PATH FINDING VISUALIZER

A PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the degree in

BACHELOR OF COMPUTER APPLICATION

Submitted By

Renaud Jaiswal Roll Number

VIth Semester 180010580033

Under the Guidance of

Dr. Puneet Misra

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF LUCKNOW

LUCKNOW

Department of Computer Science

University of Lucknow

<u>CERTIFICATE</u>

This is to certify that the project titled **"PATH FINDING VISUALIZER"** has been developed by **Renaud Jaiswal** student of BCA VIth semester under my supervision in partial fulfillment for the award of degree in **Bachelor of Computer Application (BCA)** for the session 2019-20.

Prof. Brijendra Singh	**Dr. Puneet Misra**
Head of Department	**Project Guide**
Department of Computer Science	
University of Lucknow	

ACKNOWLEDGEMENT

Firstly, it gives me immense, pleasure to give my heartily thanks and respect to the wonderful person involved in making this project. I would like to express my special thanks of gratitude to my Project Guide- **Dr. Puneet Misra**, Assistant Professor, Department of Computer Science, University of Lucknow for providing his valuable guidance and support on which rests the foundation of this project. He provided me a way to implement my knowledge in the right direction. He taught me the methodology to carry out the project and present the report as clear as possible. It is an opportunity with great privilege and honor to work and study under his guidance.

I have always considered project development a challenging task that requires lot of concentration, hard work, and understanding several concepts related to project. I have learnt various new concepts, programming styles and above all I gain wonderful experience of project development and presentation.

I am also very thankful to other faculties of Computer Science who guide me with their valuable suggestions.

I would also like to thank my parents for their love, prayers and caring. I am also thankful to almighty God; whose blessings are always with me.

Renaud Jaiswal

Roll No. 180010580033

TABLE OF CONTENT

PURPOSE & REASON

The **PATH FINDING VISUALIZER** is a software developed to demonstrate the actual execution of graph search algorithms used in computer science field. Graph search algorithms explore a graph either for general discovery or explicit search. These algorithms carve paths through the graph, but there is no expectation that those paths are computationally optimal.

Moving from one place to another is a task that we humans do almost every day. We try to find the shortest path that enables us to reach our destinations faster and make the whole process of travelling as efficient as possible. In the old days, we would trial and error with the paths available and had to assume which path taken was shorter or longer.

Now, we have algorithms that can help us find the shortest paths virtually. We just need to add costs (time, money etc.) to the graphs or maps and the algorithm finds us the path that we need to take to reach our destination as quick as possible. Pathfinding algorithms are usually an attempt to solve the shortest path problem in graph theory. They try to find the best path given a starting point and ending point based on some predefined criteria.

Path finding algorithms are important because they are used in applications like google maps, satellite navigation systems, routing packets over the internet. The usage of pathfinding algorithms isn't just limited to navigation systems. The overarching idea can be applied to other applications as well. The usage will become clearer as we talk about some examples and implementations of pathfinding algorithms.

OBJECTIVE

The objective of this project is as follows –

- To represent real time execution of graph search algorithms on the user's screen.
- Real time plotting of the shortest possible path between the start and the end node.
- To integrate graph search algorithms such as Breadth First Search and A Star algorithm into one system.
- To help user understand the working of the graph search algorithms and their applications.
- To provide better graphic interface to the user so that he/she doesn't face any difficulty to understand the working of system.
- To validate the input given by the user.

PROPOSED SYSTEM

This application enables user to visualize the graph search algorithms in action. The user can draw a maze by the choosing the grids and then set the start and end node, then user can select either breadth first search or a star option from the side menu. After obtaining the required inputs the application then starts the operation of finding the shortest path between the start and end node in a manner that it doesn't overlap with the boundaries of the maze set by the user. Once the shortest path has been found it highlights the path in dark red color.

The purpose of this project is to visualize the system by the help of programming code and full-fledged computer software, fulfilling their requirements. The required software and hardware are easily available and easy to work with.

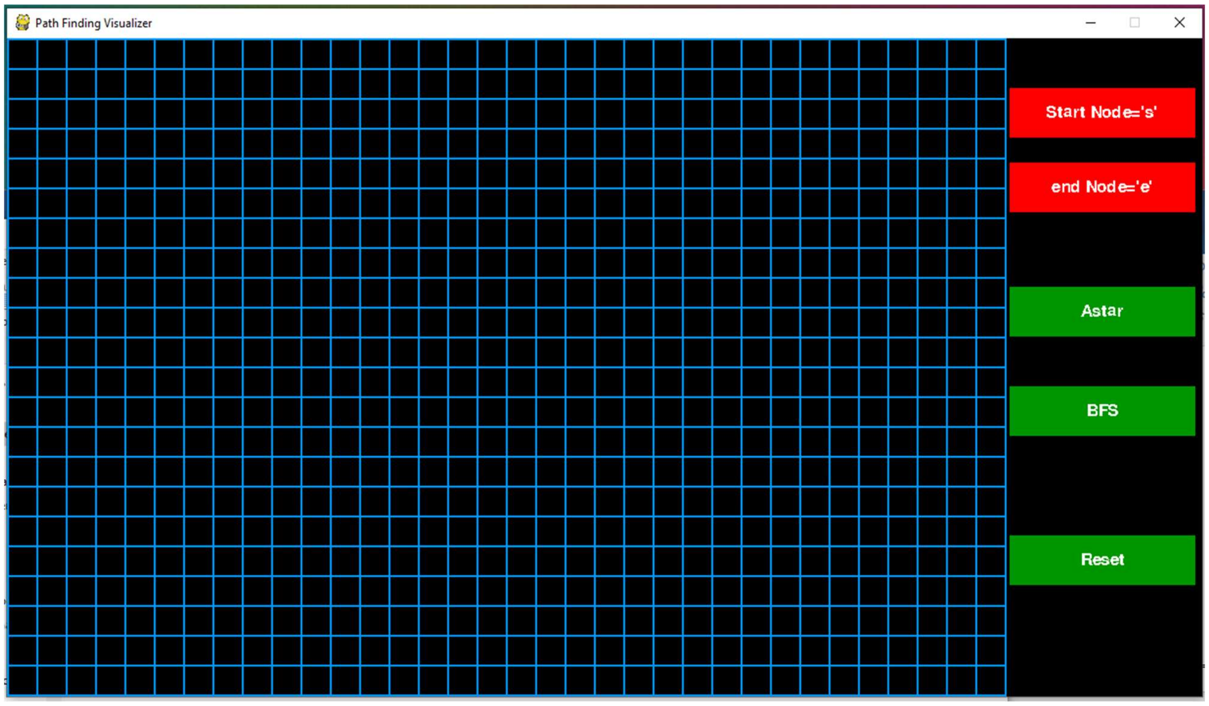
The proposed system provides a better GUI application which efficiently and effectively deals with displaying the working of the stated algorithms above.

Here, for the ease of the user, integration of all the numerical method related problem under one section and all graphical problems into another section called computer graphics.

The proposed system is flexible and user friendly. The end-user who is related to mathematics or computer science field can operate this visualizer application very easily.

The end users of this application may be scientists, researchers, CS or math students, computer science teachers. In short, anybody related to CS or mathematics field can access this application and achieve a basic demonstration of the working of graph search algorithms.

In this system user only has to set certain parameters and feed data accordingly into appropriate section and this will plot the solution. Below image demonstrates what the application's GUI looks like.



SYSTEM ANALYSIS

The word “system” is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.

A system is “an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal.”

System Constraints:

A system must have three basic constraints −

- A system must have some **structure and behaviour** which is designed to achieve a predefined objective.
- **Interconnectivity** and **interdependence** must exist among the system components.
- The **objectives of the organization** have a **higher priority** than the objectives of its subsystems.

For example, traffic management system, payroll system, automatic library system, human resources information system.

System analysis is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem-solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

Analysis specifies what the system should do.

Problem Definition:

Whenever, the problem is visualized, it is not the same as it appears. But there are some other aspects also that come into the picture only after a sharp deep study of the problem. This phase of system development is of great importance. For a system analyst, it is necessary to have a deep knowledge of the topic on which he is working.

Here are some of the problems of the current system:

- Outdated techniques and methods to visualize the working of search algorithms.

- Outdated GUIs of such applications available on the internet.

- Unable to provide better methods of interactions between the user and the application, lack of gamified approach and attention to user experience.

- Lack of appealing functionality.

Requirement Specification:

A software requirements specification (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system, we should have clear understanding of Software system. To achieve this, we need to continuous communication with customers to gather all requirements.

A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real-life scenarios. Using the Software requirements specification (SRS) document on QA lead, managers create test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.

It is highly recommended to review or test SRS documents before start writing test cases and making any plan for testing. Let's see how to test SRS and the important point to keep in mind while testing it.

To fulfill the task written in the requirement specification document, it is required to find the answer of following questions:

- What is graph theory?
- What are graph search algorithms?
- What is **Breadth First Search** algorithm?
- What is **A*** algorithm?
- What are the properties of these algorithms?
- What are the applications of these algorithms?
- What is **Pygame** in python?

To answer these questions, it is required to understand the following topics :-

GRAPH THEORY:

Graph Theory is the study of lines and points. It is a sub-field of mathematics which deals with graphs: diagrams that involve points and lines and which often pictorially represent mathematical truths. Graph theory is the study of the relationship between edges and vertices.

Formally, a graph is a pair (V, E), where V is a finite set of vertices and E a finite set of edges. Graph Theory has some unique vocabulary:

- An **arc** is a directed line (a pair of ordered vertices).
- An **edge** is line joining a pair of nodes.
- **Incident** edges are edges which share a vertex. A edge and vertex are **incident** if the edge connects the vertex to another.
- A **loop** is an edge or arc that joins a vertex to itself.
- A **vertex**, sometimes called a **node**, is a point or circle. It is the fundamental unit from which graphs are made.
- **Adjacent** vertices are vertices which are connected by an edge.
- The **degree** of a vertex is simply the number of edges that connect to that vertex. Loops count twice.
- A **predecessor** is the node (vertex) before a given vertex on a path.
- A **successor** is the node (vertex) following a given vertex on a path.
- **A walk** is a series of vertices and edges.
- A **circuit** is a closed walk with every edge distinct.
- A **closed walk** is a walk from a vertex back to itself; a series of vertices and edges which begins and ends at the same place.
- A **cycle** is a closed walk with no repeated vertices (except that the first and last vertices are the same).
- A **path** is a walk where no repeated vertices. A **u-v** path is a path beginning at u and ending at v.
- A **u-v walk** would be a walk beginning at u and ending at v.

