**NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY**

Artificial Intelligence

(CACSE11))

LAB FILE

**Name: SNEHA GUPTA**

**Roll No.: 2021UCA1859**

Branch: COMPUTER SCIENCE WITH ARTIFICIAL INTELLIGENCE

Section: 1

INDEX

1. Experiment the vacuum cleaner world example.
2. Design a program for the greedy best first search or A\* search
3. Construct the simulated annealing algorithm over the travelling salesman problem.
4. Implement a basic binary genetic algorithm for a given problem.
5. Experiment the Graph Coloring CSP or Cryptarithmetic Puzzle.
6. Implement the Tic-Tac-Toe game using any adversarial searching algorithm

**Q1. Experiment the vacuum cleaner world example.**

*Problem-*

Our vacuum cleaner is our agent(goal based) and the goal of this agent,

is to clean up the whole area.

We have two rooms and one vacuum cleaner. There is dirt in both the rooms

and it is to be cleaned. The vacuum cleaner is present in any one of these

rooms. So, we have to reach a state in which both the rooms are clean and are

dust free.

So, there are eight possible states possible in our problem :

Here, states 1 and 2 are our initial states and state 7 and state 8 are our final

states (goal states)

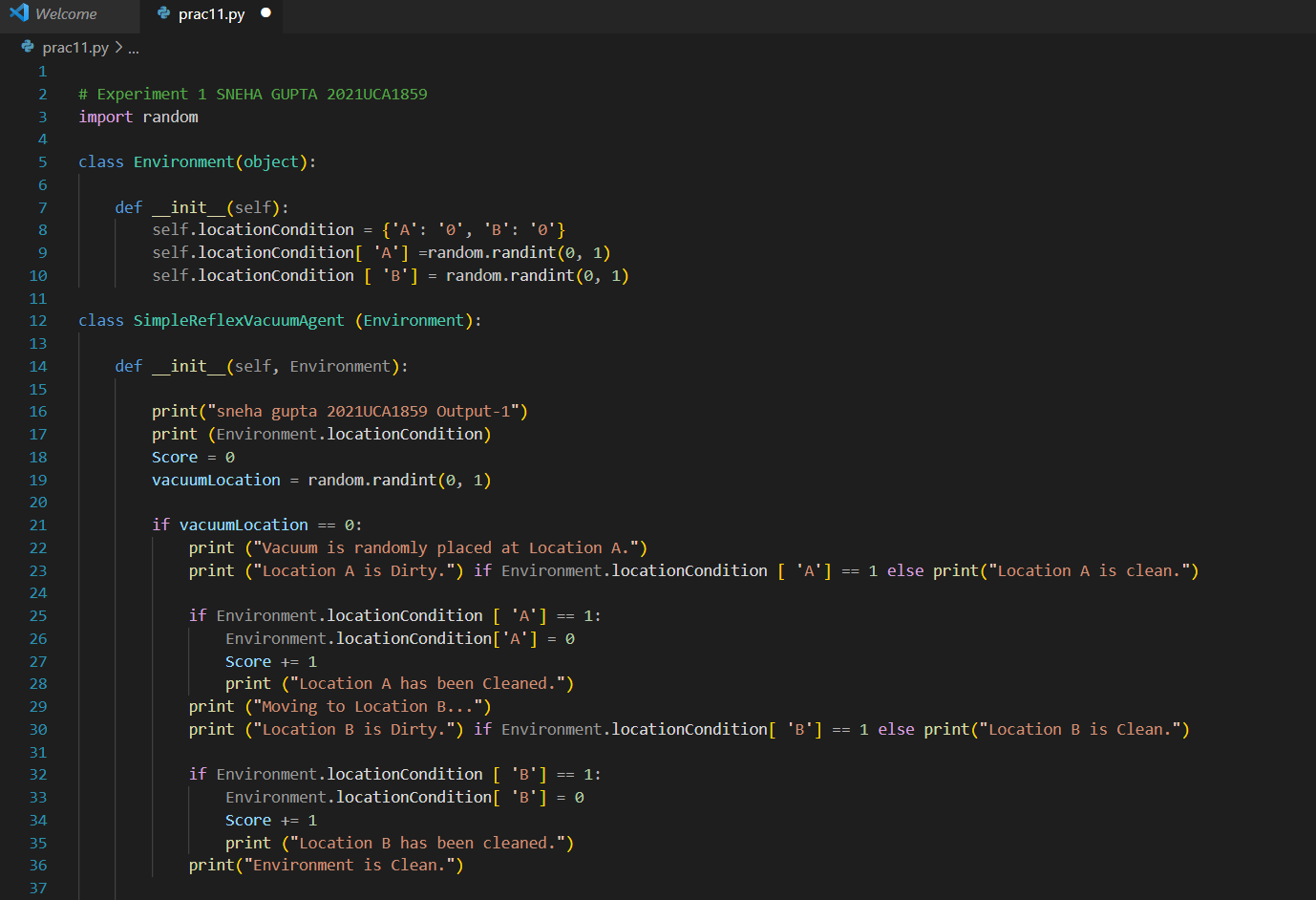
The vacuum cleaner can perform the following functions: move left, move right,

