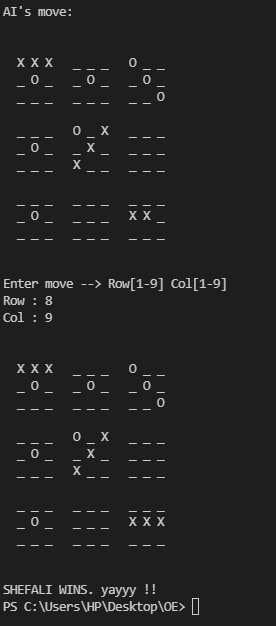
**OUTPUT:**

*Human Wins (X) :*

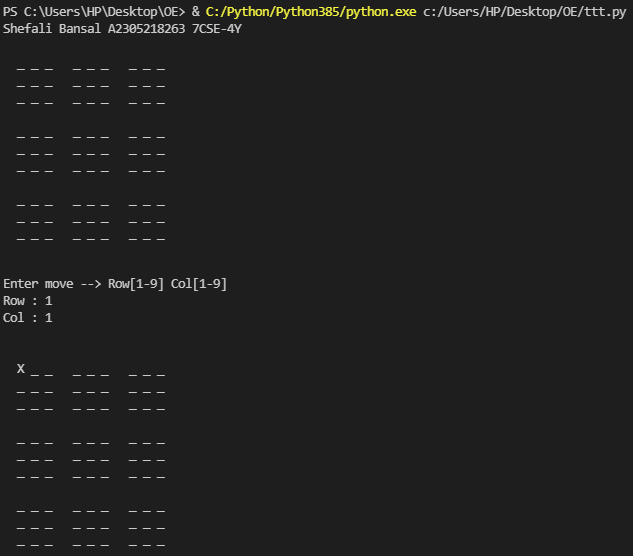
****

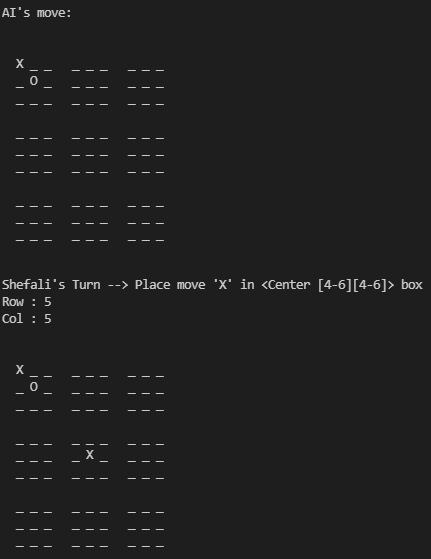
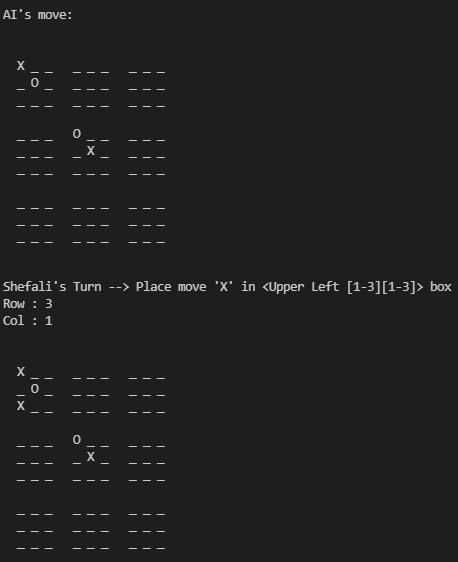
* *

