ROUTE VISUALIZER

UCS503 Software Engineering Project Report

Mid-Semester Evaluation

Submitted by:

(102003495) LOKESH SODHI

(102003502) ARYAN

(102003504) KANAN SINGLA

(102053039) AARAV GUPTA

BE-Third Year, COE

Group Name: Outliers

Submitted to: Dr. Kanupriya

Computer Science and Engineering Department

TIET, Patiala

December, 2022

TABLE OF CONTENTS

Software Bid/ Project Teams

UCS 503- Software Engineering Lab

Group: 3COE20 Dated: 09-08-2022

Team Name: Outliers
Team ID (will be assigned by Instructor):

Please enter the names of your Preferred Team Members:

· You are required to form **three to four-person** teams
· Choose your team members wisely. You will not be allowed to change teams.

Name	Roll No	Project Experience	Programming Language Used	Signature
Lokesh Sodhi	102003495	Disease Predictor Organ Donation Management System	Python SQL, PL/SQL	
Aryan	102003502	Disease Predictor Organ Donation Management System Shopping Cart Backend	Python SQL, PL/SQL C++	
Kanan Singla	102003504	Nebula – The Voice Assistance Organ Donation Management System E-commerce App	Python SQL, PL/SQL Node.js, React.js, JavaScript	
Aarav Gupta	102053039	Nebula – The Voice Assistance Organ Donation Management System Personal Resume	Python SQL, PL/SQL HTML, CSS, Bootstrap	

Programming Language / Environment Experience

List the languages you are most comfortable developing in, **as a team**, in your order of preference. Many of the projects involve Java or C/C++ programming.

1. Python
2. HTML, CSS, JavaScript
3. C++

Choices of Projects:

Please select **4 projects** your team would like to work on, by order of preference: *[Write at-least one paragraph for each choice (motivation, reason for choice, feasibility analysis, etc.)]*

First Choice	Route Visualizer: Path-finding algorithms are the algorithms that are used to find the shortest path or the shortest route between two points. It will be a visualization tool for various pathfinding algorithms and will have configurations to play around with. The major motivation behind this project is that humans tend to remember visual things more than any other thing. Also, this project could be a good way to teach ourselves about the various path-finding algorithms.
Second Choice	Optical Character Recognition: Develop an algorithm that takes in an image file, and returns the text found in the image as a string. The aim for this is to create an OCR software that recognises characters from scanned images of printed text. This project could be used to learn about the various libraries in Python language.
Third Choice	Human Action Recognition: The main purpose of this project will be to support research in the application area of analyzing activities in public places. The video dataset used here will be SPHAR. These videos have been aggregated from multiple sources, and converted to a consistent file type to contain only one action at a time.
Fourth Choice	Spell It: This app will help users to test their spelling using Google API. The applcation will dictate the word and the user types in the answer.

Additional Remarks/ Inputs

Please tell us about any other factors that we should take into consideration (e.g., if you really would like to work on a project for some particularly convincing reason).

……

We expect to improve our skill set, widen our knowledge base, and develop a robust sense of teamwork and a flair for problem-solving.

2. Planning Phase

2.1 Introduction

Pathfinding algorithms are used to find the shortest path or the shortest route between two points. Two primary problems of pathfinding are
(1) finding a path between two nodes in a graph
(2) the shortest path problem—finding the optimal shortest path.

Route Visualizer will be a visualization tool for various pathfinding algorithms and will have configurations to play around with. It is a multi-purpose website with the following configurations:

- Supported algorithms:

Algorithms	Shortest path guaranteed	Informed search	Weighted graphs	Bidirectional	Wormholes
A*	✓	✓	✓	✓	✓
Dijkstra's	✓	✗	✓	✓	✓
BFS	✓	✗	✗	✓	✓
DFS	✗	✗	✗	✓	✓
Beam Search	✗	✗	✗	✓	✓

- Modes: The pathfinder has 3 modes
    - Multiple sources: All sources simultaneously start their search for the destination, and the final path is drawn from the first source that reaches it.
    - Multiple destinations: The source ends its search at the first destination it reaches. If the algorithm is guided, the lowest heuristic value from all destinations is used.
    - Checkpoints: The path starts from the source and ends at the destination, visiting all checkpoints in the given order.

- Configurations:
    - Cut corners: Disable to prevent the path from touching the corners of obstacle cells during diagonal movement.
    - Allow diagonals: Specify whether a diagonal movement is allowed
    - Bidirectional: Specify whether the destination is a moving agent or not.

- Random mazes: The pathfinder has 2 maze generation algorithms. Both generate a completely random maze that has a possible path between any two cells.
    - Recursive Maze Algorithm
    - Randomized Prim's Algorithm

- Wormholes:
    - When an agent steps on a cell marked as the wormhole entry, it moves to the wormhole exit with 0 costs.
    - The cell marked as wormhole entry can be considered disconnected from its physical neighbours, and its only neighbour becomes the wormhole exit.

- Obstacles:
    - Obstacles are cells over which an agent cannot travel.
    - We create a boundary of obstacles around the grid so that the algorithms are bounded to the visible screen.

- Cell weights:
    - In terms of weighted edges of a directed graph, the cost of moving from one cell to another is the weight of the destination cell.
    - The cost of using a wormhole is 0, which can be considered the wormhole exit point weighting 0.

2.1.1 Project Scope:

Algorithm visualization (often called algorithm animation) uses dynamic graphics to visualize the computation of a given algorithm.
The goal here is to develop a visualization tool to visualize classic graph algorithms like BFS, DFS, Beam Search, Dijkstra, A*, etc. The major motivation behind this project is that humans tend to remember visual things more than anything else. Also, this project could be a good way to teach us about the various path-finding algorithms. Additionally, in the future, if new algorithms are created, those might be deployed in our project. This website will be accessible to all users and supported on all major web browsers.

2.1.2 Requirement:

1) Functional Requirements

a) Accuracy: The user chooses from the aforementioned features based on his preferences. The user-selected features determine how the source node moves through the grid when “Find Route” is selected. When the destination node is discovered, the route from the source to the destination is indicated by highlighting.

We believe that our model will accurately and precisely anticipate the path.

b) Error: If the user uses an obstacle to block the complete path between the source and the destination and no wormholes are employed, a prompt with the message "No path detected" displays.

2) Non-Functional Requirements

a) Performance Requirements
   i) The system's performance should be fast and accurate.
   ii) A reliable internet connection is required.

b) Software Quality Attributes
   i) Good quality framework produces robust, bug-free software which contains all necessary requirements for user satisfaction.

c) Compatibility
   i) It should be compatible with all operating systems.
   ii) No hardware requirement.

d) Usability
   i) Controls used should be self-explanatory. Also, we aim to provide a guide to operating the website.
   ii) Uniformity in the format of screen/pages on your website.
   iii) Alignment with the above goals helps in effective usability testing.

2.2 Feasibility Report:

(1) Financial feasibility

The making of the project is financially feasible. The project incurs a very low cost of maintenance. The start-up cost and operational cost are well within the budget of the project. All the software that we require during the implementation of the project is freely available as open-source. The hosting and database services will be availed at little or no prices.

(2) Technical Feasibility

- The UI/UX of the application is created using HTML, CSS and JavaScript.
- The backend architecture of the web app is implemented using a python(3.7)-based framework.
- Flask 1.1.1
- Azure App Services

(3) Resource feasibility

Resources that are required for our project include:

- Programming devices (Laptops/Desktops)
- Programming Tools (freely available)
- VS Code
- Any major web browsers(Google Chrome recommended)
- Programming Individuals

So, clearly, our project has the required resource feasibility.

(4) Legal feasibility

We are ensuring that we are not using any pirated stuff or any copyrighted stuff. If any resource is used while making the project, it will be cited in the documentation of the project in the references section. We will be doing this project from scratch with the contributed efforts from each member of the group

(5) Operational feasibility

Certainly, our project is realistic. It would be fully functional with a good user experience.
Moreover, it would be a fully deployed application thus ensuring it's operational feasibility.

(6) Scheduling feasibility

Start-26th August, 2022
End-December, 2022

The estimated time required for each phase of our project is as follows:

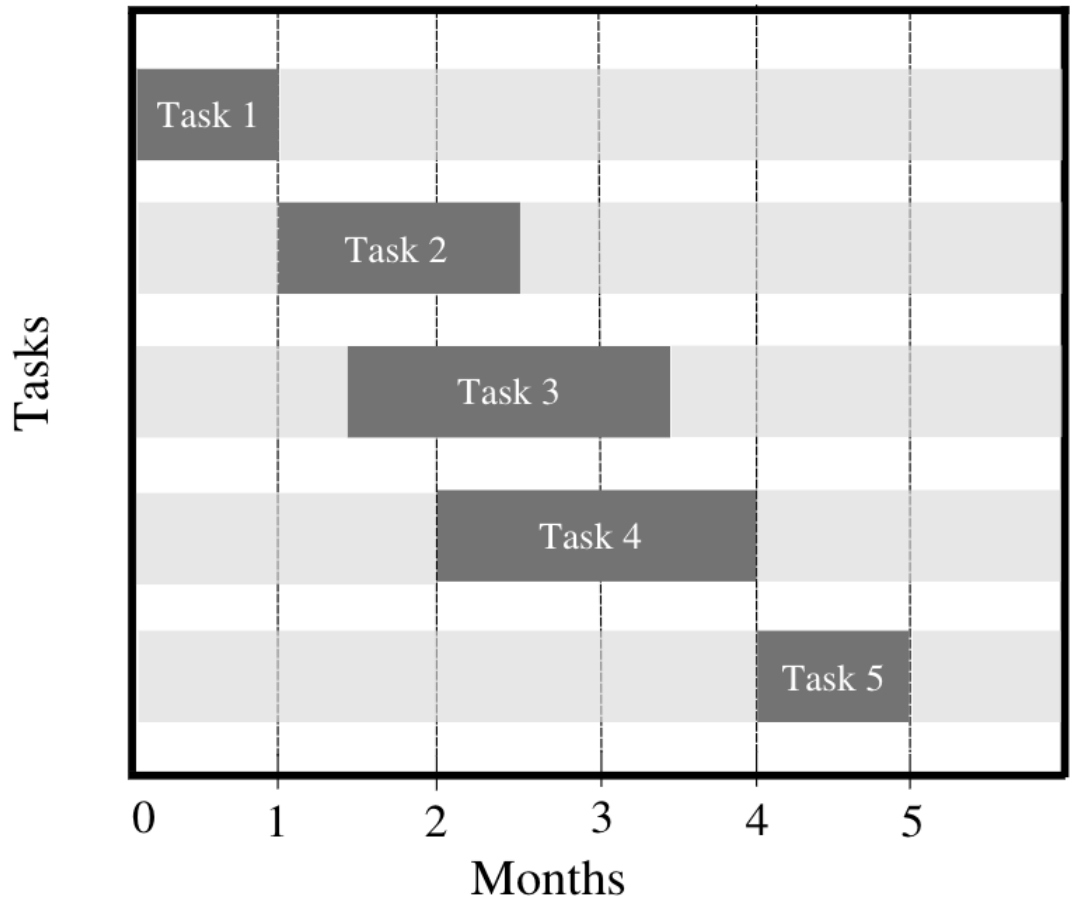
- Analysis Phase: 4 Weeks
- Design Phase: 5 Weeks
- Development Phase: 10 Weeks
- Integration and Testing: 4 Weeks

(7) Cultural/Behavioural feasibility

Since access to education is a fundamental human right, everyone worldwide can benefit from our project because it is both an extremely inclusive and reliable source of knowledge.

