**DAA LAB PROJECT REPORT**

ON

1. ***QUEEN VISUALIZER***

Submitted to-

Sovan Kumar Sahoo

BY-:

Hritika Gupta (2105720)

Satyam Rai (21051251)

Shoumiki Maulik (21051258)

Vanshika Puneet (21051865)

Sandali Dev Sinha (21052448)

***INTRODUCTION***

N QUEEN VISUALIZER

The "n-queens puzzle" is the task of placing n queens on a n × n chessboard so that no two queens may threaten one another.

Given an integer n, return every unique solution to the n-queens puzzle. You can receive the response in any order.

Each solution has a different board layout for the n-queen placement, with 'Q' denoting a queen and '.' an empty space, respectively.

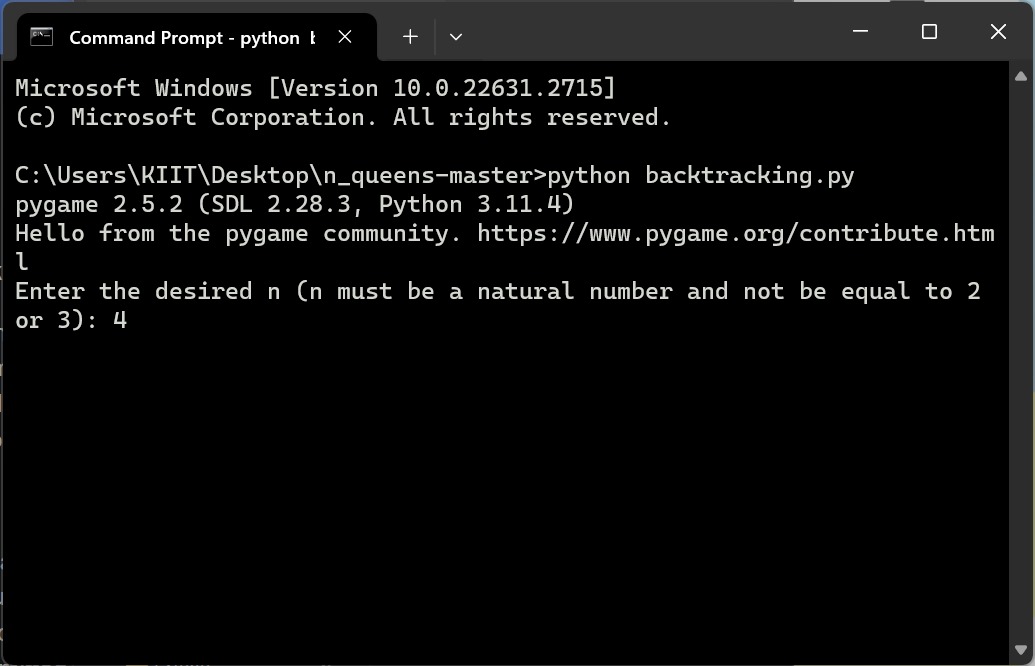
Recursion is used in the design of this algorithm, which applies the backtracking concept.

Backtracking is an algorithmic strategy for addressing problems that entail methodically going through every potential solution to an issue. It functions by gradually developing a solution and going back (reversing the previous action) when it is evident that the present course will not result in a workable solution. This procedure keeps going until a workable answer is discovered or every option has been considered.