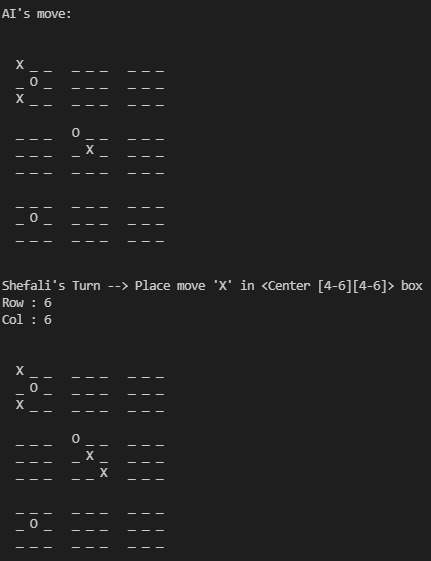
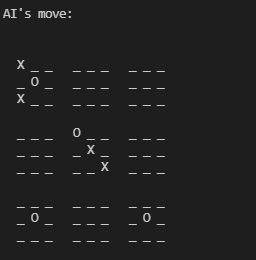
** **

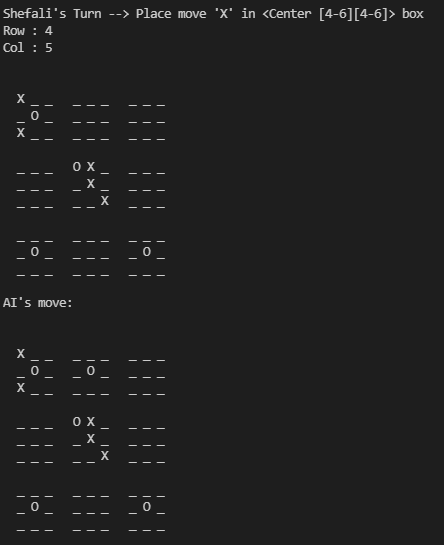
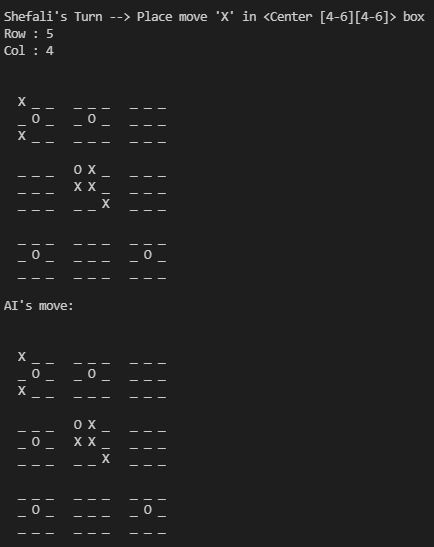
**   

