Name Lakhan Kumawat
ROLL 1906055
Assignment – CSL5402

(a) WAP to implement a performance-measuring environment simulator for the following vacuum-cleaner world diagram. Your implementation should be modular so that the sensors, actuators, and environment characteristics (size, shape, dirt placement etc.) can be changed easily.

(b) In the addition of the above program (i.e., in 1(a)), WAP to implement a simple reflex agent for the vacuum environment. Run the environment with this agent for all possible initial dirt configurations and agent locations.

```
import numpy as np
from time import sleep
import random
class VacuumCleaner:
    def __init__(self):
        self.position = [2,2]
        self.environ = np.random.choice([True,False],size=(8,8))
        self.performance = 0

    # SENSOR
    def scan_dirty_regions(self):
        curr_pos = self.position
        environment = self.environ

        dirty = [
                [False,False,False,False,False,False,False],
                [False,False,False,False,False,False,False],
                [False,False,False,False,False,False,False],
                [False,False,False,False,False,False,False],
                [False,False,False,False,False,False,False],
                [False,False,False,False,False,False,False],
                [False,False,False,False,False,False,False]
```

```
                ]

        for i in range(-3, 3):
            for j in range(-3, 3):
                x = curr_pos[0] + i
                y = curr_pos[1] + j

                if x >= 0 and x < 8 and y >= 0 and y < 8:
                    dirty[i+3][j+3] = environment[x][y]

        return dirty

    # ACTUATOR
    def move(self,dir_vec):
        new_x = self.position[0] + dir_vec[0]
        new_y = self.position[1] + dir_vec[1]

        if new_x < 0 or new_x > 7 or new_y > 7 or new_y < 0:
            print("At edge")
            return

        self.position = [ new_x, new_y ]

        if dir_vec == [0,0]:
            print("Staying")
        else:
            print("Moving to", self.position)

    def clean(self):
        # SENSOR
        locations = self.scan_dirty_regions()
        self.performance += len(locations) / 25

        # AGENT - Chose dirty locations
        dirty = []
        for i in range(len(locations)):
            for j in range(7):
```

```
                if self.environ[i][j] == True:
                    dirty.append([i,j])

        if len(dirty) == 0:
            print("No dirt found near cleaner")
            return [0,0]

        loc = random.choice(dirty)

        move = [ loc[0] - 2, loc[1] - 2 ]

        self.move(move)

        new_pos = self.position
        if self.environ[ new_pos[0] ][ new_pos[1] ]:
            print("Cleaning: ", new_pos)
            self.environ[ new_pos[0] ][ new_pos[1] ] = False
            self.performance += len(locations)*0.4;  # TODO: per
f += dirt_found/total_boxes

            for i in range(8):
                for j in range(8):
                    if self.environ[i][j]:
                        print("#", end=" ")
                    else:
                        print(" ", end=" ")
                print()
        else:
            print("Already Clean")

vacuum_cleaner = VacuumCleaner()
while True:
    vacuum_cleaner.clean()
    sleep(1)
```

OUTPUT :

```
Moving to [0, 1]
Cleaning:  [0, 1]
      #
  # #   # #
  #   #   # #
# #       # #
#     # # #
# # # # # # #
    #         #
# # #         #
Moving to [1, 0]
Already Clean
Moving to [1, 4]
Cleaning:  [1, 4]
      #
  # #     #
  #   #   # #
# #       # #
#     # # #
# # # # # # #
    #         #
# # #         #
Moving to [3, 7]
Already Clean
At edge
Already Clean
At edge
Already Clean
At edge
Already Clean
Moving to [7, 7]
Cleaning:  [7, 7]
      #
  # #     #
```

```
  #   #   # #
# #       # #
#     # # #
# # # # # # #
    #         #
# # #
Moving to [7, 6]
Already Clean
```

2. WAP to implement a model-based reflex agent for the automatic taxi driver environment. Run the environment with this agent for all possible assumptions made by you which makes it model-based.

```
from time import sleep
import numpy as np
import random
import math

environ = np.random.choice([True,False],p=[0.1,0.9],siz
e=(15,15))
```

