**ADVANCED DATA STRUCTURES MINI PROJECT**

**TOPIC : PATH FINDING THROUGH OBSTACLES**

**BRANCH : SE-IT**

**BATCH : A**

**GROUP MEMBERS :**

**NAME : UID :**

**JAYNIL GAGLANI 2017140013**

**SHUBHAM GOGATE 2017140015**

**Introduction :**

**We have implemented BFS & A-STAR algorithms to**

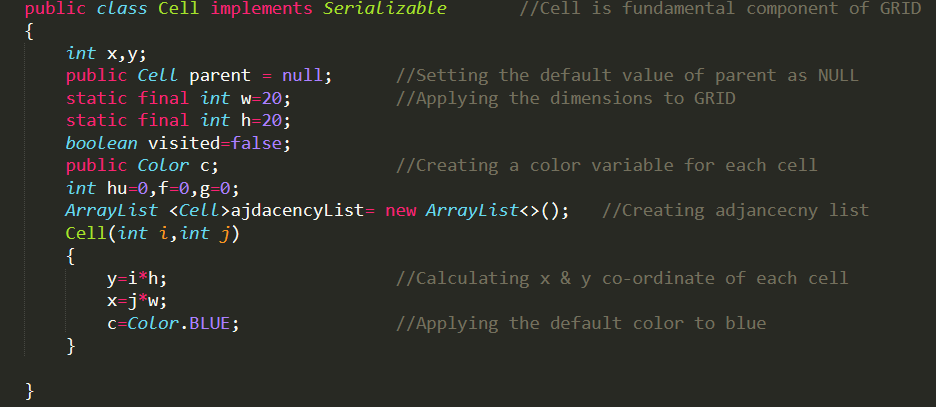
**find best possible paths from given source to destination.**

**And also comparison of the two algorithms is shown while finding path form same source and destination.**

**RGB color scheme is used to represent the distance array. Hence , we found that A-STAR algorithm is more efficient of the two for finding the best possible path.**

# Procedure:…………..

🡪The Data of each cell is stored in a Object of type Cell



The Class `Test` contains the main function and serves as the entry point of the program.

It creates the Windows on which the the Game is Displayed And Creates

The Graphics Context Of the program “MyCanvas” Object

It Has a Public Static member known as Map which is an array of ‘Cell’ Objects.

The Array is initialized by the in the main function where the Cells Are Constructed and their Adjacency lists are populated.

Another Way to Initialise the Cells is by loading the data from a pre-saved binary file

Function saveFile() serializes the cell objects and stored them in a file so that loadFile() can read The file and initialize the map with Cells with already ready map

updateAdjacencyList() takes in a cell as a parameter and removes itself from the adjacency List of all its Neighbours

class MyCanvas extends java.awt.Canvas and overrides its paint method.

Overridden paint method paints all the cells according to their coordinates and color on the screen

MyCanvas also implements MouseListener and overrides all its methods.

All methods except public void mousepressed() are left empty

In MousePresses the X and Y coordinates of the mouse are used to determine the cell on which the click event has occurred.

Now based on the type of mouse button the following things may occur

* If Middle mouse button is pressed then the cell is blocked(greyed out)and updateAdjacencyList is called.
* If left mouse button is pressed then the Cell is declared as the source in the following operations
* If right mouse button is pressed then the Cell is set as destination in the following operations

In the Main function there are two pathfinding algorithms used

* BFS
* A\*

1. **BFS Algorithm :**

**It starts at the root and explores all of it’s children in the next level(neighbors) before moving to each of the root children, and then, it explores the children of the root children, and so on. The algorithm uses a queue to perform the BFS.**

**1. Add root node to the queue, and mark it as visited(already explored).**

**2. Loop on the queue as long as it's not empty.**

**1. Get and remove the node at the top of the queue(current).**

**2. For every non-visited child of the current node, do the following:**

**1. Mark it as visited.**

**2. Check if it's the goal node, If so, then return it.**

**3. Otherwise, push it to the queue.**

**3. If queue is empty, then goal node was not found!**

**2. A-STAR Algorithm :**

A\* is a combination of Dijkstra and Greedy. It uses distance from the root node plus heuristics distance to the goal. The algorithm terminates when we find the goal node.