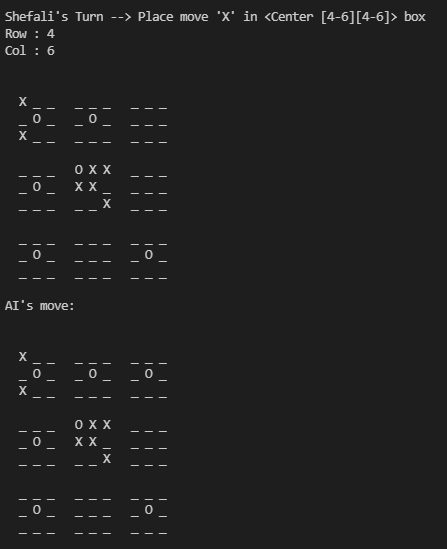
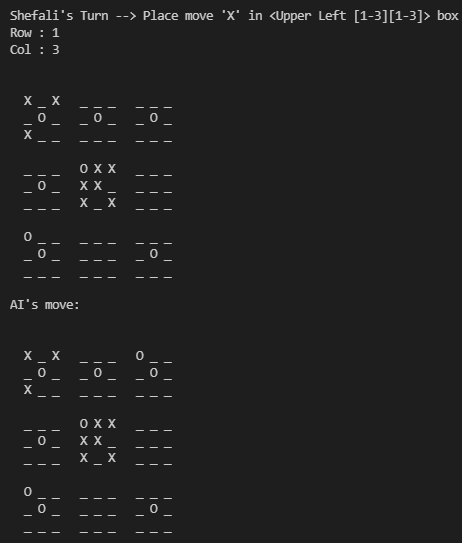
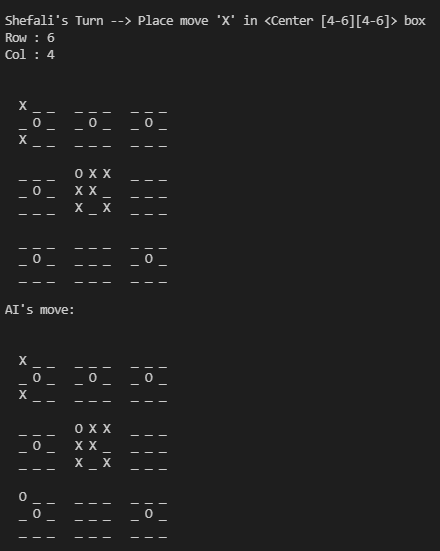
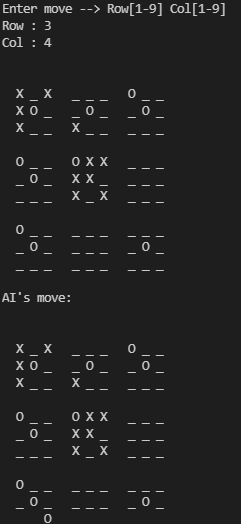
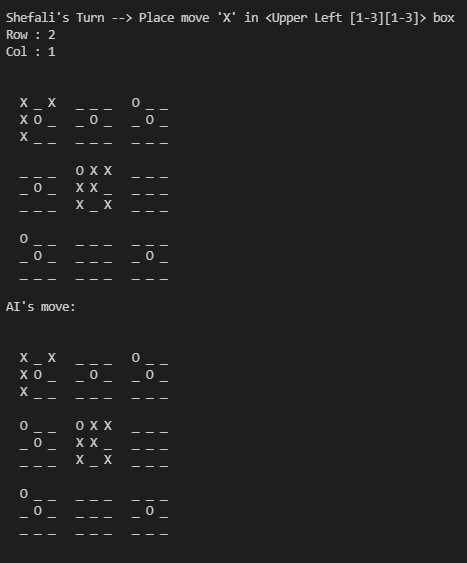
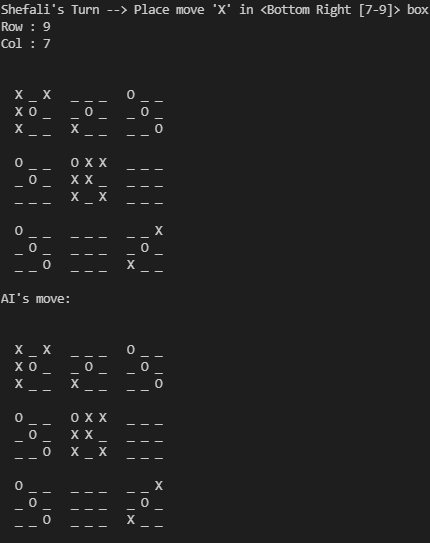
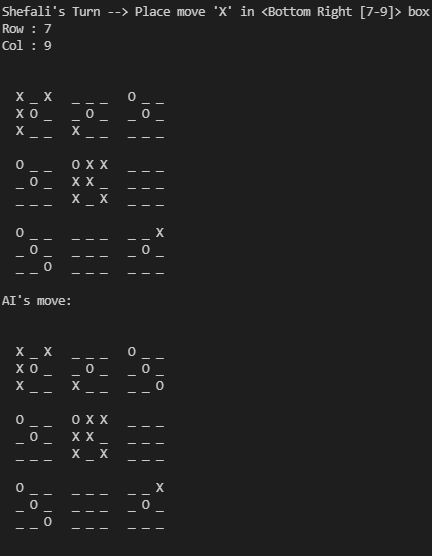
