18CSC204J- Design and Analysis of Algorithms
A Project Report on
Optimal Path for Car Parking

Submitted in partial fulfillment of the requirement for the IV semester

of

BACHELOR OF TECHNOLOGY

IN

DESIGN AND ANALYSIS OF ALGORITHMS

Submitted By:

Subhasish Kumar (RA2011029010055)
Debdatta Singha (RA2011029010056)

Under the supervision of

Dr. Balasaraswathi R

(Asst. Professor)

DEPARTMENT OF COMPUTING AND TECHNOLOGY,

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SESSION – 2022

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

18CSC204J- Design and Analysis of Algorithms

Mini Project
Record Work

Registration Numbers- RA2011029010055, RA2011029010056

Names: Subhasish Kumar, Debdatta Singha

Semester: IV

Department: Computer Networking

SRMI INSTITUTE OF SCIENCE AND TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR – 603203

BONAFIDE CERTIFICATE

Register No. RA2011029010055

Certified to be bonafide record of the work done by Subhasish Kumar and Debdatta Singha of Computer Science and Engineering with specialization in Computer Networking, B.Tech degree course in the practical 18CSC204J – Design and Analysis of Algorithm in SRM Institute of Science and Technology, Kattankulathur during the academic year 2021-22.

Date: **Lab Incharge**

<u>DECLARATION</u>

I, hereby declare that the work presented in this dissertation entitled "**<u>Optimal Path for Car Parking</u>**" has been done by me and my team, and this dissertation embodies my own work.

Approved By:
V.R. Balasaraswathi

CERTIFICATE

This is to certify that the project entitled “Optimal Path for Car Parking” was carried out by “Debdatta Singha” under my supervision at the Department of Computing and Technology, SRM Institute of Technology, Kattankulathur, Chennai.

The work is original, as it has not been submitted earlier either in part or full for any purpose before.

V.R Balasaraswathi

(Asst. Professor)

ACKNOWLEDGEMENT

I would like to thank V.R. Balasaraswathi (Asst. Professor) who has been a great inspiration and who has provided sufficient background knowledge and understanding of this subject.

Our humble prostration goes to her, for providing all the necessary resources and environment, which have aided me to complete this project successfully.

TABLE OF CONTENTS

CONTRIBUTION TABLE

NAME	REG. NO.	CONTRIBUTION
Subhasish Kumar	RA2011029010055	Brought in the idea/concept and worked on developing the algorithm for the same and complexity analysis. Contributed towards the making of this project report.
Debdatta Singha	RA2011029010056	Contributed towards the successful implementation of the idea and worked on developing the algorithm. Contributed towards the making of this project report.

AIM: To design a Optimal Car Parking Suing Backtracking using Backtracking

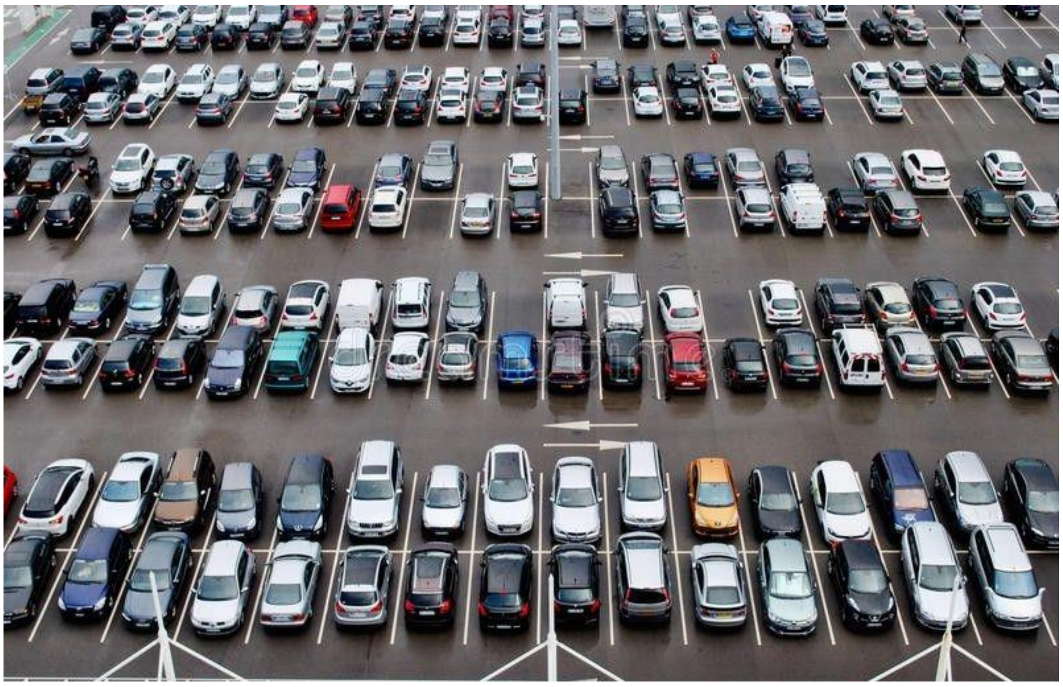
Subhasish Kumar & Debdatta Singha
Department of Engineering
SRM Institute of Science and Technology
Chennai-603203