So, our project is feasible.

2.3 Gantt Chart:



Task1 : Project Selection and documentation
The "Route Visualizer" proposal was chosen from our selection of four projects. Then the corresponding documents were created.

Task2: Figma Prototype
Figma, a well-liked no-code application used by designers and creators to prototype websites, is what we are planning to use to design our website.

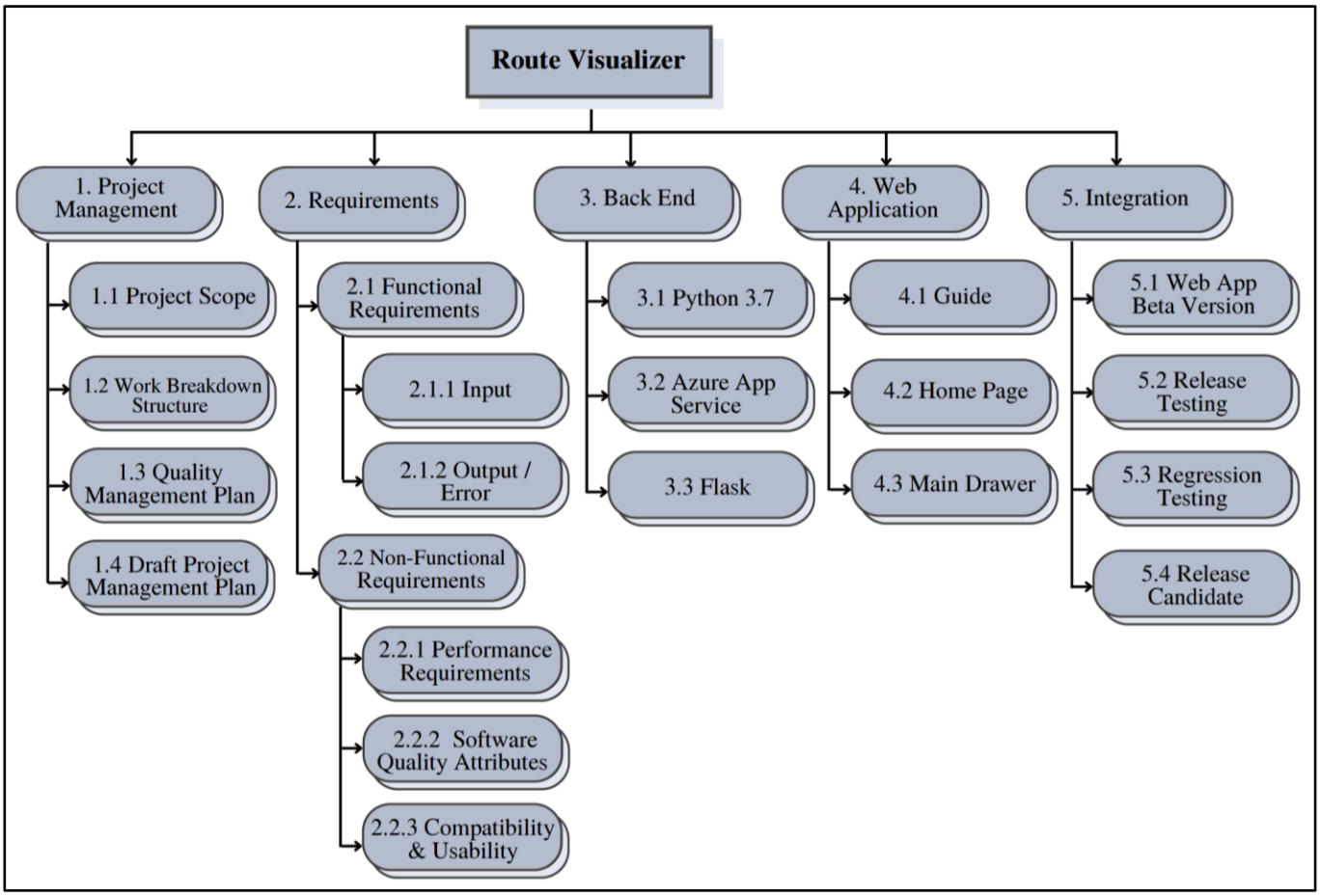
Task3: Back End (algorithms)
Code creation using python libraries.

Task4: Front End
Designing the website's user interface by implementing the FIGMA prototype with HTML, CSS, and JavaScript.

Task5: Integration and Testing
The final steps include integrating the front end and back end and testing it using different algorithms.

2.4 Work Breakdown Structure:

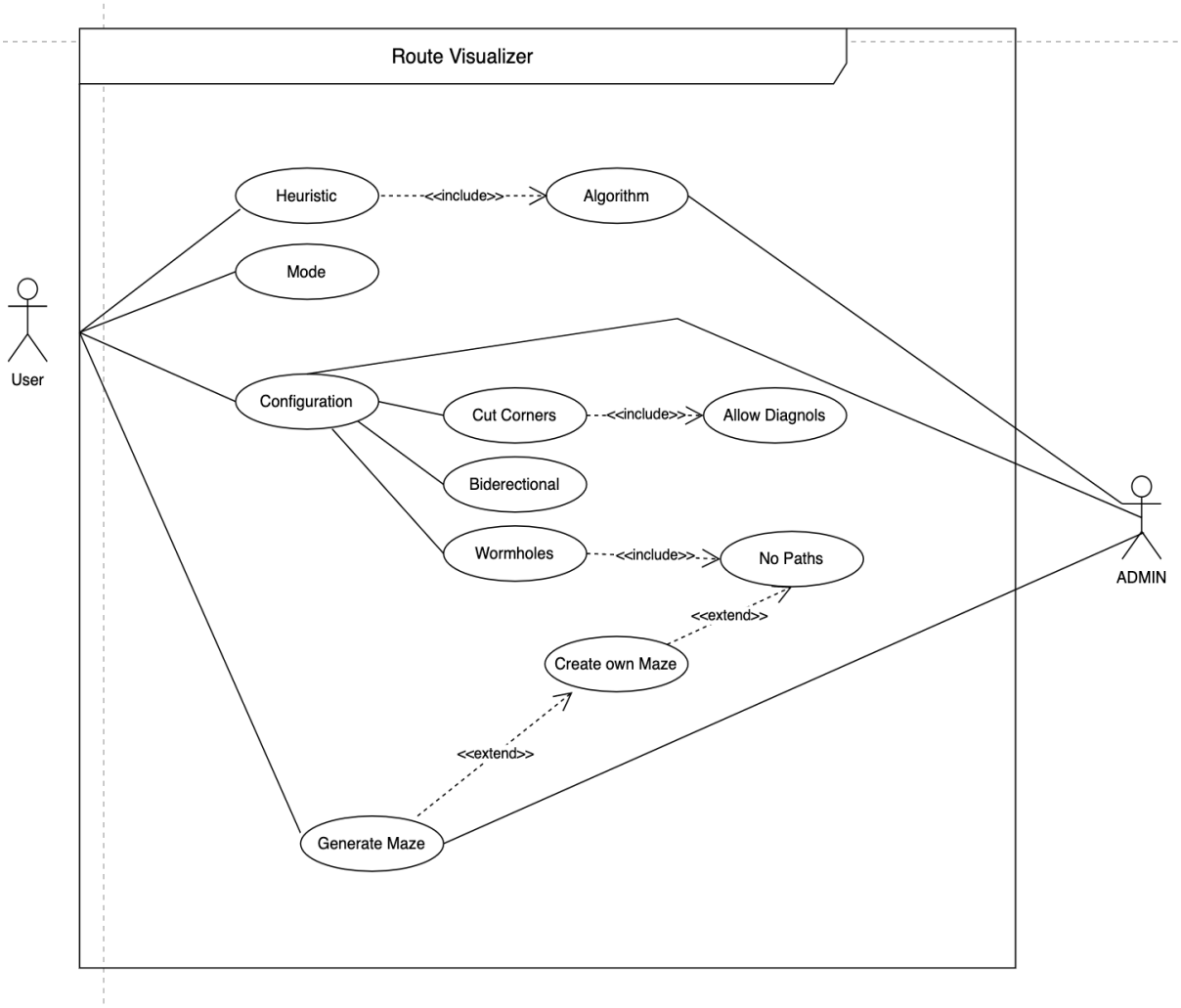
A work breakdown structure (WBS) in project management and systems engineering is a deliverable-oriented breakdown of a project into smaller components. A work breakdown structure is a key project deliverable that organises the team's work into manageable sections.
A work breakdown structure (WBS) is a tool that can be used for projects, programs, and even initiatives to understand the work that has to be done to successfully produce a deliverable(s). The benefits of creating a WBS include: it defines and organises the work required.



3. Analysis Phase

3.1 Use Case Diagram:

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. In the Figure below, we have two actors -the user and the admin.