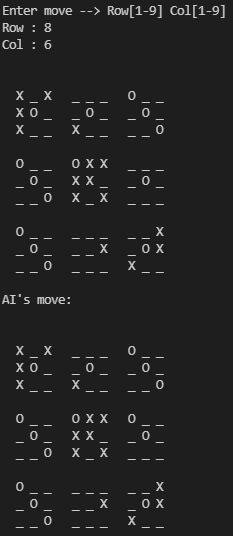
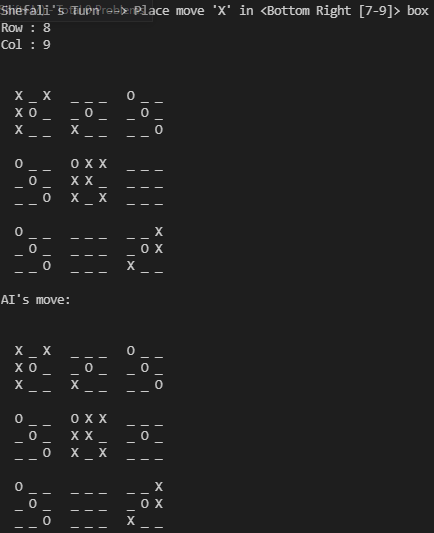
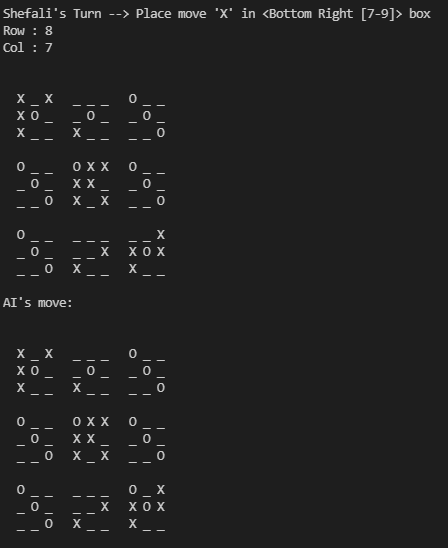
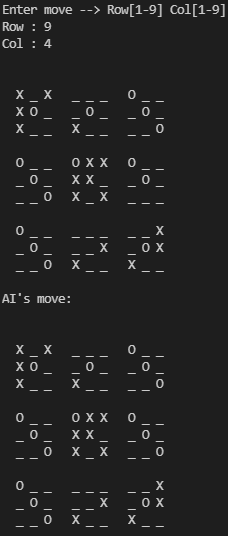
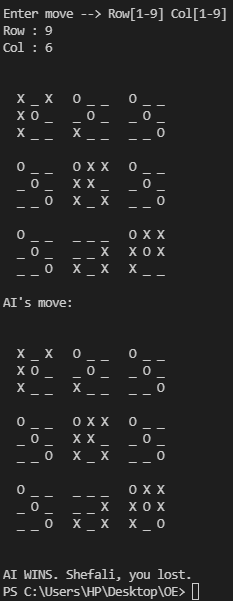
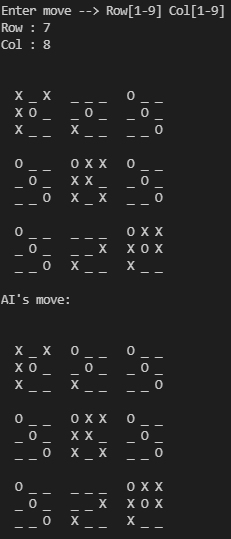
*AI Wins (0) :*