Problem definition:

With rapid urbanization and increased population density in cities, there is a heightened need for mobility solutions. Private vehicles are a preferred mode of transportation for many people in developed economies. As the standard of living continues to go up in several parts of the world, more and more people and companies buy new cars. Malls and other commercial centers are also increasing in cities, attracting more and more people who prefer to drive in with their cars. **Parking a car in the allotted place in the parking area is always a cumbersome task in enormous malls and hospitals etc**. This has increased the need for an efficient parking lot management system in malls to ensure people have a hassle-free experience. An efficient algorithm is required to make sure that a person is able to park his/her car in the allotted parking space.

2. Problem Explanation with diagram and example

When you are parking a car, in what way do you have to park so that it would be easy and safe? A lot of time is wasted without an efficient way of parking a car in big parking areas. So, there is a need for an efficient parking lot management system in malls to ensure people have a hassle-free experience. Shortage of parking space, high parking tariffs, and traffic congestion due to visitors in search of a parking place are only a few examples of everyday parking problems. The paper examines the car parking problem in the city; its different causes and the conventions. By using modern solutions we should adopt a new methodology for parking so that the Safety of the car becomes high and people can have a stress-free experience while parking.



<u>Design Technique Used:</u>

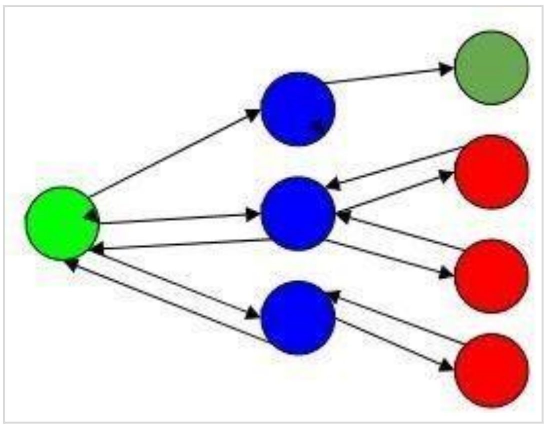
The backtracking technique is used to solve this problem. It uses recursive calling to find the solution by building a solution step by step increasing values with time. It removes the solutions that don't give rise to the problem's solution based on the constraints given to solve the problem.

Backtracking algorithm is applied to some specific types of problems,

- Decision problem is used to find a feasible solution of the problem.
- Optimisation problem used to find the best solution that can be applied.
- Enumeration problem used to find the set of all feasible solutions of the problem.

In a backtracking problem, the algorithm tries to find a sequence path to the solution which has some small checkpoints from where the problem can backtrack if no feasible solution is found for the problem.

Example,



Here,

Green is the start point, blue is the intermediate point, red are points with no feasible solution, dark green is the end solution.

Here, when the algorithm propagates to an end to check if it is a solution or not, if it is then returns the solution otherwise backtracks to the point one step behind it to find the next point to find solution.

Algorithm:

Step 1 − if current_position is goal, return success

Step 2 − else,

Step 3 − if current_position is an end point, return failed.

Step 4 − else, if current_position is not the end point, explore and repeat above steps.

Algorithm:

Create a solution matrix, initially filled with 0’s.