1. Assign dis[v] for all nodes = INT\_MAX (distance from root node + heuristics of every node).

2. Assign dis[root] = 0 + heuristic(root, goal) (distance from root node to itself + heuristics).

2. Add root node to priority queue.

3. Loop on the queue as long as it's not empty.

1. In every loop, choose the node with the minimum distance from the root node in the queue + heuristic (root node will be selected first).

2. Remove the current chosen node from the queue (vis[current] = true).

3. If the current node is the goal node, then return it.

4. For every child of the current node, do the following:

1. Assign temp = distance(root, current) + distance(current, child) + heuristic(child, goal).

2. If temp < dis[child], then, assign dist[child] = temp. This denotes a shorter path to child node has been found.

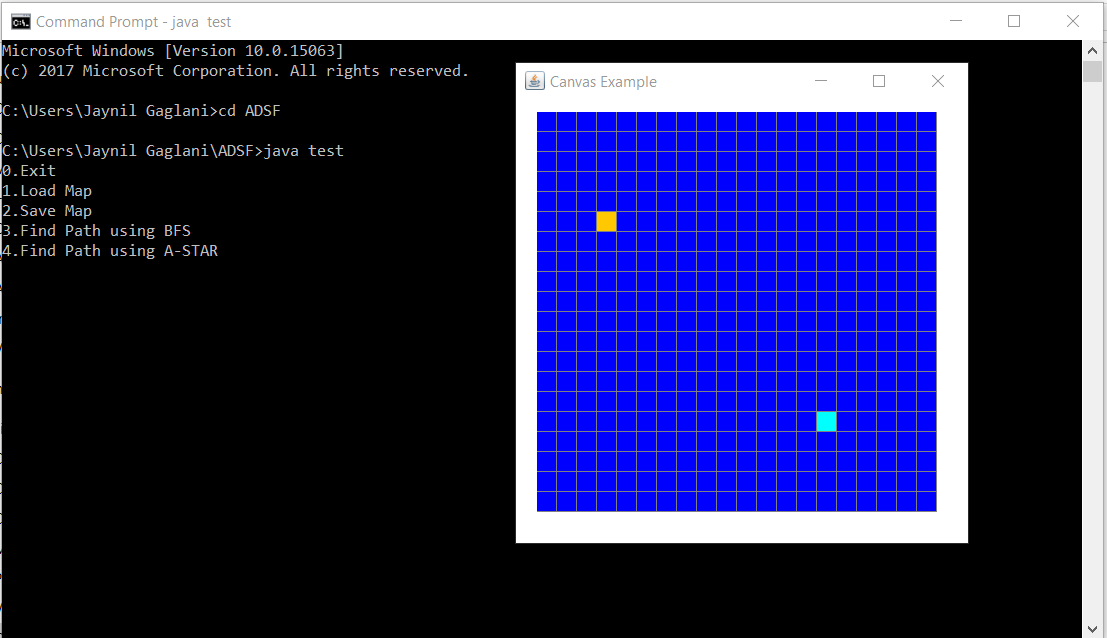
3. And, add child node to the queue if not already in the queue (thus, it's now marked as not visited again).