3.2 Use Case Templates

Use Case ID:	1
Use Case Name:	Heuristics

Actor:	User
Description:	This option will allow the user to choose from listed various types of heuristics which will govern the calculations for finding the appropriate route.
Preconditions:	1.User must have already selected a route finding algorithm. 2. User must have stable internet connection.
Task Sequence	1. User will select an appropriate algorithm. 2. Now user can select the desired heuristics.
Post Condition	User's choice of heuristic will be stored in the database.
Modification history	10th September, 2022
Authors	Lokesh Sodhi, Aryan, Kanan Singla, Aarav Gupta

USE CASE TEMPLATE - 1

Use Case ID:	2
Use Case Name:	Mode

Actor:	User
Description:	This option will allow the user to select from different modes which will define the number of sources and destinations.
Preconditions:	1.User must have already selected a route finding algorithm. 2.User must have already selected a heuristic. 3.User must have stable internet connection.
Task Sequence	1. User will select an appropriate algorithm. 2.After selecting the algorithm, the user will select a heuristic of their choice. 3.Now the user will select from the modes provided.
Post Condition	User's choice of mode will be stored in the database.
Modification history	10th September, 2022
Authors	Lokesh Sodhi, Aryan, Kanan Singla, Aarav Gupta

USE CASE TEMPLATE - 2

Use Case ID:	3
Use Case Name:	Configuration

Actor:	User
Description:	This option will allow the user to select the configurations which will hence restrict the required path as desired.
Preconditions:	1.User must have already selected a route finding algorithm. 2.User must have already selected a heuristic. 3.User must have stable internet connection.
Task Sequence	1.User will select an appropriate algorithm. 2.After selecting the algorithm, the user will select a heuristic of their choice. 3.User will further select a mode from the options provided. 4.Now the user will select the configuration they prefer.
Post Condition	User's choice of configuration will be stored in the database.
Modification history	10th September, 2022
Authors	Lokesh Sodhi, Aryan, Kanan Singla, Aarav Gupta

USE CASE TEMPLATE - 3

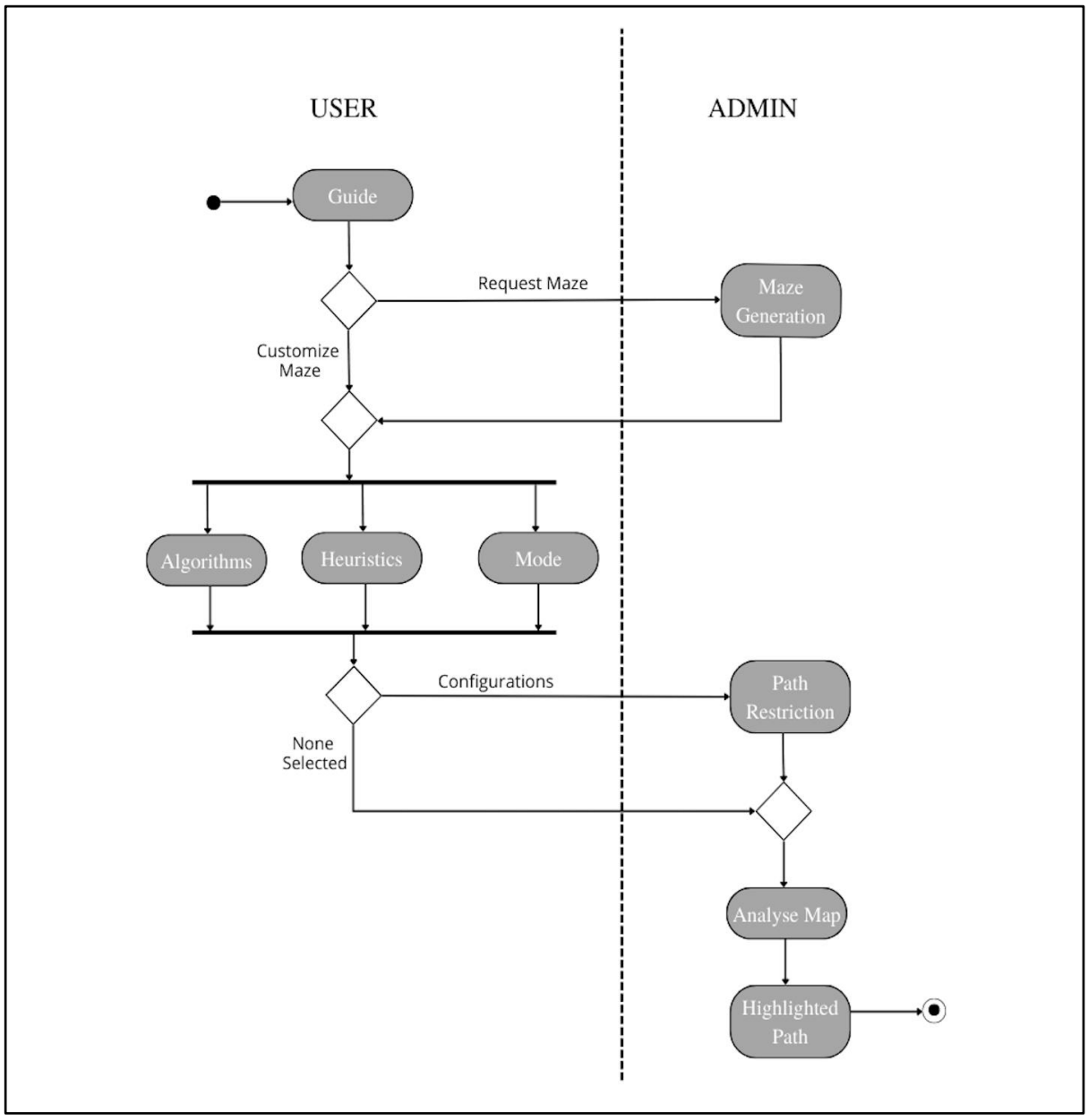
Use Case ID:	4
Use Case Name:	Generate Maze

Actor:	User
Description:	This option will allow the user to choose algorithm required to generate random maze.
Preconditions:	1.User must have already selected a route finding algorithm. 2.User must have already selected a heuristic. 3. Internet connection must be stable.
Task Sequence	1. User will select an appropriate algorithm. 2.After selecting the algorithm, the user will select a heuristic of their choice. 3.Now the user will select from the maze generating algorithms provided.
Post Condition	Random Maze will be generated.
Modification history	10th September, 2022
Authors	Lokesh Sodhi, Aryan, Kanan Singla, Aarav Gupta

USE CASE TEMPLATE - 4

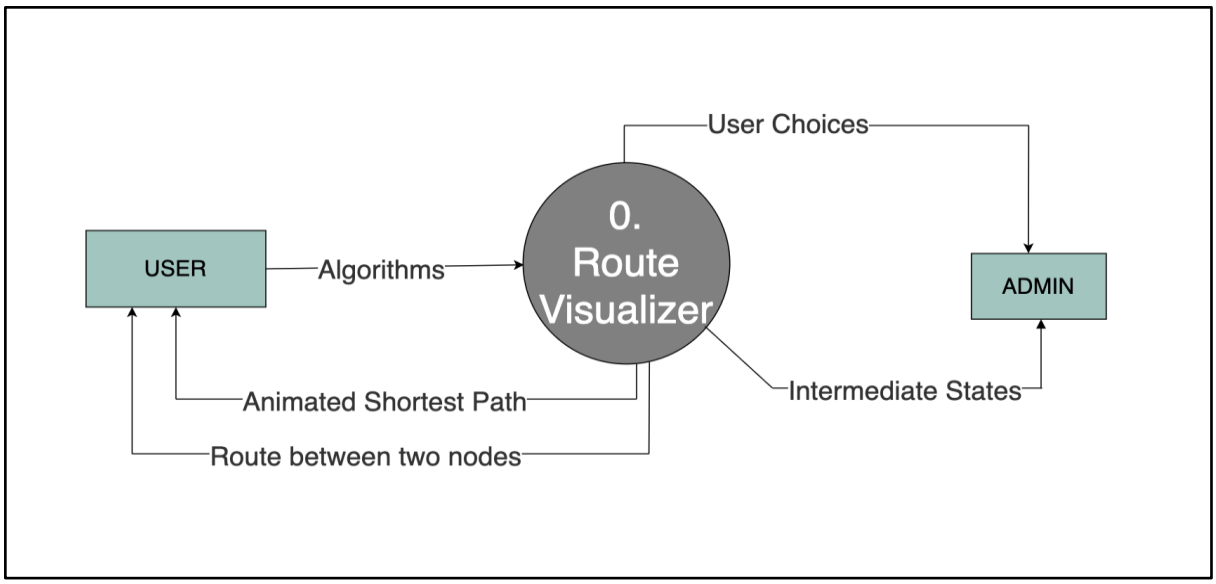
3.3 Activity diagram and Swimlane Diagrams:

An activity diagram is a behavioural diagram i.e. it depicts the behaviour of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. In Figure below, two objects are user and admin.

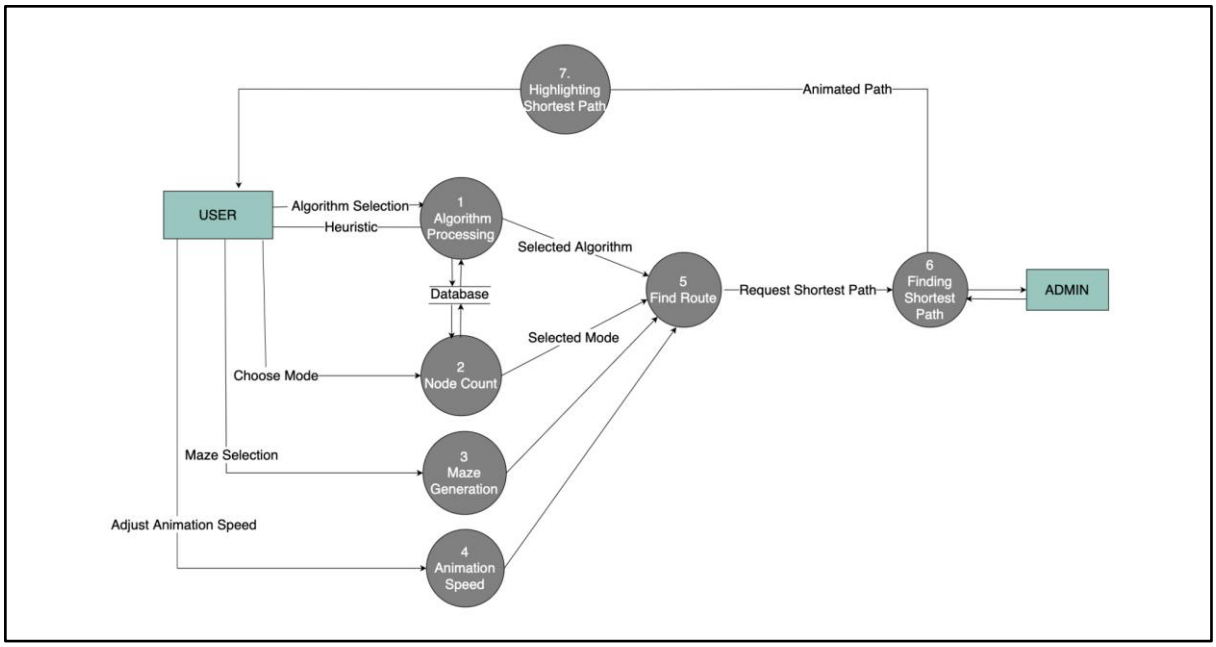


3.4 Data Flow Diagram (DFD):

DFD(data flow diagram) is drawn to represent the system of different levels of abstraction. Higher-level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. In Figure below, we will see mainly 2 levels in the data flow diagram, which are: 0-level DFD, 1-level DFD.



0- Level DFD



1- Level DFD

3.5 Software Requirement Specification in IEEE Format

1. Introduction

1.1 Purpose of Document:

The purpose of this document is to provide an overview of the software product that we aim to develop and its requirements, i.e., hardware, software specifications, and intended audience. The document has been formatted to divide the deliverables into smaller components, describing the functions, goals, and tasks the system can perform.