We create the environment

```
class ModelBasedTaxi:
    def __init__(self, environ, start_loc):
        self.environ = environ
        self.start_loc = start_loc
        self.position = self.start_loc
        self.model = ModelBasedReflexAgent(self)

    def move_next(self):
        # MODEL+RULES
        action = self.model.act( self.environ )

        # ACTUATOR
```

```
        self._move(direction=action)

    def _move(self, direction):
        if direction[0] == 0 and direction[1] == 0:
            print("Model Taxi Not Move")
            return

        print("Model Taxi moving", end=" ")
        if direction[0] == -1:
            print("Left", end=" ")
        elif direction[0] == 1:
            print("Right", end=" ")
        if direction[1] == -1:
            print("Up", end=" ")
        elif direction[1] == 1:
            print("Down", end=" ")
        print()

        self.environ[ self.position[0] ][ self.position[1] ] = False

        new_x = self.position[0] + direction[0]
        new_y = self.position[1] + direction[1]

        self.environ[ new_x ][ new_y ] = True
        self.position = [ new_x, new_y ]
```

Model-based reflex agent

```
class ModelBasedReflexAgent:
    def __init__(self,parent_taxi):
        self.taxi = parent_taxi
        # STATE
        self.state = {
            "obstacles": []
        }
```

```
    def _state_update(self):
        curr_pos = self.taxi.position
        environ = self.taxi.environ
        obstacles = []

        dirs = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-1],[1,0],[1,1]]
        for dir in dirs:
            x = curr_pos[0] + dir[0]
            y = curr_pos[1] + dir[1]
            if environ[x][y]:
                obstacles.append([x,y])

        self.state["obstacles"] = obstacles

    def act(self,percept):
        # UPDATE STATE
        self._state_update()

        pos = self.taxi.position

        movable_directions = []

        dirs = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-1],[1,0],[1,1]]

        for dir in dirs:
            x = pos[0] + dir[0]
            y = pos[1] + dir[1]

            # TODO
            if x < 0 or x >= len(self.taxi.environ) or y < 0 or y >= l
en(self.taxi.environ):
                continue

            if [x,y] not in self.state["obstacles"]:
                movable_directions.append(dir)

        return random.choice(movable_directions)
```

```
model_taxi = ModelBasedTaxi(
        environ=environ,
        start_loc=[5,4],
    )
```