****

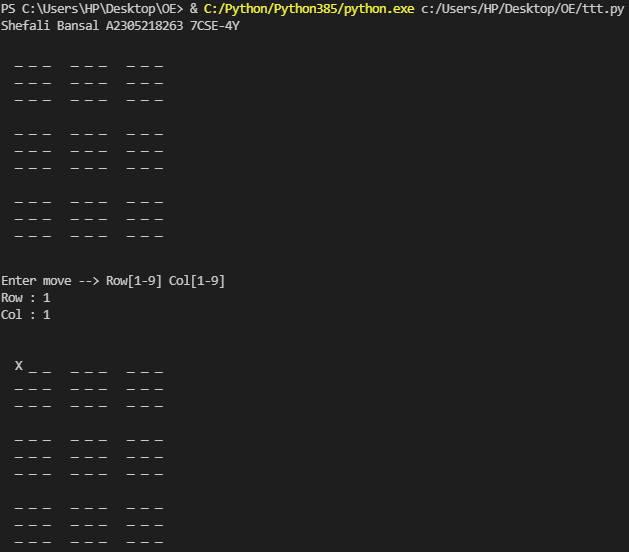
** ****

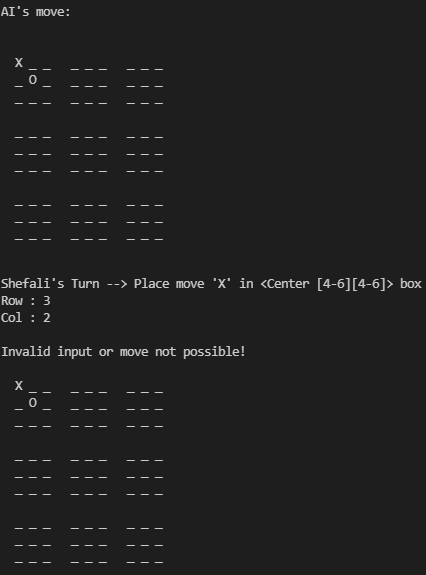
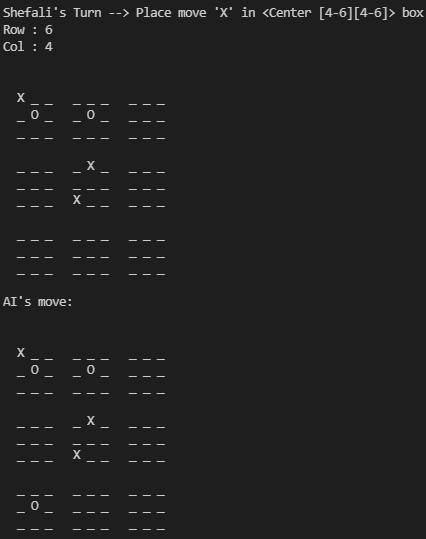
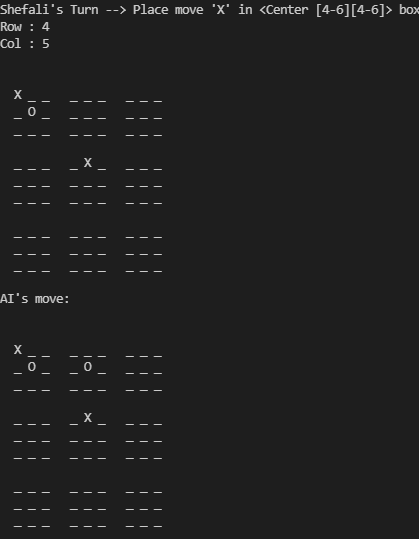
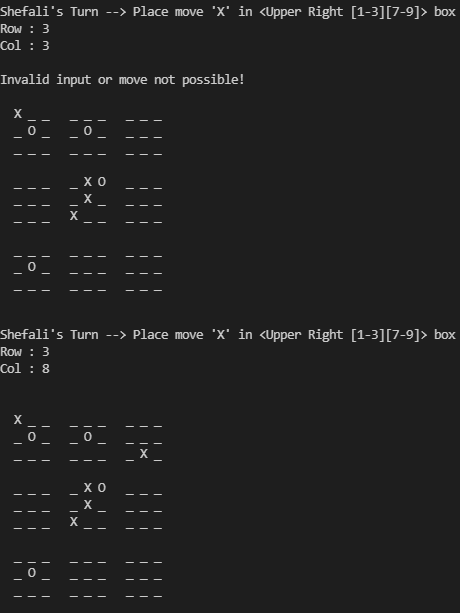
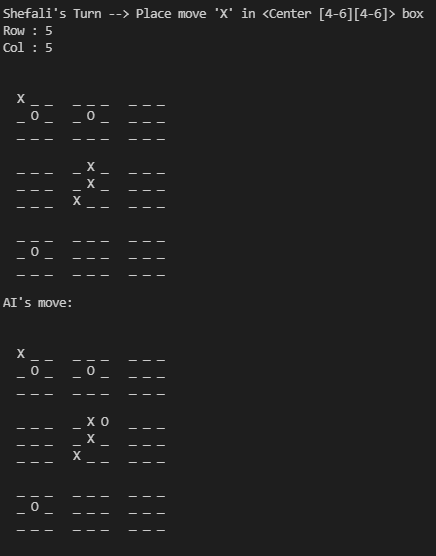
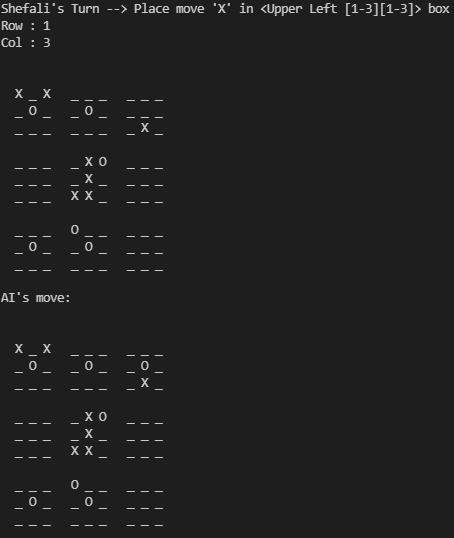