The major role of graph theory in computer applications is the development of graph algorithms. Numerous algorithms are used to solve problems that are modeled in the form of graphs. These algorithms are used to solve the graph theoretical concepts which intern used to solve the corresponding computer science application problems. Some algorithms are as follows:

1. Shortest path algorithm in a network

2. Finding a minimum spanning tree

3. Finding graph planarity

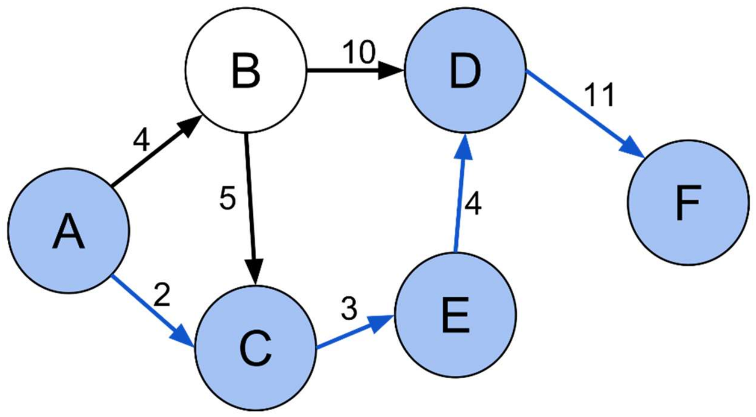
4. Algorithms to find adjacency matrices.

5. Algorithms to find the connectedness

6. Algorithms to find the cycles in a graph

7. Algorithms for searching an element in a data structure (DFS, BFS) and so on. Various computer languages are used to support the graph theory concepts. The main goal of such languages is to enable the user to formulate operations on graphs in a compact and natural manner.

Graphs are mathematical structures used to model pairwise relationships in between objects. A graph consists of nodes/vertices/points that are connected edges/links/lines. Look at the diagram below:



The circles with letters in them are nodes/vertices/points and the lines connecting/relating the circles are the edges/links/lines. The letters in the circles uniquely represent that circle. The numbers on the lines represent how much it will cost to move along that line.

APPLICATION OF GRAPH THEORY:

- In **Computer science** graphs are used to represent the flow of computation.
- **Google maps** uses graphs for building transportation systems, where intersection of two (or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
- In **Facebook**, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook's Friend suggestion algorithm uses graph theory. Facebook is an example of **undirected graph**.
- In **World Wide Web**, web pages are considered to be the vertices. There is an edge from a page 'u' to another page 'v' if there is a link of page 'v' on page 'u'. This is an example of **Directed graph**. It was the basic idea behind Google Page Ranking Algorithm.
- In **Operating System**, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.

BREADTH FIRST SEARCH ALGORITHM:

Breadth first search is one of the basic and essential searching algorithms on graphs. Breadth-first search (BFS) is an important graph search algorithm that is used to solve many problems including finding the shortest path in a graph and solving puzzle games (such as Rubik's Cubes). Many problems in computer science can be thought of in terms of graphs. For example, analysing networks, mapping routes, and scheduling are graph problems. Graph search algorithms like breadth-first search are useful for analysing and solving graph problems.

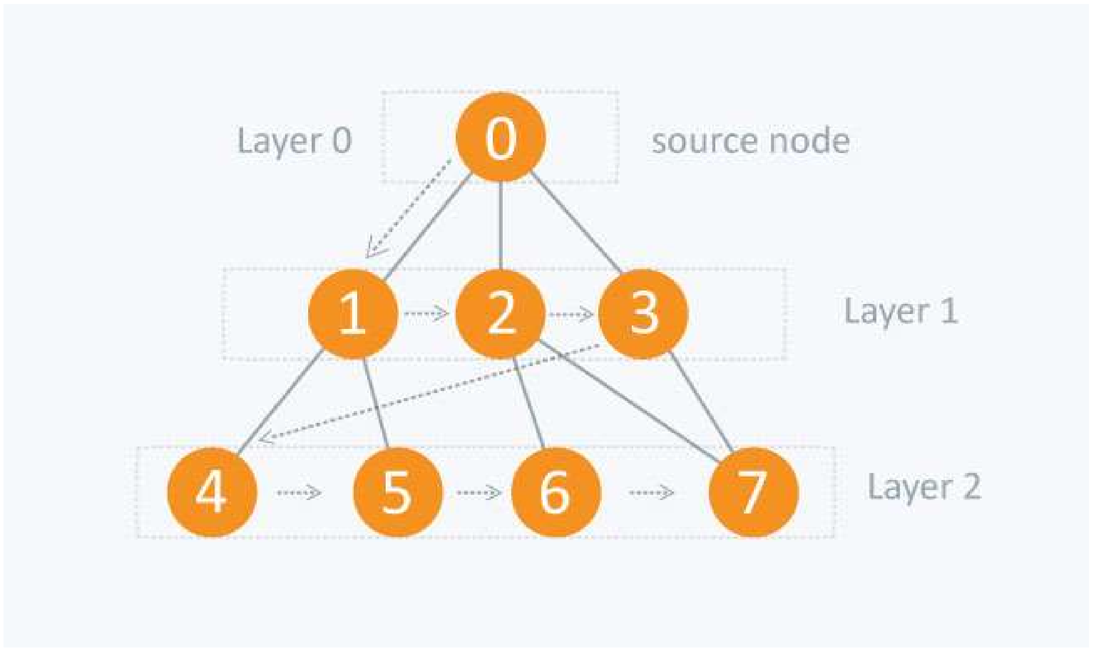
As a result of how the algorithm works, the path found by breadth first search to any node is the shortest path to that node, i.e. the path that contains the smallest number of edges in unweighted graphs.

The algorithm works in "O(n+m)" time, where 'n' is number of vertices and 'm' is the number of edges.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer



Consider the diagram given above. The distance between the nodes in layer 1 is comparatively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

The algorithm takes as input an unweighted graph and the id of the source vertex s. The input graph can be directed or undirected, it does not matter to the algorithm.

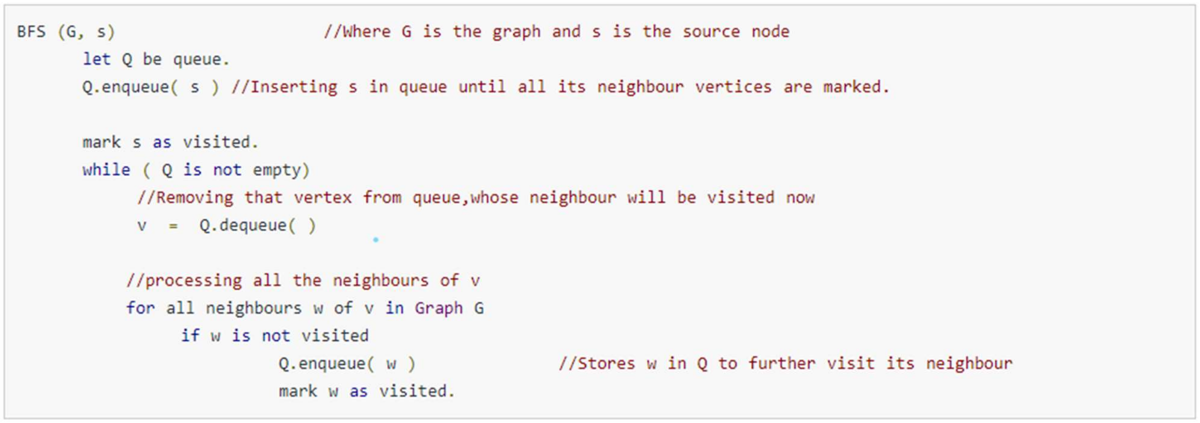
The algorithm can be understood as a fire spreading on the graph: at the zeroth step only the source s is on fire. At each step, the fire burning at each vertex spreads to all of its neighbours. In one iteration of the algorithm, the "ring of fire" is expanded in width by one unit (hence the name of the algorithm).

More precisely, the algorithm can be stated as follows: Create a queue q which will contain the vertices to be processed and a Boolean array used [] which indicates for each vertex, if it has been lit (or visited) or not.