Run whatever cars/taxis were added

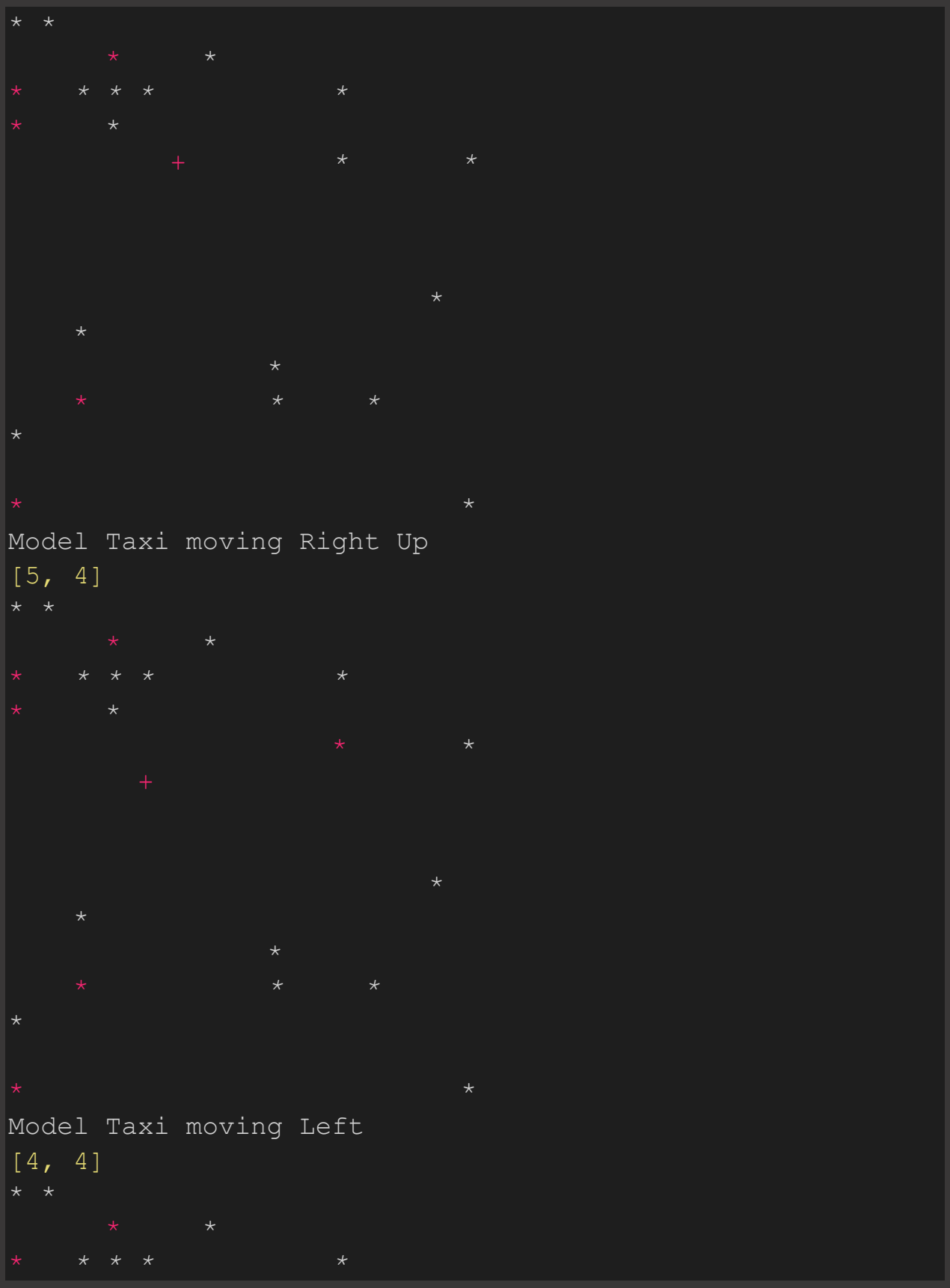
```
while True:
    print(model_taxi.position)
    for i in range(15):
        for j in range(15):
            if environ[i][j]:
                if [i,j] == model_taxi.position:
                    print("+", end = " ")
                else:
                    print("*", end = " ")
            else:
                print(" ", end = " ")
        print()

    model_taxi.move_next()
    sleep(1)
```