In short, the purpose of this SRS document is to provide a detailed overview of our software product, its parameters, and its goals. This document describes the project's target audience, user interface, hardware, and software requirements. It defines how our client, team, and audience see the product and its functionality. Nonetheless, it helps any designer and developer to assist in software delivery lifecycle (SDLC) processes.

1.2 Document Convention:

The bold-faced text has been used to emphasize section and subsection headings. The font is uniform across the document, which is Times New Roman with size 13. Line spacing for the whole document is 1.07 with 2 lines between different sections. The text in paragraphs is justified. Lists are used to show subsections of a section.

1.3 Intended Audience and Reading Suggestions:

The audience that the software product being developed targets:

1. The student who wants to learn more about graph theory, its concept, and its applications and who wants to use visualization to comprehend algorithms more thoroughly.
2. The professors who, in addition to imparting theory to their students, can also help them visualize the concept of how algorithms work so that they fully grasp them.

The document has been divided into sections and further sub-sections depending on the aspects that must be covered while developing the product to meet the required specifications. The first section gives an overview of the project- the scope, the intended audience, and the conventions that are to be followed throughout the document. The second section examines the various perspectives of the product, including the features, viewpoints of different users to be considered, the limitations and challenges to be faced while developing the product, the dependencies required, and the assumptions to be kept in mind. This section needs to be studied in detail by

the developers since it covers every detail which will be required while developing the product. The third section covers a detailed description of each system feature. The users and testers must carefully read this section. These sections dealt with the functional requirements of the project. The last section covers various non-functional aspects such as the safety, security, and performance requirements that need to be taken care of by the project manager.

1.4 Project Scope:

Algorithm visualization (often called algorithm animation) uses dynamic graphics to visualize the computation of a given algorithm.

The goal is to develop a visualization tool to visualize classic graph algorithms like BFS, DFS, Beam Search, Dijkstra, A*, etc. The major motivation behind this project is that humans tend to remember visual things more than anything else. Also, this project could be a good way to teach us about the various path-finding algorithms. Additionally, in the future, if new algorithms are created, those might be deployed in our project. This website will be accessible to all users and supported on all major web browsers.

A real-world use case scenario for this project could be a navigation-based application that lets the user find the shortest path between two user-defined locations as defined in our project by source and destination nodes and considering the buildings and other obstacles as a maze-like structure used in this project.

1.5 References:

- https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
- https://en.wikipedia.org/wiki/Maze_generation_algorithm
- https://www.c-sharpcorner.com/article/software-requirements-specification/
- https://belitsoft.com/custom-elearning-development/srs-for-e-learning-management-system

2. Overall Description:

2.1 Product Perspective:

2.2 Product Functions:

1. Our projects use a variety of **Algorithms**, including BFS, DFS, Beam Search, Dijkstra, A*, etc to find a path between two nodes in a graph and the shortest path,i.e, finding the optimal shortest path
2. The pathfinder has three **Modes**, **Multiple sources** in which all sources simultaneously start their search for the destination, and the final path is drawn

from the first source that reaches it, **Multiple destinations**, i.e. The source ends its search at the first destination it reaches. The algorithm is guided with the lowest heuristic value from all destinations. **Checkpoints** are points that are visited in order by the path from the source to the destination.

3. There are various **Configurations** available, some of which are **Cut corners** that prevent the path from touching the corners of obstacle cells during diagonal movement, **Allow diagonals** that specify whether a diagonal movement is allowed or not, and **Bidirectional** which specify whether the destination is a moving agent or not.
4. **Random mazes** are one of the features of the pathfinder, which has 2 maze generation algorithms. Both generate a random maze with a possible path between the two cells.
5. **Wormholes** are agents that step on a cell marked as the wormhole entry; it moves to the wormhole exit with 0 costs. The cell marked as a wormhole entry can be considered disconnected from its physical neighbours, and its only neighbour becomes the wormhole exit.
6. Obstacles are cells over which an agent cannot travel. We create a boundary of obstacles around the grid so that the algorithms are bound to the visible screen.

2.3 User Classes and Characteristics:

The goal is to design software for a user who wants to learn more about graph theory, its concept, and applications and wants to use visualization to comprehend algorithms more thoroughly. These user types are listed below as follows:

1. Student

2. Professors

3. Student cum Staff

4. Group Leader

As seen from the list, each user can have a different educational background and expertise in using the system. Our goal is to develop software that should be easy to use for all types of users. Thus while designing the software, one can assume that each user type has the following characteristics:

- The user is computer-literate and has little or vast information about path-finding algorithms.
- Only a reliable internet connection is needed for this application's widespread use.

2.4 Operating Environment

As this is a web-based application, no such operating environment requirements exist. But as the website relies on browsers and their capability to load data and handle

requests, browsers with the latest updates are preferred for the best experience. Old and discontinued browsers such as Microsoft internet explorer or older versions of opera are not preferred. Also, as the screens are to be rendered in real-time, a system with 2 GB of ram and processors powerful enough to support a web browser will be preferred for the best-intended experience.

2.5 General Constraints, Assumptions and Dependencies

The following list presents the constraints, assumptions, dependencies, or guidelines that are imposed upon the implementation of Route Visualizer:

- As we are using hosting from third-party websites so the availability and speed of the connection will depend upon the services provided by them.
- The files are stored on other third-party servers, so the performance and storage capacity will depend on the service provider's capability and services.
- The product has a user-friendly interface that is simple enough for all types of users to understand.
- A general knowledge of basic graph theory and its concepts is required to use the product.
- Response time for loading the software and for finding a route should be no longer than six seconds.

2.6 Apportioning of requirements

The Route Visualizer can be implemented in the following phases:

- FIGMA:

  Here we design the website’s general layout using a collaborative web application for interface design.

- Deployment Phase

  After the FIGMA step is successfully completed, we will build the front end and back end of our project before integrating the two.

- Launching website

  The integrated project module is then launched using a third-party hosting platform which in turn would be accessible for global application.

Here, the same functionalities will be implemented in each phase; the only difference will be the purpose of using it and the scale of implementation.

3. Specific Requirements

3.1 External Interface Requirements

3.1.2 User Interfaces

- When a user enters our website, a guide panel appears on the screen. This help section would give the user all the necessary instructions for using our website.
- After that, the user is presented with a grid containing two flags representing the source and destination.
- The main drawer on the left has a list of all the necessary features listed in our document.

The following list presents the external interface requirements:

- The product requires very limited graphics usage with just a simple keypad/click for taking the user input.
- The product does not require the usage of sound.
- Sound is not an essential feature, but can be considered for future system variants wherein the guide can be vocalized.

3.2 Functional Requirements

3.2.1 Guide Panel -

Description:

When a user initially visits our website, the first thing they see is a guide panel that is meant to assist them to utilise the functionalities of the website as effectively and helpfully as possible.

Stimulus/Response Sequences:

1. The user will click on the next button.
2. The user can go to the previous instruction with a single click on the previous button.
3. If the user clicks the “don’t show again” button on the guide panel's last instruction page, the guide panel will not reappear for them.
4. Finally, the user must click the submit button to access the website's main page.

Functional Requirements:

1. **Navigate to Guide Panel:**
    a. Input: The user needs to click on the Next/Previous buttons and Finish button at last.
    b. Output: The user is presented with a guide panel having all instructions.
2. **Closing the Panel:**

a. Input: The user must click on the “Don’t Show Again” button before the Submit button.
b. Output: The user is presented with the main page after submitting.

3.2.2 Main Drawer -

Description:

All of the aforementioned elements are included in the main drawer, some of which must be chosen in order to find the shortest path or the distance between the two nodes.

Stimulus/Response Sequences:

1. The user will select the algorithms followed by the heuristics (if required).
2. Following algorithms and heuristics selection, the user will select modes.
3. The user can choose configurations to limit the route.
4. In order to create random mazes, there are two different algorithms that the user may select based on their requirements.
5. Finally, the user clicks "Find Route" to find the correct route.

Functional Requirements:

1. **Accessing the features:**
    a. Input: The user needs to select the required features mentioned above.
    b. Output: The user is presented with the grid page.

3.2.3 Grid Page -

Description:

On our website's home page, "Grid Page," which features a matrix, the user may customize the maze. The two flags on the grid represent the source node and destination node, respectively.

Stimulus/Response Sequences:

1. Initially, the user customizes the maze on the grid page.
2. Depending on the mode chosen in the main drawer, the user can increase the flag count.
3. When the user selects "Find Route," the route is shown as a highlighted path.

Functional Requirements:

1. **Generating maze:**
    a. Input: In order to create maze walls (obstacles), the user needs to click on desired grid elements (pixels).

b. Output: Highlighted path is displayed representing the route between the nodes.

2. **Flag count:**
    a. Input: The user must choose a "Mode" and then click on a grid element that they wish to be a checkpoint or an additional source.
    b. Output: The selected grid element is replaced by a flag.

3.3 Performance Requirements

- The performance of the system should be fast and accurate.
- The system should be able to “Find Route” by applying various algorithms as many times as the user requests.

3.4 Quality Attributes

The product is targeted towards a wide variety of users such as students, staff, student cum staff, etc. The product must load quickly and work well on a variety of terminals. It must also tolerate a wide variety of input possibilities from a user, such as incorrect responses or unforeseen keystrokes.

3.5 Other Requirements

3.5.1 Software Quality Attributes

1. Good quality of framework produces robust, bug-free software which contains all necessary requirements for user satisfaction.
2. Software should be fully serviceable, especially in case of maximum load. Tests should always have positive results and estimates, i.e., accurately give the path.