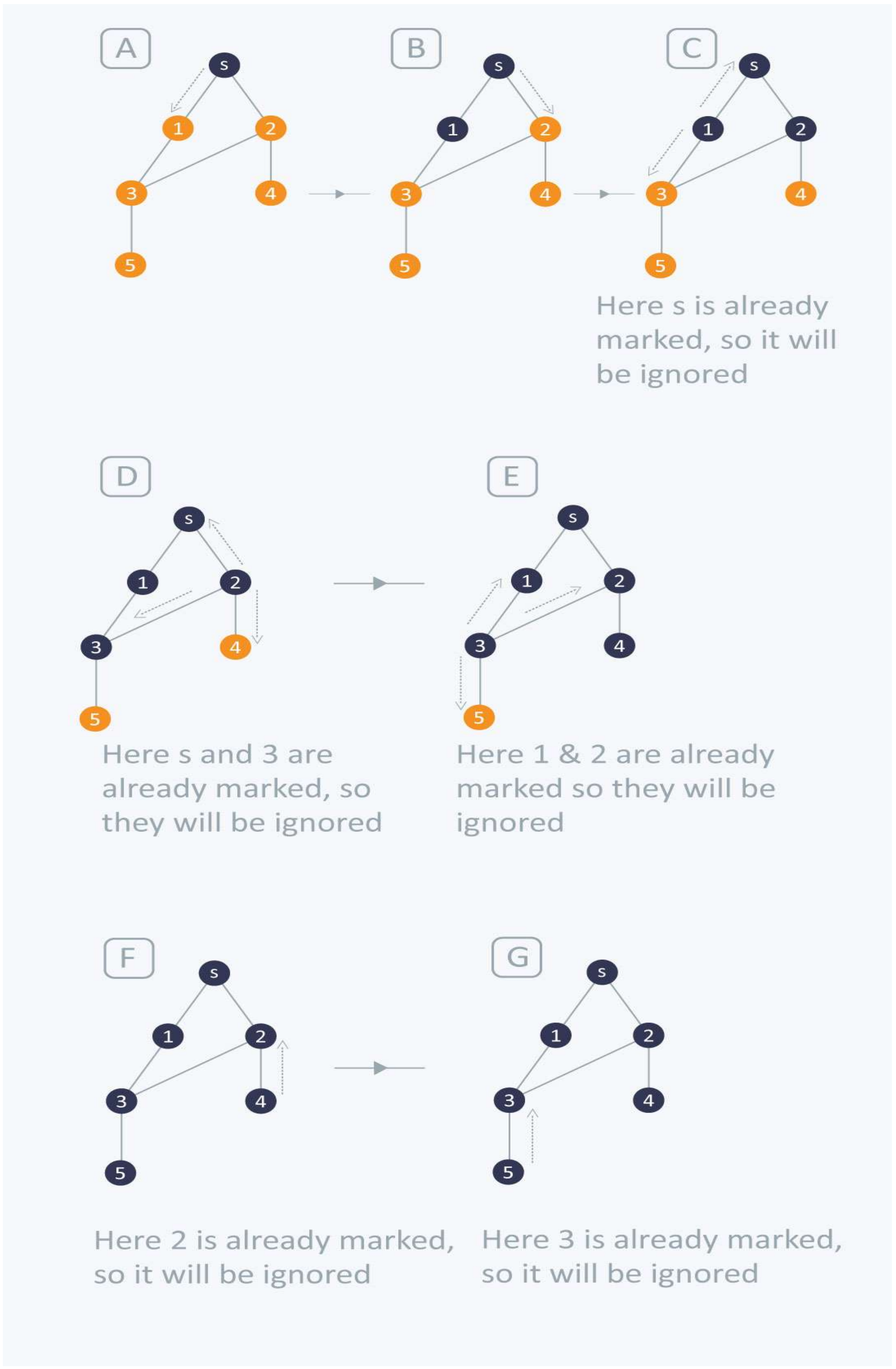
Initially, push the source s to the queue and set used[s]=true, and for all other vertices v set used[v]=false. Then, loop until the queue is empty and, in each iteration, pop a vertex from the front of the queue. Iterate through all the edges going out of this vertex and if some of these edges go to vertices that are not already lit, set them on fire and place them in the queue.

As a result, when the queue is empty, the "ring of fire" contains all vertices reachable from the source s, with each vertex reached in the shortest possible way. You can also calculate the lengths of the shortest paths (which just requires maintaining an array of path lengths d[ ]) as well as save information to restore all of these shortest paths (for this, it is necessary to maintain an array of "parents" p[ ], which stores for each vertex the vertex from which we reached it).

PSEUDOCODE:



```
BFS (G, s)                    //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

      mark s as visited.
      while ( Q is not empty)
           //Removing that vertex from queue,whose neighbour will be visited now
           v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                        Q.enqueue( w )             //Stores w in Q to further visit its neighbour
                        mark w as visited.
```



Here s is already marked, so it will be ignored

Here s and 3 are already marked, so they will be ignored

Here 1 & 2 are already marked so they will be ignored

Here 2 is already marked, so it will be ignored

Here 3 is already marked, so it will be ignored

TRANSVERSING PROCESS:

The traversing will start from the source node and push *s* in queue. *s* will be marked as 'visited'.

First iteration:

- s will be popped from the queue
- Neighbours of s i.e. 1 and 2 will be traversed
- 1 and 2, which have not been traversed earlier, are traversed. They will be:
    - Pushed in the queue
    - 1 and 2 will be marked as visited

Second iteration:

- 1 is popped from the queue
- Neighbours of 1 i.e. s and 3 are traversed
- s is ignored because it is marked as 'visited'
- 3, which has not been traversed earlier, is traversed. It is:
    - Pushed in the queue
    - Marked as visited

Third iteration:

- 2 is popped from the queue
- Neighbours of 2 i.e. s, 3, and 4 are traversed
- 3 and s are ignored because they are marked as 'visited'
- 4, which has not been traversed earlier, is traversed. It is:
    - Pushed in the queue
    - Marked as visited

Fourth iteration:

- 3 is popped from the queue
- Neighbours of 3 i.e. 1, 2, and 5 are traversed
- 1 and 2 are ignored because they are marked as 'visited'
- 5, which has not been traversed earlier, is traversed. It is:
    - Pushed in the queue
    - Marked as visited

Fifth iteration:

- 4 will be popped from the queue
- Neighbours of 4 i.e. 2 is traversed
- 2 is ignored because it is already marked as 'visited'

Sixth iteration:

- 5 is popped from the queue
- Neighbours of 5 i.e. 3 is traversed
- 3 is ignored because it is already marked as 'visited'

The queue is empty and it comes out of the loop. All the nodes have been traversed by using BFS.

<u>**APPLICATIONS:**</u>

1. Find the shortest path from a source to other vertices in an unweighted graph.

2. Find all connected components in an undirected graph in O(n+m) time: To do this, we just run BFS starting from each vertex, except for vertices which have already been visited from previous runs. Thus, we

perform normal BFS from each of the vertices, but do not reset the array used[] each and every time we get a new connected component, and the total running time will still be O(n+m) (performing multiple BFS on the graph without zeroing the array used[] is called a series of breadth first searches).

3. Finding a solution to a problem or a game with the least number of moves, if each state of the game can be represented by a vertex of the graph, and the transitions from one state to the other are the edges of the graph.

4. Finding the shortest path in a graph with weights 0 or 1: This requires just a little modification to normal breadth-first search: Instead of maintaining array used[], we will now check if the distance to vertex is shorter than current found distance, then if the current edge is of zero weight, we add it to the front of the queue else we add it to the back of the queue. This modification is explained in more detail in the article 0-1 BFS.

5. Finding the shortest cycle in a directed unweighted graph: Start a breadth-first search from each vertex. As soon as we try to go from the current vertex back to the source vertex, we have found the shortest cycle containing the source vertex. At this point we can stop the BFS, and start a new BFS from the next vertex. From all such cycles (at most one from each BFS) choose the shortest.

6. Find all the edges that lie on any shortest path between a given pair of vertices (a,b). To do this, run two breadth first searches: one from a and one from b. Let da[] be the array containing shortest distances obtained from the first BFS (from a) and db[] be the array containing

shortest distances obtained from the second BFS from b. Now for every edge (u,v) it is easy to check whether that edge lies on any shortest path between a and b: the criterion is the condition da[u]+1+db[v]=da[b].

7. Find all the vertices on any shortest path between a given pair of vertices (a,b). To accomplish that, run two breadth first searches: one from a and one from b. Let da[] be the array containing shortest distances obtained from the first BFS (from a) and db[] be the array containing shortest distances obtained from the second BFS (from b). Now for each vertex it is easy to check whether it lies on any shortest path between a and b: the criterion is the condition da[v]+db[v]=da[b].

8. Find the shortest path of even length from a source vertex s to a target vertex t in an unweighted graph: For this, we must construct an auxiliary graph, whose vertices are the state (v,c), where v - the current node, c=0 or c=1 - the current parity. Any edge (a,b) of the original graph in this new column will turn into two edges ((u,0),(v,1)) and ((u,1),(v,0)). After that we run a BFS to find the shortest path from the starting vertex (s,0) to the end vertex (t,0).

A* ALGORITHM:

A* (pronounced as "A star") is a computer algorithm that is widely used in pathfinding and graph traversal. The algorithm efficiently plots a walkable path between multiple nodes, or points, on the graph.

On a map with many obstacles, pathfinding from points AA to BB can be difficult. A robot, for instance, without getting much other direction, will continue until it encounters an obstacle, as in the path-finding example to the left below.

However, the A* algorithm introduces a heuristic into a regular graph-searching algorithm, essentially planning ahead at each step so a more optimal decision is made. With A*, a robot would instead find a path in a way similar to the diagram on the right below.

A* is an extension of Dijkstra's algorithm with some characteristics of breadth-first search (BFS).

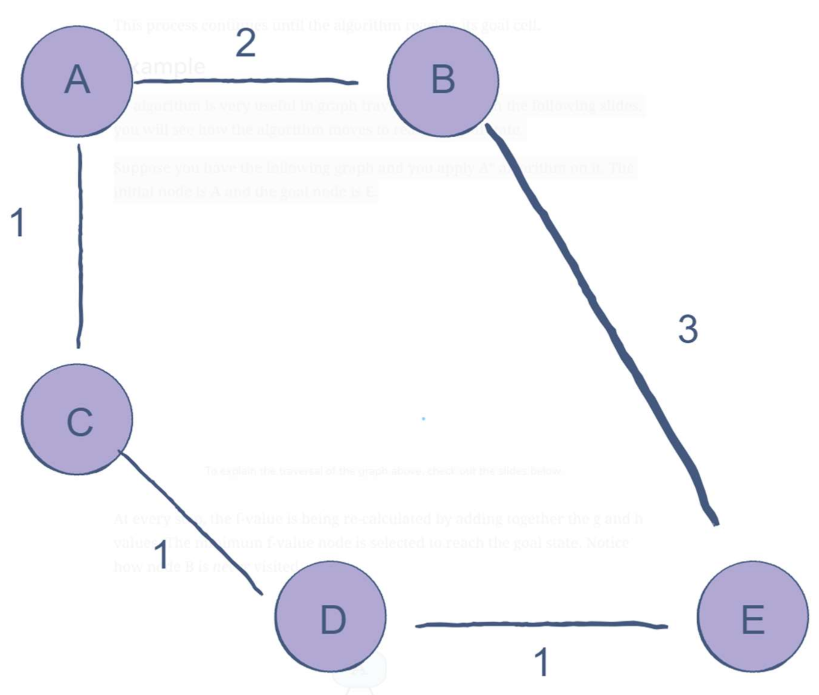
A* algorithm has 3 parameters:

- **g** : the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
- **h** : also known as the *heuristic value,* it is the **estimated** cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We **must** make sure that there is **never** an over estimation of the cost.
- **f** : it is the sum of g and h. So, **f = g + h**

The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the *smallest f-valued cell* and moves to that cell. This process continues until the algorithm reaches its goal cell.