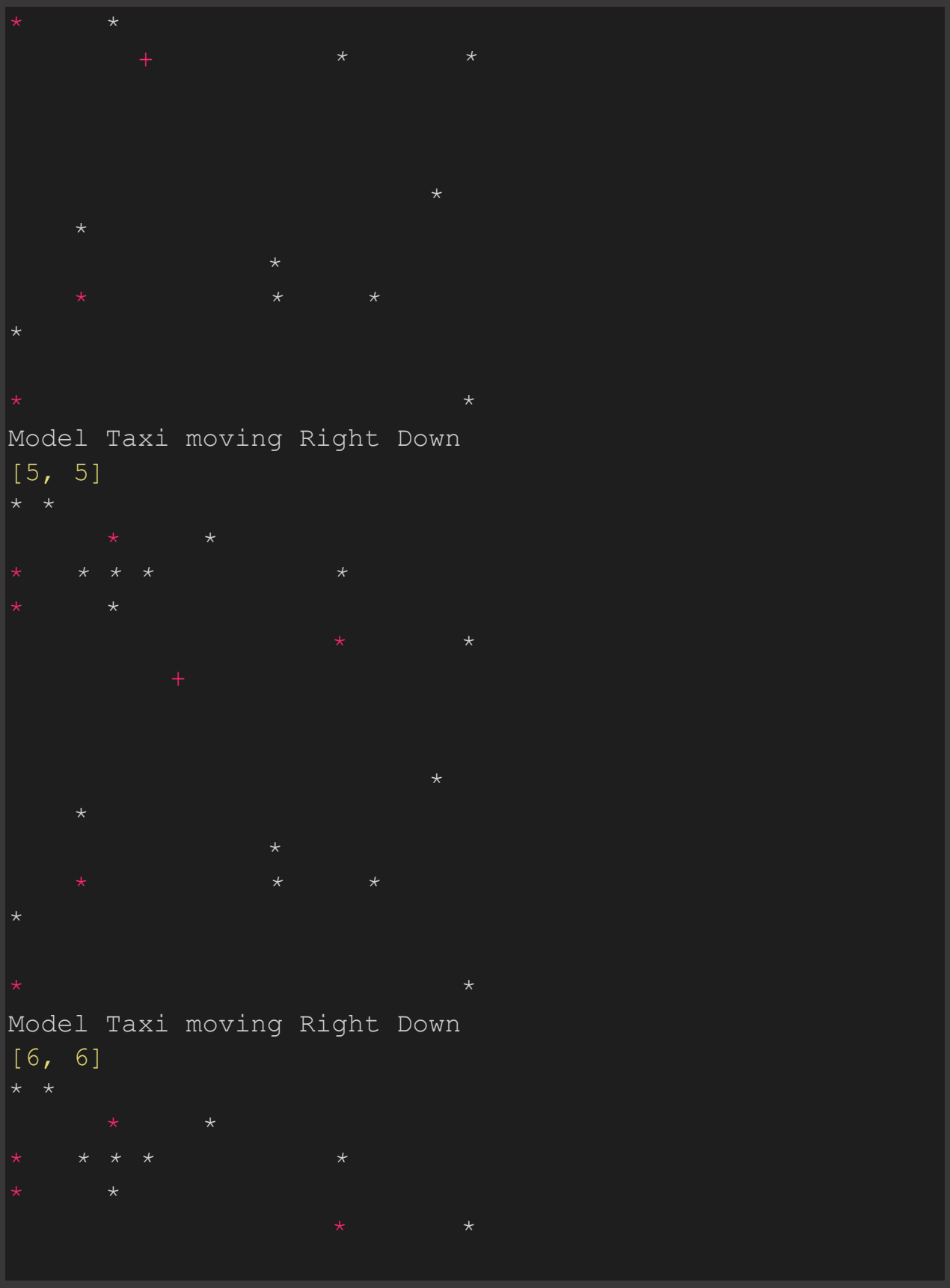
OUTPUT :