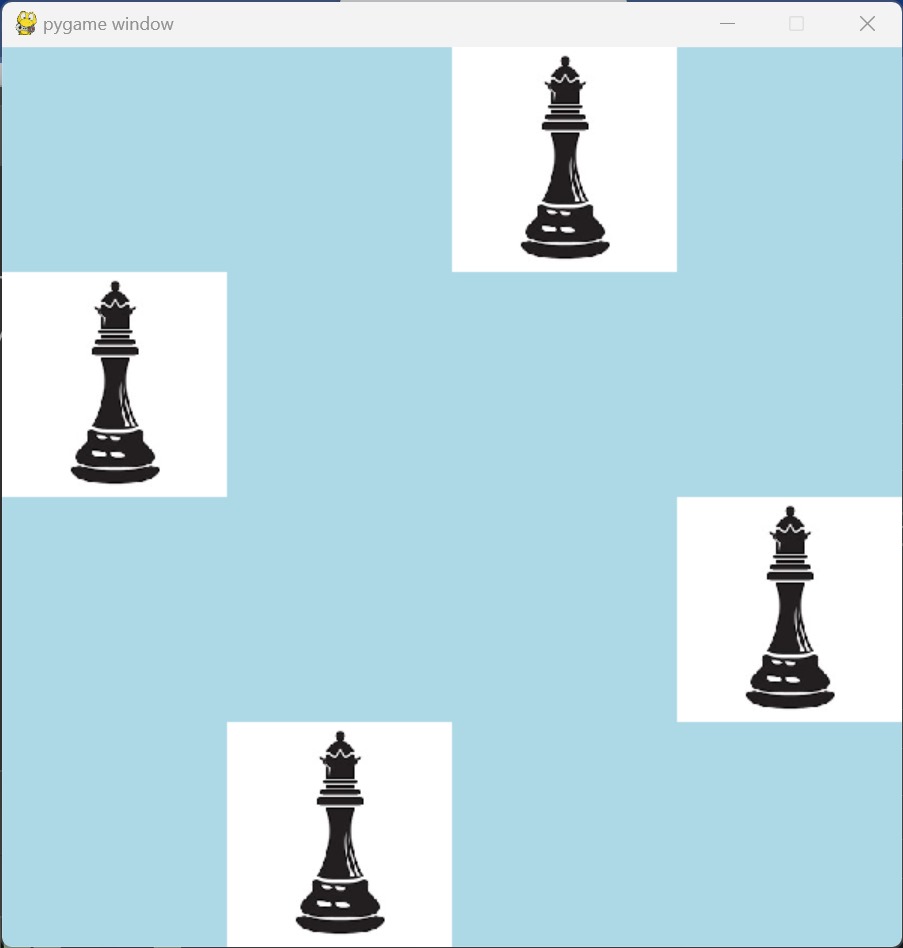
It also uses different libraries of Python for the complete representation and visualization.

***INPUT/OUTPUT***

Input:-



Output:-



***CODE***

**In c++ :-**

class Solution {

public:

vector<vector<string>> solveNQueens(int n) {

vector<vector<string>> res;

vector<string> nQueens(n, string(n, '.'));

solveNQueens(res, nQueens, 0, n);

return res;

}

private:

void solveNQueens(vector<vector<string>> &res, vector<string> &nQueens, int row, int &n) {

if (row == n) {

res.push\_back(nQueens);

return;

}

for (int col = 0; col != n; ++col)

if (isValid(nQueens, row, col, n)) {

nQueens[row][col] = 'Q';

solveNQueens(res, nQueens, row + 1, n);

nQueens[row][col] = '.';

}

}

bool isValid(vector<string> &nQueens, int row, int col, int &n) {

//check if the column had a queen before.

for (int i = 0; i != row; ++i)

if (nQueens[i][col] == 'Q')

return false;

//check if the 45° diagonal had a queen before.

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; --i, --j)

if (nQueens[i][j] == 'Q')

return false;

//check if the 135° diagonal had a queen before.

for (int i = row - 1, j = col + 1; i >= 0 && j < n; --i, ++j)

if (nQueens[i][j] == 'Q')

return false;

return true;

}

};