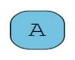
A* algorithm is very useful in graph traversals as well. In the following slides, you will see how the algorithm moves to reach its goal state.

Suppose you have the following graph and you apply A* algorithm on it. The initial node is **A** and the goal node is **E**.

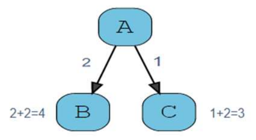


At every step, the f-value is being re-calculated by adding together the g and h values. The minimum f-value node is selected to reach the goal state. Notice how node B is *never* visited.

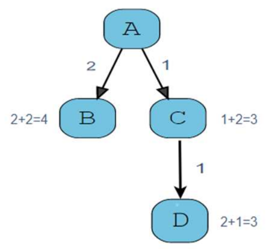
Example:



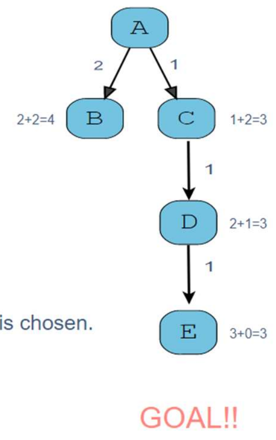
Root node A.



Node C is chosen.



Node D is chosen.



Node E is chosen.

STEPS INVOLVED:

1. Add the starting square (or node) to the open list.

2. Repeat the following:

A) Look for the lowest F cost square on the open list. We refer to this as the current square.

B) Switch it to the closed list.

C) For each of the 8 squares adjacent to this current square …

- If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
- If it isn't on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
- If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If you are keeping your open list sorted by F score, you may need to resort the list to account for the change.

D) Stop when you:

- Add the target square to the closed list, in which case the path has been found, or
- Fail to find the target square, and the open list is empty. In this case, there is no path.

3. Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path.

PSUEDOCODE:

```
// A* (star) Pathfinding

// Initialize both open and closed list
let the openList equal empty list of nodes
let the closedList equal empty list of nodes

// Add the start node
put the startNode on the openList (leave it's f at zero)
```

```
// Loop until you find the end
while the openList is not empty


    // Get the current node
    let the currentNode equal the node with the least f value
    remove the currentNode from the openList
    add the currentNode to the closedList


    // Found the goal
    if currentNode is the goal
        Congratz! You've found the end! Backtrack to get path


    // Generate children
    let the children of the currentNode equal the adjacent nodes

    for each child in the children


        // Child is on the closedList
        if child is in the closedList
            continue to beginning of for loop


        // Create the f, g, and h values
        child.g = currentNode.g + distance between child and current
        child.h = distance from child to end
        child.f = child.g + child.h


        // Child is already in openList
        if child.position is in the openList's nodes positions
            if the child.g is higher than the openList node's g
                continue to beginning of for loop


        // Add the child to the openList
        add the child to the openList
```

APPLICATION:

A-star (A*) is a mighty algorithm in Artificial Intelligence with a wide range of usage. However, it is only as good as its heuristic function(which can be highly variable considering the nature of a problem). A* is the most popular choice for pathfinding because it's reasonably flexible.

It has found applications in many software systems, from Machine Learning and search Optimization to game development.

PYGAME:

Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language.

Pygame is highly portable and runs on nearly every platform and operating system.

Pygame itself has been downloaded millions of times.

Pygame is free. Released under the LGPL licence, you can create open source, freeware, shareware, and commercial games with it. See the licence for full details.

Multi core CPUs can be used easily. With dual core CPUs common, and 8 core CPUs cheaply available on desktop systems, making use of multi core CPUs allows you to do more in your game. Selected pygame functions release the dreaded python GIL, which is something you can do from C code.

Uses optimized C and Assembly code for core functions. C code is often 10-20 times faster than python code, and assembly code can easily be 100x or more times faster than python code.

Comes with many operating systems. Just an apt-get, emerge, pkg_add, or yast install away. No need to mess with installing it outside of your operating system's package manager. Comes with binary pos system installers (and uninstallers) for Windows or MacOSX. Pygame does not require setup tools with even ctypes to install.

Truly portable. Supports Linux (pygame comes with most main stream linux distributions), Windows (95, 98, ME, 2000, XP, Vista, 64-bit Windows, etc), Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX. The code contains support for AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS and OS/2, but these are not officially supported. You can use it on hand held devices, game consoles and the One Laptop Per Child (OLPC) computer.

It's Simple and easy to use. Kids and adults make shooter games with pygame. Pygame is used in the OLPC project and has been taught in essay courses to young kids and college students. It's also used by people who first programmed in z80 assembler or c64 basic.

Many games have been published. Including Indie Game Festival finalists, Australian Game festival finalists, popular shareware, multimedia projects and open source games. Over 660 projects have been published on the pygame websites such as: list needed. Many more games have been released with SDL (which pygame is based on), so you can be sure much of it has been tested well by millions of users.

You control your main loop. You call pygame functions, they don't call your functions. This gives you greater control when using other libraries, and for different types of programs.

Does not require a GUI to use all functions. You can use pygame from a command line if you want to use it just to process images, get joystick input, or play sounds.

Fast response to reported bugs. Some bugs are patched within an hour of being reported. Do a search on our mailing list for BUG... you'll see for yourself. Sometimes we suck at bug fixes, but mostly we're pretty good bug fixers. Bug reports are quite rare these days, since a lot of them have been fixed already.

Small amount of code. It does not have hundreds of thousands of lines of code for things you won't use anyway. The core is kept simple, and extra things like GUI libraries, and effects are developed separately outside of pygame.

Modular. You can use pieces of pygame separately. Want to use a different sound library? That's fine. Many of the core modules can be initialized and used separately.

General Constraints:

- **Technology Constraints**: Proposed application will be implemented using python programming language and using pygame library.
- **Interface Constraints**: It should work on operating system such as Linux, Windows etc. Python must be installed on these systems.
- **Safety and Security Constraints**: Since, application is intended for the authenticated users only, so anonymous person should not be able to access and operate over the user data.

Non-functional Requirements:

- **Performance Requirements:**
  The application should be able to operate on all operating systems with all of its fundamental functions. It should not slow-down the system even at peak hours without affecting the quality of service of the system.

FEASIBILITY STUDY

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below:

- To analyze whether the software will meet organizational requirements.
- To determine whether the software can be implemented using the current technology and within the specified budget and schedule.
- To determine whether the software can be integrated with other existing software.

<u>Technical feasibility:</u>

It assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, the software development team ascertains whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements. Technical feasibility also performs the following tasks:

- Analyses the technical skills and capabilities of the software development team members.
- Determines whether the relevant technology is stable and established.
- Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

Operational feasibility:

It assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources (software development team) and involves visualizing whether the software will operate after it is developed and be operative once it is installed. Operational feasibility also performs the following tasks:

- Determines whether the problems anticipated in user requirements are of high priority.
- Determines whether the solution suggested by the software development team is acceptable.
- Analyses whether users will adapt to a new software.
- Determines whether the organization is satisfied by the alternative solutions proposed by the software development team.

Economic feasibility:

It determines whether the required software is capable of generating financial gains for an organization. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on. For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition, it is necessary to consider the benefits that can be achieved by developing the software. Software is said to be economically feasible if it focuses on the issues listed below:

- Cost incurred on software development to produce long-term gains for an organization.
- Cost required to conduct full software investigation (such as requirements elicitation and requirements analysis).
- Cost of hardware, software, development team, and training.

SOFTWARE ENGINEERING MODEL

Software engineering is a technological discipline that combines the concepts of computer science, economics, communication skills, and management science with the problem-solving approach of engineering. It also involves a standardized approach to program development, both in its managerial and technical aspects.

The profound knowledge of computer science both theoretical and practical forms the basis of software engineering. The theoretical knowledge provides an understanding of which problems are resolvable, what data structures and algorithms are appropriate, when and how they are to be used, etc. On the other hand, the practical knowledge provides an understanding of how hardware functions, how to utilize the power of programming languages and tools while developing software, etc.

One of the main objectives of software engineering is to help developers obtain high quality software. This quality is achieved through use of Total Quality Management (TQM), which enables continuous process improvement custom that leads to the development of more established approaches to software engineering.

<u>Basic Objective:</u>

Software engineering is the systematic approach to the development, operation, maintenance and retirement of software. Software Engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentations.

The basic objective of software engineering is to develop methods and procedures for software development that can scale up for large systems and that can be used consistently to produce high-quality software at low cost and with a small cycle of time.