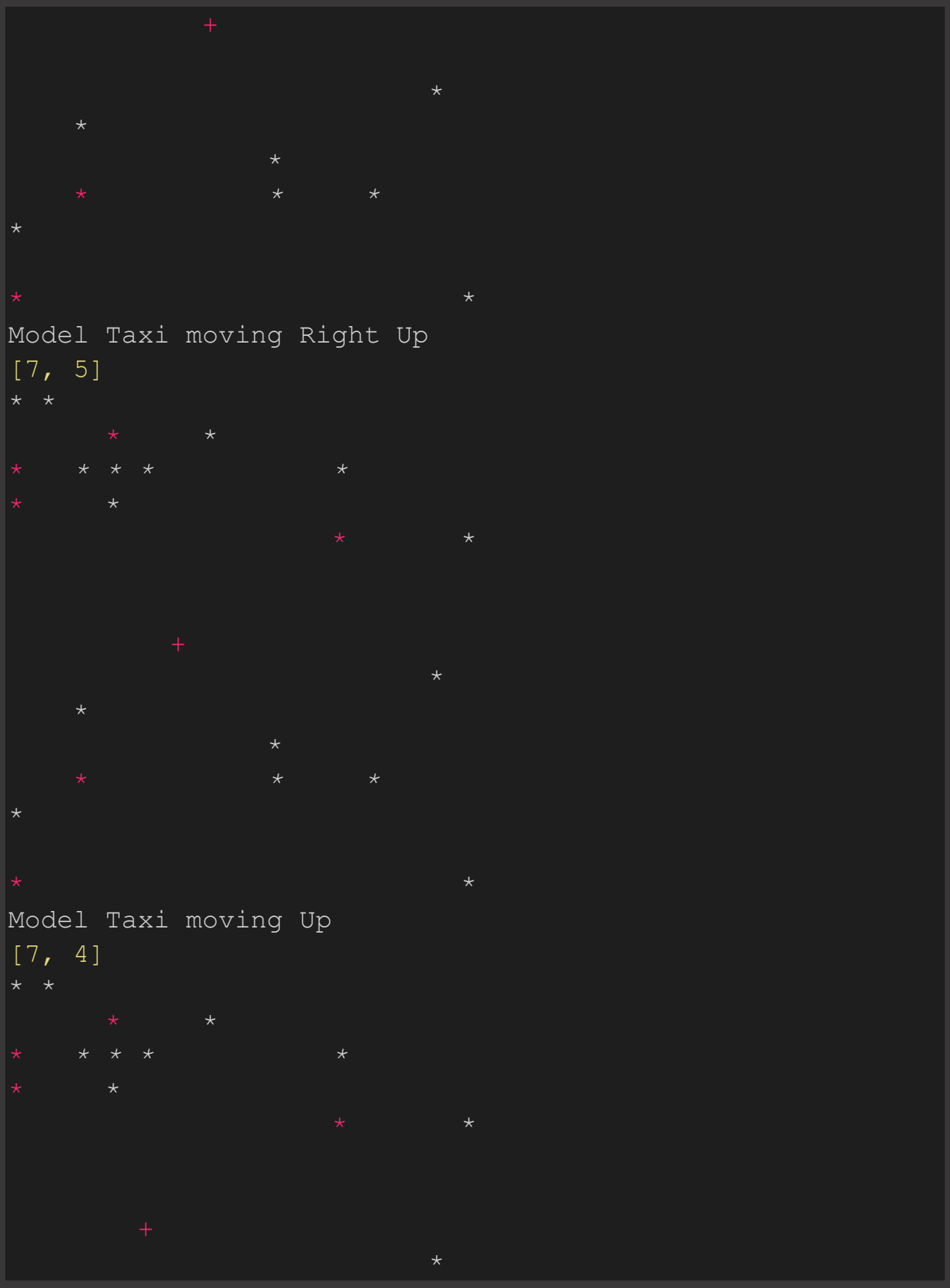
```
* *
        *       *
*     *  *  *              *
*       *
                          *         *
```

```
     *              *      *
*


*                             *
Model Taxi moving Up
[5, 3]
* *
       *      *
*    * * *             *
*      *
                      *         *
       +



                               *
     *
                   *
     *             *      *
*


*                             *
Model Taxi moving Left Down
[4, 4]
* *
       *      *
*    * * *             *
*      *
         +             *         *



                               *
     *
                   *
     *             *      *
*
```

```
*                                *
Model Taxi moving Down
[4, 5]
* *
       *      *
*    * * *             *
*      *
            +           *        *
                                 *
     *
                   *
     *             *      *
*

*                                *
Model Taxi moving Left Down
[3, 6]
* *
       *      *
*    * * *             *
*      *      +
                       *        *
                                 *
     *
                   *
     *             *      *
*

*                                *
Model Taxi moving Right Up
[4, 5]
```


Model Taxi moving Right Up
[5, 4]
Model Taxi moving Left
[4, 4]


Model Taxi moving Right Down
[5, 5]
Model Taxi moving Right Down
[6, 6]