4. Document Approvers

SRS for Route Visualizer is approved by:

_____

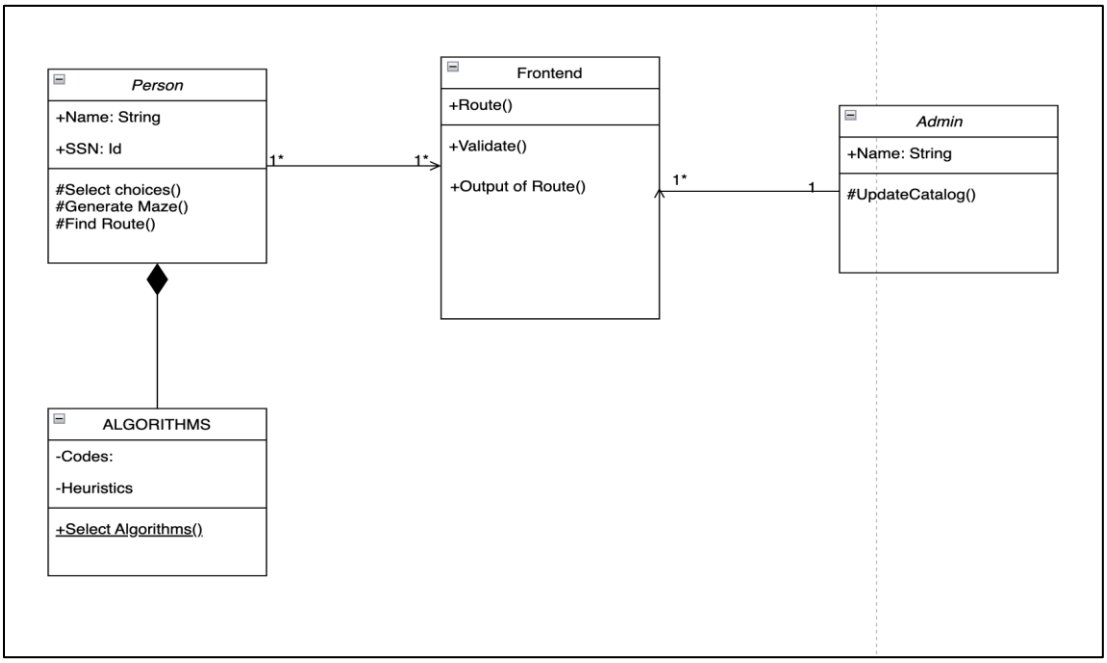
(name)

Designation

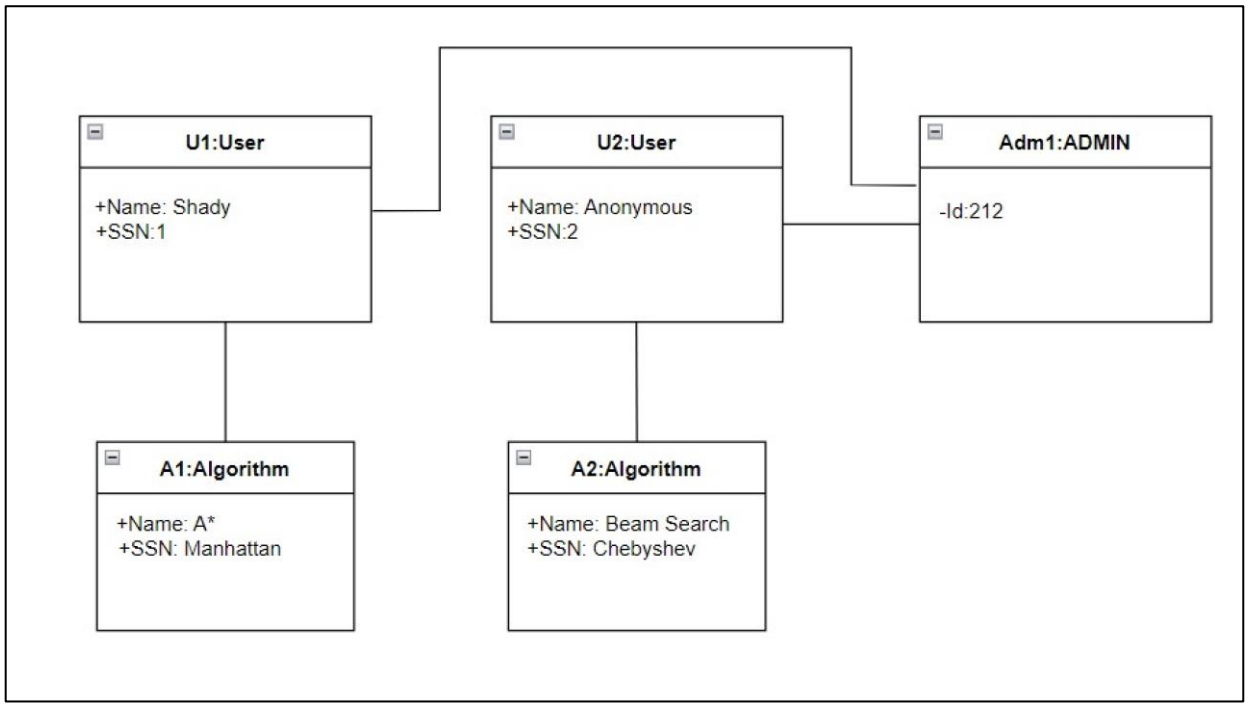
4. Design Phase

4.1 Class Diagram and Object Diagram:

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

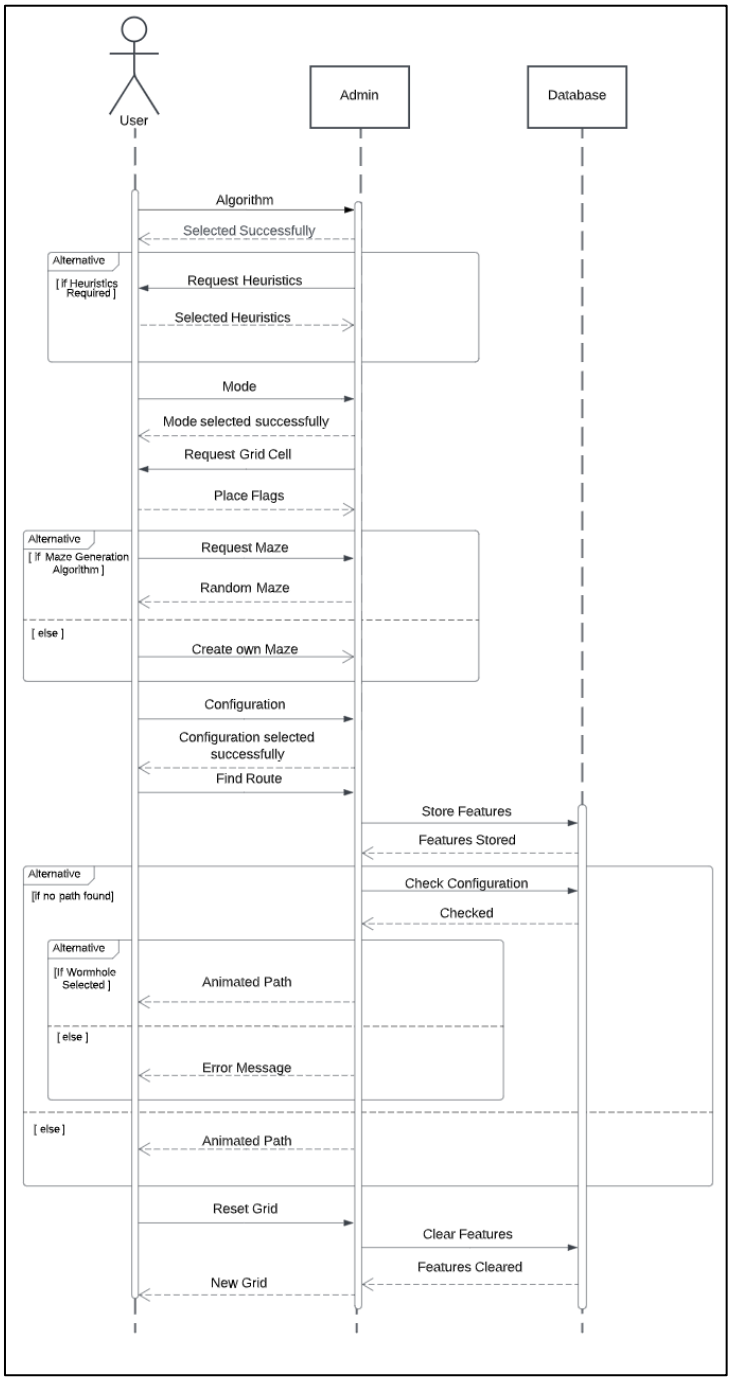


Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams. Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment. Object diagrams are used to render a set of objects and their relationships as an instance.



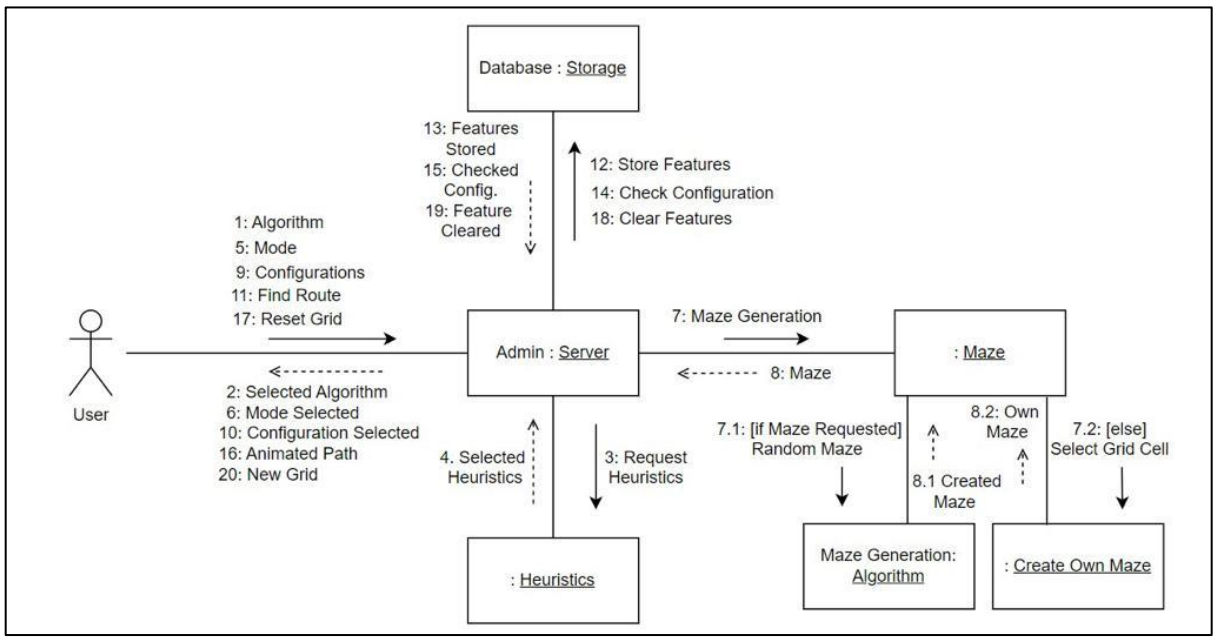
4.2 Sequence Diagram:

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.



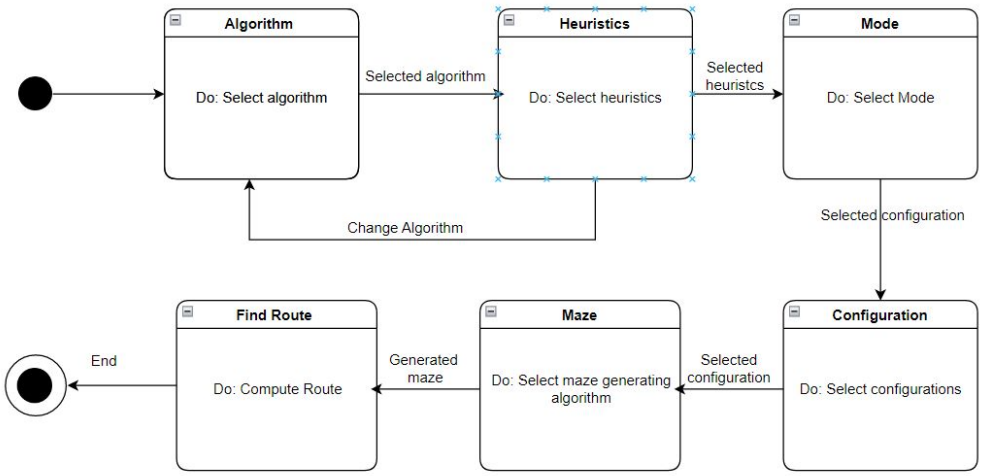
4.3 Collaboration Diagram

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.