Need for Software Engineering:

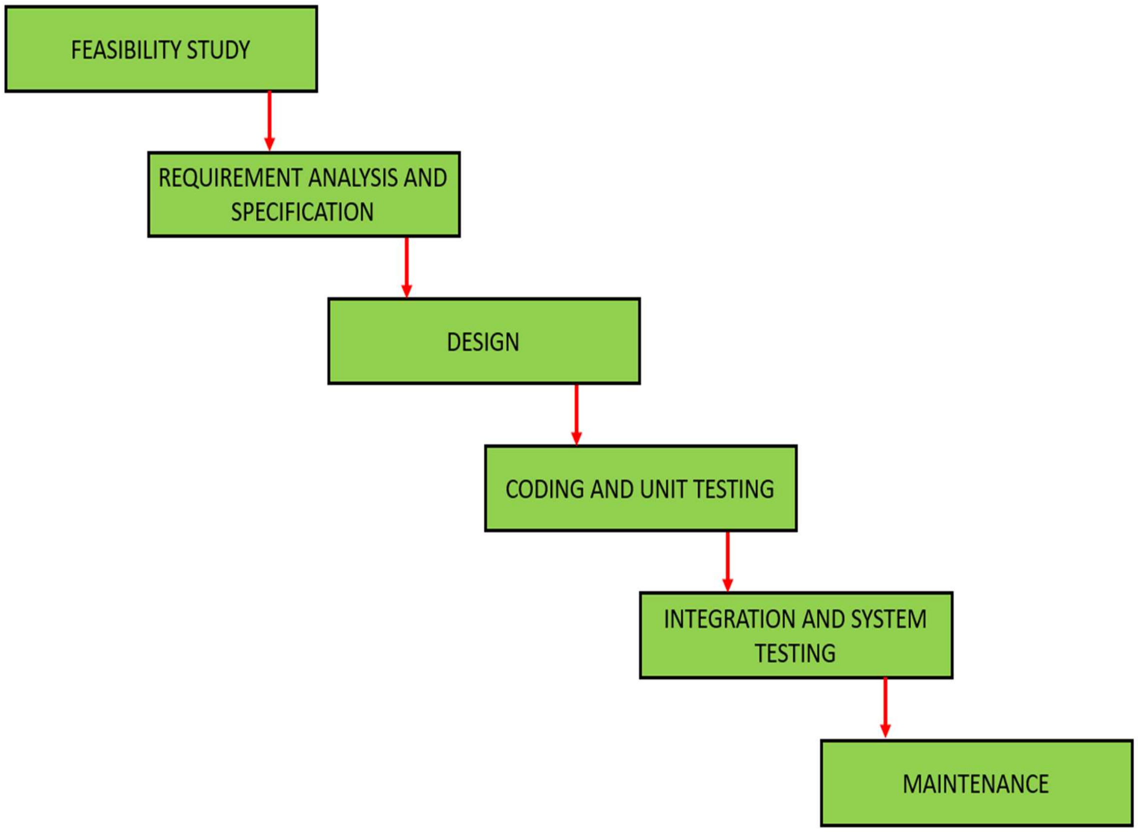
- As Software development is expensive so proper measures are required so that the resources are used efficiently and effectively.
- Cost and time considerations are another factor, which arises the need for Software Engineering.
- Reliability factors

In this project, **waterfall model** is used for the development. The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.

Classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus, the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:


FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility study involves understanding the problem and then determine the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, the best solution is chosen and all the other phases are carried out as per this solution strategy.

2. **Requirements analysis and specification**: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. It includes:

    - **Requirement gathering and analysis:** Firstly, all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (inconsistent requirement is one in which some part of the requirement contradicts with some other part).
    - **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. **Design**: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

4. **Coding and Unit testing**: In coding phase software design is translated into source code using any suitable programming language. Thus, each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant

system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists three different kinds of testing activities as described below:

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.

6. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance:

- **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
- **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model:

Classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered as the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well understood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before- design,
  design-before-code.
- This model works well for smaller projects and projects where requirements are well
  understood.

Drawbacks of Classical Waterfall Model:

Classical waterfall model suffers from various shortcomings, basically we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

- **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.

- **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers' requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.

- **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase the efficiency and reduce the cost, phases may overlap.

SYSTEM DESIGN

System design is the process of designing the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system. Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software. Initially, the blueprint depicts a holistic view of software. In other words, the design is represented at a high level of abstraction and after that slowly it is converted into a detailed design.

System design activities include the allocation of resources to equipment tasks, personnel tasks, and computer program tasks. In the design phase, the technical specifications are prepared for the performance of all allocated tasks. This phase is sometimes split into sub phases- high level design and detailed design. High-level design specifies overall module structure and how they interact with each other to produce the desired results.

During detailed design, the internal logic of each of modules is decided. During this step of system development data structures and procedural logic of each modules are designed. Data structures are designed with the help of Entity-Relationship Diagram and modules are identified with the help of Data Flow Diagram. The logic of a module is usually specified in a high-level design description language, which is independent of the target language in which the software will eventually be implemented.

The purpose of the System Design process is to provide sufficient detailed data and information about the system and its system elements to enable the implementation consistent with architectural entities as defined in models and views of the system architecture.

Elements of a System:

- **Architecture** – This is the conceptual model that defines the structure, behavior and more views of a system. We can use flowcharts to represent and illustrate the architecture.
- **Modules –** This are components that handle one specific tasks in a system. A combination of the modules makes up the system.
- **Components –** This provides a particular function or group of related functions. They are made up of modules.
- **Interfaces –** This is the shared boundary across which the components of the system exchange information and relate.
- **Data –** This the management of the information and data flow.

Major Tasks Performed During the System Design Process:

1. **Initialize design definition:**

- Plan for and Identify the technologies that will compose and implement the systems elements and their physical interfaces.
- Determine which technologies and system elements have a risk to become obsolete, or evolve during the operation stage of the system. Plan for their potential replacement.
- Document the design definition strategy, including the need for and requirements of any enabling systems, products, services to perform design.

2. Establish design characteristics:

- Define the design characteristics relating to the architectural characteristics and check that they are implementable.

- Define the interfaces that were not defined by the System Architecture process or that need to be refined as the design details evolve.
- Define and document the design characteristics of each system element2.

3. **Assess alternatives for obtaining system elements:**

- Assess the design options
- Select the most appropriate alternatives.
- If the decision is made to develop the system element, rest of the design definition process and the implementation process are used. If the decision is to buy or reuse a system element, the acquisition process may be used to obtain the system element.

4. **Manage the design:**

- Capture and maintain the rationale for all selections among alternatives and decisions for the design, architecture characteristics.
- Assess and control the evolution of the design characteristics.

SOURCE CODE

STARTING.PY

```
import pygame
from astar import Astar
import time
from bfs import Bfs
class Game(Astar,Bfs):

    def __init__(self):
        Astar.__init__(self)
        Bfs.__init__(self)

        pygame.init()
        self.display_width = 1020+2+200
        self.display_height = 570+2+300-210
        self.screen = pygame.display.set_mode((self.display_width,self.display
_height))
        self.clock = pygame.time.Clock()
        pygame.display.set_caption(u'Path Finding Visualizer')
        self.width = 34
        self.height =22

        self.walls = list()
        self.maze = [[]]

        self.white = (255,255,255)
        self.red = (255,0,0)
        self.less_red = (150,0,0)
        self.blue = (0,255,0)
        self.less_blue = (0,150,0)
        self.green = (0,0,255)
        self.less_green = (0,0,150)
        self.black = (0,0,0)
        self.grid_color = (0,150,255)

    def text_objects(self,message,font):
        textSurface = font.render(message,False,self.white)
        return textSurface,textSurface.get_rect()

    def button(self,msg,x,y,w,h,ic,ac,action=None):
        mouse = pygame.mouse.get_pos()
        click = pygame.mouse.get_pressed()

        pygame.draw.rect(self.screen,ic,(x,y,w,h))
        if x+w > mouse[0] > x and y+h > mouse[1] > y:
                pygame.draw.rect(self.screen,ac,(x,y,w,h))
                if click[0] == 1 and action != None:
                    time.sleep(0.5)
                    action()

        font = pygame.font.SysFont("freesansbold.ttf",25)
        textSurface ,textRect = self.text_objects(msg,font)
        textRect.center = ( (x+(w/2)), (y+(h/2)) )
        self.screen.blit(textSurface, textRect)


    def welcome_screen(self):

        game_exit = False

        while not game_exit:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    game_exit = True
                    pygame.quit()


            message = ['Welcome to this game, Samarth is lonely as fuck','all
he know are different pathfinding algorithms,','he will use different algorihm
s to find his girlfriend']
            width_counter = 0
            for message in message:
                text = pygame.font.Font('freesansbold.ttf',30)
                TextSurf, TextRect = self.text_objects(message,text)
                TextRect.center = ((self.display_width/2),(self.display_height
/2-150+width_counter))
                width_counter+=30
                self.screen.blit(TextSurf, TextRect)

                self.button("G O !",500,450,100,50,self.red,self.less_red,self
.construct_maze)

                pygame.display.update()
                self.clock.tick(60)

    def construct_maze(self):
        game_exit = False
        draw_grid = False
        self.screen.fill(self.black)
        while not game_exit:
            self.button("Start Node='s'",1025,50,190,50,self.red,(0,0,50),acti
on = None)
            self.button("end Node='e'",1025,125,190,50,self.red,(0,0,50),actio
n = None)
```

```
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    gameExit = True

                elif event.type == pygame.KEYDOWN:
                    x,y = pygame.mouse.get_pos()
                    x = x//30
                    y = y//30
                    if event.key == pygame.K_s:
                        pygame.draw.rect(self.screen,self.red,(x*30,y*30,29,29
))

                        self.start_node = (x,y)
                    if event.key == pygame.K_e:
                        pygame.draw.rect(self.screen,self.green,(x*30,y*30,29,
29))

                        self.end_node = (x,y)

            if not draw_grid:
                for x in range(0,1020+30,30):
                    pygame.draw.line(self.screen,self.grid_color,(0,x),(1020,x
),2) #horizontal line

                    pygame.draw.line(self.screen,self.grid_color,(x,0),(x,1020
), 2)

                    self.clock.tick(60)
                    pygame.display.update()
                    draw_grid = True




            x,y = pygame.mouse.get_pos()
            x = x//30
            y = y//30
            if pygame.mouse.get_pressed()[0] and x*30<1020 and y*30<400+8*30:

                pygame.draw.rect(self.screen,self.grid_color,(x*30,y*30,29,29)
)

                if (x,y) not in self.walls:
                    self.walls.append((x,y))

            self.maze = [[0 for i in range(self.height)]for j in range(self.wi
dth)]
            for x in self.walls:
                self.maze[x[0]][x[1]] = 1

            self.button("Astar",1025,250,190,50,self.less_blue,(0,0,50),action
 = self.animate_astar)
```

```
            self.button("BFS",1025,350,190,50,self.less_blue,(0,0,50),action =
 self.bfs)
            self.button("Reset",1025,500,190,50,self.less_blue,(0,0,50),action
 = start_game)

            pygame.display.update()

    def animate_astar(self):
        a,path =Astar.search(self,self.maze,1,self.start_node,self.end_node)

        for a in a:

            pygame.draw.rect(self.screen,self.less_red,(a.position[0]*30,a.pos
ition[1]*30,29,29))
            pygame.display.update()
            self.clock.tick(60)

        draw = []
        for i,j in enumerate(path):
            for k,l in enumerate(j):
                if l is not -1:
                    draw.append((i,k))

        for i in draw[::-1]:


            pygame.draw.rect(self.screen,self.red,(i[0]*30,i[1]*30,29,29))
            self.clock.tick(200)
            pygame.display.update()

    def bfs(self):
        a,path = Bfs.search(self,self.maze,self.start_node,self.end_node)
        for a in a:

            pygame.draw.rect(self.screen,self.less_red,(a[0]*30,a[1]*30,29,29)
)
            pygame.display.update()
            self.clock.tick(60)

        draw = []
        for i,j in enumerate(path):
            for k,l in enumerate(j):
                if l is not -1:
                    draw.append((i,k))

        for i in draw[::-1]:

            pygame.draw.rect(self.screen,self.red,(i[0]*30,i[1]*30,29,29))
            self.clock.tick(200)
            pygame.display.update()

def start_game():
    try:
        a = Game()
        a.construct_maze()
    except:
        print(":) awwww, snap!")

start_game()
```

BFS.PY

```
class Node:


    def __init__(self, parent=None, position=None):

        self.parent = parent
        self.position = position


    def __eq__(self,other):
        return self.position == other.position

class Bfs():
    def __init__(self):
        self.visited = []
        self.queue = []


    def search(self,maze,start,end):
        start_node = Node(None,tuple(start))
        end_node = Node(None,tuple(end))


        self.queue.append(start_node)

        no_rows = len(maze)
        no_columns = len(maze[0])

        move = [[-1, 0 ], [ 0, -1], [ 1, 0 ],[ 0, 1 ]]

        while (len(self.queue)>0):
            children = []
            current_node = self.queue.pop(0)

            if current_node.position not in self.visited:

                self.visited.append(current_node.position)


            for new_position in move:
                node_position = (current_node.position[0] + new_position[0], c
urrent_node.position[1] + new_position[1])
```

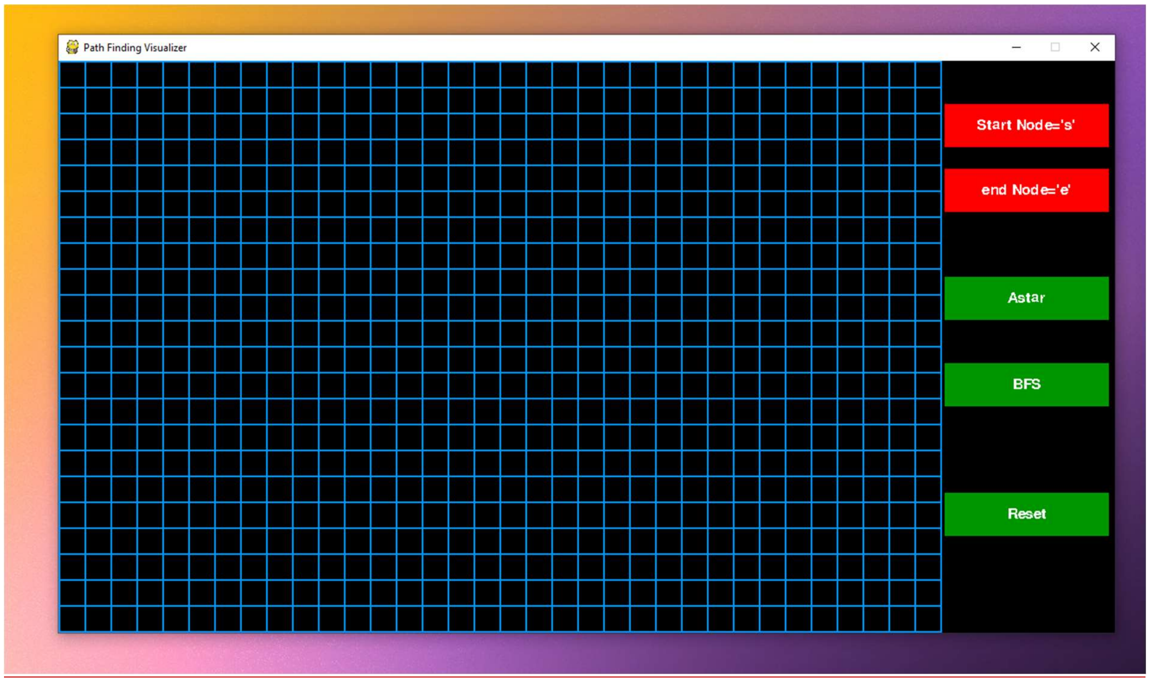
```
            if (node_position[0] > (no_rows - 1) or
                node_position[0] < 0 or
                node_position[1] > (no_columns -1) or
                node_position[1] < 0):
                continue
            if maze[node_position[0]][node_position[1]] != 0:
                continue
            if node_position in self.visited:
                continue

            new_node = Node(current_node,node_position)




            if new_node not in self.queue:
                self.queue.append(new_node)




            if new_node.position == end_node.position:
                return self.return_path(self.visited,current_node,maze)
def return_path(self,visited,current_node,maze):
    path = []
    no_rows = len(maze)
    no_columns = len(maze[0])
    result = [[-1 for i in range(no_columns)] for j in range(no_rows)]
    current = current_node
    while current is not None:
        path.append(current.position)
        current = current.parent
    path = path[::-1]
    start_value = 0
    for i in range(len(path)):
        result[path[i][0]][path[i][1]] = start_value
        start_value += 1
    return visited,result
```

ASTAR.PY

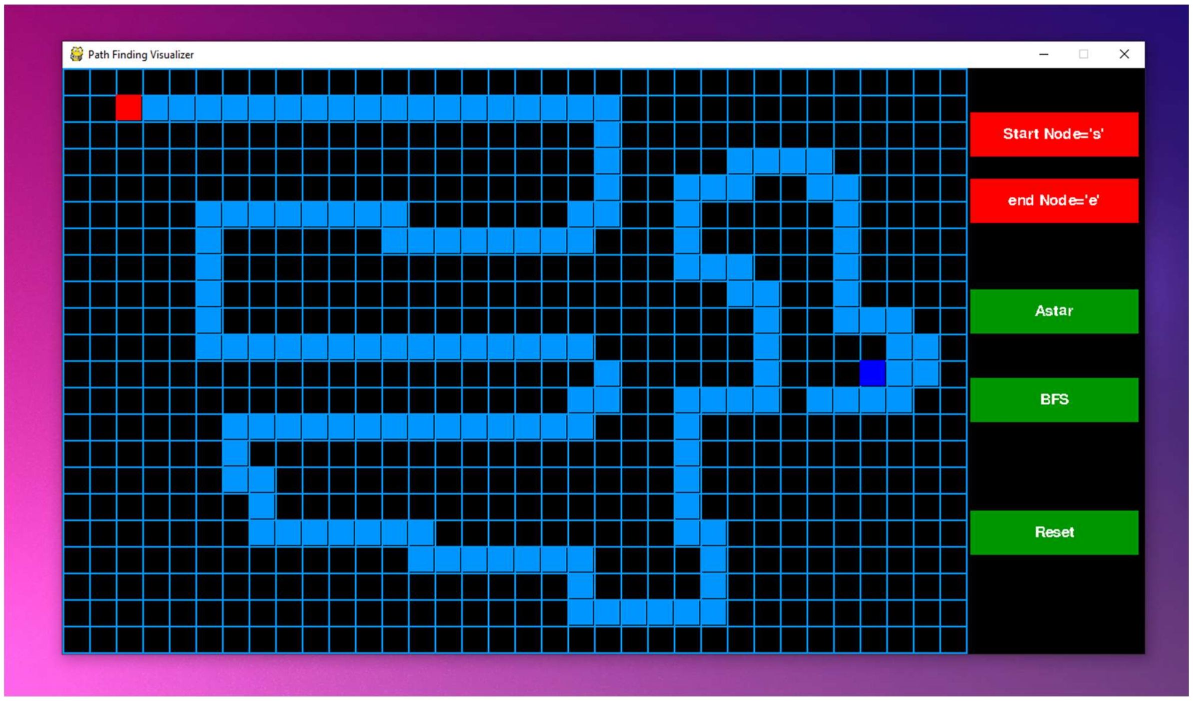
```
class Node:


    def __init__(self, parent=None, position=None):

        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self,other):
        return self.position == other.position


class Astar():

    def __init__(self):
        self.visited = []


    def search(self,maze,cost,start,end):
        start_node = Node(None, tuple(start))
        start_node.g = start_node.h = start_node.f = 0
        end_node = Node(None, tuple(end))
        end_node.g = end_node.h = end_node.f = 0

        yet_to_visit_list = []
        visited_list = []

        yet_to_visit_list.append(start_node)

        outer_iterations = 0
        max_iterations = (len(maze) // 2) ** 10

        move  =  [[-1, 0 ], # go up
                  [ 0, -1], # go left
                  [ 1, 0 ], # go down
                  [ 0, 1 ]] # go right
        no_rows = len(maze)
        no_columns = len(maze[0])
        while len(yet_to_visit_list) > 0:
            outer_iterations += 1
            current_node = yet_to_visit_list[0]
            current_index = 0

            for index, item in enumerate(yet_to_visit_list):
                if item.f < current_node.f:
                    current_node = item
                    current_index = index

            if outer_iterations > max_iterations:
                print ("giving up on pathfinding too many iterations")
                return self.return_path(self.visited,current_node,maze)

            yet_to_visit_list.pop(current_index)
            visited_list.append(current_node)

            if current_node == end_node:
                return self.return_path(self.visited,current_node,maze)

            children = []

            for new_position in move:
                node_position = (current_node.position[0] + new_position[0], c
urrent_node.position[1] + new_position[1])

                if (node_position[0] > (no_rows - 1) or
                    node_position[0] < 0 or
                    node_position[1] > (no_columns -1) or
                    node_position[1] < 0):
                    continue

                if maze[node_position[0]][node_position[1]] != 0:
                    continue

                new_node = Node(current_node, node_position)

                children.append(new_node)
                self.visited.append(new_node) #use this to animate

            for child in children:
                if len([visited_child for visited_child in visited_list if vis
ited_child == child]) > 0:
                    continue

                child.g = current_node.g + cost
                child.h =2* int(abs(child.position[0] - end_node.position[0]))
 + int(abs(child.position[1] - end_node.position[1]))
```

```
                # child.h = (((child.position[0] - end_node.position[0]) ** 2)
 +
                #             ((child.position[1] - end_node.position[1]) ** 2
))
                print(child.h)

                child.f = child.g + child.h

                if len([i for i in yet_to_visit_list if child == i and child.g
 > i.g]) > 0:
                    continue
                if child not in yet_to_visit_list:
                    yet_to_visit_list.append(child)

    def return_path(self,visited,current_node,maze):
        path = []
        no_rows = len(maze)
        no_columns = len(maze[0])
        result = [[-1 for i in range(no_columns)] for j in range(no_rows)]
        current = current_node
        while current is not None:
            path.append(current.position)
            current = current.parent
        path = path[::-1]
        start_value = 0
        for i in range(len(path)):
            result[path[i][0]][path[i][1]] = start_value
            start_value += 1
        return visited,result
```