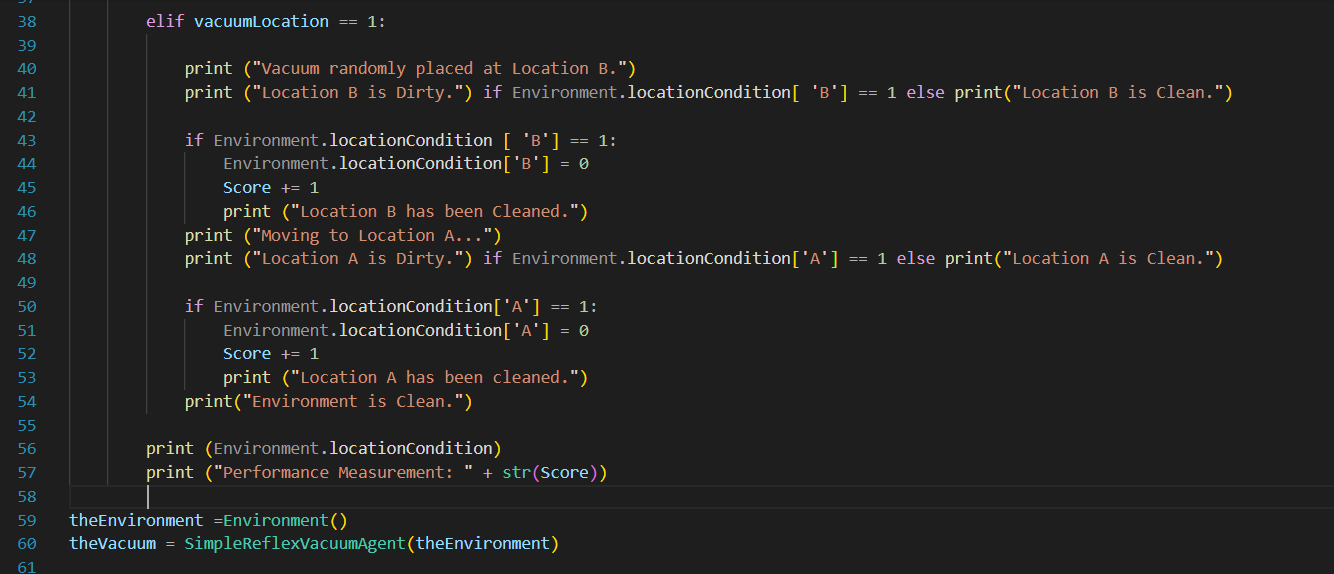
move forward, move backward and to suck dust. But as there are only two rooms

in our problem, the vacuum cleaner performs only the following functions here:

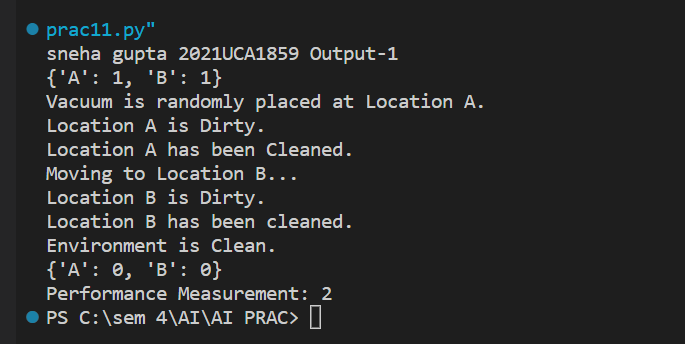
move left, move right and suck

*Program-*

**

**

*Output* –



**Q2) Design a program for the greedy best first search or A\* search**

BFS-) It is of the most common search strategies. It generally starts from the

root node and examines the neighbor nodes and then moves to the next level.

It uses First-in First-out (FIFO) strategy as it gives the shortest path to

achieving the solution.

Advantages -

BFS will never be trapped in any unwanted nodes.

If the graph has more than one solution, then BFS will return the optimal

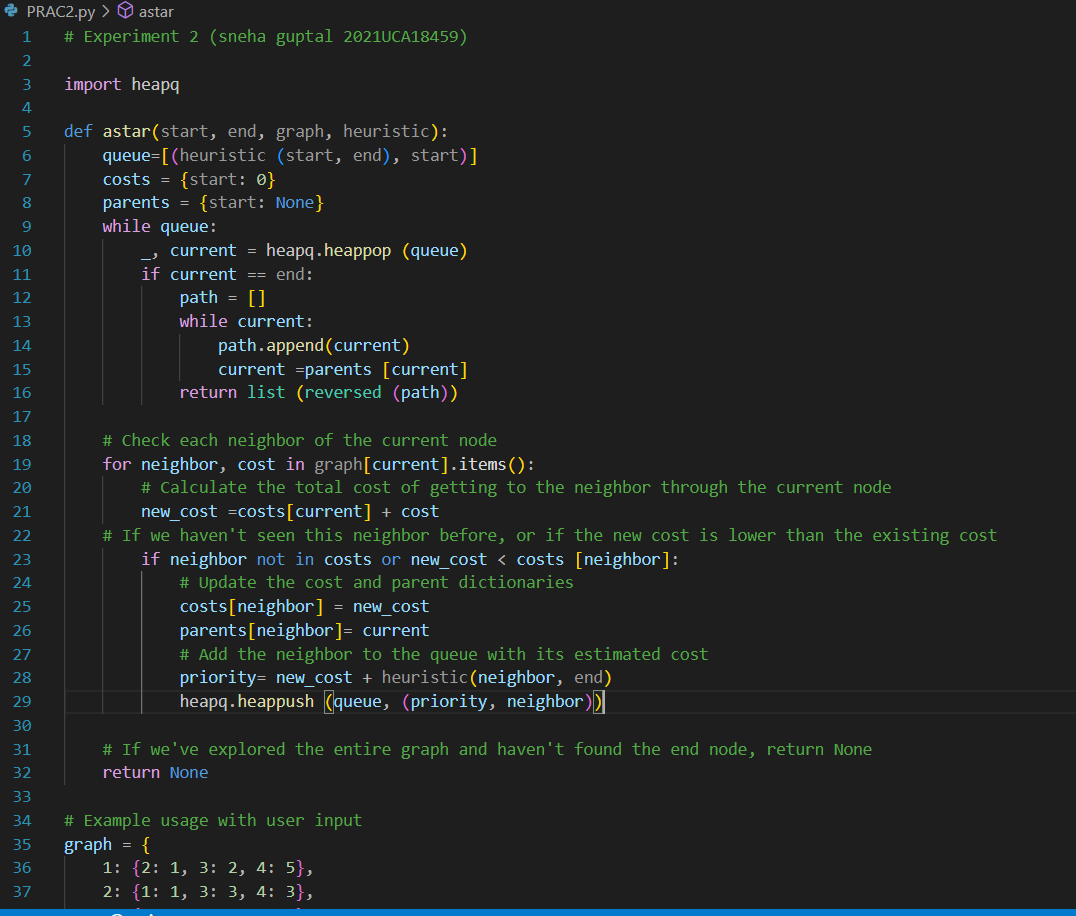
solution which provides the shortest path.