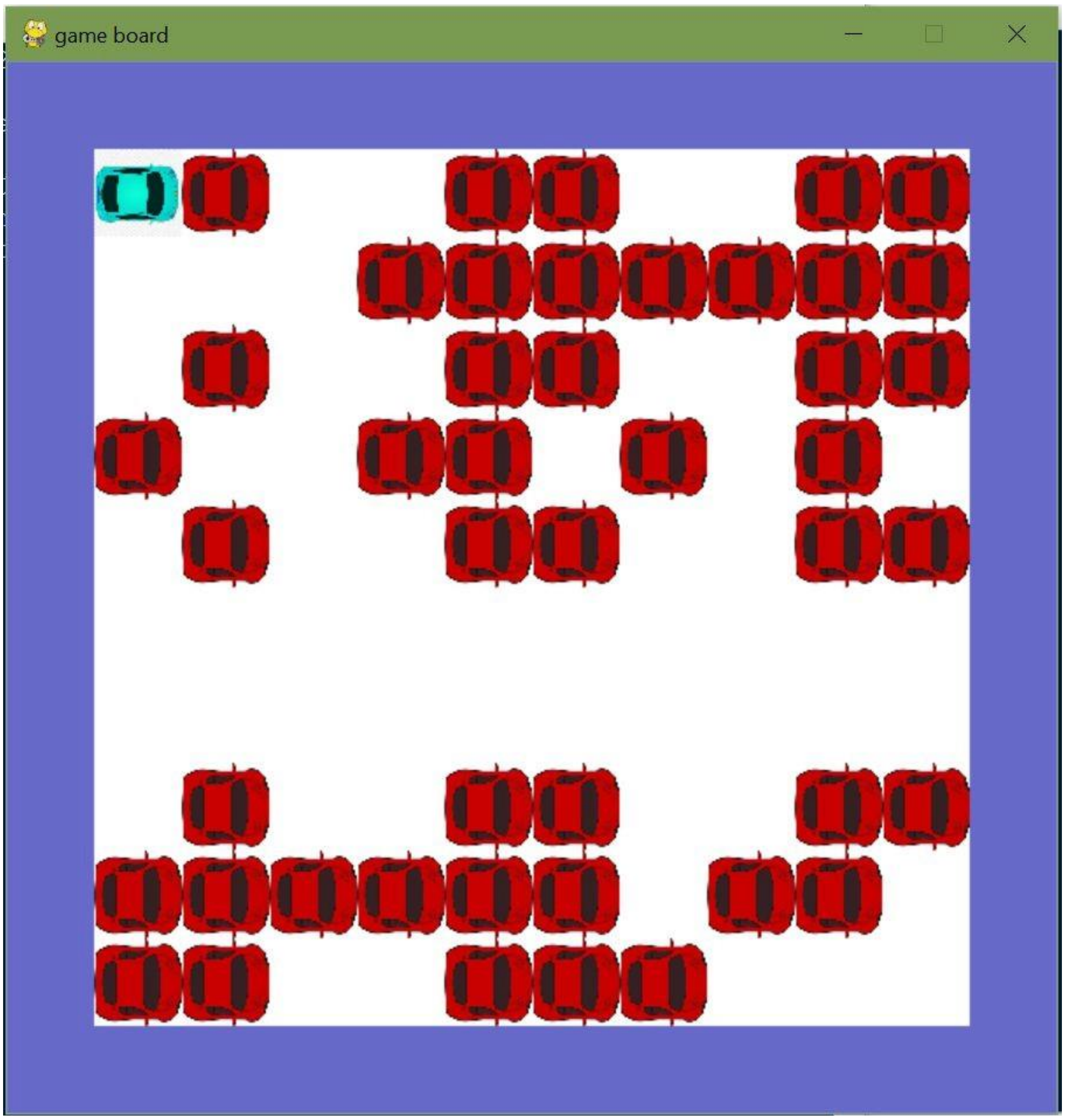
Create a recursive function, which takes initial matrix, output matrix and position of car(i, j). If the position is out of the matrix or the position is not valid then return.Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.

