Siddaganga Institute of Technology, Tumakuru

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

OPEN ENDED PROJECT

on

"BRICKS AND BALL"

submitted
in the partial fulfillment of the requirements IV semester
Bachelor of Engineering
In
Computer Science and Engineering

by

ADITYA RANJAN 1SI20CS005
AKASH RANJAN 1SI20CS007
ANKIT KUMAR 1SI20CS013

Department of Computer Science & Engineering
(Program Accredited by NBA)
Siddaganga Institute of Technology
B.H Road, Tumakuru-572103, Karnataka, India.
Web: www.sit.ac.in

Siddaganga Institute of Technology, Tumakuru

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

Department of Computer Science and Engineering

(Program Accredited by NBA)

CERTIFICATE

This is to certify that open ended project entitled "BRICKS AND BALL" is a bona fide work carried out by **ADITYA RANJAN, AKASH RANJAN, ANKIT KUMAR** of IV semester **Bachelor of Engineering in Computer Science and Engineering** of the **SIDDAGANGA INSTITUTE OF TECHNOLOGY** (An Autonomous Institution, affiliated to VTU, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified) during the academic year 2021-2022.

Name of Faculty	**Signature**
Mr. Vedamurthy H K	
Mr. Raghvendra M	

ABSTRACT

PROBLEM STATEMENT

In this problem, there is a given maze of size M x N. The source and the destination location is top-left cell and bottom right cell respectively. Some cells are valid to move and some cells are blocked. If we start moving from start vertex to destination vertex, we have to find that is there any way to complete the path, if it is possible then mark the correct path for the ball. The maze is given using a binary matrix, where it is marked with 1, it is a valid path, otherwise 0 for a blocked cell.

The ball can only move in two directions, either to the right or to the down. To solve these types of mazes, we first start with the source cell and move in a direction where the path is not blocked. If the path taken makes us reach the destination, then the puzzle is solved. Otherwise, we come back and change our direction of the path taken.

TABLE OF CONTENTS

Sl. No	**Contents**	**Page No.**
1	Introduction	5
2	Design of Algorithm and Objectives	6
3	Design and Implementation of code	9
4	Result Analysis	13
5	Conclusion	14
6	Reference	14

INTRODUCTION

Backtracking is a technique based on algorithm to solve 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 doesn't give rise to the solution of the problem based on the constraints given to solve the problem.

Backtracking algorithm is applied to some specific types of problems,

- Decision problem 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 the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

Approach: Form a recursive function, which will follow a path and check if the path reaches the destination or not. If the path does not reach the destination, then backtrack and try other paths.

DESIGN OF ALGORITHM

```
isSafe (maze, x, y)
//check whether a position is blocked or safe
//Input: x and y point in the maze.
//Output: true if the (x, y) place is valid, otherwise false.

Begin
  if x and y are in range and (x, y) place is not blocked, then
    return true
  return false
End

Algorithm: bricksAndBall (maze, x, y, solutionMaze)
//traverse the maze and find the solution maze
//Input: The starting point x and y.
//Output: The path to follow by the ball to reach the destination, //otherwise false.

Begin
  if (x, y) is the bottom right corner, then
    mark the place as 1
    return true
  if bricksAndBall (x, y) = true, then
    mark (x, y) place as 1
    if bricksAndBall (x+1, y) = true, then //for forward movement
      return true
    if bricksAndBall (x, y+1) = true, then //for down movement
      return true
    mark (x, y) as 0 when backtracks
    return false
  return false
End
```

OBJECTIVE

Input:

A binary matrix of size m*n with 0 as bricks and 1 as a path.

Constraints:

Time Complexity: O(2^(n^2)).

The recursion can run upper-bound 2^(n^2) times.

Space Complexity: O(n^2).

Output:

A solution matrix having a path from start to end of the maze.

Sample Input:

m = 5 and n = 5

```
1 0 1 0 1
1 1 1 1 1
1 1 0 1 0
0 0 0 1 1
1 1 1 0 1
```

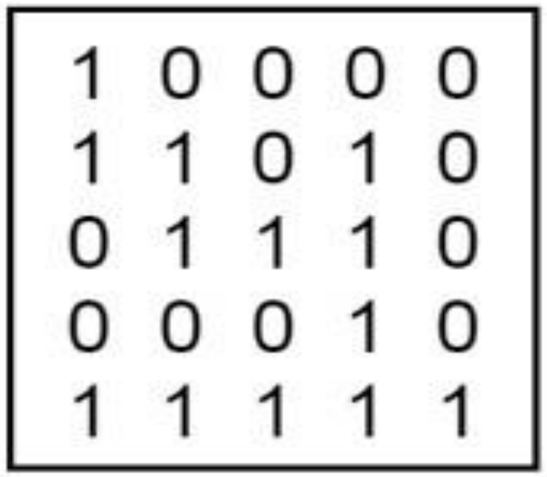
Sample Output:

```
1 0 0 0 0
1 1 1 1 0
0 0 0 1 0
0 0 0 1 1
0 0 0 0 1
```