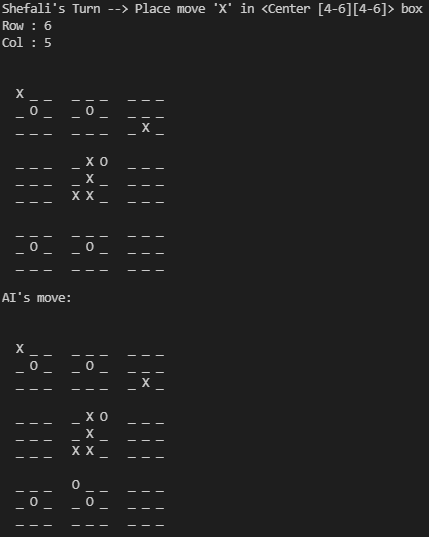
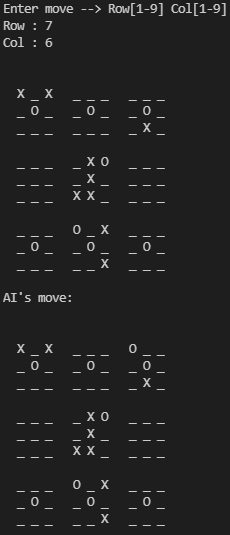
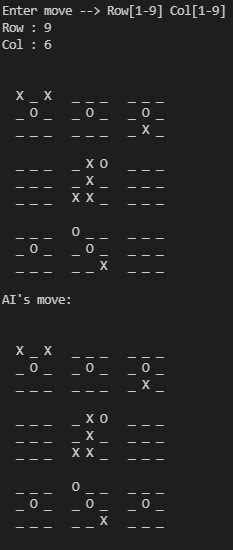
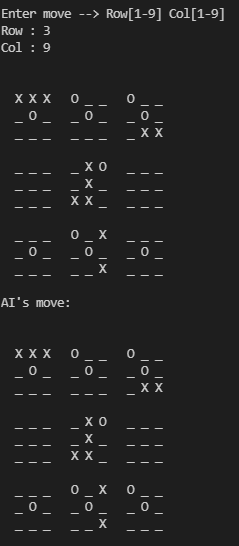
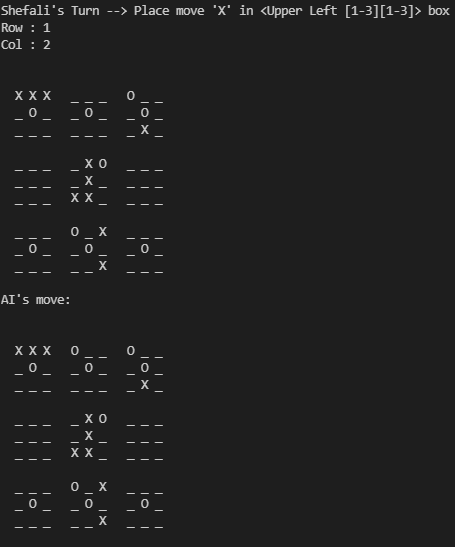
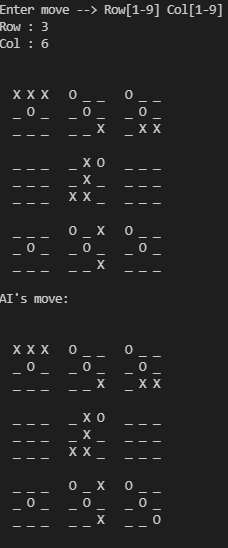
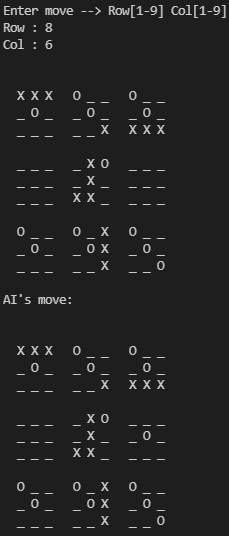
