* Lab : 02 :

vaccum cleaner :—

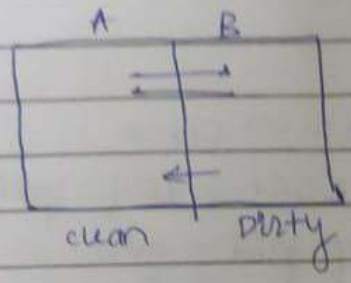
* Algorithm:
  step 1: there are two rooms which are room A and room B
* Intalize : start with room A
  * check room A
    → * If Room A is dirty clean it
    → * If Room A is clean move to next step or room
* Note : ~~Both~~ each rooms can either be clean or dirty.
* check room B
  → If Room B is dirty clean it.
  → If Room B is clean move to next Room A.
* Repeat : continue until both the rooms are clean.
* Stop: when both rooms are clean.
* Result : ~~Dis~~ play ~~when~~ that both the rooms are clean.

* percept sequence :—

i) ('Room A' 'Dirty')
('Room B', 'Dirty')
('Room A', 'clean')
('Room B', 'clean')

('Room A' 'clean')
('Room B' 'clean')

Rooms



① (A, left)
① (A, left) ② (B, clean)
① (A, left) ② (B, clean) ③ (A, right)

Goal: B

* code:

```
class vacuum cleaner :
    def __init__ (self, grid) :
        self.grid = grid
        self.position = (0,0)
def clean (self) :
    x, y = self.position
    if self.grid [x][y] == 1:
        print (f"cleaning position {self.position}")
        self.grid [x][y] = 0
    else :
        print (f" position {self.position} is already
                clean")
def move (self, direction) :
    x, y = self.position
    if direction == 'up' and x > 0 :
        self.position = (x, -1, y)
    elif direction == 'down' and x < len (self.grid)-1:
        self.position = (x+1, y)
    elif direction == 'left' and y > 0 :
        self.position = (x, y-1)
    elif direction == 'right' and y < len (self.grid[0]
                -1 :
        self.position = (x, y+1)
    else :
        print ("move not possible")
def run (self) :
    rows = len (self.grid)
    cols = len (self.grid [0])
for i in range (rows) :
    for j in range (cols) :
```

```
        self.position = (i, j)
        self.clean()
    print("Final grid state:")
        for row in self.grid:
            print(row)

def get_dirty_coordinates(rows, cols, num_dirty_cells):
    dirty_cells = set()
    while len(dirty_cells) < num_dirty_cells:
        try:
        coords = input(f"Enter coordinates for dirty
                cell {len(dirty_cells)+1}
(format: row, col):")
        x, y = map(int, coords.split(','))
        if 0 <= x < rows and 0 <= y < cols:
            dirty_cells.add((x, y))
        else:
            print("co-ordinates are out of bounds. try again")
    except value error:
        print("Invalid input. pls enter co-ordinates in
        the format: row, col")
    return dirty_cells

rows = int(input("Enter the no. of rows"))
cols = int(input("  "  "  " of columns"))
num_dirty_cells = int(input("Enter the number
                of dirty cells:"))
```

DATE: PAGE:

```
intial_grid = [[0 for _ in range(cols)] for _ in
    range(rows)]
dirty_coordinates = get_dirty_coordinates
    (rows, cols, num_dirty_cells)

vaccum.run()
```

→ output :

```
Enter the no of rows : 2
Enter the no of cols : 2
Enter the no of dirty cells : 1
intial grid state :
    [0, 1]
    [0, 0]
position (0, 0) is already clean
  cleaning position (0, 1)
position (1, 0) is already clean
position (1, 1) is already clean
  Final grid state
    [0, 0]
    [0, 0]
```

7/10/24

Lab 02:-

Vacuum Cleaner

Code:-

```
class VacuumCleaner:
    def __init__(self, grid):
        self.grid = grid
        self.position = (0, 0)

    def clean(self):
        x, y = self.position
        if self.grid[x][y] == 1:
            print(f"Cleaning position {self.position}")
            self.grid[x][y] = 0
        else:
            print(f"Position {self.position} is already clean")

    def move(self, direction):
        x, y = self.position
        if direction == 'up' and x > 0:
            self.position = (x - 1, y)
        elif direction == 'down' and x < len(self.grid) - 1:
            self.position = (x + 1, y)
        elif direction == 'left' and y > 0:
            self.position = (x, y - 1)
        elif direction == 'right' and y < len(self.grid[0]) - 1:
            self.position = (x, y + 1)
        else:
            print("Move not possible")

    def run(self):
        rows = len(self.grid)
        cols = len(self.grid[0])


        for i in range(rows):
            for j in range(cols):
                self.position = (i, j)
                self.clean()

        print("Final grid state:")
        for row in self.grid:
            print(row)

def get_dirty_coordinates(rows, cols, num_dirty_cells):
    dirty_cells = set()
    while len(dirty_cells) < num_dirty_cells:
```

```
        try:
            coords = input(f"Enter coordinates for dirty cell {len(dirty_cells) + 1} (format: row,col): ")
            x, y = map(int, coords.split(','))
            if 0 <= x < rows and 0 <= y < cols:
                dirty_cells.add((x, y))
            else:
                print("Coordinates are out of bounds. Try again.")
        except ValueError:
            print("Invalid input. Please enter coordinates in the format: row,col")
    return dirty_cells

rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))
num_dirty_cells = int(input("Enter the number of dirty cells: "))

if num_dirty_cells > rows * cols:
    print("Number of dirty cells exceeds total cells in the grid. Adjusting to maximum.")
    num_dirty_cells = rows * cols

initial_grid = [[0 for _ in range(cols)] for _ in range(rows)]
dirty_coordinates = get_dirty_coordinates(rows, cols, num_dirty_cells)

for x, y in dirty_coordinates:
    initial_grid[x][y] = 1

vacuum = VacuumCleaner(initial_grid)
print("Initial grid state:")
for row in initial_grid:
    print(row)

vacuum.run()
```

