MAZE SOLVER USING BFS

(breadth-first search)

-RISHIT	191IT141
-SARTHAK JAIN	191IT145
-YASH GUPTA	191IT158

INTRODUCTION

It is a very common game to find the path from source to destination in a grid with some obstacles (maze) on which we cannot walk. To solve this manually is quite troublesome and time consuming as we explore the different paths which could possibly lead us to destination. In this project we will be making that work easy and fast by using the Breadth First Search algorithm. The code will provide us with the shortest possible path from the source to the destination. In this algorithm we traverse the grid in such a way that each grid block will be checked in order of its increasing distance from the source node. It can be compared to the pattern observed at the surface of water when a stone is dropped in the river. So it keeps checking every block of grid and when we find the destination block it surely is from the shortest path.

The project will find the shortest path for any grid as long as it is uniform(size of each block is same) because BFS algorithm finds the shortest path only in the case of uniformly weighted graph(grid).

In order to show to the user the exact paths checked by this algorithm, we have also provided a graphical interface in which the user will be able to see in real time, how the algorithm is checking the blocks in breadth first fashion and when it finally finds the destination, it traces back that shortest path back to the destination.

WORK DONE

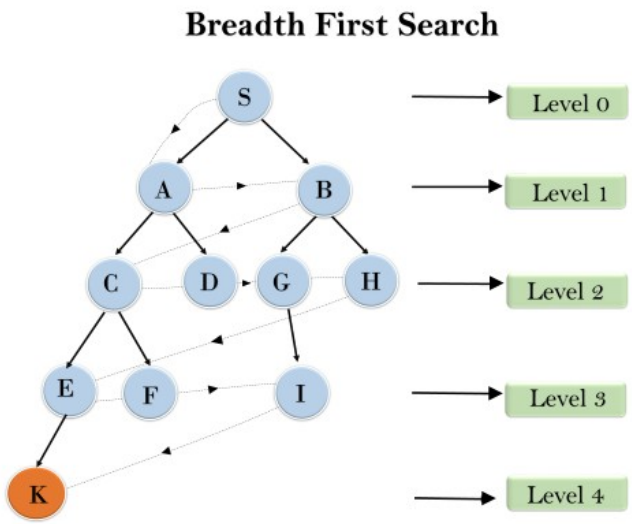
SHORTEST PATH:

The user provides two points on the grid, the start and the destination.The bfs algorithm then explores all paths from a point on the grid.It checks if a path is possible in the four directions (left,right,up,down).If a path is possible then the next point on the path is stored in the dictionary data structure of python where key is the current position and the value of the dictionary is the next point on the path.

The dictionary of python is used to find the shortest path from the source to destination using the concept of backtracking.The destination is first colored an then the destination's key is colored.The destination's key is then made the value and the corresponding key is colored.This process is looped out until we reach the source as the value of our dictionary.Thus,we have succesfully obtained our shortest path between the source and the destination using the concept of backtracking with the help of dictionaries data structure of python.

BFS: pseudo code

```
Set all nodes to "not visited";

q = new Queue();

q.enqueue(initial node);

while ( q ≠ empty ) do
{
   x = q.dequeue();

   if ( x has not been visited )
   {
      visited[x] = true;          // Visit node x !

      for ( every edge (x, y)  /* we are using all edges ! */
         if ( y has not been visited )
            q.enqueue(y);        // Use the edge (x,y) !!!
   }
}
```



BFS will search for all possible paths in order to find the shortest path we will be saving parent node of each vertex while doing BFS traversal of graph.

From destination we will be going in the backward direction till we encounter the source

Pseudo code:

```
def back_route(end_x,end_y):

while (x, y) != (start_x, start_y):

x,y=parent[x,y]
```

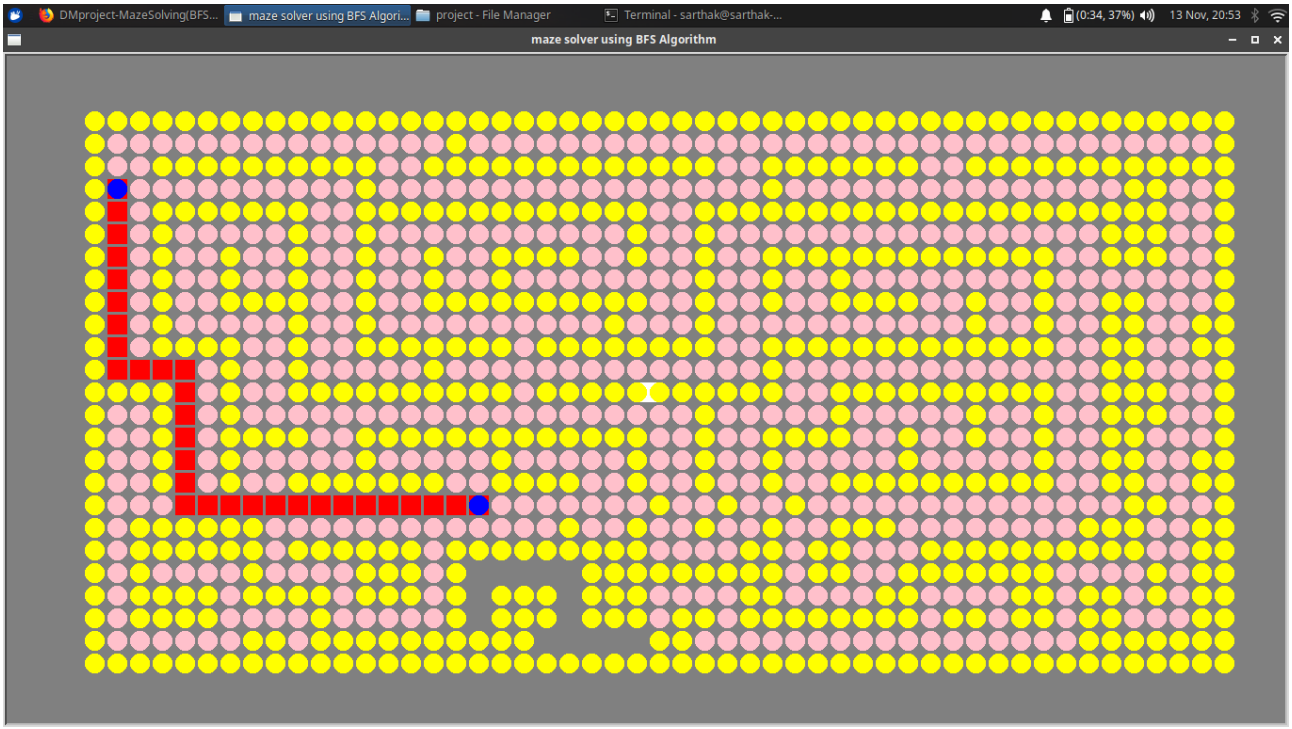
GUI:

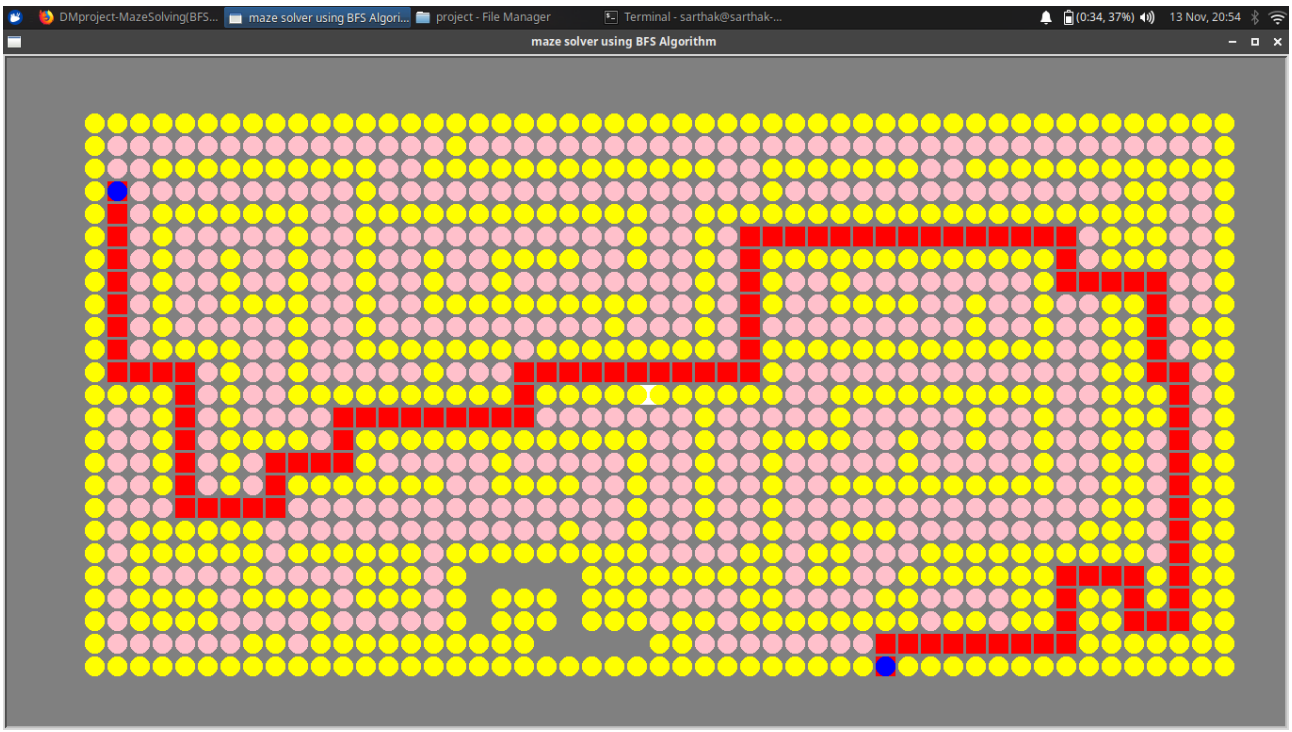
For displaying maze we have used python's turtle module. Turtle is basically like a drawing board on which we can draw various shapes.

Turtle module uses tkinter for the underlying graphics.
The TurtleScreen class defines graphics windows as a playground for the drawing turtles. Its constructor needs a tkinter.Canvas or a ScrolledCanvas as argument.
For the generation of maze we have used stamp function which stamps the copy of created turtle at the given coordinates. For giving features to turtle like shape,color,penup/pendown feature , a constructor is used to initialize these values for particular turtle.
To prevent screen from disappearing once the results are calculated turtle.exitonclick() is used.

Such stamps of various colors are generated and are drawn at certain coordinates

RESULTS AND DISCUSSIONS





CONCLUSION

This program finds the shortest path between two points on a grid, provided there are some/none obstacles on the path.The algorithm of our project first finds out all the paths from the source in all directions.It then uses backtracking to retrieve the shortest path from the source to the destination and highlights it using a separate color.

The user with the help of this program can visualise how a bfs algorithm works in finding all paths from a certain point on thre grid.The program further provides a graphical represantion of the shortest path between two paths by backtracking the path from the destination to the source using a significant color.

Our program,thus enables a person in understanding the working of a bfs algorithm on a grid along with the concept of backtracking using visual graphics.

THE END