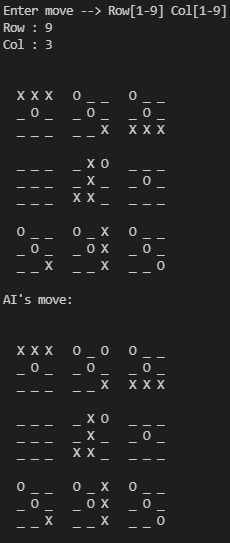
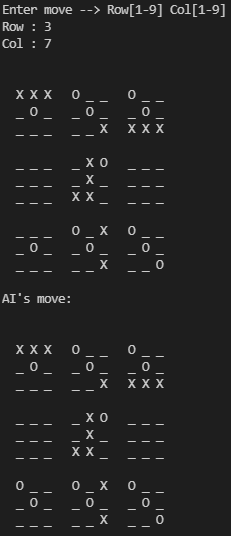
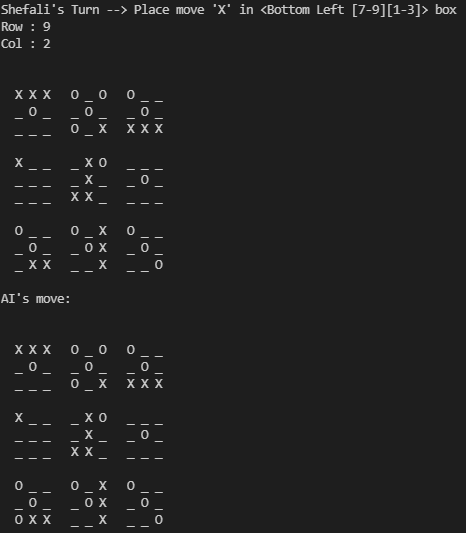
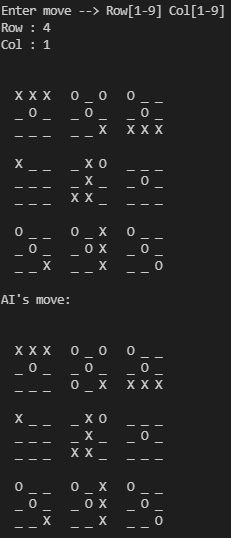
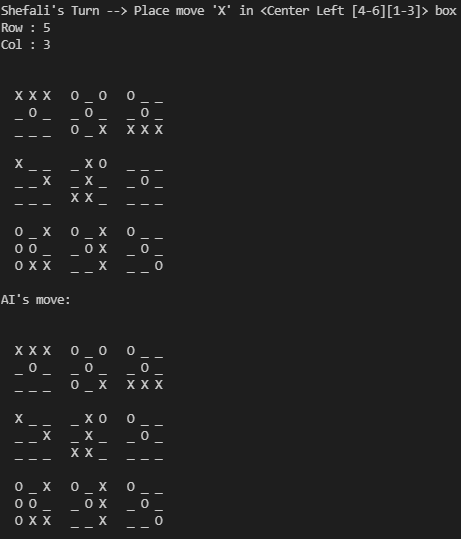
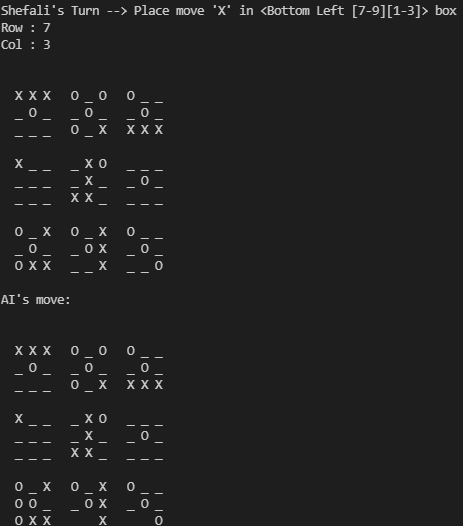
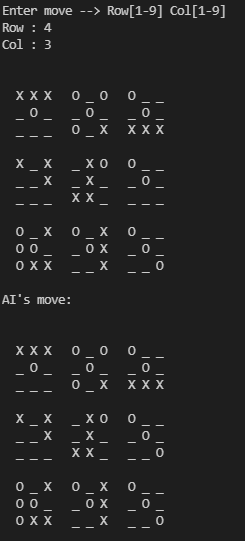
** **