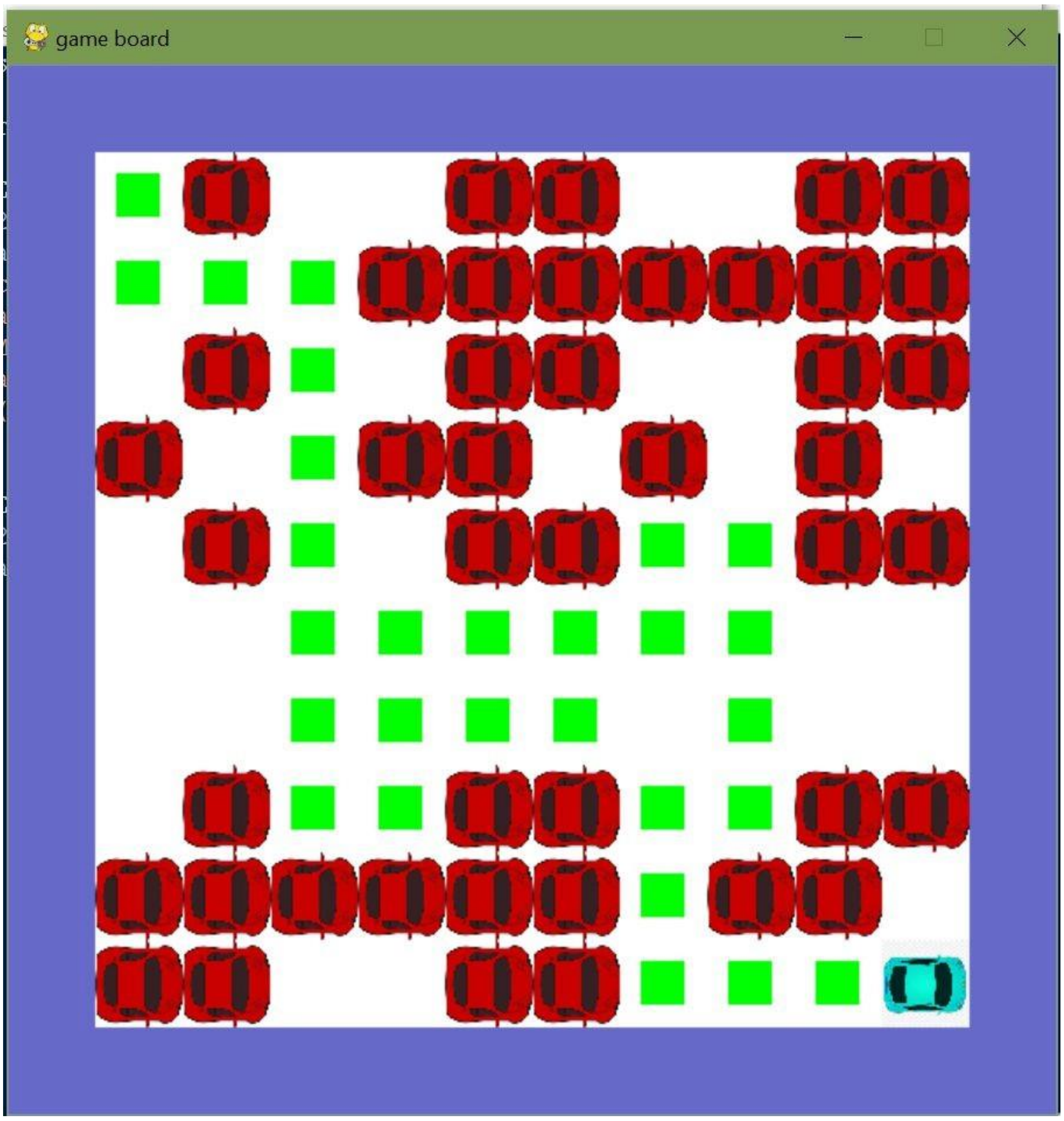
Recursively call for position (i-1,j), (I,j-1), (i+1, j) and (i, j+1).Unmark position (i, j), i.e output[i][j] = 0