Model Taxi moving Right Up
[7, 5]
Model Taxi moving Up
[7, 4]

```
     *
                    *
     *              *       *
*

*                                 *
Model Taxi moving Left Up
[6, 3]
* *
        *       *
*     * * *               *
*       *
                          *         *

        +

                                  *
     *
                    *
     *              *       *
*

*                                 *
Model Taxi moving Down
[6, 4]
* *
        *       *
*     * * *               *
*       *
                          *         *

          +

                                  *
     *
                    *
     *              *       *
```

```
*

*                                 *
Model Taxi moving Up
[6, 3]
* *
       *       *
*    * * *              *
*      *
                        *         *

       +

                                  *
     *
                    *
     *              *       *
*

*                                 *
Model Taxi moving Up
[6, 2]
* *
       *       *
*    * * *              *
*      *
                        *         *

     +

                                  *
     *
                    *
     *              *       *
*

*                                 *
```

```
Model Taxi moving Right Up
[7, 1]
* *
       *      *
*    * * *             *
*      *
                       *         *



  +
                                 *
     *
                    *
     *              *      *
*

*                                *
Model Taxi moving Up
[7, 0]
* *
       *      *
*    * * *             *
*      *
                       *         *



+
                                 *
     *
                    *
     *              *      *
*

*                                *
Model Taxi moving Down
[7, 1]
* *
```

```
       *       *
*    * * *              *
*       *
                        *          *
  +
                                  *
     *
                    *
     *               *       *
*
*                                  *
Model Taxi moving Left Down
```

3. WAP to implement a goal-based reflex agent for the automatic taxi driver in a road map environment. Run the environment with this agent for all possible assumptions made by you which makes it goal-based agent.

```
from time import sleep
import numpy as np
import random
import math
environ = np.random.choice([True,False],p=[0.1,0.9],size=
(15,15))
```

We create the environment

```
class GoalBasedTaxi:
    def __init__(self, environ, start_loc, end_loc):
        self.environ = environ

        self.start_loc = start_loc
```

```
        self.end_loc = end_loc

        self.position = self.start_loc

        self.agent = GoalBasedReflexAgent(self)

    def move_next(self):
        # MODEL+RULES
        action = self.agent.act( self.environ )

        # ACTUATOR
        self._move(direction=action)

    def _move(self, direction):
        if direction[0] == 0 and direction[1] == 0:
            print("Goal Taxi Not Moving")
            return

        print("Goal Taxi moving", end=" ")
        if direction[0] == -1:
            print("Left", end=" ")
        elif direction[0] == 1:
            print("Right", end=" ")
        if direction[1] == -1:
            print("Up", end=" ")
        elif direction[1] == 1:
            print("Down", end=" ")
        print()

        self.environ[ self.position[0] ][ self.position[1
] ] = False

        new_x = self.position[0] + direction[0]
        new_y = self.position[1] + direction[1]

        self.environ[ new_x ][ new_y ] = True
        self.position = [ new_x, new_y ]
```

Goal-based reflex agent

```
class GoalBasedReflexAgent:
    def __init__(self,parent_taxi):
        self.taxi = parent_taxi
        self.state = {
            "goal": self.taxi.end_loc,
            "obstacles": []
        }

    def _state_update(self):
        curr_pos = self.taxi.position
        environ = self.taxi.environ
        obstacles = []

        dirs = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-
1],[1,0],[1,1]]
        for dir in dirs:
            x = curr_pos[0] + dir[0]
            y = curr_pos[1] + dir[1]
            if environ[x][y]:
                obstacles.append([x,y])

        self.state["obstacles"] = obstacles

    def act(self,percept):
        # UPDATE STATE
        self._state_update()

        movable_directions = []

        dirs = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-
1],[1,0],[1,1]]
        pos = self.taxi.position

        if pos == self.state["goal"]:
            print("Goal based Reached Destination...")
```

```
            return [0,0]

        for dir in dirs:
            x = pos[0] + dir[0]
            y = pos[1] + dir[1]
            if [x,y] not in self.state["obstacles"]:
                movable_directions.append(dir)

        better_direction = [0,0]
        goal = self.state["goal"]
        for dir in movable_directions:
            x = pos[0] + dir[0]
            y = pos[1] + dir[1]
            nearer_x = pos[0] + better_direction[0]
            nearer_y = pos[1] + better_direction[1]

            nearest_dist = math.hypot( goal[1] - nearer_y
, goal[0] - nearer_x )
            new_dist = math.hypot( goal[1] - y, goal[0] -
 x )

            if new_dist <= nearest_dist:
                better_direction = dir

        return better_direction
goal_taxi = GoalBasedTaxi(
        environ=environ,
        start_loc=[2,9],
        end_loc=[12,1]
    )
```