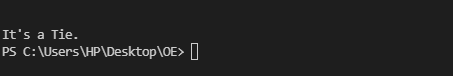
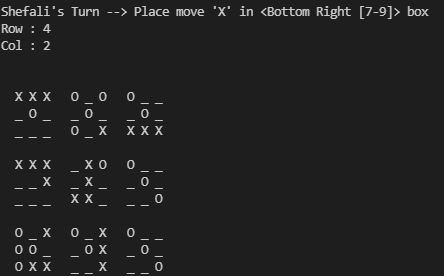
** **

**      **

*Tie:*

****

**           **

****

**LEARNING OUTCOME:**

While tic-tac-toe is elementary to solve and can be done nearly instantly using [depth-first search](https://en.wikipedia.org/wiki/Depth-first_search), ultimate tic-tac-toe cannot be reasonably solved using any brute-force tactics. Therefore, more creative computer implementations are necessary to play this game.

The most common [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) (AI) tactic, [minimax](https://en.wikipedia.org/wiki/Minimax), may be used to play ultimate tic-tac-toe, but has difficulty playing this. This is because, despite having relatively simple rules, ultimate tic-tac-toe lacks any simple [heuristic evaluation function](https://en.wikipedia.org/wiki/Evaluation_function). This function is necessary in minimax, for it determines how good a specific position is. Although elementary evaluation functions can be made for ultimate tic-tac-toe by taking into account the number of local victories, these largely overlook positional advantage that is much harder to quantify. Without any efficient evaluation function, most typical computer implementations are weak, and therefore there are few computer opponents that can consistently outplay humans.

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Design Based Experiment) sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP)** | | | |
| Programme | B. Tech CSE | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | Semester | 7 |
| Student Name |  | Enrollment No. |  |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Designing Concept (D) | 3 |  |  |
| Application of Knowledge (E) | 2 |  |  |
| Performance (F) | 3 |  |  |
| Result (G) | 2 |  |  |
| Total | 10 |  |  |