**In Python :-**

**import pygame**

**# get user input for n**

**def get\_input():**

**n = int(input("Enter the desired n (n must be a natural number and not be equal to 2 or 3): "))**

**if n == 2 or n == 3:**

**print("Invalid input. n must not be 2 or 3")**

**exit()**

**return n**

**# create the board**

**def create\_board(n):**

**board = []**

**for i in range(n):**

**row = [0] \* n**

**board.append(row)**

**return board**

**# if queen exists on board position, board value = 1 else value = 0**

**def solve\_n\_queens(board):**

**if not problem\_util(board, 0):**

**print("Solution does not exist")**

**return False**

**print\_solution(board)**

**return True**

**# check if queen can be placed in the position**

**def is\_safe(board, row, col):**

**n = len(board)**

**clock = pygame.time.Clock()**

**# check the row on left side**

**for i in range(col):**

**if board[row][i] == 1:**

**for j in range(col):**

**if board[row][j] == 2:**

**board[row][j] = 0**

**return False**

**board[row][i] = 2**

**draw(board, n)**

**clock.tick(2)**

**# check for queens in upper diagonal on the left side**

**for i, j in zip(range(row, -1, -1), range(col, -1, -1)):**

**if board[i][j] == 1:**

**for x, y in zip(range(row, -1, -1), range(col, -1, -1)):**

**if board[x][y] == 2:**

**board[x][y] = 0**

**return False**

**board[i][j] = 2**

**draw(board, n)**

**clock.tick(2)**

**# check for queens in lower diagonal on the left side**

**for i, j in zip(range(row, n, 1), range(col, -1, -1)):**

**if board[i][j] == 1:**

**for x, y in zip(range(row, -1, -1), range(col, -1, -1)):**

**if board[x][y] == 2:**

**board[x][y] = 0**

**return False**

**board[i][j] = 2**

**draw(board, n)**

**clock.tick(2)**

**return True**

**def problem\_util(board, col):**

**n = len(board)**

**clock = pygame.time.Clock() # to delay display**

**# if all queens are placed**

**if col >= n:**

**return True**

**# for given column, try placing queens in each row**

**for i in range(n):**

**if is\_safe(board, i, col):**

**# place the queen on board[i][col] if it is safe to place it**

**board[i][col] = 1**

**draw(board, n)**

**clock.tick(2)**

**# recursively place the rest of the queens**

**if problem\_util(board, col + 1):**

**return True**

**# if placing the queen at board[i][col] does not provide a solution, remove it**

**board[i][col] = 0**

**draw(board, n)**

**clock.tick(2)**

**# if the queen cannot be placed in any row in this column**

**return False**

**def draw(board, n):**

**win = pygame.display.set\_mode((600, 600))**

**size = 600 // n**

**queen\_image = pygame.transform.scale(pygame.image.load('queen.jpg'), (size, size))**

**# fill the window white**

**win.fill((255, 255, 255))**

**for i in range(n):**

**for j in range(n):**

**if board[j][i] == 0:**

**pygame.draw.rect(win, (173, 216, 230), [i\*size, j\*size, size, size])**

**pygame.display.update()**

**elif board[j][i] == 1:**

**pygame.draw.rect(win, (255, 255, 255), [i\*size, j\*size, size, size])**

**win.blit(queen\_image,(i\*size,j\*size))**

**pygame.display.update()**

**elif board[j][i] == 2:**

**pygame.draw.rect(win, (255, 255, 153), [i\*size, j\*size, size, size])**

**pygame.display.update()**

**# Print the solution and draw it on the board**

**def print\_solution(board):**

**n = len(board)**

**for i in range(n):**

**for j in range(n):**

**if board[i][j] == 2:**

**board[i][j] = 0**

**print(board[i][j], end=" ")**

**print()**

**draw(board, n)**

**# main function**

**def main():**

**pygame.init()**

**n = get\_input()**

**board = create\_board(n)**

**solve\_n\_queens(board)**

**run = True**

**while run:**

**for event in pygame.event.get():**

**if event.type == pygame.QUIT:**

**run = False**

**if event.type == pygame.KEYDOWN:**

**if event.key == pygame.K\_SPACE:**

**solve\_n\_queens(board)**

**main()**

***Plagiarism Report :-***