4. If queue is empty, then goal node was not found!

**Output :**

**1. SOURCE & DESTINATION**

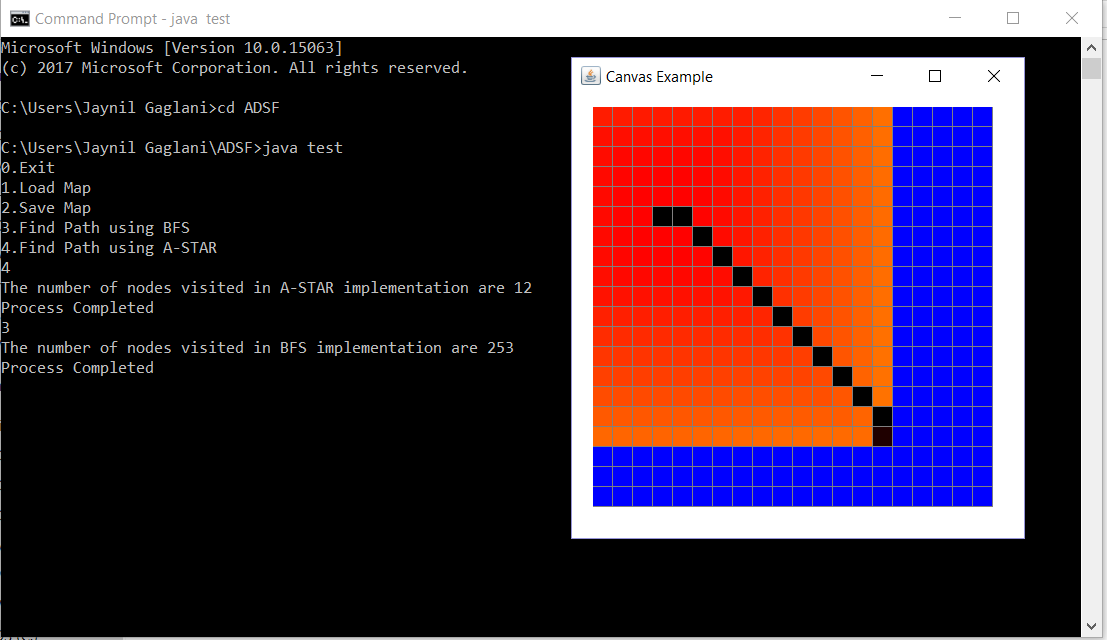
**( YELLOW , CYAN )**

****

**2. NORMAL A-STAR**

****

**3. NORMAL BFS**

****