4.4 State Chart Diagram:

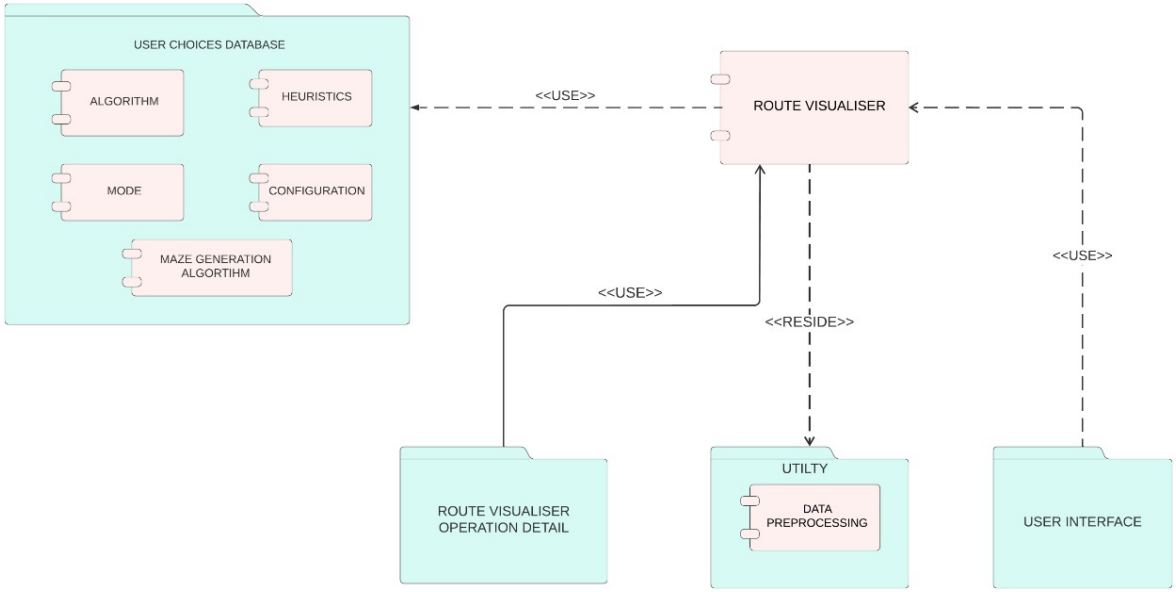
A **state diagram** is used to represent the condition of the system or part of the system at finite instances of time. It's a **behavioural** diagram and it represents the behaviour using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams**. These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behaviour of a class in response to time and changing external stimuli. We can say that each and every class has a state but we don't model every class using State diagrams. We prefer to model the states with three or more states.



5. Implementation:

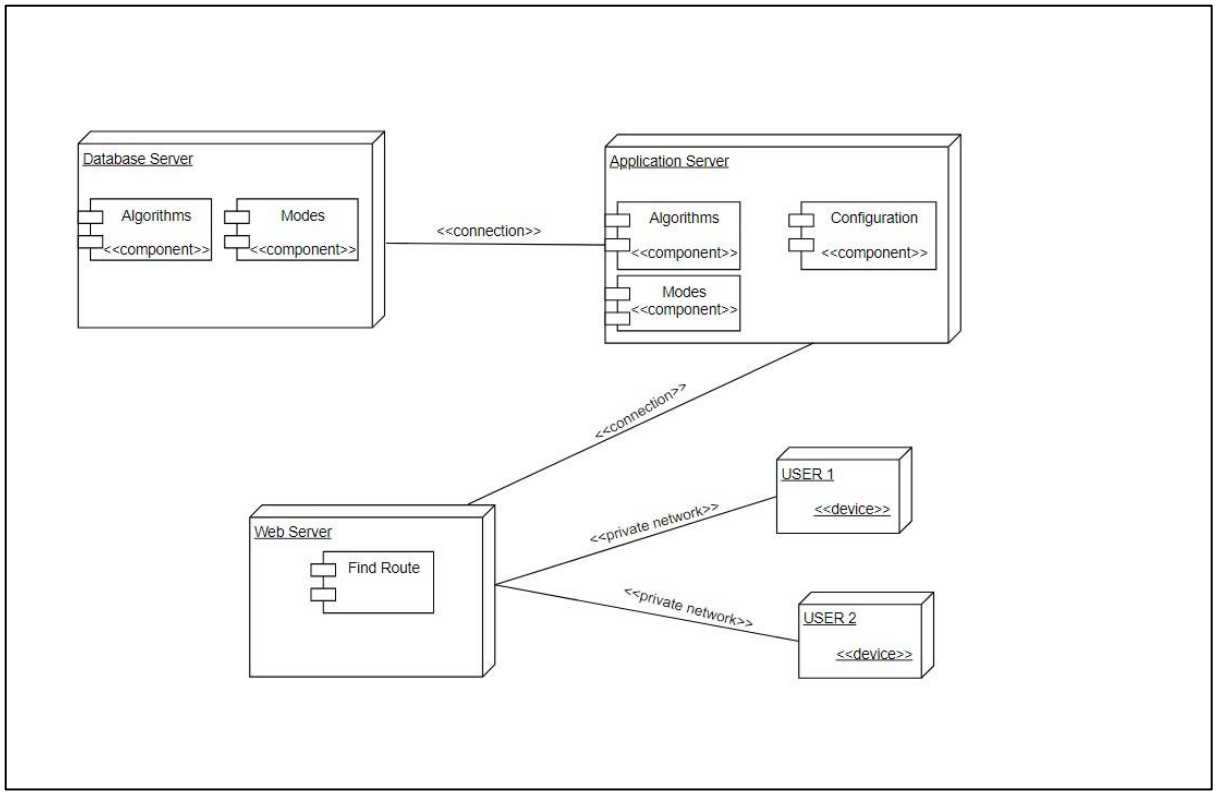
5.1 Component Diagrams:

A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node. It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behaviour is explained by the provided and required interfaces.



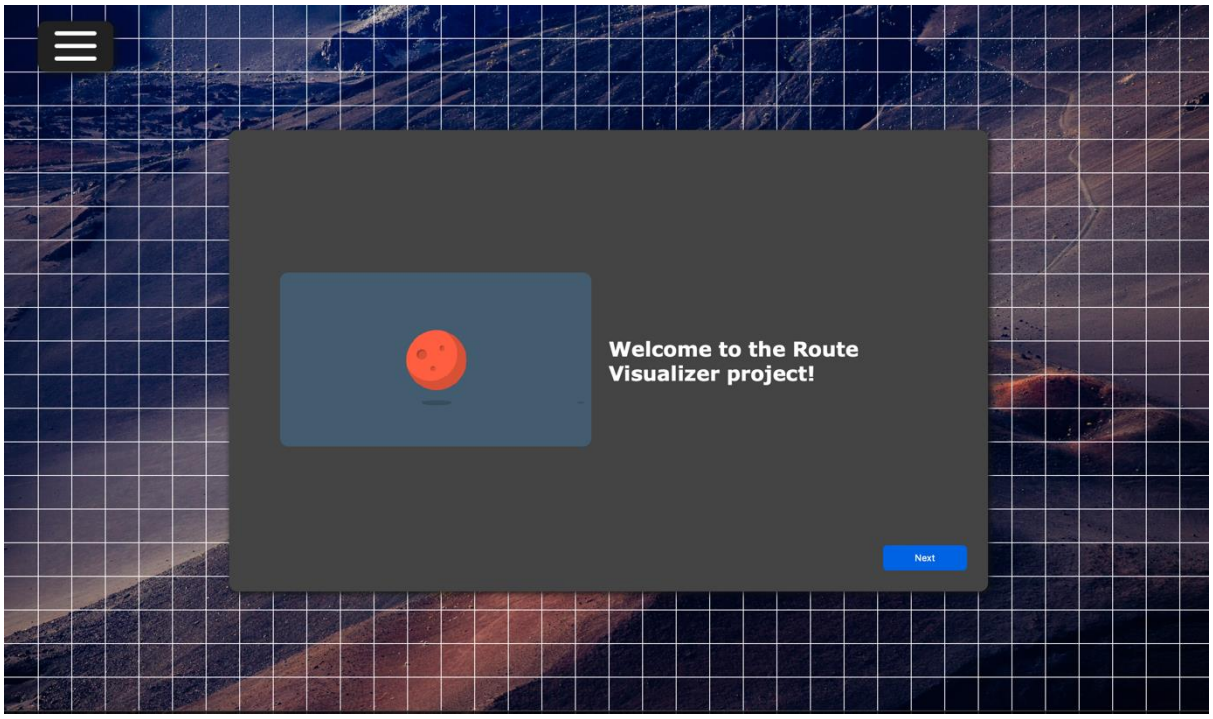
5.2 Deployment Diagrams:

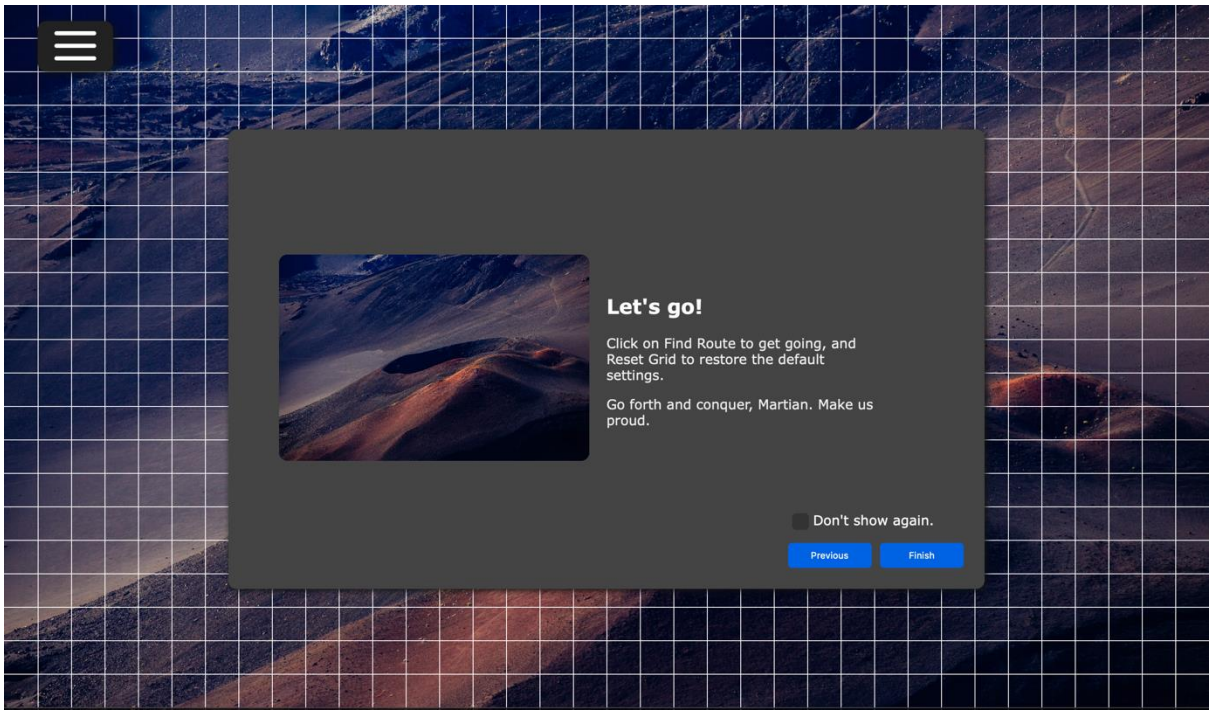
The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships. It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.