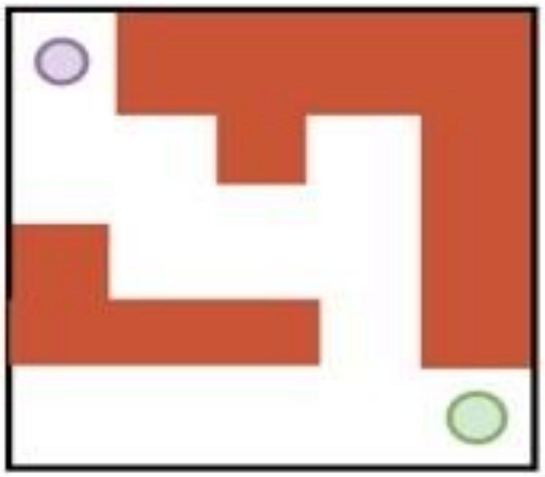
EXPLANATION

This algorithm will take the maze as a matrix.

In the matrix, the value 1 indicates the free space and 0 indicates the wall or blocked area.

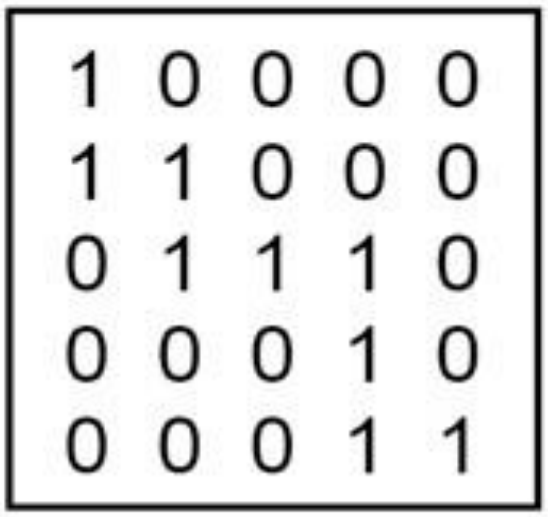


1	0	0	0	0
1	1	0	1	0
0	1	1	1	0
0	0	0	1	0
1	1	1	1	1

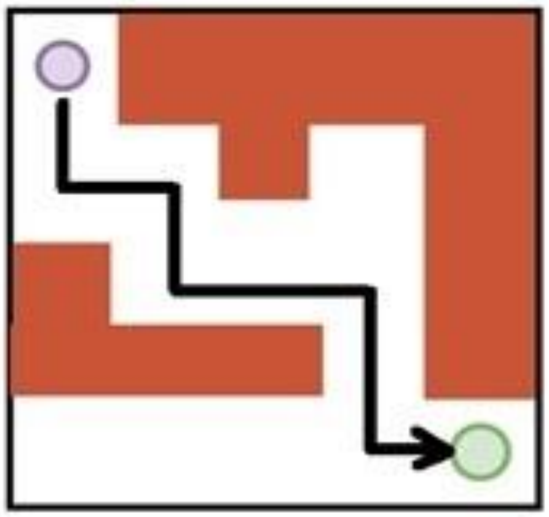


In this diagram, the top-left circle indicates the starting point and the bottom-right circle indicates the ending point.

It will display a matrix. From that matrix, we can find the path of the rat to reach the destination point.



