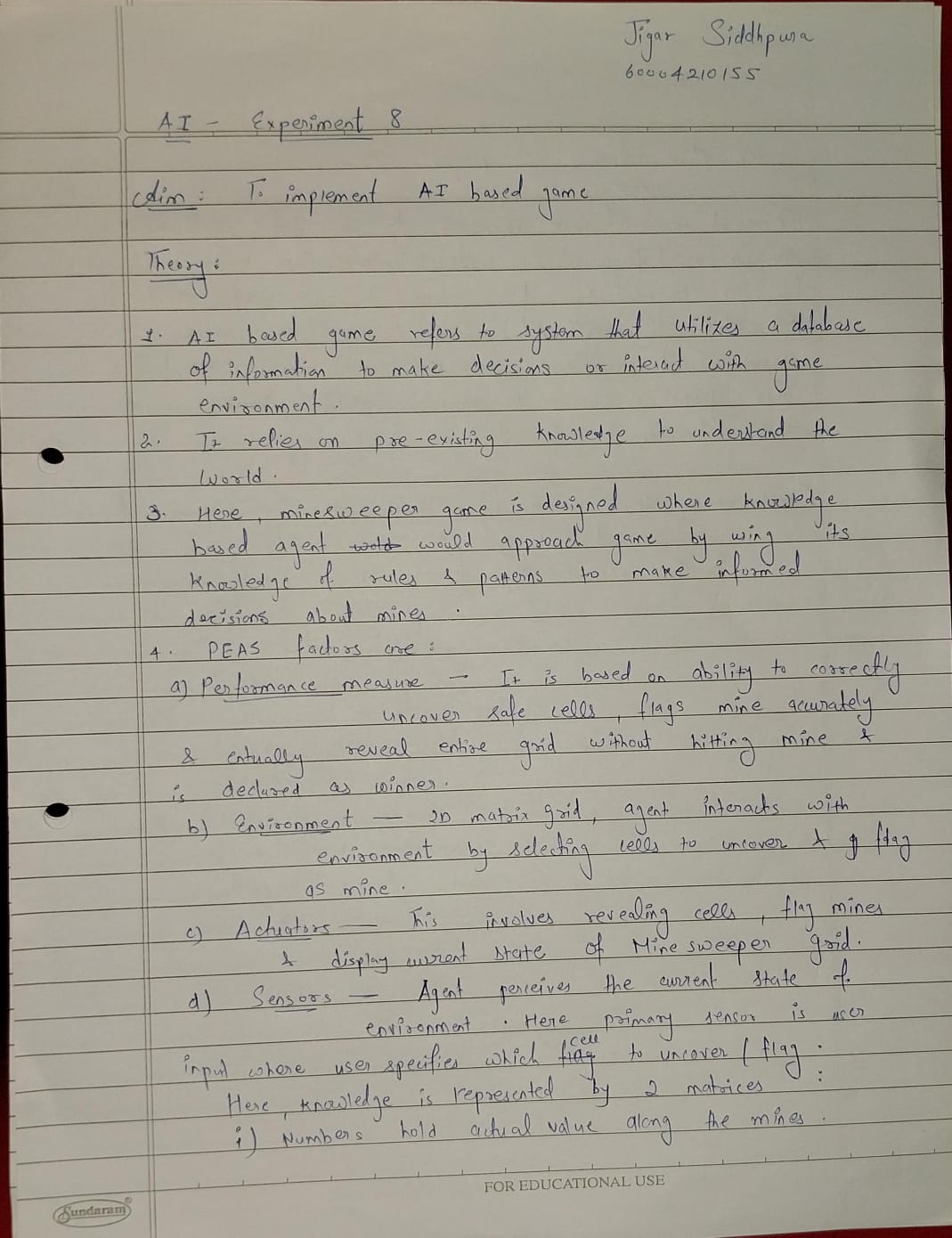
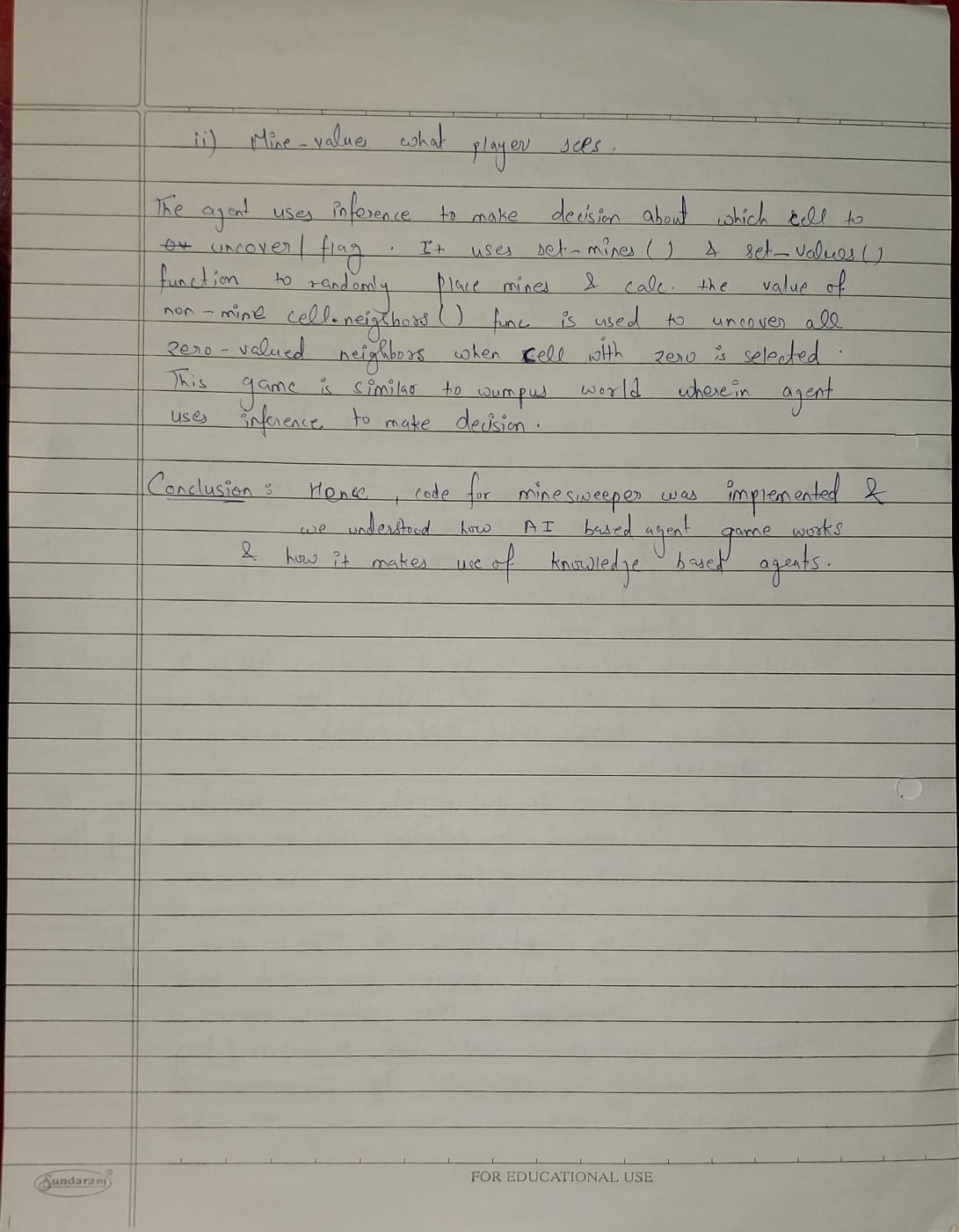
**Name: Jigar Siddhpura SAPID:** 60004200155

**DIV: C/C2** **Branch:** Computer Engineering

AI EXPERIMENT 8





**Code :**

*# Importing packages*

import random

import os

*# Printing the Minesweeper Layout*

def print\_mines\_layout():

*global* mine\_values

*global* n

    print()

print("\t\t\tMINESWEEPER\n")

    st = "   "

    for i in range(n):

        st = st + "     " + str(i + 1)

print(st)

    for r in range(n):

        st = "     "

        if r == 0:

            for col in range(n):

                st = st + "\_\_\_\_\_\_"

            print(st)

        st = "     "

        for col in range(n):

            st = st + "|     "

        print(st + "|")

        st = "  " + str(r + 1) + "  "

        for col in range(n):

            st = st + "|  " + str(mine\_values[r][col]) + "  "

        print(st + "|")

        st = "     "

        for col in range(n):

            st = st + "|\_\_\_\_\_"

        print(st + '|')

print()

*# Function for setting up Mines*

def set\_mines():

*global* numbers

*global* mines\_no

*global* n

*# Track of number of mines already set up*

    count = 0

while count < mines\_no:

*# Random number from all possible grid positions*

        val = random.randint(0, n \* n - 1)

*# Generating row and column from the number*

        r = val // n

        col = val % n

*# Place the mine, if it doesn't already have one*

        if numbers[r][col] != -1:

            count = count + 1

            numbers[r][col] = -1

*# Function for setting up the other grid values*

def set\_values():

*global* numbers

*global* n

*# Loop for counting each cell value*

    for r in range(n):

        for col in range(n):

*# Skip, if it contains a mine*

            if numbers[r][col] == -1:

                Continue

*# Check up*

            if r > 0 and numbers[r - 1][col] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check down*

            if r < n - 1 and numbers[r + 1][col] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check left*

            if col > 0 and numbers[r][col - 1] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check right*

            if col < n - 1 and numbers[r][col + 1] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check top-left*

            if r > 0 and col > 0 and numbers[r - 1][col - 1] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check top-right*

            if r > 0 and col < n - 1 and numbers[r - 1][col + 1] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check below-left*

            if r < n - 1 and col > 0 and numbers[r + 1][col - 1] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Check below-right*

            if r < n - 1 and col < n - 1 and numbers[r + 1][col + 1] == -1:

                numbers[r][col] = numbers[r][col] + 1

*# Recursive function to display all zero-valued neighbours*

def neighbours(r, col):

*global* mine\_values

*global* numbers

*global* vis

*# If the cell already not visited*

if [r, col] not in vis:

*# Mark the cell visited*

        vis.append([r, col])

*# If the cell is zero-valued*

        if numbers[r][col] == 0:

*# Display it to the user*

            mine\_values[r][col] = numbers[r][col]

*# Recursive calls for the neighbouring cells*

            if r > 0:

                neighbours(r - 1, col)

            if r < n - 1:

                neighbours(r + 1, col)

            if col > 0:

                neighbours(r, col - 1)

            if col < n - 1:

                neighbours(r, col + 1)

            if r > 0 and col > 0:

                neighbours(r - 1, col - 1)

            if r > 0 and col < n - 1:

                neighbours(r - 1, col + 1)

            if r < n - 1 and col > 0:

                neighbours(r + 1, col - 1)

            if r < n - 1 and col < n - 1:

                neighbours(r + 1, col + 1)

*# If the cell is not zero-value*

        if numbers[r][col] != 0:

            mine\_values[r][col] = numbers[r][col]

*# Function for clearing the terminal*

def clear():

os.system("clear")

*# Function to display the instructions*

def instructions():

    print("Instructions:")

    print("1. Enter row and column number to select a cell, Example \"2 3\"")

    print(

        "2. In order to flag a mine, enter F after row and column numbers, Example \"2 3 F\""

)

*# Function to check for completion of the game*

def check\_over():

*global* mine\_values

*global* n

*global* mines\_no

*# Count of all numbered values*

count = 0

*# Loop for checking each cell in the grid*

    for r in range(n):

        for col in range(n):

*# If cell not empty or flagged*

            if mine\_values[r][col] != ' ' and mine\_values[r][col] != 'F':

                count = count + 1

*# Count comparison*

    if count == n \* n - mines\_no:

        return True

    else:

        return False

*# Display all the mine locations*

def show\_mines():

*global* mine\_values

*global* numbers

*global* n

    for r in range(n):

        for col in range(n):

            if numbers[r][col] == -1:

                mine\_values[r][col] = 'M'

if \_\_name\_\_ == "\_\_main\_\_":

*# Size of grid*

    n = 8

*# Number of mines*

mines\_no = 8

*# The actual values of the grid*

    numbers = [[0 for y in range(n)] for x in range(n)]

*# The apparent values of the grid*

    mine\_values = [[' ' for y in range(n)] for x in range(n)]

*# The positions that have been flagged*

flags = []

*# Set the mines*

set\_mines()

*# Set the values*

set\_values()

*# Display the instructions*

instructions()

*# Variable for maintaining Game Loop*

over = False

*# The GAME LOOP*

    while not over:

        print\_mines\_layout()

*# Input from the user*

        inp = input(

            "Enter row number followed by space and column number = ").split()

*# Standard input*

        if len(inp) == 2:

*# Try block to handle errant input*

            try:

                val = list(map(int, inp))

            except ValueError:

                clear()

                print("Wrong input!")

                instructions()

                Continue

*# Flag input*

        elif len(inp) == 3:

            if inp[2] != 'F' and inp[2] != 'f':

                clear()

                print("Wrong Input!")

                instructions()

                Continue

*# Try block to handle errant input*

            try:

                val = list(map(int, inp[:2]))

            except ValueError:

                clear()

                print("Wrong input!")

                instructions()

                Continue

*# Sanity checks*

            if val[0] > n or val[0] < 1 or val[1] > n or val[1] < 1:

                clear()

                print("Wrong input!")

                instructions()

                Continue

*# Get row and column numbers*

            r = val[0] - 1

            col = val[1] - 1

*# If cell already been flagged*

            if [r, col] in flags:

                clear()

                print("Flag already set")

                Continue

*# If cell already been displayed*

            if mine\_values[r][col] != ' ':

                clear()

                print("Value already known")

                Continue

*# Check the number for flags*

            if len(flags) < mines\_no:

                clear()

                print("Flag set")

*# Adding flag to the list*

                flags.append([r, col])

*# Set the flag for display*

                mine\_values[r][col] = 'F'

                continue

            else:

                clear()

                print("Flags finished")

                Continue

        else:

            clear()

            print("Wrong input!")

            instructions()

            Continue

*# Sanity checks*

        if val[0] > n or val[0] < 1 or val[1] > n or val[1] < 1:

            clear()

            print("Wrong Input!")

            instructions()

            Continue

*# Get row and column number*

        r = val[0] - 1

        col = val[1] - 1

*# Unflag the cell if already flagged*

        if [r, col] in flags:

            flags.remove([r, col])

*# If landing on a mine --- GAME OVER*

        if numbers[r][col] == -1:

            mine\_values[r][col] = 'M'

            show\_mines()

            print\_mines\_layout()

            print("Landed on a mine. GAME OVER!!!!!")

            over = True

            continue

*# If landing on a cell with 0 mines in neighboring cells*

        elif numbers[r][col] == 0:

            vis = []

            mine\_values[r][col] = '0'

            neighbours(r, col)

*# If selecting a cell with atleast 1 mine in neighboring cells*

        else:

            mine\_values[r][col] = numbers[r][col]

*# Check for game completion*

        if (check\_over()):

            show\_mines()

            print\_mines\_layout()

            print("Congratulations!!! YOU WIN")

            over = True

            continue

        clear()

**Output :**