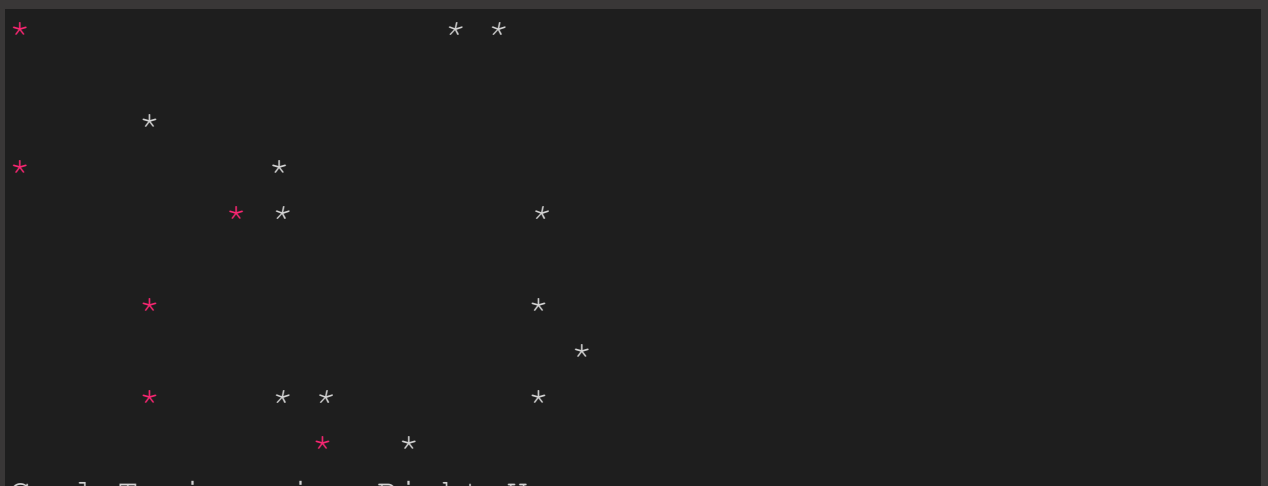
Run whatever cars/taxis were added

```
while True:
    print(goal_taxi.position)
    for i in range(15):
        for j in range(15):
            if environ[i][j]:
```

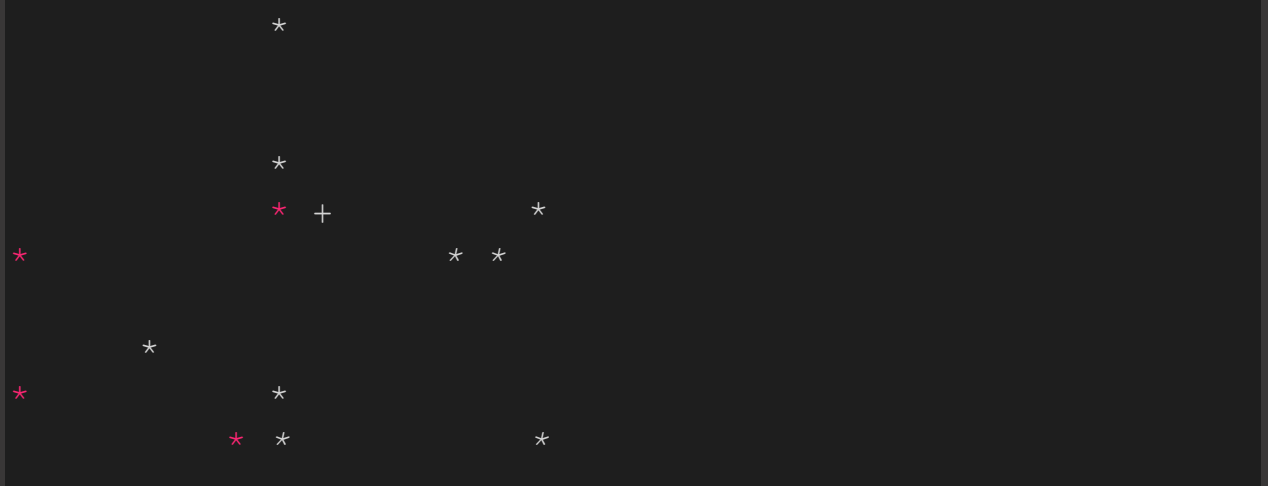
```
                if [i,j] == goal_taxi.position:
                    print("+", end = " ")
                else:
                    print("*", end = " ")
            else:
                print(" ", end = " ")
        print()

    goal_taxi.move_next()
    sleep(1)
```

