AI LAB 6 – 9763-Harsh Parmar– Batch D

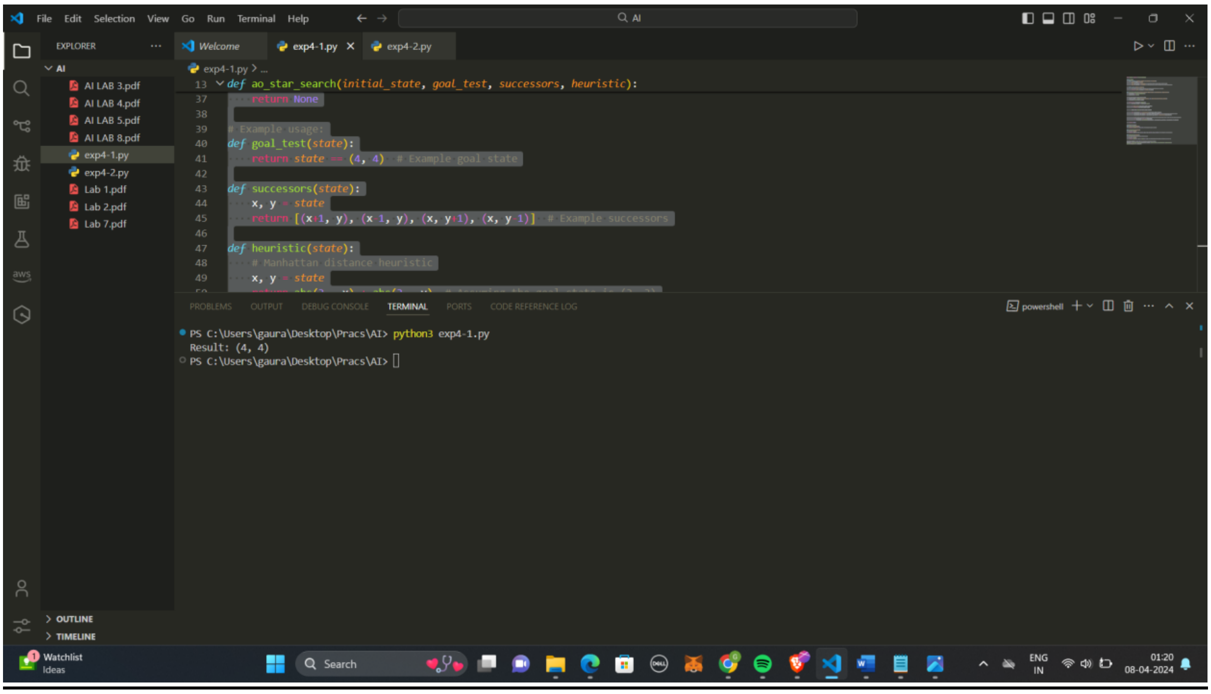
Implementation of AO* algorithm

Code:

```
import random

# Define the initial state of the block world
initial_state = ['A', 'B', 'C', 'D']

# Define the goal state of the block world
goal_state = ['D', 'C', 'B', 'A']

# Define a function to calculate the heuristic (number of misplaced blocks)
def heuristic(state):
    return sum([1 for i, j in zip(state, goal_state) if i != j])

# Define a function to generate neighboring states (move a block to the top)
def generate_neighbors(state):
    neighbors = []
    for i in range(len(state)):
        for j in range(i+1, len(state)):
            neighbor = state[:i] + [state[j]] + state[i:j] + state[j+1:]
            neighbors.append(neighbor)
    return neighbors

# Define the Hill Climbing algorithm
```

```
def hill_climbing(initial_state, goal_state):
    current_state = initial_state
    while True:
        current_heuristic = heuristic(current_state)
        neighbors = generate_neighbors(current_state)
        best_neighbor = min(neighbors, key=lambda neighbor: heuristic(neighbor))
        if heuristic(best_neighbor) >= current_heuristic:
            return current_state
        current_state = best_neighbor


# Run the Hill Climbing algorithm
final_state = hill_climbing(initial_state, goal_state)


# Print the result
print("Initial State:", initial_state)
print("Final State:", final_state)
```

OUTPUT:



```python
def ao_star_search(initial_state, goal_test, successors, heuristic):
    return None

# Example usage:
def goal_test(state):
    return state == (4, 4)  # Example goal state

def successors(state):
    x, y = state
    return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]  # Example successors

def heuristic(state):
    # Manhattan distance heuristic
    x, y = state
```

```
PS C:\Users\gaura\Desktop\Pracs\AI> python3 exp4-1.py
Result: (4, 4)
PS C:\Users\gaura\Desktop\Pracs\AI> 
```