**4. SAVED MAP “MAZE”**

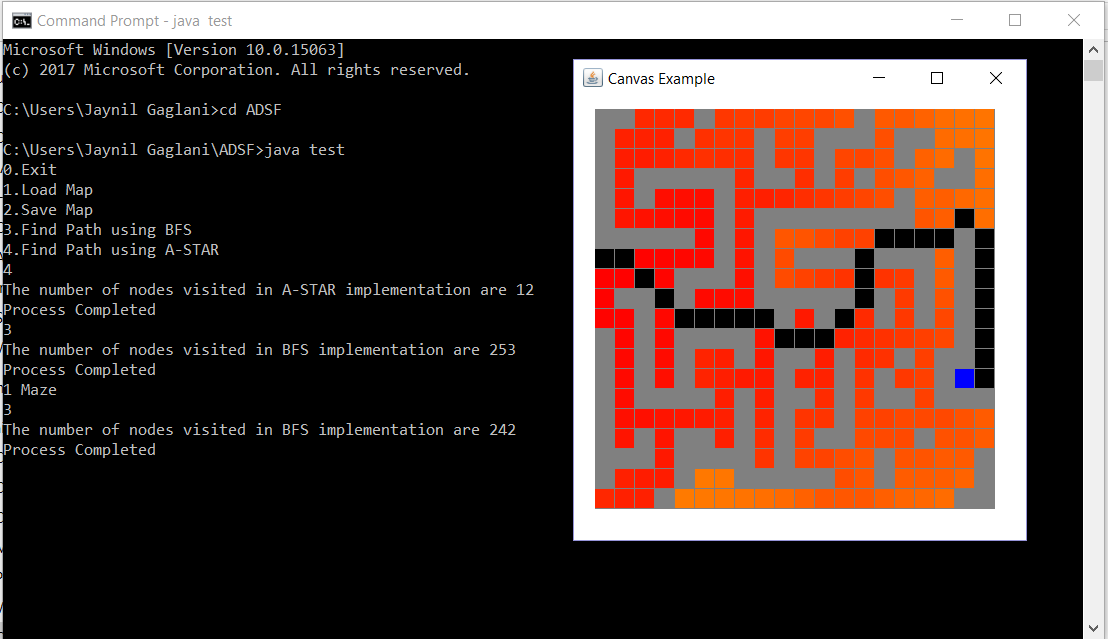
****

1. **SOURCE & DESTINATION IN “MAZE”**

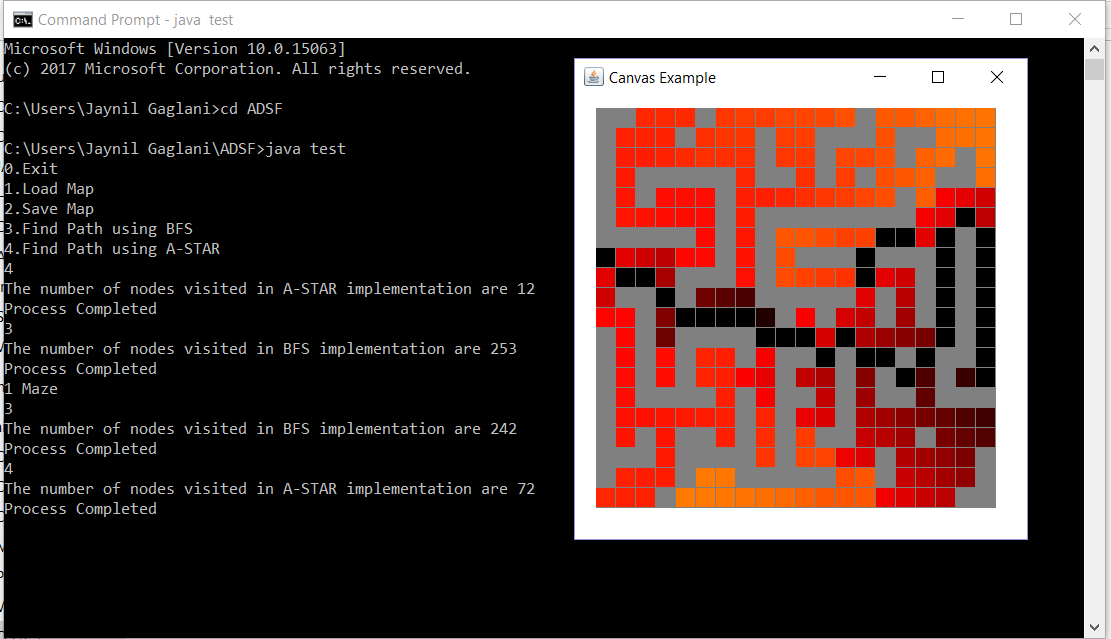
**( YELLOW , CYAN )**

****

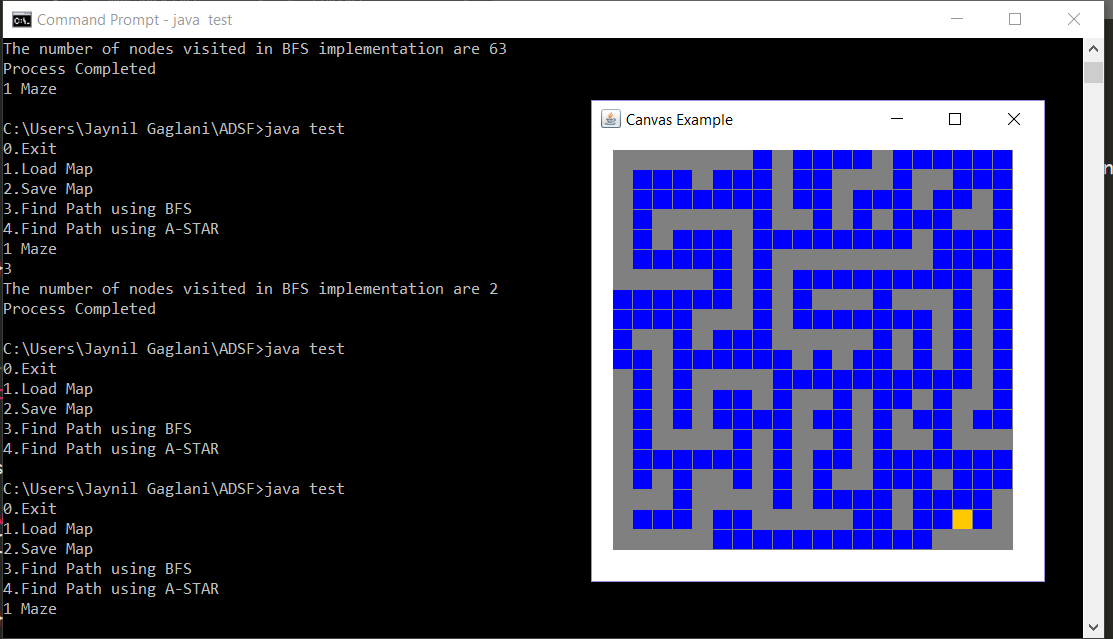