Output:-

```
Python 3.7.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Int
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Users/bmsce/Desktop/1bm22cs195/vc2.py =============
Enter the number of rows: 2
Enter the number of columns: 2
Enter the number of dirty cells: 1
Enter coordinates for dirty cell 1 (format: row,col): 0,1
Initial grid state:
[0, 1]
[0, 0]
Position (0, 0) is already clean
Cleaning position (0, 1)
Position (1, 0) is already clean
Position (1, 1) is already clean
Final grid state:
[0, 0]
[0, 0]
>>>
```

LAB-2

Vacuum cleaner

Algorithm:-

1. Initialize
   - Create two 2D grids to represent room1 and room2
   - set initial position to top of room1 [0,0]

2. Perceive environment
   - At each step vacuum cleaner checks if cell is dirty, then it clean it, if it's clean then it moves to the next location

3. clean
   - If the current cell / grid is dirty clean it and the change the state to dirty to clean

4. Movement
   - Follows Left-Right pattern
   - if not end to the row move right
   - if end of the row stepdown one column and start cleaning.

5. check
   if Room1 is clean then move to Room2

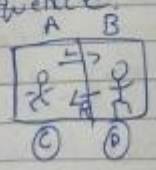
6. Repeat steps 2, 3, 4, 5 for Room2

7. End
   the vacuum cleaner terminates it algorithm when both Rooms are clean.

8. Results
   Display the final status of both the rooms

Percept Sequence

Room



Observation Book:-

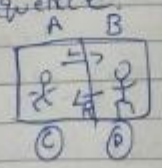
LAB-2

Vacuum cleaner

Algorithm:-

1. Initialize
   - Create two 2D grids to represent room1 and room2
   - Set initial position to top of room1 [0,0]
2. Perceive environment
   - At each step vacuum cleaner checks if cell is dirty, then it clean it, if it's clean then it moves to the next location
3. clean
   - If the current cell / grid is dirty clean it and the change the state to dirty to clean
4. Movement
   - Follows Left-Right pattern
   - if not end to the row move right
   - if end of the row stepdown one column and start cleaning.
5. check
   if Room1 is clean then move to Room2
6. Repeat steps 2, 3, 4, 5 for Room2
7. End
   the vacuum cleaner terminates it algorithm when both Rooms are clean.
8. Results
   Display the final status of both the rooms

Percept Sequence



①, (A, Left)

①, (B, Left), ② (B, clean)

①, (A, Left), ② (B, clean) ③ (A, right)

Code:-

```
class vac :
    def __init__(self, grid):
        self.grid = grid
        self.position = (0,0)
    def clean(self):
        x,y = self.position
        if self.grid[x][y] = 1 :
            print("cleaning position {self.position}")
            self.grid[x][y] = 0
        else :
            print("Position {self.position} is already
                                clean")
def move(self, direction):
    x,y = self.position
    if direction == 'up' and x>0:
        self.position = (x-1, y)
    elif direction == 'down' and x<
            len(self.grid) -1:
        self.position = (x+1, y)
    elif direction == "left" and y>0:
        self.position = (x, y-1)
    elif direction = 'right', and y <
            len(self.grid[0]) -1:
        self.position = (x, y+1)
    else:
        print("Move not possible")
```

```
def run(self):
    rows = len(self.grid)
    cols = len(self.grid[0])
    for i in range(rows):
        for j in range(cols):
            self.position[i,j]
            self.clean()
    print("Final grid state:")
    for row in self.grid:
        print(row)
def get_dirty_coord(rows, cols, num_dirty):
    dirty_cells = set()
    while len(dirty_cells) < num_dirty_cells:
        try:
            coords = input("Enter coordinates for dirty cell {len(dirty_cells)+1} (format: row,col) : ")
            x,y = map(int, coords.split(','))
            if 0<= x < rows and 0<= y < cols:
                dirty_cells.add((x,y))
            else:
                print("Coordinates are out of bounds")
        except ValueError:
            print("Invalid input")
    return dirty_cells
rows = int(input("Enter the no of rows"))
cols = int(input("Enter the no of cols"))
num_dirty_cell = int(input("Enter the no of dirty cells"))
if num_dirty_cells > rows * cols:
    print("No of dirty cells exceeds total cell")
num_dirty_cells = rows * cols
```

```
initial_grid = [[0]*col for _ in
        range(row)]
dirty_coordinates = get_dirty_coordinates
        (row, col, num_dirty_cells)
for x,y in dirty_coord:
        initial_grid [x][y] = 1
vac_num = VacuumCleaner (initial_grid)
print ("Initial grid state")
for row in initial_grid:
        print(row)
vacuum.run()
```

Output:-

```
Enter the no of row : 2
Enter the no of col : 2
Enter the no of dirty cells : 1
Enter coordinates for dirty cell : 0,1
Initial grid state :
[0, 1]
[0, 0]
Position (0, 0) is already clean
cleaning position (0, 1)
Position (1,0) is already clean
Position (1,1) is already clean
Final grid state :
[0 0]
[0 0]
```

1/10/24