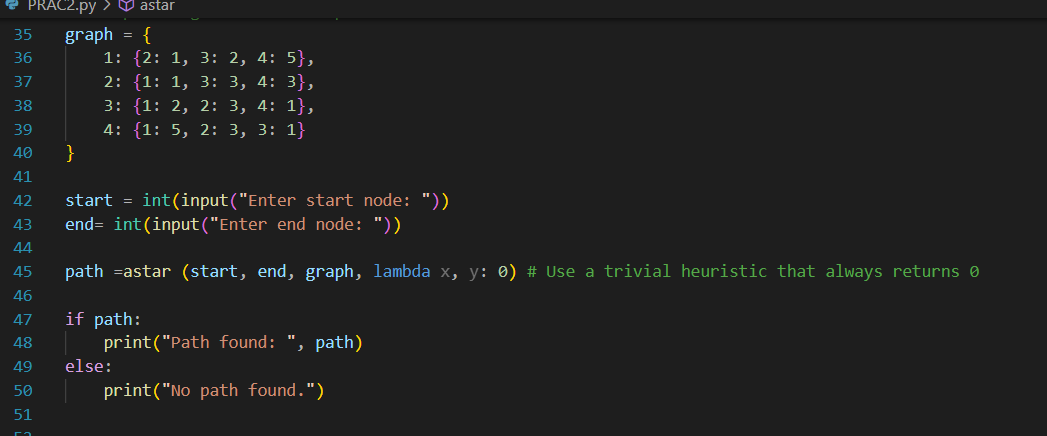
DisadvantagesBFS stores all the nodes in the current level and then go to the next level. It

requires a lot of memory to store the nodes.

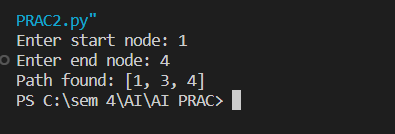
BFS takes more time to reach the goal state which is far away.

*Program-*

**

**

*Output:*

**

**Q3) Construct the simulated annealing algorithm over the travelling salesman problem.**

The traveling salesman problem (TSP) is a classic optimization problem in computer

science and mathematics. The problem can be stated as follows: Given a list of cities

and the distances between each pair of cities, what is the shortest possible route that

visits each city exactly once and returns to the origin city

The problem is often stated in the context of a traveling salesman who needs to visit a

set of cities and return to the starting point while minimizing the total distance traveled.

However, the problem has applications beyond just logistics and transportation, such as

in circuit board design and DNA sequencing

The TSP is an NP-hard problem, which means that as the size of the problem grows, the