OUTPUT :

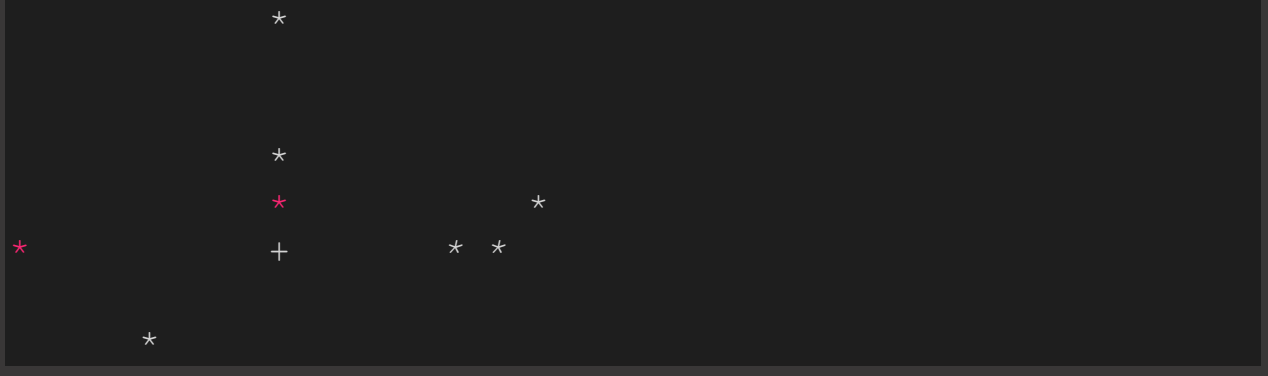
```
[2, 9]
             *


             *
             *            *
*                    * *

       *
*            *
           * *            *

       *                  *
                            *
       *      * *           *
                *    *
Goal Taxi moving Right Up
[3, 8]
             *


             *    +
             *            *
```



Goal Taxi moving Right Up
[4, 7]



Goal Taxi moving Right Up
[5, 6]



```
*             *
            *  *             *


       *                    *
                              *
       *       * *           *
                 *    *
Goal Taxi moving Right Up
[6, 5]
              *



              *
              *             *
*                       * *
            +
       *
*             *
            *  *             *


       *                    *
                              *
       *       * *           *
                 *    *
Goal Taxi moving Right Up
[7, 4]
              *



              *
              *             *
*                       * *


       * +
*             *
            *  *             *
```

```
       *                    *
                             *
       *       * *          *
                 *    *
Goal Taxi moving Right Up
[8, 3]
               *


               *
               *             *
*                      * *

       *
*      +       *
            *  *             *

       *                    *
                             *
       *       * *          *
                 *    *
Goal Taxi moving Right Up
[9, 2]
               *


               *
               *             *
*                      * *

       *
*              *
     +       * *             *

       *                    *
                             *
       *       * *          *
```

```
                *    *
Goal Taxi moving Right Up
[10, 1]
              *


              *
              *             *
*                     * *

       *
*              *
            *  *             *
  +
       *                     *
                               *
       *       * *           *
                *    *
Goal Taxi moving Right
[11, 1]
              *


              *
              *             *
*                     * *

       *
*              *
            *  *             *

  +    *                     *
                               *
       *       * *           *
                *    *
Goal Taxi moving Right
[12, 1]
```

```
Goal based Reached Destination...
Goal Taxi Not Moving
[12, 1]
```

```
Goal based Reached Destination..
```

End Of Assignment