**6. BFS IN “MAZE”**

****

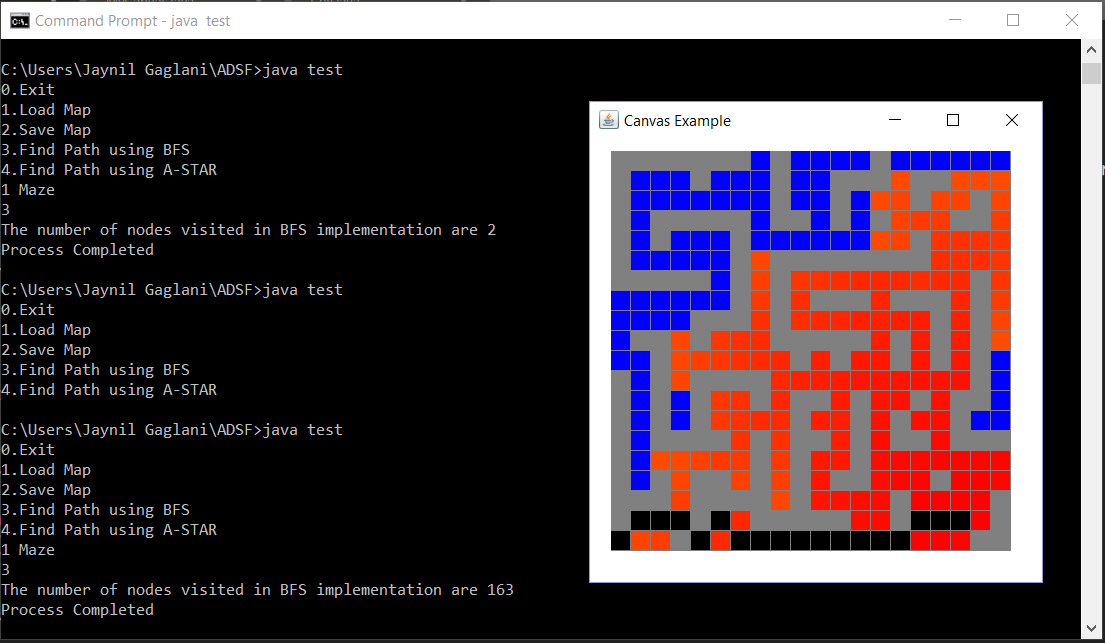
**7. A-STAR IN “MAZE”**

****

**8. SETTING A SOURCE & RESTRICTING THE DESTINATION TO FIRST ROW & COLUMN.**

****

1. **BFS IMPLEMENTATION OF ABOVE CASE**

****