time required to find the optimal solution grows exponentially. Therefore, exact

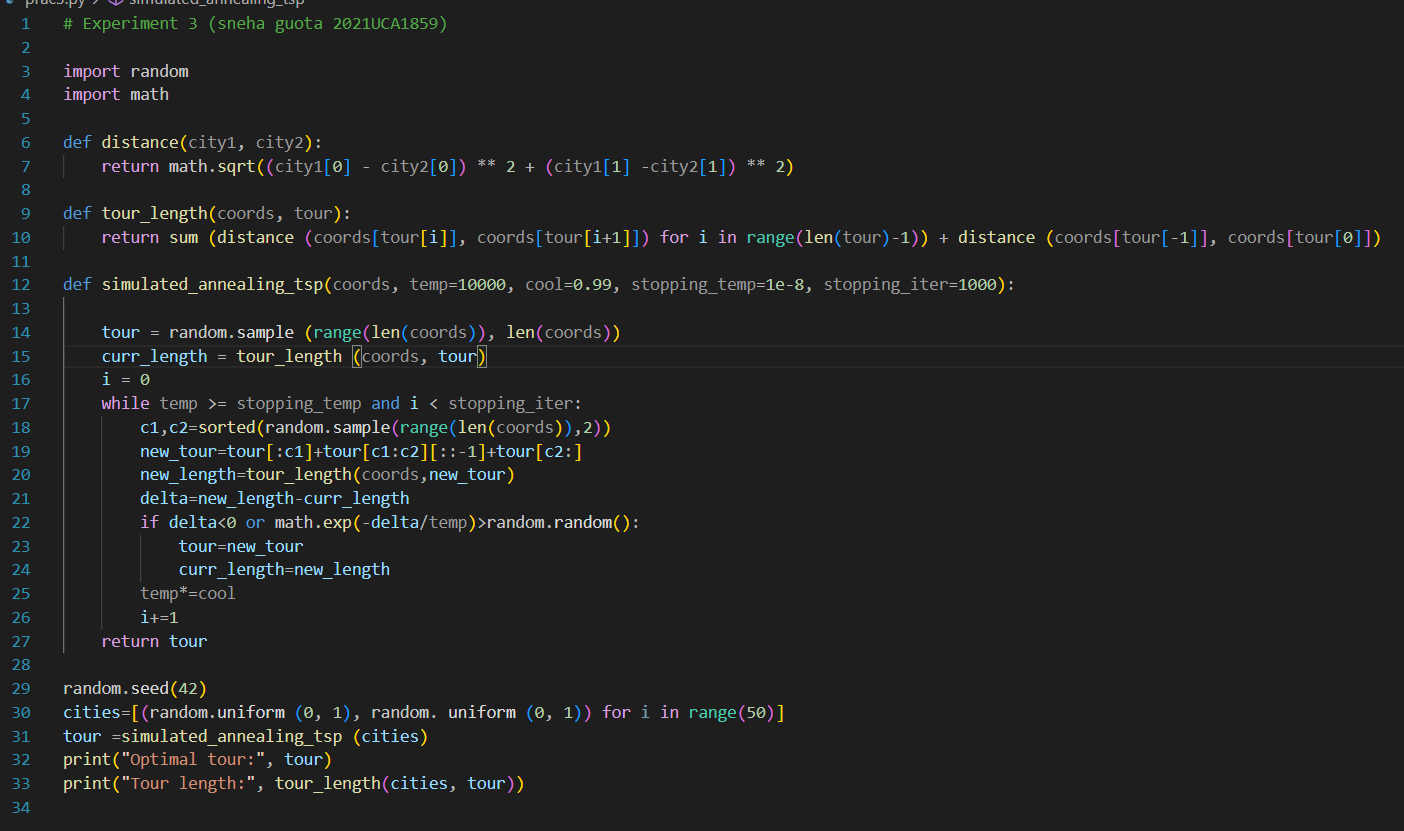
algorithms for solving the TSP are only practical for small instances of the problem.

Instead, heuristic and metaheuristic algorithms such as simulated annealing, genetic

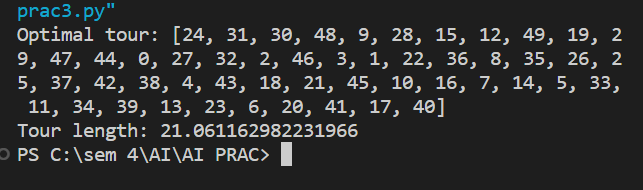
algorithms, and ant colony optimization are often used to find approximate solutions

that are close to the optimal solution in a reasonable amount of time

*Program*

**

*Output-*

**

**Q4) Implement a basic binary genetic algorithm for a given problem.**

A genetic algorithm is a heuristic search and optimization technique inspired by the

process of natural selection. It is often used to find approximate solutions to

optimization and search problems. A binary genetic algorithm operates on a population

of binary strings, also called chromosomes, where each bit in the string represents a

decision variable

1)The problem to be solved is defined by the objective\_function, which takes a binary

string (or chromosome) as input and returns a score representing the quality of the

solution.

2)The fitness\_function is defined to calculate the fitness score of a chromosome based

on its objective score

3)The genetic algorithm initializes a population of population\_size chromosomes with

random bits.

4)The algorithm then evaluates the fitness of each chromosome in the population by

applying the fitness\_function

5)The algorithm then applies crossover to the selected parent chromosomes based on

the crossover\_rate parameter. In this implementation, single-point crossover is used,

where a random point in the chromosome is selected and the tail of one parent is

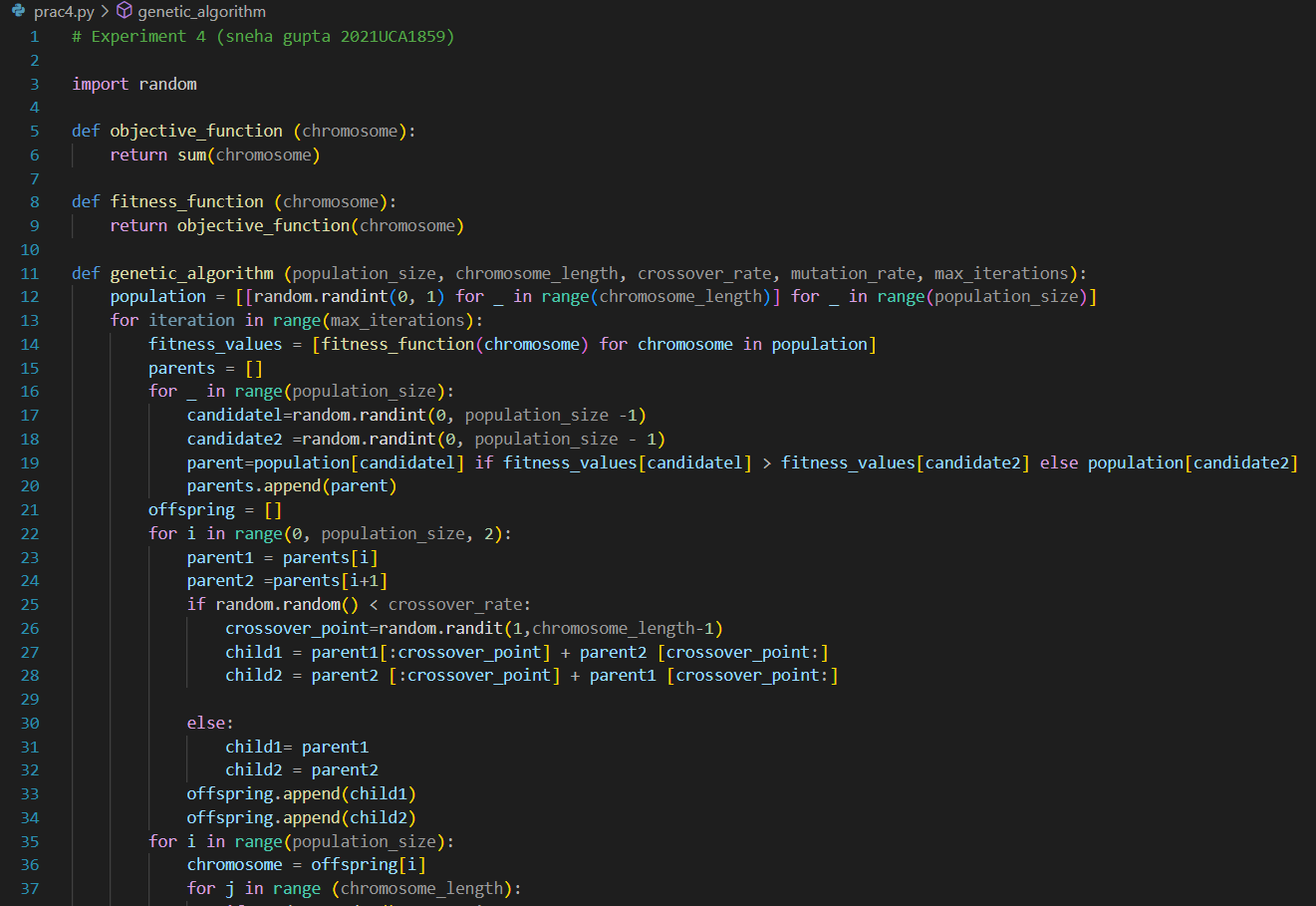
swapped with the tail of the other parent to create two offspring.

6)The algorithm then applies mutation to the offspring chromosomes based on the

mutation\_rate parameter. In this implementation, a single bit in the chromosome is

flipped with a small probability.

*Program-*

**

**

*Output-*

**

**Q5)Experiment the Graph Coloring CSP or Cryptarithmetic Puzzle**

A function is\_valid that checks whether a given vertex can be colored with a particular

color without violating the constraint that no two adjacent vertices have the same color.

The next function defined is backtrack, which is the main function that performs the

backtracking search to find a valid coloring of the graph. The function takes in the

current vertex to be colored, the current coloring of the vertices, and the number of

colors to be used

The function first checks if all vertices have been colored. If so, it means we have found

a valid coloring and the function returns True

If not, the function loops through all possible colors for the current vertex and checks if

the color is valid using the is\_valid function. If the color is valid, it sets the color for the

current vertex and recursively calls backtrack for the next vertex

If the recursive call returns True, it means a valid coloring was found and the function

returns True. If not, the color is removed for the current vertex and the function

continues with the next color

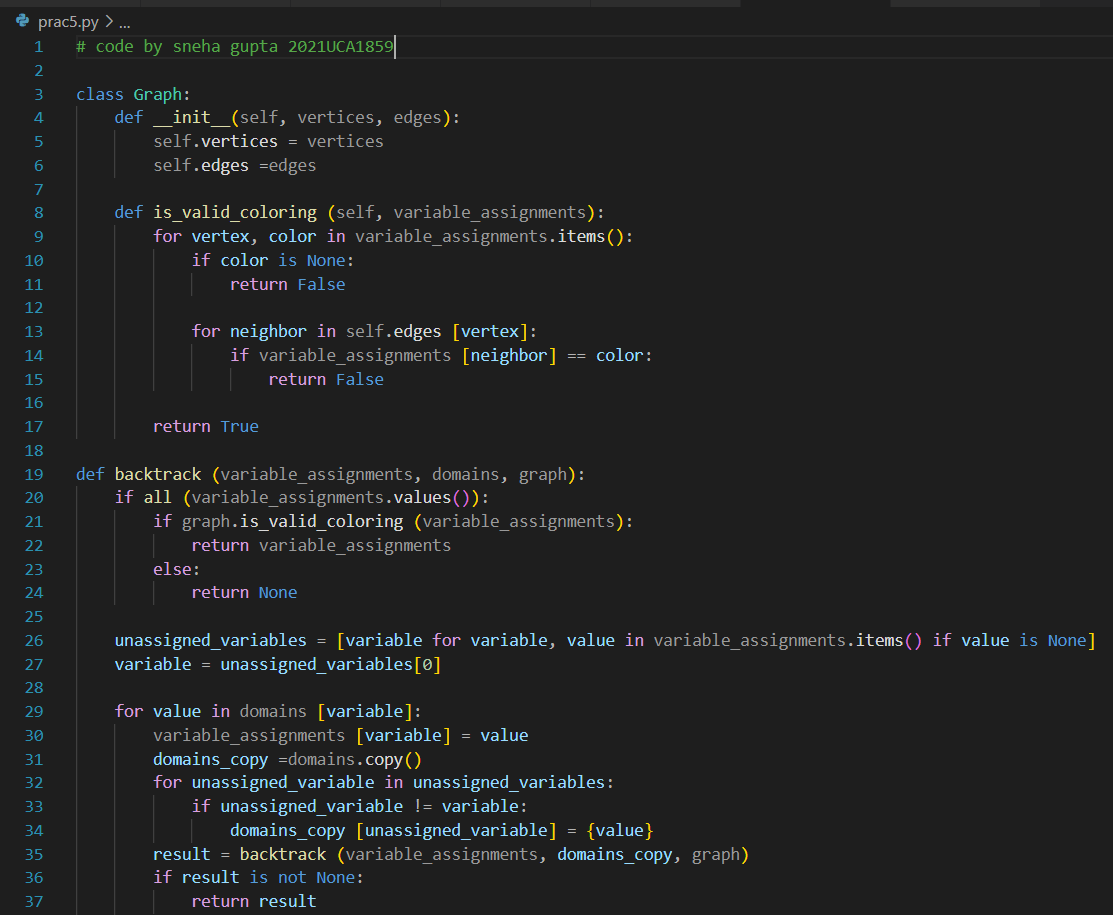
Finally, the main program reads in the input graph and number of colors from the user,