1	0	0	0	0
1	1	0	0	0
0	1	1	1	0
0	0	0	1	0
0	0	0	1	1



IMPLEMENTATION OF CODE

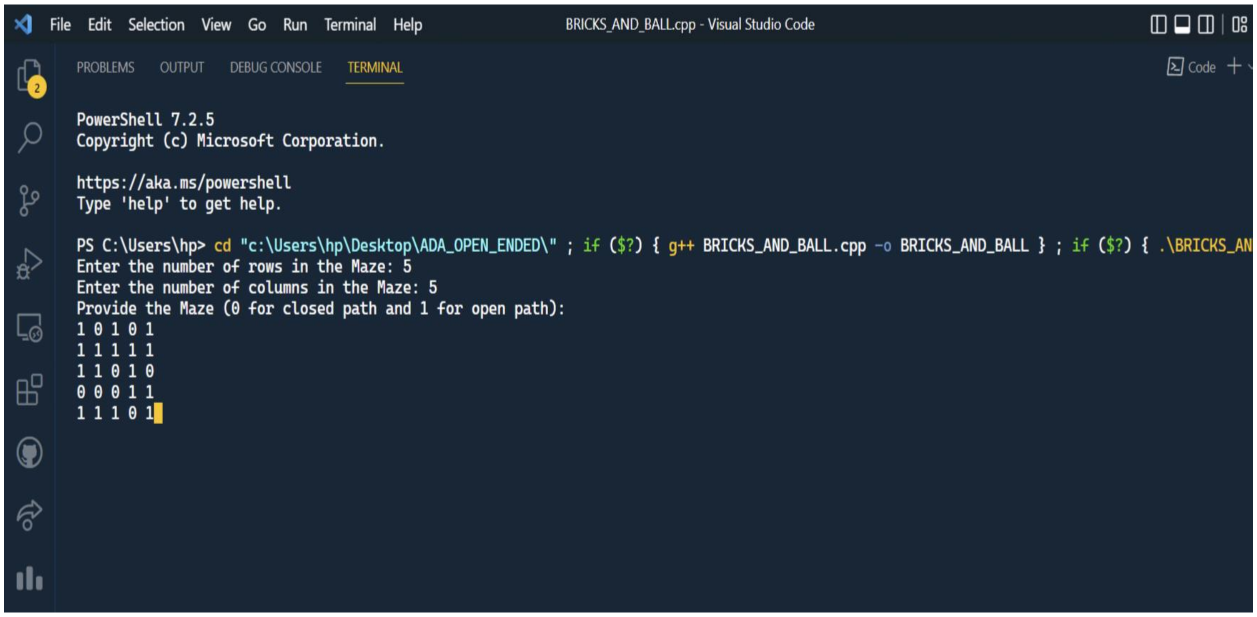
```
#include <bits/stdc++.h>
using namespace std;

// function to check whether a position is safe or not in the maze
// it checks whether a position is 1 if so then it returns true otherwise false
bool isSafe(int **maze, int x, int y, int mazeRow, int mazeColumn)
{
    if (x < mazeRow && y < mazeColumn && maze[x][y] == 1)
    {
        return true;
    }
    return false;
}

// function to traverse the maze
// It stores the solution in a solution array
bool bricksAndBall(int **maze, int x, int y, int mazeRow, int mazeColumn, int
**solutionMaze)
{
    // base case
    // update the last position with 1 when it is reached
    if (x == mazeRow - 1 && y == mazeColumn - 1)
    {
        solutionMaze[x][y] = 1;
        return true;
    }
    // update the current position with 1 in solutionMaze
    if (isSafe(maze, x, y, mazeRow, mazeColumn))
    {
        solutionMaze[x][y] = 1;
        // check the maze for the position down to it recursively
        if (bricksAndBall(maze, x + 1, y, mazeRow, mazeColumn, solutionMaze))
        {
            return true;
        }
        // check the maze for the position right to it recursively
        if (bricksAndBall(maze, x, y + 1, mazeRow, mazeColumn, solutionMaze))
        {
            return true;
        }
        // backtracking
        // if we cannot go anywhere update that position with 0 and backtrack
        solutionMaze[x][y] = 0;
        return false;
    }
    return false;
}

// Driver Code
int main()
{
    // maze row and maze column size input
    int mazeRow, mazeColumn;
    cout << "Enter the number of rows in the Maze: ";
    cin >> mazeRow;

    cout << "Enter the number of columns in the Maze: ";
    cin >> mazeColumn;

    // dynamic memory allocation for maze
    int **maze = new int *[mazeRow];
```

```
{
    maze[i] = new int[mazeColumn];
}

// maze input
cout << "Provide the Maze (0 for closed path and 1 for open path): " << endl;
for (int i = 0; i < mazeRow; i++)
{
    for (int j = 0; j < mazeColumn; j++)
    {
        cin >> maze[i][j];
    }
}

system("CLS");
cout << "\t\t\t\t\t  MAZE" << endl;
cout << endl
     << endl;

for (int i = 0; i < mazeRow; i++)
{
    cout << "\t\t\t\t\t";
    for (int j = 0; j < mazeColumn; j++)
    {
        cout << maze[i][j] << " ";
    }
    cout << endl;
}

// solution maze declaration and initialization
int **solutionMaze = new int *[mazeRow];
for (int i = 0; i < mazeRow; i++)
```

```
        solutionMaze[i] = new int[mazeColumn];
        for (int j = 0; j < mazeColumn; j++)
            solutionMaze[i][j] = 0;
    }
    cout << endl;

    // ratInMaze function call and display the solution Maze
    cout << "\t\t\t\t      SOLUTION MAZE" << endl;
    cout << endl
         << endl;

    if (bricksAndBall(maze, 0, 0, mazeRow, mazeColumn, solutionMaze))
    {
        for (int i = 0; i < mazeRow; i++)
        {
            cout << "\t\t\t\t\t";
            for (int j = 0; j < mazeColumn; j++)
            {
                cout << solutionMaze[i][j] << " ";
            }
            cout << endl;
        }
    }
    return 0;
}

// 1 0 1 0 1
// 1 1 1 1 1
// 1 1 0 1 0
// 0 0 0 1 1
// 1 1 1 0 1
```