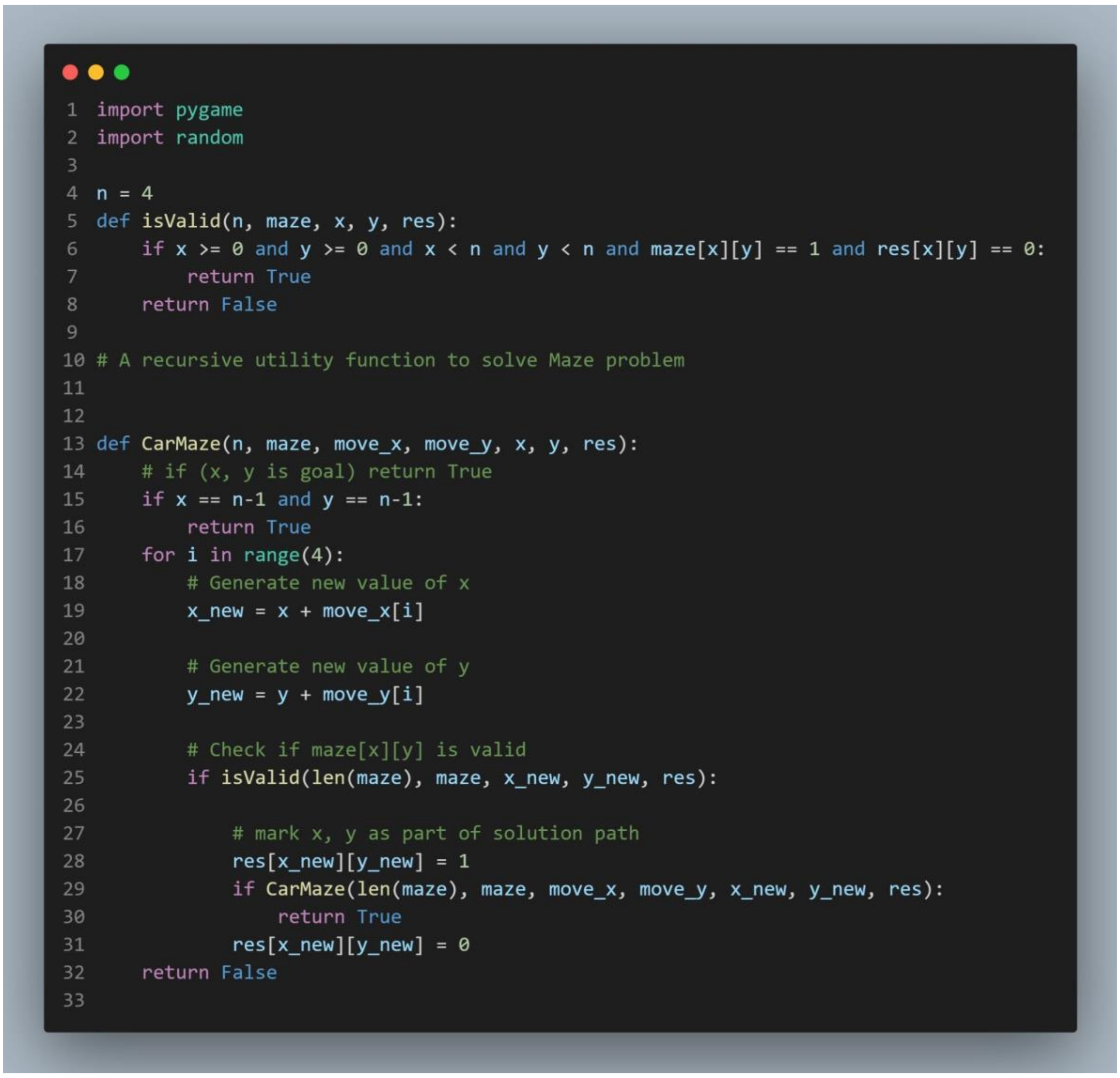
Complexity Analysis:

Time Complexity: O(2^(n^2)).
The recursion can run upper-bound 2^(n^2) times.
Space Complexity: O(n^2).
Output matrix is required so an extra space of size n*n is needed.

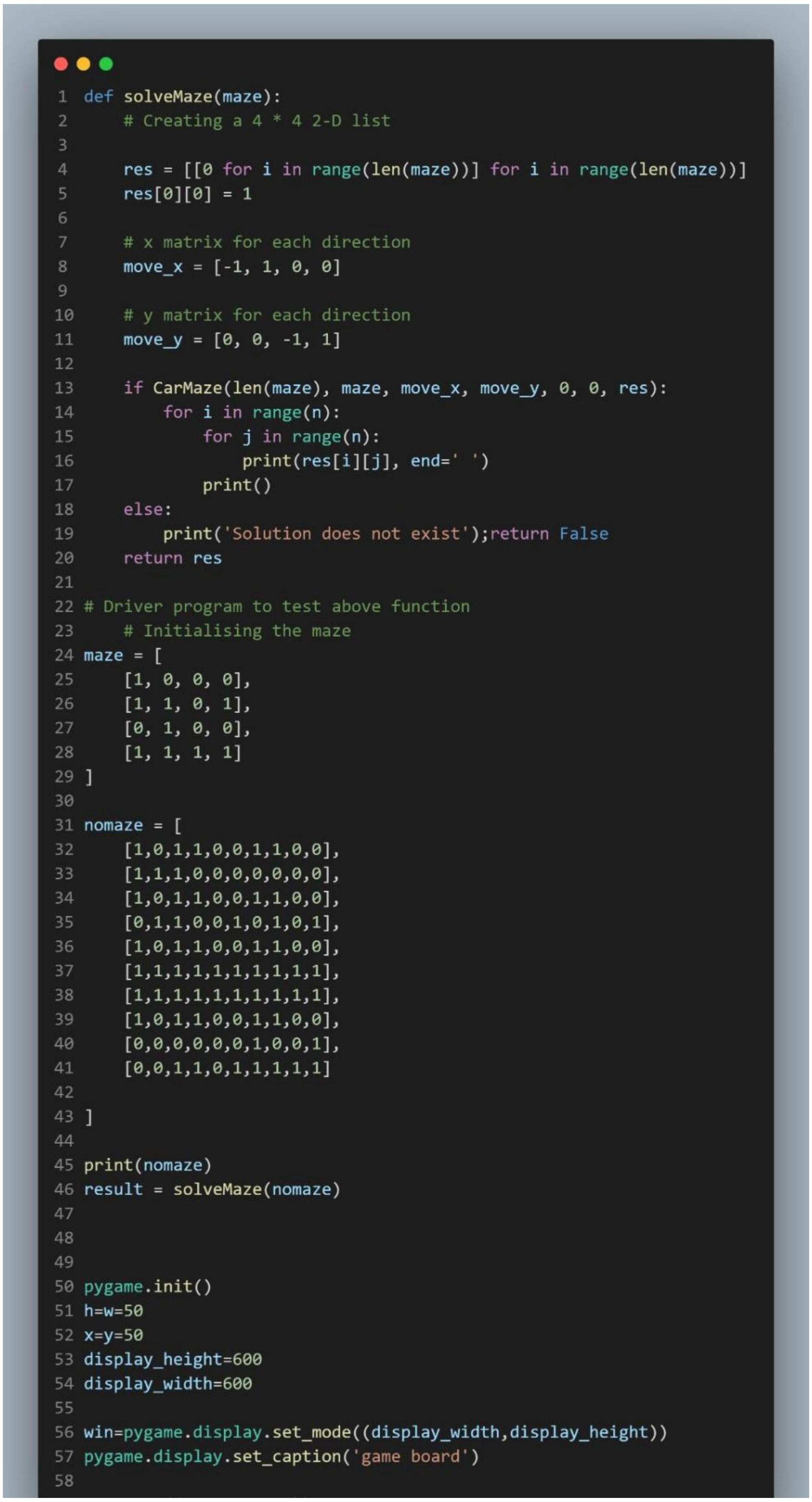
considering a grid of size MxN, and car has to move from 0,0 to M-1,N-1 (some cells may be blocked and car can move either down or right)
Now at every point car has two choices, go down or go left and to reach destination, car has to make (M+N) moves, so total recursion complexity turns out as O(2^(M+N)) which is very high, but there are many overlapping recursive calls, so using DP reduces the complexity to O(MN).


game board

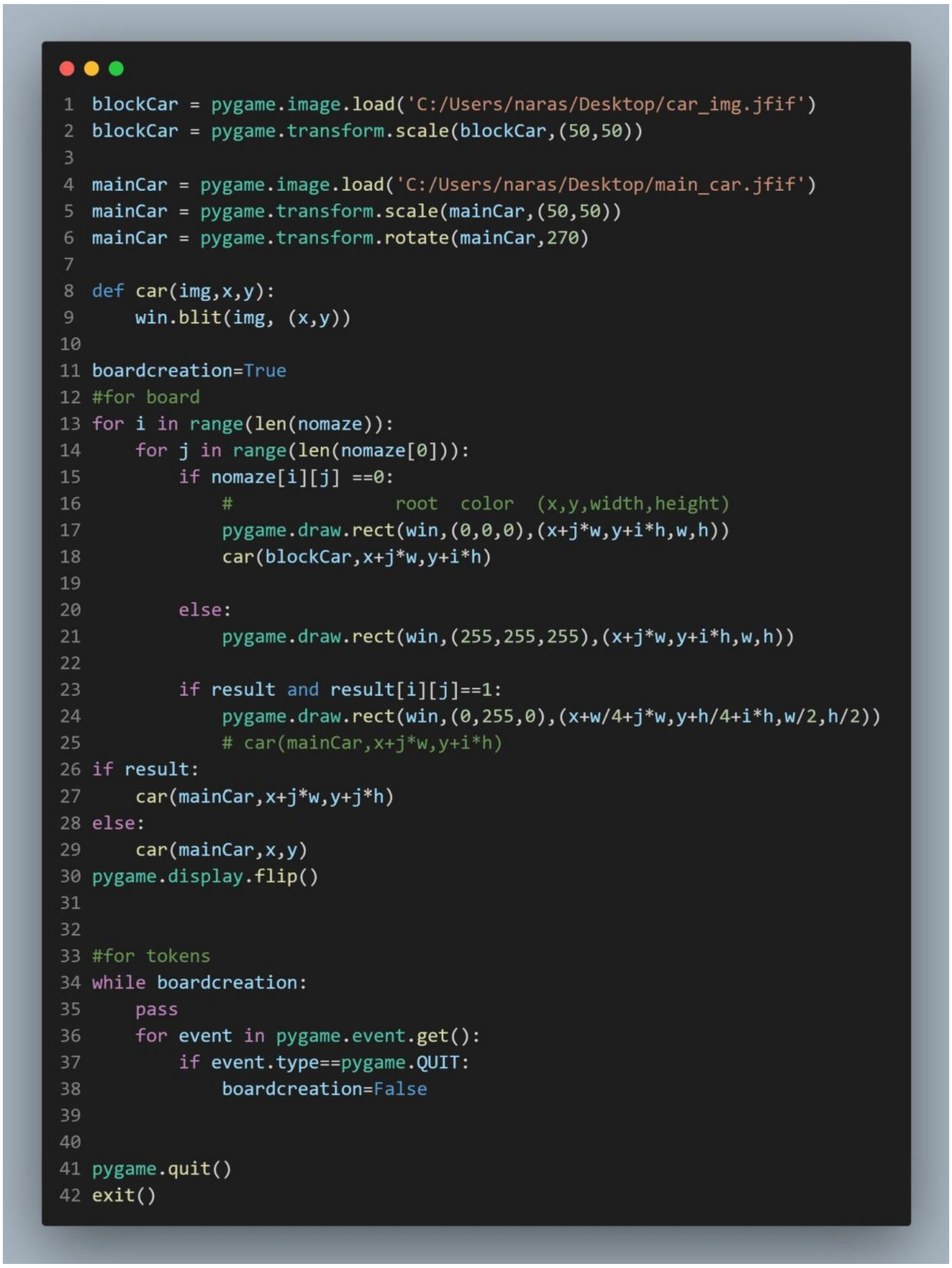

game board



```python
import pygame
import random

n = 4
def isValid(n, maze, x, y, res):
    if x >= 0 and y >= 0 and x < n and y < n and maze[x][y] == 1 and res[x][y] == 0:
        return True
    return False

# A recursive utility function to solve Maze problem


def CarMaze(n, maze, move_x, move_y, x, y, res):
    # if (x, y is goal) return True
    if x == n-1 and y == n-1:
        return True
    for i in range(4):
        # Generate new value of x
        x_new = x + move_x[i]

        # Generate new value of y
        y_new = y + move_y[i]

        # Check if maze[x][y] is valid
        if isValid(len(maze), maze, x_new, y_new, res):

            # mark x, y as part of solution path
            res[x_new][y_new] = 1
            if CarMaze(len(maze), maze, move_x, move_y, x_new, y_new, res):
                return True
            res[x_new][y_new] = 0
    return False

```



```python
def solveMaze(maze):
    # Creating a 4 * 4 2-D list

    res = [[0 for i in range(len(maze))] for i in range(len(maze))]
    res[0][0] = 1

    # x matrix for each direction
    move_x = [-1, 1, 0, 0]

    # y matrix for each direction
    move_y = [0, 0, -1, 1]

    if CarMaze(len(maze), maze, move_x, move_y, 0, 0, res):
        for i in range(n):
            for j in range(n):
                print(res[i][j], end=' ')
            print()
    else:
        print('Solution does not exist');return False
    return res

# Driver program to test above function
    # Initialising the maze
maze = [
    [1, 0, 0, 0],
    [1, 1, 0, 1],
    [0, 1, 0, 0],
    [1, 1, 1, 1]
]

nomaze = [
    [1,0,1,1,0,0,1,1,0,0],
    [1,1,1,0,0,0,0,0,0,0],
    [1,0,1,1,0,0,1,1,0,0],
    [0,1,1,0,0,1,0,1,0,1],
    [1,0,1,1,0,0,1,1,0,0],
    [1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,1,1,0,0,1,1,0,0],
    [0,0,0,0,0,0,1,0,0,1],
    [0,0,1,1,0,1,1,1,1,1]

]

print(nomaze)
result = solveMaze(nomaze)



pygame.init()
h=w=50
x=y=50
display_height=600
display_width=600

win=pygame.display.set_mode((display_width,display_height))
pygame.display.set_caption('game board')

```



```python
blockCar = pygame.image.load('C:/Users/naras/Desktop/car_img.jfif')
blockCar = pygame.transform.scale(blockCar,(50,50))

mainCar = pygame.image.load('C:/Users/naras/Desktop/main_car.jfif')
mainCar = pygame.transform.scale(mainCar,(50,50))
mainCar = pygame.transform.rotate(mainCar,270)

def car(img,x,y):
    win.blit(img, (x,y))

boardcreation=True
#for board
for i in range(len(nomaze)):
    for j in range(len(nomaze[0])):
        if nomaze[i][j] ==0:
            #               root  color  (x,y,width,height)
            pygame.draw.rect(win,(0,0,0),(x+j*w,y+i*h,w,h))
            car(blockCar,x+j*w,y+i*h)

        else:
            pygame.draw.rect(win,(255,255,255),(x+j*w,y+i*h,w,h))

        if result and result[i][j]==1:
            pygame.draw.rect(win,(0,255,0),(x+w/4+j*w,y+h/4+i*h,w/2,h/2))
            # car(mainCar,x+j*w,y+i*h)
if result:
    car(mainCar,x+j*w,y+j*h)
else:
    car(mainCar,x,y)
pygame.display.flip()


#for tokens
while boardcreation:
    pass
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            boardcreation=False


pygame.quit()
exit()
```

Algorithm Explanation :

Input Array :

```
    nomaze = [
    [1,0,1,1,0,0,1,1,0,0],
    [1,1,1,0,0,0,0,0,0,0],
    [1,0,1,1,0,0,1,1,0,0],
    [0,1,1,0,0,1,0,1,0,1],
    [1,0,1,1,0,0,1,1,0,0],
    [1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,1,1,0,0,1,1,0,0],
    [0,0,0,0,0,0,1,0,0,1],
    [0,0,1,1,0,1,1,1,1,1]


]
```

From the above array we will starting at index nomaze[0][0] and we will be reaching to index[9][9]

In the above array 1 is represented as free space and 0 is represented as occupied space

Now at index[0][0] we have two options either to go to index[0][1] or index[1][0] we can't go to index[0][0] so we will be going to index[1][0]

From index[1][0] we can go to index[1][1] or index[2][0] but we can't go to index[2][0] because from there our car will be blocked

From index[1][1] we can go to index[1][2] and from index[1][2] we will be directly going to index[6][2] because

index[2][2],index[3][2],index[4][2],index[5][2],index[6][2] all the values are 1’s

From index[6][2] we will be directly going to index[6][6] because all the values in between are 1’s

From index[6][6] we will be going to index[9][6]

From there we will be reaching our target index

Conclusion:

The problem of parking a car in the allotted place is solved using the backtracking method and an example is provided with the algorithm and the user interface. This method can be used to park the car in a hassle-free manner in a parking lot in areas like malls and big hospitals.

References:

Pygame

Python

Algorithm

THANK YOU