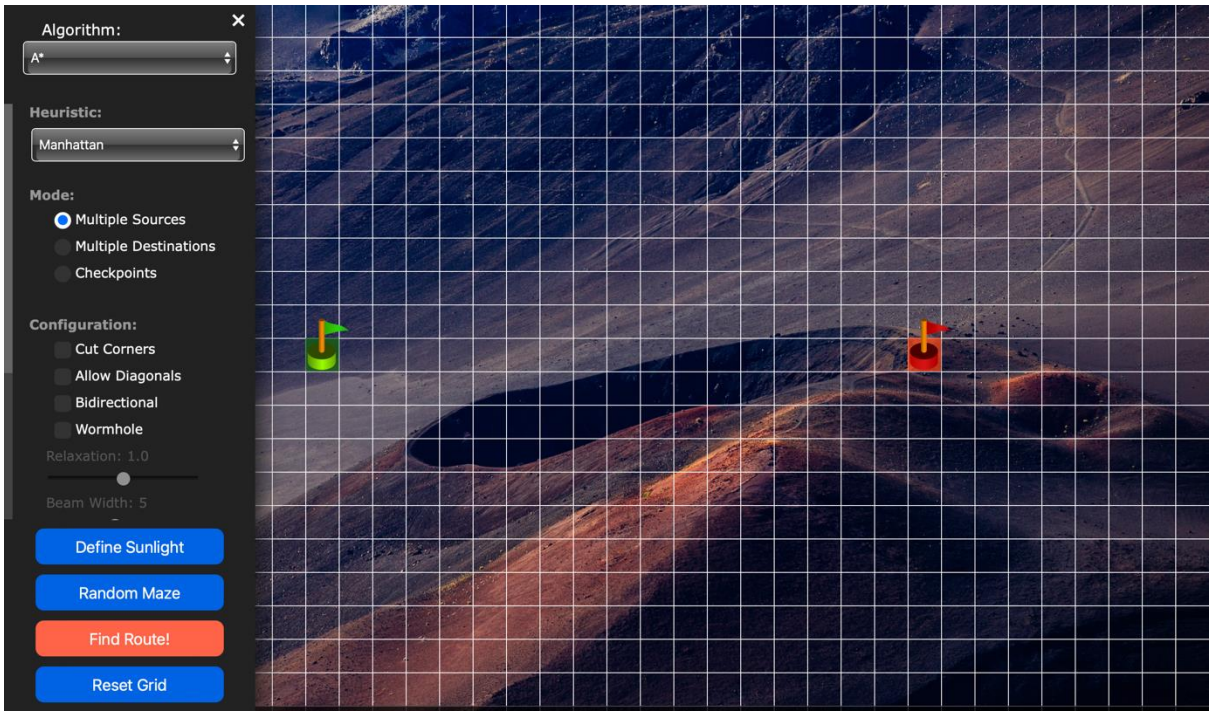
5.3 Screenshots of Working Project

Guide Page



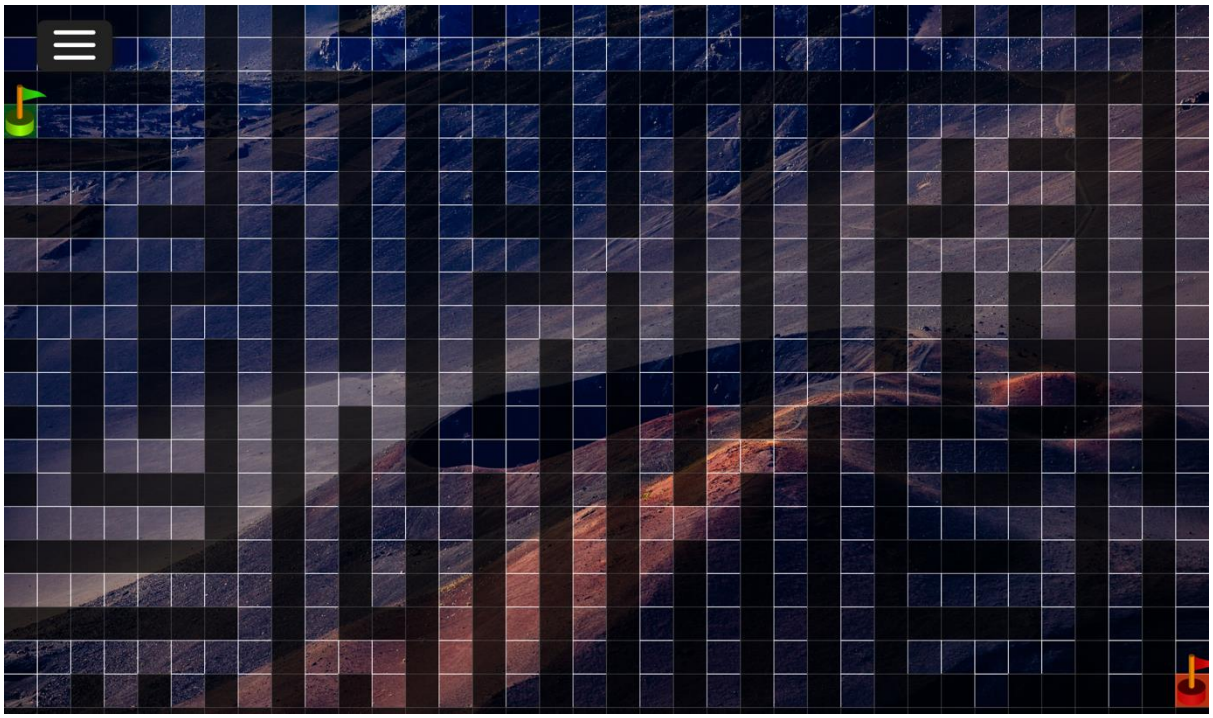


Main Drawer:

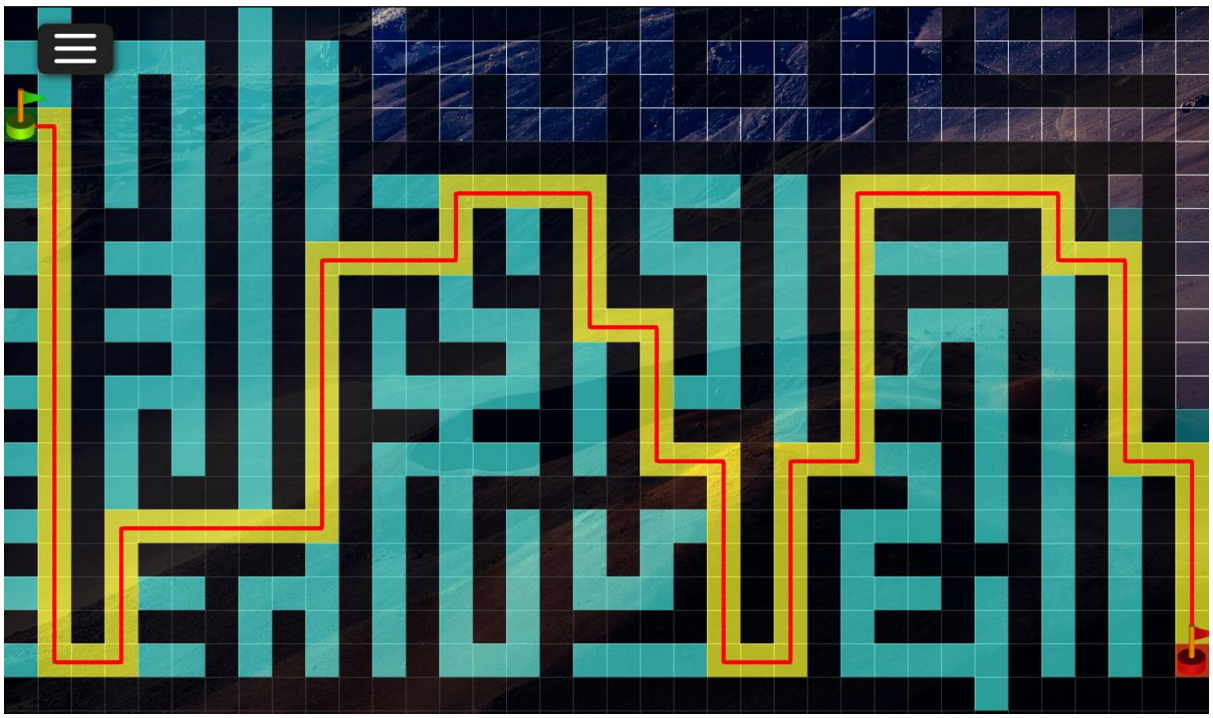

Algorithm:
A*
Heuristic:
Manhattan
Mode:
Multiple Sources
Multiple Destinations
Checkpoints
Configuration:
Cut Corners
Allow Diagonals
Bidirectional
Wormhole
Relaxation: 1.0
Beam Width: 5
Define Sunlight
Random Maze
Find Route!
Reset Grid


Algorithm:
A*
Configuration:
Cut Corners
Allow Diagonals
Bidirectional
Wormhole
Relaxation: 1.0
Beam Width: 5
Maze Generation Algorithm:
Recursive Maze
Randomized Prim
Animation Speed:
Medium
Define Sunlight
Random Maze
Find Route!
Reset Grid

