TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS

A Project Report
On
Algorithm Visualizer

Submitted By:

Kirtan Kunwar	(THA078BCT19)
Navin Nepal	(THA078BCT23)
Prashant Aryal	(THA078BCT30)
Prateek Poudel	(THA078BCT31)

Submitted To:
Department of Electronics and Computer Engineering
Thapathali Campus
Kathmandu, Nepal

December, 2023

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS

A Project Report On AlgViz

Submitted By:

Kirtan Kunwar	(THA078BCT19)
Navin Nepal	(THA078BCT23)
Prashant Aryal	(THA078BCT30)
Prateek Poudel	(THA078BCT31)

Submitted To:
Department of Electronics and Computer Engineering
Thapathali Campus
Kathmandu, Nepal

December, 2023

ACKNOWLEDGEMENT

This project is prepared in partial fulfilment of the requirement for the the bachelor's degree in Computer Engineering. We owe our deepest gratitude to the Department of Electronics and Computer Engineering IOE, Thapathali Campus for providing us with an opportunity to work on a project as a part of our syllabus. We would also like to owe our gratitude to our OOP teacher, Er.Saroj Sakhya, for his guidance. The experience of working on this project will surely enrich our technical knowledge and also gives experience of working on a project and also develop our team works skills to a great extent.

Kirtan Kunwar	(THA078BCT019)
Navin Nepal	(THA078BCT023)
Prashant Aryal	(THA078BCT030)
Prateek Paudel	(THA078BCT031)

ABSTRACT

Algorithm Visualizer is an GUI based interactive tool developed using C++ SFML that aims to enhance understanding of various search and sort algorithms. Its primary goal is to provide an intuitive platform to visualize and comprehend various algorithms. Developed using the powerful C++ programming language, the project offers a comprehensive toolset for algorithm visualization. Users can observe the step-by-step execution of sorting, searching, and graph algorithms, gaining a deeper understanding of their inner workings with custom animations. With an emphasis on object-oriented programming principles, the project promotes code modularity and reusability.

Keywords: Algorithm Visualizer, SFML, GUI , OOP

TABLE OF CONTENTS

List of Figures

LIST OF ABBREVIATIONS

BFS	Bredth For Search
CPP	C Plus Plus
CPU	Central Processing Unit
DFS	Depth-First Search
GUI	Graphical user interface
OOP	Object-oriented programming
SFML	Simple and Fast Multimedia Library

1. INTRODUCTION

1.1. Background

In the field of computer science, algorithms play a crucial role in solving complex problems and optimizing processes. However, understanding and visualizing these algorithms can often be challenging for students, particularly those in computer software engineering programs. Recognizing the need for a user-friendly and intuitive platform to bridge this gap, the Algorithm Visualizer CPP project was developed. This project focuses on providing students with a comprehensive toolset to visualize and comprehend various algorithms using the powerful C++ programming language. By offering a platform that allows step-by-step execution of sorting, searching, and graph algorithms, the Algorithm Visualizer CPP project aims to enhance students' understanding of the inner workings of these algorithms. Moreover, by promoting code modularity and reusability through an object-oriented programming approach, students can develop efficient and scalable algorithm implementations. With features such as a code editor and debugging environment, the project also empowers students to experiment and refine their algorithms, fostering a hands-on learning experience. Overall, the Algorithm Visualizer CPP project serves as a vital resource for computer software engineering students, enabling them to enhance their algorithmic understanding and problem-solving skills in a visually engaging manner.

1.2. Motivation

The Algorithm Visualizer CPP project is driven by the motivation to enhance students' understanding and comprehension of algorithms. Recognizing the challenges faced in grasping the practical aspects of algorithms, this project offers a user-friendly and interactive platform that provides a visual representation of algorithmic processes. By observing step-by-step executions, students gain valuable insights into the inner workings of sorting, searching, and graph algorithms. The choice of the C++ programming language further strengthens their programming skills and prepares them for real-world applications. Additionally, by promoting code modularity, reusability, and object-oriented programming principles, this project equips students with good coding practices and the ability to write efficient code. Ultimately, the Algorithm Visualizer CPP project aims to bridge the gap between theory and practice, empowering students to become proficient software engineers with a deep understanding of algorithms and problem-solving skills.

1.3. Objectives

1. The primary objective of the Algorithm Visualizer CPP project is to provide students with a user-friendly and intuitive platform to visualize various algorithms, allowing them to gain a deeper understanding of their inner workings through step-by-step execution

2. The project aims to promote the use of the C++ programming language and reinforce programming skills by emphasizing code modularity, reusability, and object-oriented programming principles, enabling students to develop efficient and scalable algorithm implementations.

1.4. