

**Divyadharshini K**

**241501053**

## **Experiment 9**

### **IMPLEMENTATION OF BLOCKS WORLD**

#### **PROGRAM**

##### **Aim:**

To implement Blocks World Program.

##### **Scenario:**

A robotic arm in a warehouse is programmed to rearrange blocks according to a given goal state. The Blocks World problem involves moving blocks from an initial configuration to a desired goal configuration while following specific constraints.

A robotic system is given an initial state and a goal state:

Initial State:

A is on B

B is on table

C is on table

Goal State

B is on C

A is on B

C is on table

##### **Procedure:**

1. Initialize the world with an initial state of blocks.
2. Define the goal state that needs to be achieved.
3. Check if the current state matches the goal state:

- If yes, stop the execution.
- If no, continue planning moves.

4. For each block in the goal state:

- If the block is not in its desired position, move it to the correct place. ■ Print the move action.
- Update the current state after each move.

5. Repeat until the goal state is reached.
6. Print the final arrangement of blocks when the goal state is met.

### **Program:**

```
class BlocksWorld:
    def __init__(self):
        self.state = {
            "A": "B", # A is on B
            "B": "table", # B is on table
            "C": "table" # C is on table
        }
        self.goal = {
            "A": "B",
            "B": "C",
            "C": "table"
        }

    def is_goal_state(self):
        return self.state == self.goal

    def move(self, block, destination):
        if block in self.state and self.state[block] != destination:
            print(f"Moving {block} from {self.state[block]} to {destination}")
            self.state[block] = destination

    def plan_moves(self):
        print("\nInitial State:", self.state)
        while not self.is_goal_state():
            for block, target in self.goal.items():
                if self.state[block] != target:
                    self.move(block, target)
```

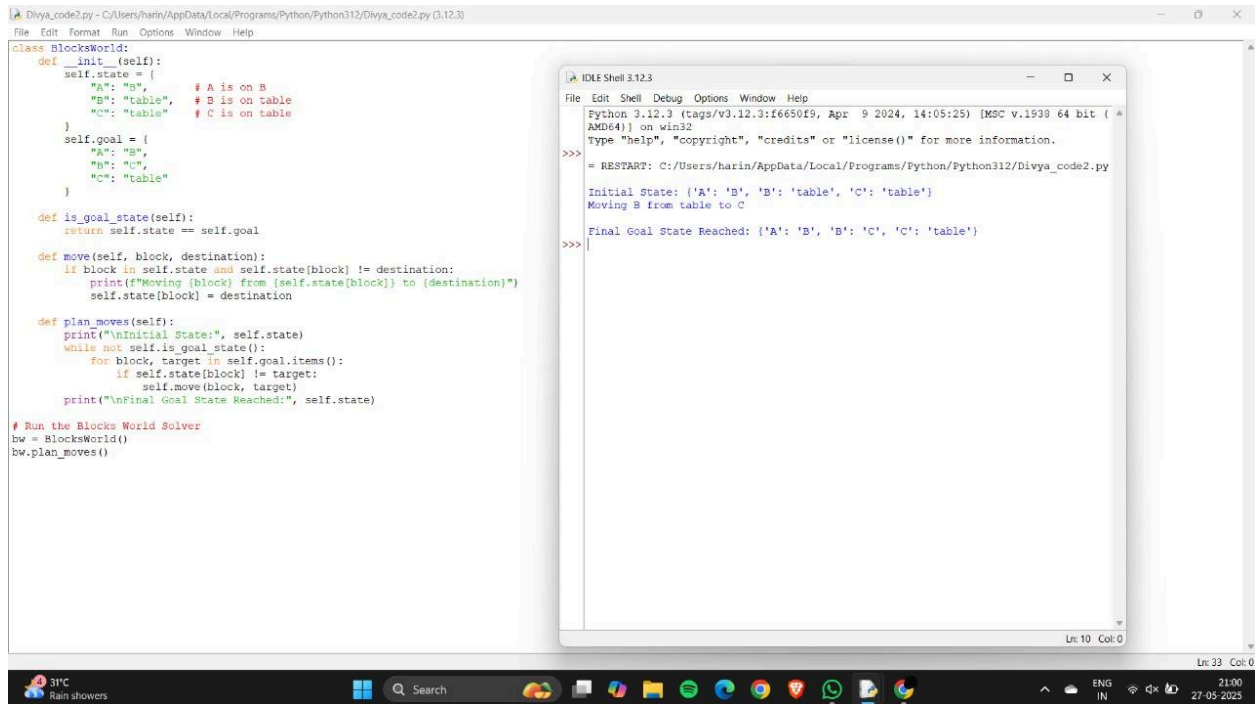
```
print("\nFinal Goal State Reached:", self.state)

# Run the Blocks World Solver

bw = BlocksWorld()

bw.plan_moves()
```

## Output:



The screenshot displays a Python IDE with two windows. The main window shows a Python script for a BlocksWorld solver. The script defines a `BlocksWorld` class with methods for initializing the state, checking the goal, moving blocks, and planning moves. It then creates an instance of the class and calls `plan_moves()`. The output window shows the execution results, including the initial state, the move of block B from the table to position C, and the final goal state reached.

```
class BlocksWorld:
    def __init__(self):
        self.state = {
            "A": "B",      # A is on B
            "B": "table",  # B is on table
            "C": "table"   # C is on table
        }
        self.goal = {
            "A": "B",
            "B": "C",
            "C": "table"
        }

    def is_goal_state(self):
        return self.state == self.goal

    def move(self, block, destination):
        if block in self.state and self.state[block] != destination:
            print(f"Moving {block} from {self.state[block]} to {destination}")
            self.state[block] = destination

    def plan_moves(self):
        print("\nInitial State:", self.state)
        while not self.is_goal_state():
            for block, target in self.goal.items():
                if self.state[block] != target:
                    self.move(block, target)
            print("\nFinal Goal State Reached:", self.state)

# Run the Blocks World Solver
bw = BlocksWorld()
bw.plan_moves()
```

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/harin/AppData/Local/Programs/Python/Python312/Divya_code2.py
Initial State: {'A': 'B', 'B': 'table', 'C': 'table'}
Moving B from table to C
Final Goal State Reached: {'A': 'B', 'B': 'C', 'C': 'table'}
>>>
```