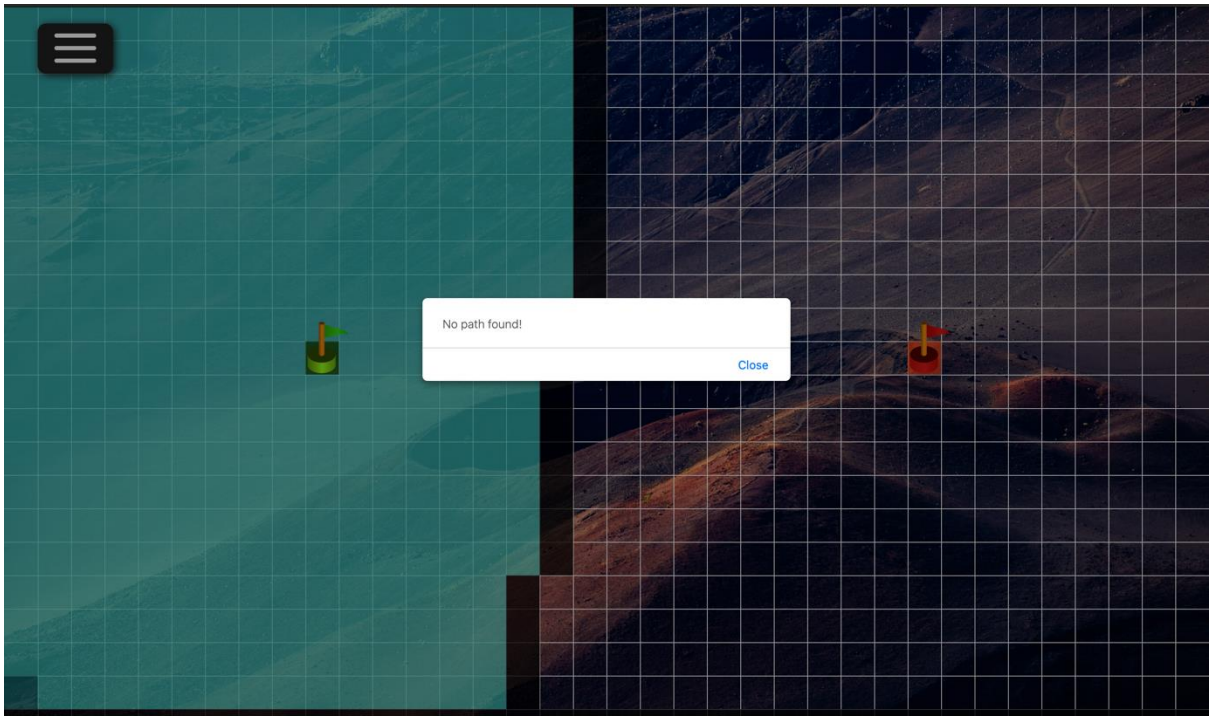
Random Maze:



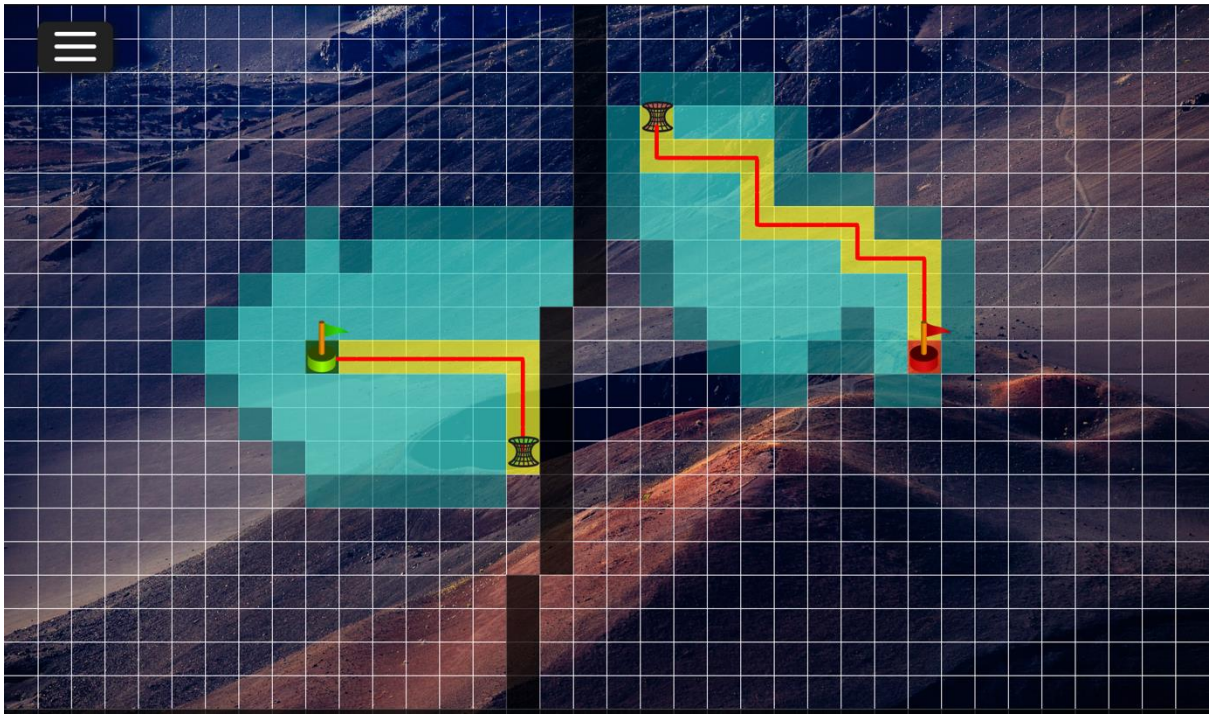
Find Route:



No Path Found:



Wormholes:



6. Testing
6.1 Test Cases

Test Case #: 1.1	**Test Case Name:** With Heuristic
System: Route Visualizer	**Subsystem:** Heuristics
Designed by: Lokesh, Aryan, Kanan, Aarav	**Design Date:** 30/11/2022
Executed by: Himank, Sahil, Ishan	**Execution Date:** 30/11/2022
Short Description: Test Algorithms that use Heuristics	

Pre-conditions
1. The user must have a stable internet connection.
2. The user must have basic knowledge of algorithms and heuristics.

Step	Action	Expected System Response	Pass/ Fail	Comment
1	Click the 'Navigation' button	The system displays the main menu with various features.	Pass	
2	Select the 'A*' algorithm.	The system assigns A* in the algorithm slot.	Pass	
3	Select 'Manhattan' as Heuristics	The system assigns Manhattan to the heuristics slot.	Pass	
4	Create obstacles such that at least one path exists between the source and destination.	The system highlights the grid cell selected by the user.	Pass	
5	Click the 'Find Route' button	The system stores the selected features in the database. The source flag starts navigating on the basis of the heuristics selected.	Pass	

Post-conditions
1. The highlighted path from the source to the destination is displayed on the screen.

Test Case #: 1.2 **Test Case Name:** Without Heuristic

System: Route Visualizer **Subsystem:** Heuristics

Designed by: Lokesh, Aryan, Kanan, Aarav **Design Date:** 30/11/2022

Executed by: Himank, Sahil, Ishan **Execution Date:** 30/11/2022

Short Description: Test Algorithms that do not require Heuristics

Pre-conditions

1. The user must have a stable internet connection.
2. The user must have basic knowledge of algorithms and heuristics.

Step	Action	Expected System Response	Pass/ Fail	Comment
1	Click the 'Navigation' button	The system displays the main menu with various features.	Pass	
2	Select the 'Dijkstra' algorithm.	The system assigns Dijkstra to the algorithm slot.	Pass	
3	Create obstacles such that at least one path exists between the source and destination.	The system highlights the grid cell selected by the user.	Pass	
4	Click the 'Find Route' button	The system stores the selected features in the database. The source flag starts navigating on the basis of the heuristics selected.	Pass	

Post-conditions

1. The highlighted path from the source to the destination is displayed on the screen.

Test Case #: 2.1 **Test Case Name:** Wormholes

System: Route Visualizer **Subsystem:** Path Configuration

Designed by: Lokesh, Aryan, Kanan, Aarav **Design Date:** 30/11/2022

Executed by: Himank, Sahil, Ishan **Execution Date:** 30/11/2022

Short Description: Test Configurations functionality

Pre-conditions

1. The user must have a stable internet connection.
2. The user must select the required features (algorithms and heuristics).

Step	Action	Expected System Response	Pass/ Fail	Comment
1	Create obstacles such that no path exists between the source and destination.	The system highlights the grid cell selected by the user.	Pass	
2	Click the 'Navigation' button	The system displays the main menu with various features.	Pass	
3	Select 'Wormhole' as configuration.	The system creates two Wormholes that can be placed according to the user's requirements.	Pass	
4	Click the 'Find Route' button	The system stores the selected features in the database. The source flag starts navigating on the basis of the precondition.	Pass	
	Check Post-condition 1			
5	Repeat steps 1, and 2 and unselect the 'Wormhole' configuration.	The system deletes wormholes.	Pass	
6	Repeat step 4.	The system stores the selected features in the database. The source flag starts navigating on the basis of the precondition.	Pass	

Post-conditions

1. The highlighted path from the source to the destination through wormholes is displayed on the screen.
2. Dialog box with an error message 'No Path Found' is displayed on the screen.

Test Case #: 3.1 **Test Case Name:** Random Maze and Reset Grid

System: Route Visualizer **Subsystem:** Maze Generation

Designed by: Lokesh, Aryan, Kanan, Aarav **Design Date:** 30/11/2022

Executed by: Himank, Sahil, Ishan **Execution Date:** 30/11/2022

Short Description: Test maze generation and resetting services

Pre-conditions

1. The user must have a stable internet connection.
2. The user must select the required features (algorithms and heuristics).

Step	Action	Expected System Response	Pass/ Fail	Comment
1	Click the 'Navigation' button	The system displays the main menu with various features.	Pass	
2	Select the 'Recursive Maze' algorithm.	The system selects the Recursive Maze from the list of Random Maze algorithms.	Pass	
3	Click the 'Random Maze' button.	The system creates a random maze on the basis of the maze algorithm selected.	Pass	
4	Click the 'Find Route' button.	The system stores the selected features and current state in the database. The source flag starts navigating on the basis of the preconditions.	Pass	
	Check Post-condition 1			
5	Repeat step 1 and click on the 'Reset Grid' button.	The system clears out all the features and grids to default.	Pass	
6	Select grid cells to create a customized maze.	The system highlights the grid cell selected by the user which are obstacles.	Pass	

Post-conditions

1. The highlighted path from the source to the destination through the maze is displayed on the screen.

6.2 Cyclomatic Complexity :-

Since, the project contains multiple files hence, there is a need of calculating the average cyclomatic complexity in such case:

The individual complexity of files sums up to –
11+3+43+72+6+31+31+10+7+44+13+7+5+7+7+7+21+57+7+7 = 396

Hence, the average cyclometric complexity per code is **19.8**