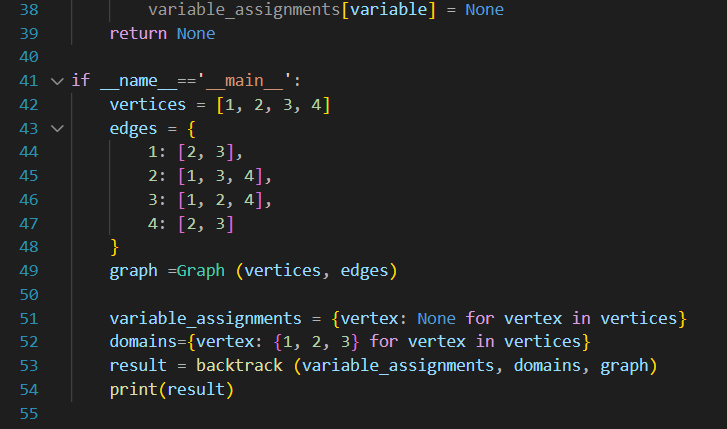
and calls the backtrack function to find a valid coloring of the graph. If a valid coloring is

found, it is printed to the console. If not, a message indicating that no valid coloring was

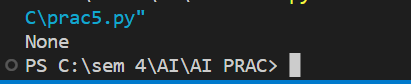
found is printed

*Program-*

**

**

*Output-*

**

**Q6) Implement the Tic-Tac-Toe game using any adversarial searching algorithm.**

This is an implementation of the Minimax algorithm

The code uses a few pre-defined constants:

1) HUMAN = -1 - this represents the human player

2) COMP = +1 - this represents the AI/Computer player

3) board = [ [0, 0, 0], [0, 0, 0], [0, 0, 0], ] - this is the initial game state. A

two-dimensional array of size 3x3, representing the game board.

evaluate(state): returns the score of the game state. +1 if the computer wins, -1 if the

human wins, and 0 for a tie

wins(state, player): tests if a specific player wins. Returns True if the player wins

game\_over(state): tests if the game is over (either player wins or tie)

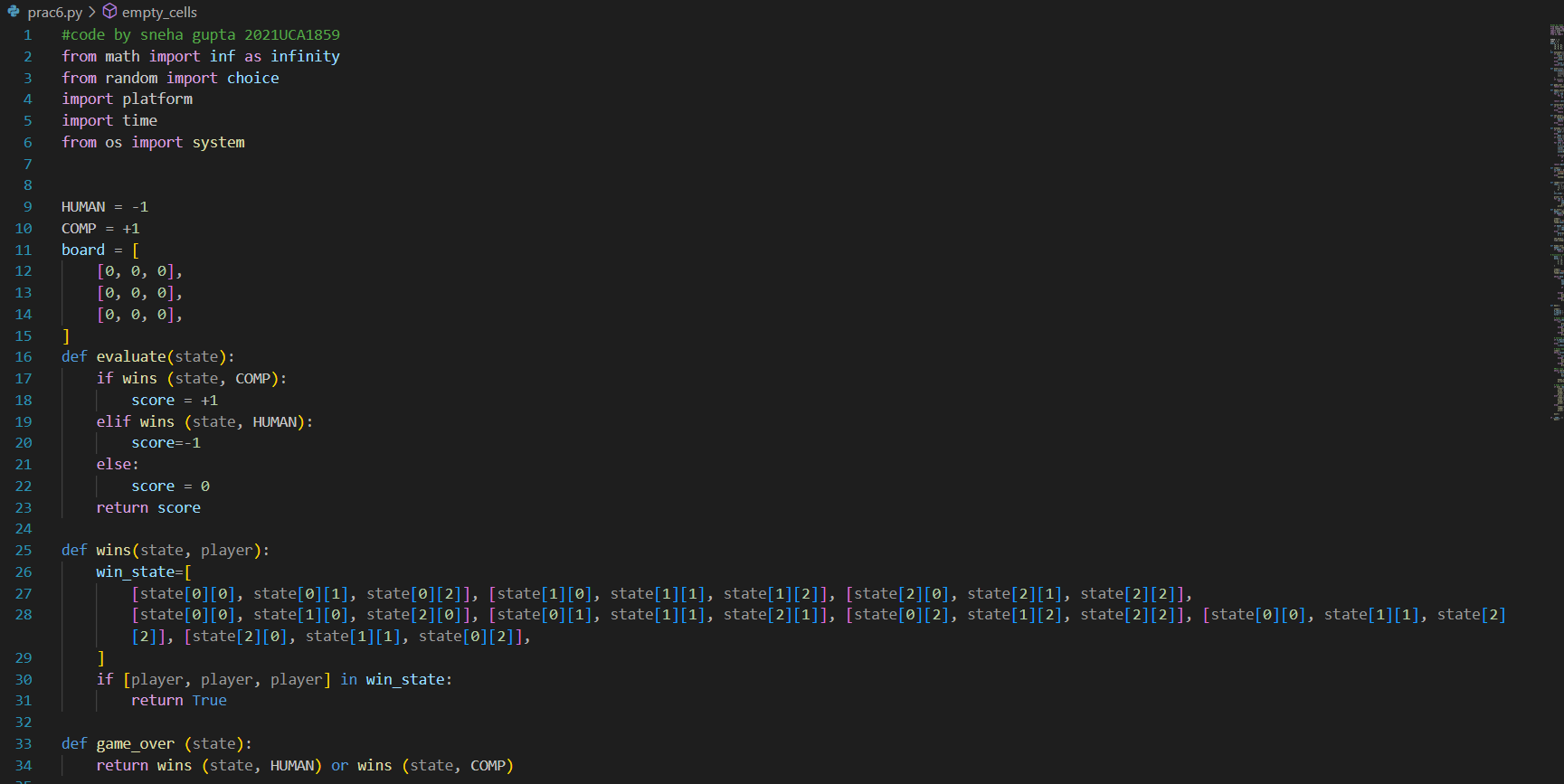
empty\_cells(state): returns a list of empty cells (positions) on the board.