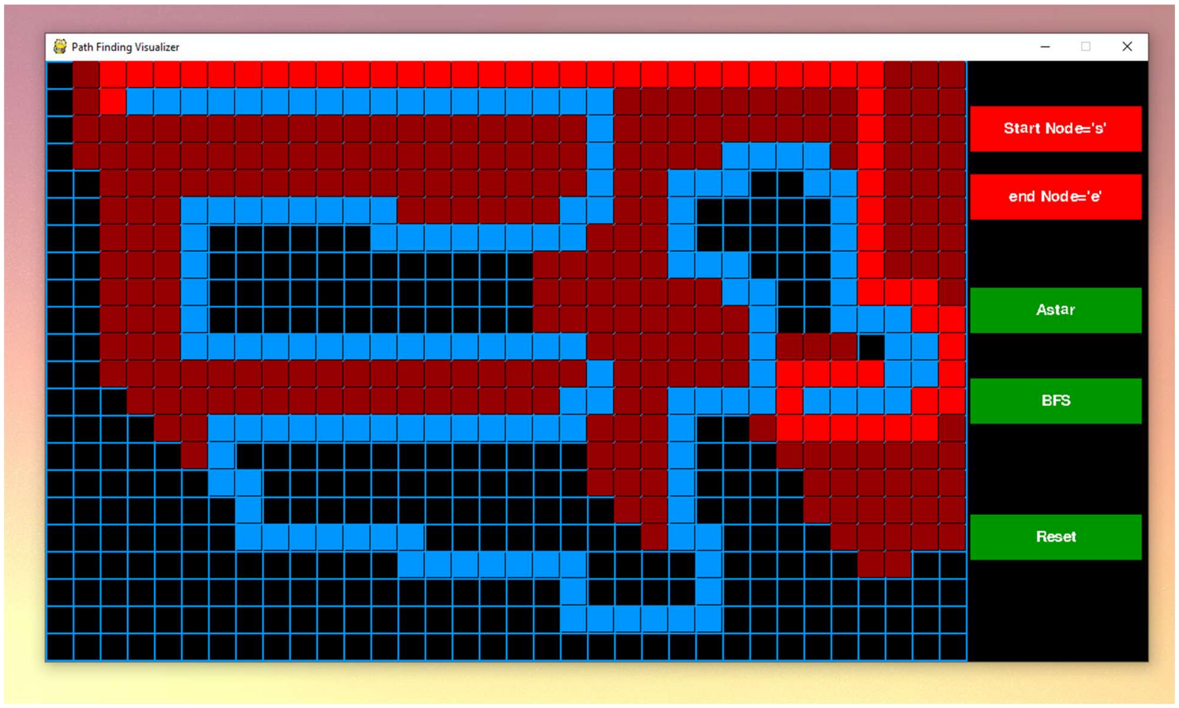
OUTPUT



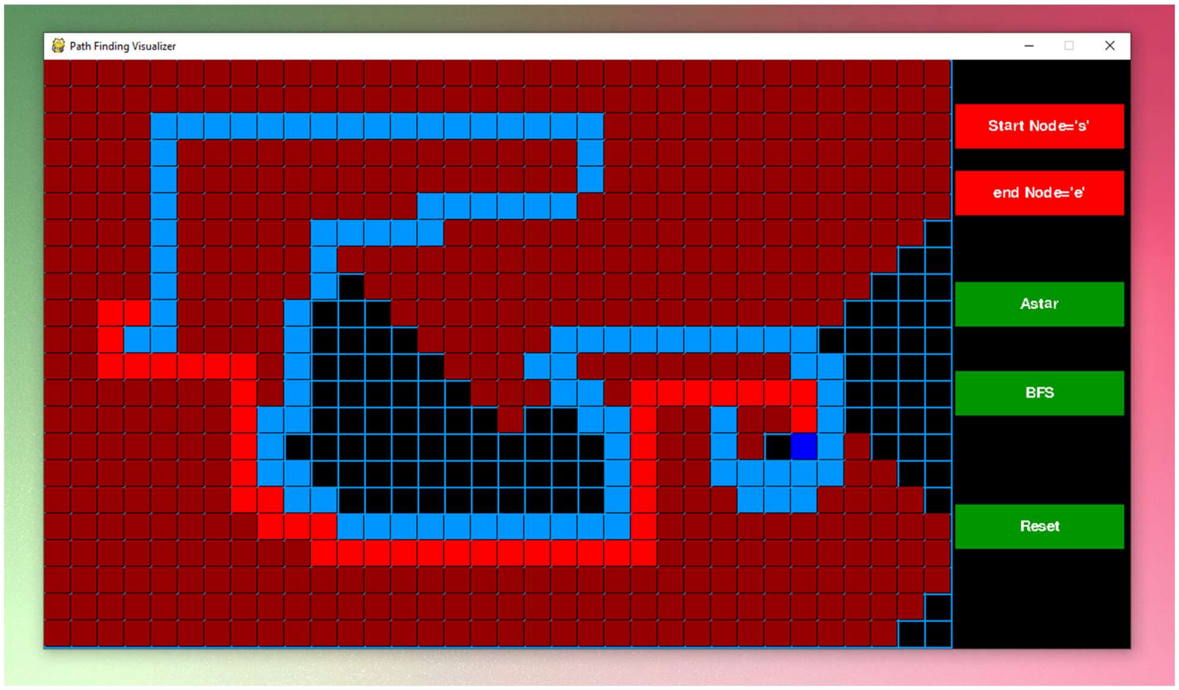
This is how the application looks life after running the source code.



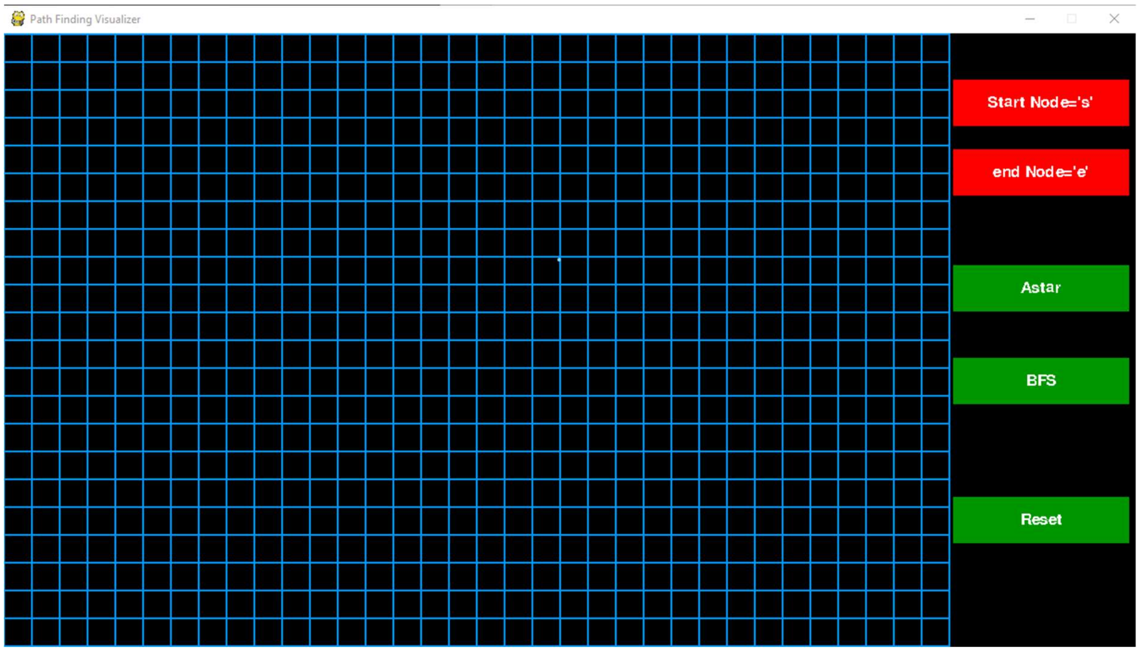
Creating the maze and setting start and end node



Shortest path found by Astar Algorithm



Shortest path found by Breadth First Algorithm



Reseting the application

TESTING

Once source code has been generated, software must be tested to uncover and correct many errors as possible before delivery to customer. Testing must be conducted to uncover errors in internal logic of software components, and in program function, behaviour and performance. Moreover, it is working according to specifications, if it's behavioural and performance requirements appear to be met. It improves the reliability and thus quality of the software.