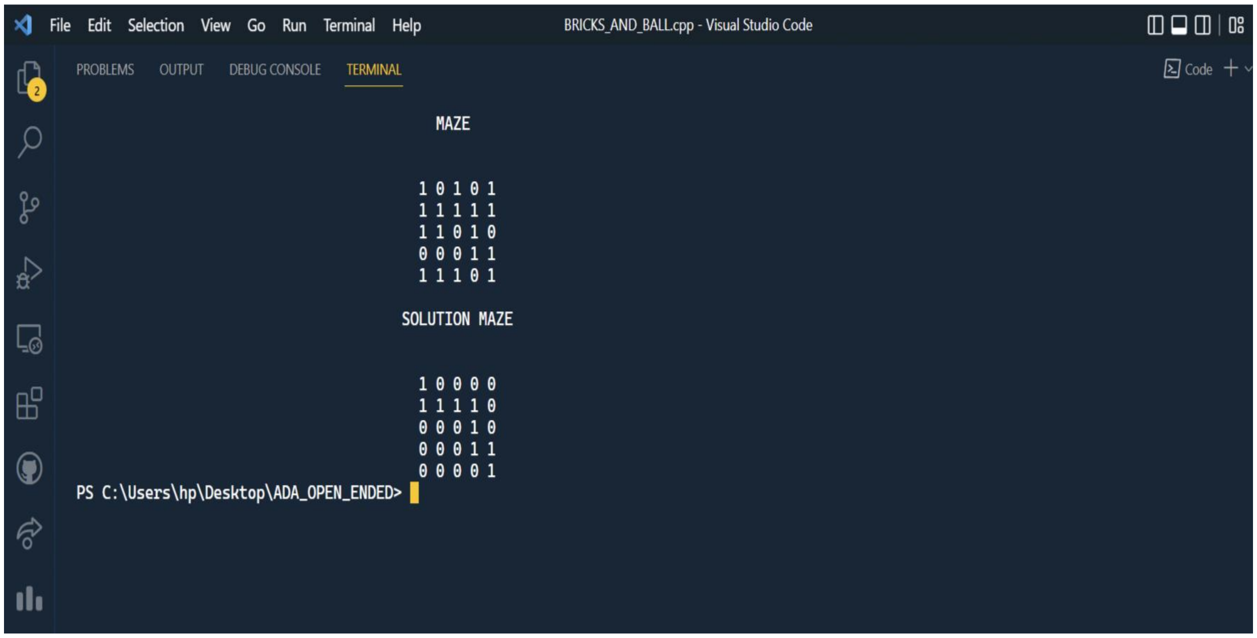
RESULT ANALYSIS DETAILS

1. Maze Input


File Edit Selection View Go Run Terminal Help
BRICKS_AND_BALL.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code

```
PowerShell 7.2.5
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\hp> cd "c:\Users\hp\Desktop\ADA_OPEN_ENDED\" ; if ($?) { g++ BRICKS_AND_BALL.cpp -o BRICKS_AND_BALL } ; if ($?) { .\BRICKS_AN
Enter the number of rows in the Maze: 5
Enter the number of columns in the Maze: 5
Provide the Maze (0 for closed path and 1 for open path):
1 0 1 0 1
1 1 1 1 1
1 1 0 1 0
0 0 0 1 1
1 1 1 0 1
```

2. Solution Maze


File Edit Selection View Go Run Terminal Help
BRICKS_AND_BALL.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code

```
                                        MAZE


                                      1 0 1 0 1
                                      1 1 1 1 1
                                      1 1 0 1 0
                                      0 0 0 1 1
                                      1 1 1 0 1

                                    SOLUTION MAZE


                                      1 0 0 0 0
                                      1 1 1 1 0
                                      0 0 0 1 0
                                      0 0 0 1 1
                                      0 0 0 0 1
PS C:\Users\hp\Desktop\ADA_OPEN_ENDED>
```

CONCLUSION

In this way, a “bricks and ball maze problem” can be solved in which we get solution maze for a given problem maze. This project is helpful in solving complex mazes and get a solution easily and efficiently. We solved this problem using the concept of recursion and backtracking. Without using backtracking, we cannot store all the possible paths out of the maze. In the end, we developed a C++ program to demonstrate this algorithm.

REFERENCES

https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/
https://www.tutorialspoint.com/Rat-in-a-Maze-Problem
[Narasimha Karumanchi] "Data Structures and Algorithms Made Easy".