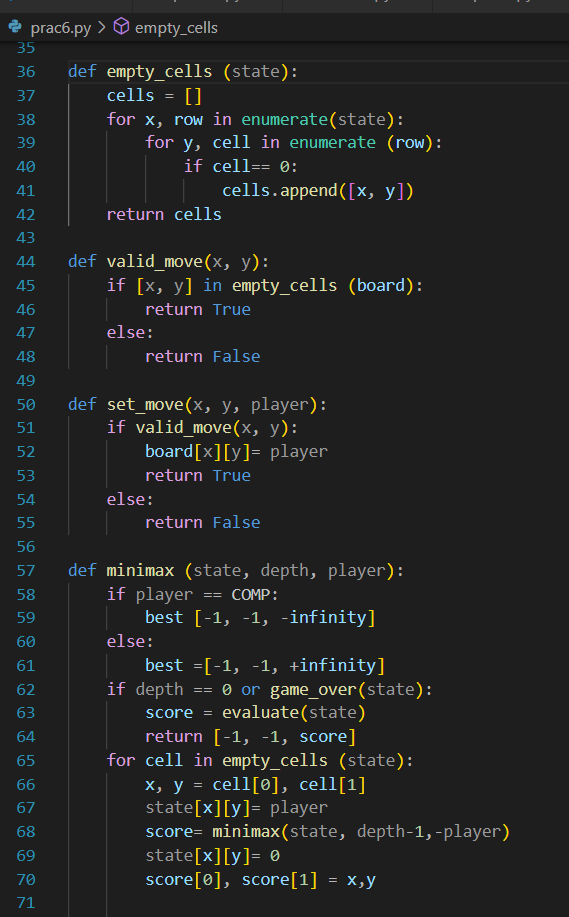
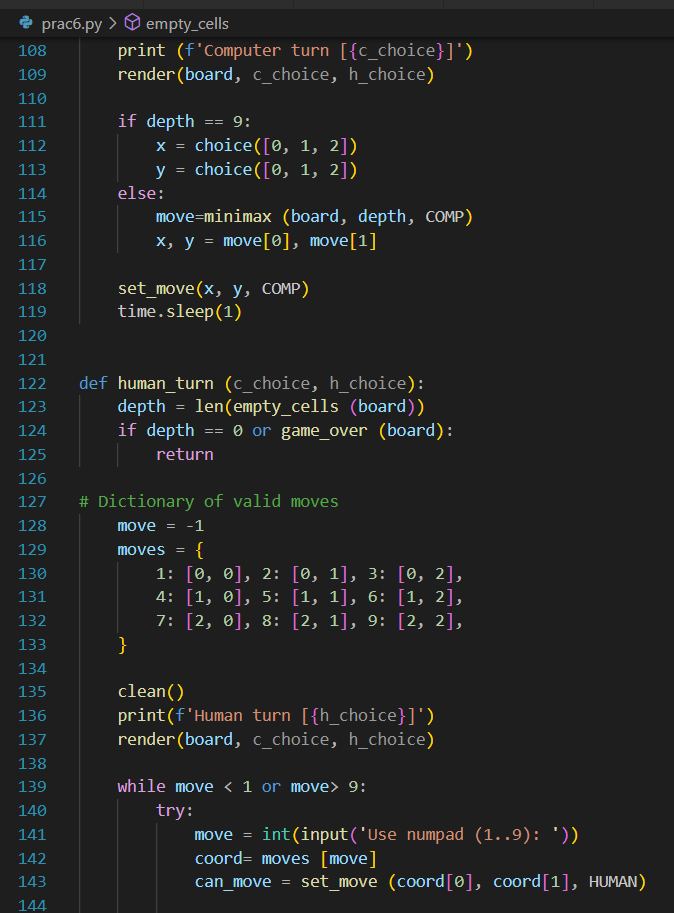
valid\_move(x, y): checks if a move (x, y) is valid, i.e., if the cell is empty