Testing of the software starts with small components of it. After completion of the Software, it is tested as a whole. The testing that has been performed for this project are white box testing, black box testing, alpha testing and beta testing.

Principles of Testing: -

- All the test should meet the customer requirements
- To make our software testing should be performed by third party
- Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk assessment of the application.
- All the test to be conducted should be planned before implementing it
- It follows pareto rule (80/20 rule) which states that 80% of errors comes from 20% of program components.
- Start testing with small parts and extend it to large parts.

List of some common types of Software Testing:

Functional Testing types include:

- Unit Testing
- Integration Testing
- System Testing
- Sanity Testing
- Smoke Testing
- Interface Testing
- Regression Testing
- Beta/Acceptance Testing

Non-functional Testing types include:

- Performance Testing
- Load Testing
- Stress Testing
- Volume Testing
- Security Testing
- Compatibility Testing
- Install Testing
- Recovery Testing
- Reliability Testing
- Usability Testing
- Compliance Testing
- Localization Testing

TESTING METHODS USED:

WHITE BOX TESTING:

It is also called as Glass Box, Clear Box, and Structural Testing. White Box Testing is based on application's internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, is used to design test cases. This testing is usually done at the unit level.

In order to perform white-box testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

BLACK BOX TESTING:

It is also called as Behavioural/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

ALPHA AND BETA TESTING:

After performing the white and black box testing most of the errors are removed by the developer, but it is virtually impossible for him to foresee how the customer will actually use a program. Instruction for uses may be misinterpreted; strange combination of data may be regularly used, output that seemed clear to the tester may be unintelligible for the user in the field. Alpha and beta testing testing is performed when there are many users of the software to ensure that the customers/users are fully satisfied. A customer conducts alpha test at the developer's site. Developer note downs the errors encountered during the use of the software. Alpha tests are conducted in a controlled environment. Beta-test is conducted at one or more customer sites by the end user of the software. Unlike alpha testing, the developer is not present here.

In this project the unit testing is performed after the development of each component i.e. a page. Unit testing of interface, local data structure, boundary condition, independent paths and error handling paths. Unit testing makes heavy use of white box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection. After completion of development of all web pages black box testing is performed to confirm that none of the categories of errors mentioned earlier while discussing black box testing, exists in the developed software/website. After this alpha and beta testing have been conducted.

The validation checks have been applied to all the entry forms are tested if they are working or not by the following test cases.

UNIT TESTING:

It is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

IMPLEMENTATION

After designing and testing, the system is required to be implemented. The implementation phase includes installation of hardware and software, training the user and conversion to new system.

Conversion means replacement of old system by new system. The objective is to put the tested system into operation while holding costs, risks and customer's irritation to minimum.

Before implementation of the new system following tasks should be performed:

- Conversion begins with a review of project plan, the system test documentation and the implementation plan.
- The conversion portion of implementation plan is finalized and approved.
- Files are converted.
- Parallel processing between the existing system and the new system is initiated.
- Results of computer runs and operation for the new system are logged on a special form.
- Assuming no problem, parallel processing is discontinued. Implementation results are documented for reference.
- Conversion is completed. Plans for the post implementation review are prepared.
- Following the review, the new system is officially operational.
- For the purpose of training a user manual has been provided, it is an item that must accompany every system.

MAINTAINANCE

Software maintenance is a part of Software Development Life Cycle. Its main purpose is to modify and update software application after delivery to correct faults and to improve performance. Software is a model of the real world. When the real-world changes, the software requires alteration wherever possible.

Software maintenance is a vast activity which includes optimization, error correction, deletion of discarded features and enhancement of existing features. Since these changes are necessary, a mechanism must be created for estimation, controlling and making modifications. The essential part of software maintenance requires preparation of an accurate plan during the development cycle. Typically, maintenance takes up about 40-80% of the project cost, usually closer to the higher pole. Hence, a focus on maintenance definitely helps keep costs down.

Corrective maintenance:

It deals with the repair of faults or defects found in day-today system functions. A defect can result due to errors in software design, logic and coding. Design errors occur when changes made to the software are incorrect, incomplete, wrongly communicated, or the change request is misunderstood. Logical errors result from invalid tests and conclusions, incorrect implementation of design specifications, faulty logic flow, or incomplete test of data. All these errors, referred to as residual errors, prevent the software from conforming to its agreed specifications. Note that the need for corrective maintenance is usually initiated by bug reports drawn by the users.

Adaptive Maintenance:

Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system. Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system. The term environment in this context refers to the conditions and the influences which act (from outside) on the system. For example, business rules, work patterns, and government policies have a significant impact on the software system.

For instance, a government policy to use a single 'European currency' will have a significant effect on the software system. An acceptance of this change will require banks in various member countries to make significant changes in their software systems to accommodate this currency. Adaptive maintenance accounts for 25% of all the maintenance activities.

Perfective Maintenance:

Perfective maintenance mainly deals with implementing new or changed user requirements. Perfective maintenance involves making functional enhancements to the system in addition to the activities to increase the system's performance even when the changes have not been suggested by faults. This includes enhancing both the function and efficiency of the code and changing the functionalities of the system as per the users' changing needs.

Examples of perfective maintenance include modifying the payroll program to incorporate a new union settlement and adding a new report in the sales analysis system. Perfective maintenance accounts for 50%, that is, the largest of all the maintenance activities.

Preventive Maintenance:

Preventive maintenance involves performing activities to prevent the occurrence of errors. It tends to reduce the software complexity thereby improving program understandability and increasing software maintainability. It comprises documentation updating, code optimization, and code restructuring. Documentation updating involves modifying the documents affected by the changes in order to correspond to the present state of the system. Code optimization involves modifying the programs for faster execution or efficient use of storage space. Code restructuring involves transforming the program structure for reducing the complexity in source code and making it easier to understand.

Preventive maintenance is limited to the maintenance organization only and no external requests are acquired for this type of maintenance. Preventive maintenance accounts for only 5% of all the maintenance activities.

SECURITY MECHANISM

The system security problem can be divided into four related issues: -

- Security
- Integrity
- Privacy
- Confidentiality

System security:

It refers to the technical innovations and procedures applied to the hardware and operating systems to protect against deliberate or accidental damage from a defined threat. In contrast, data security is the protection against data from loss disclosure, modification and destruction.

System integrity:

It refers to the proper functioning of hardware and programs, Appropriate physical security and safe against external threats such as eavesdropping and wire-tapping. In contrast data integrity makes sure that data do not differ from their original form and have not been accidentally or intentionally disclosed, altered or destroyed.

Privacy:

It defines the rights of the users or organizations to determine what information they are willing to share with or accept from others and how the portal can be protected against unwelcome, unfair or excessive dissemination of information about it.

Confidentiality:

It is a special status given to sensitive information in a database to minimize possible invasion of privacy. It is an attribute of information that characterizes its need for protection.

Control Measures:

After system security risks have been evaluated, the next step is to select the measures that are internal and external to the facility. The measures are generally classified under the following:

Identification:

There are three schemes for identifying persons to the computer:

1] Something you know- such as password. A password is the most commonly as means for authenticating the identity of people. Passwords should be hard to guess and easy to remember.

2] Something you are- such as fingerprints or voiceprints.

3] Something you have- such as the credit card, key or special terminal.

Access Control:

Various steps are taken to control access to a computer facility. One way is to use an encoded card system with a log-keeping capability. Encryption is and effective and practical way to safeguard data transmitted over an unprotected communication channel.

Audit Controls:

It protects a system from external security breaches and internal fraud or embezzlement. The resources invested in audit controls, however, should balance with the sensitivity of the data being manipulated. One problem with audit controls is that it is difficult to prove their worth until the system has been violated or a company officer imprisoned. For this reason, audibility must be supported at all management levels and planned into every system.

System Integrity:

It is a line of defence that concentrates on the functioning hardware, database and supportive software, physical security and operating procedures. The costliest software loss is program error. It is possible to eliminate such error through proper testing routines. Parallel runs should be Implemented whenever possible.

TOOLS, HARWARE & SOFTWARE REQUIREMENTS

HARDWARE:

Processor: 1.5GHz or above

Memory: 2GB

Free Disk Space: 200MB

Monitor

Keyboard

Mouse

SOFTWARE:

Operating System: Windows(All versions), Linux

Code Editor : Sublime text (Any)

Python version : 3 or above

ADVANTAGES OF PROPOSED SYSTEM

- It satisfies the user requirements.
- Easy to understand by the user.
- Easy to operate application
- Has a simple and easy to use user interface.
- Integration of two algorithms in one system
- The visualizer maintains the accuracy, by displaying the visited nodes as well as the shortest path.
- Expandable
- Minimum time needed for various processing
- Efficient
- Not too heavy on the user system.

FUTURE SCOPE

- In future, more graph search algorithms such as **DEPTH FIRST SEARCH ALGORITHM, DIJKSTRA's ALGORITHM, SWARM ALGORITHM etc .**

- More graphics and icons can be added to the graphic user interface to enhance the user experience.

- We can host the application on online server to make it accessible to much more vast number of users.

- New features can be added to the present system, these include:
  1. **Adding a bomb:** Adding a bomb will change the course of the chosen algorithm. In other words, the algorithm will first look for the bomb (in an effort to diffuse it) and will then look for the target node.
  2. **Dragging nodes:** The user will be abe drag nodes even after an algorithm has finished running. This will allow user to instantly see different paths.
  3. **Speed:** The user will be able to set the speed by which the application will seek the shortest path. User can tweak the speed of visiting the nearby nodes from fast to slow.
  4. **Maze & Pattern:** This feature will randomly draw the maze or place the wall nodes in the matrix.