set\_move(x, y, player): sets the move on the board if the coordinates are valid

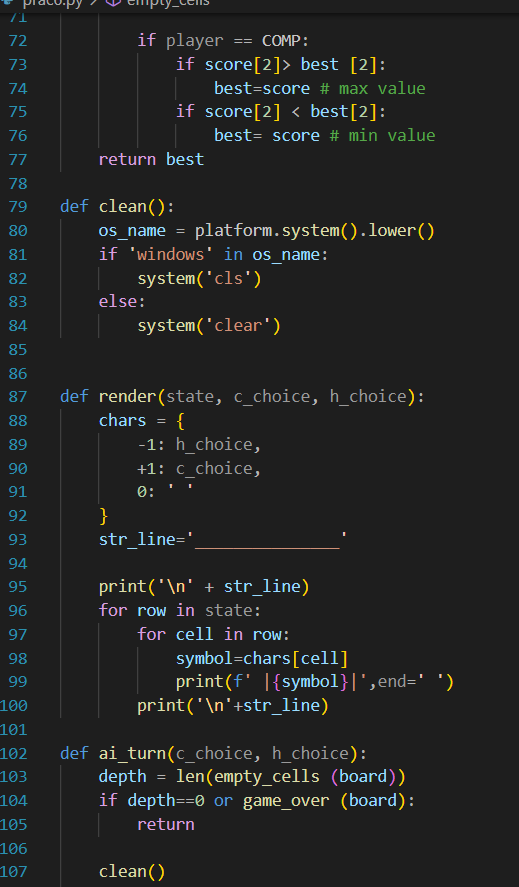
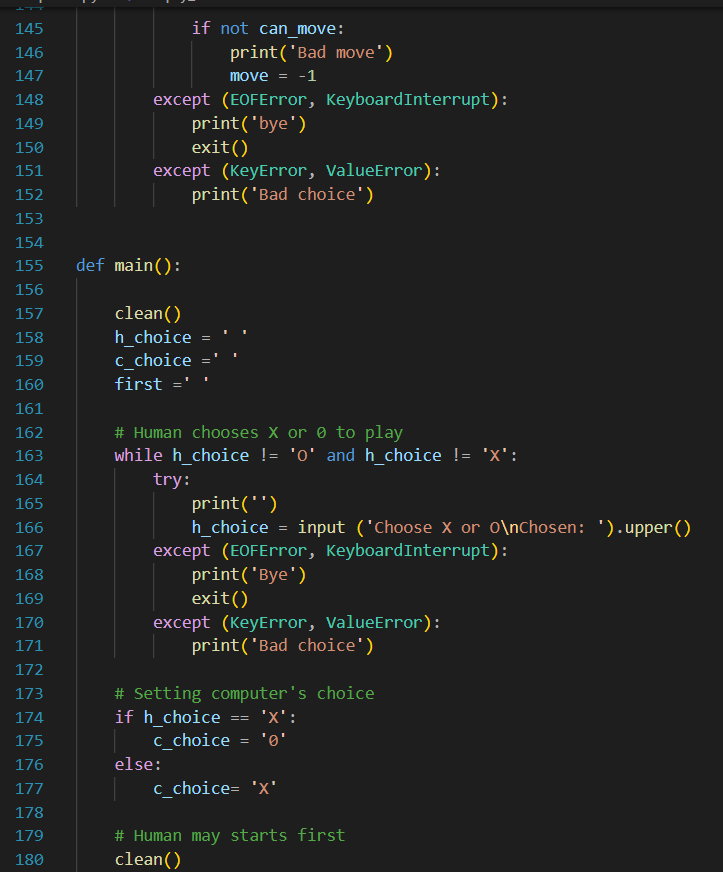
clean(): clears the console screen

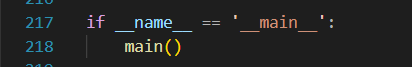
*Program-*

**

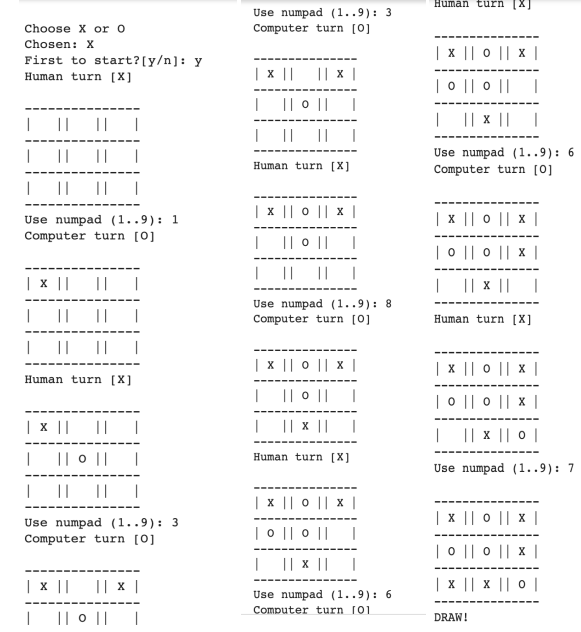
**

*Output*

**

**

*Output:*

**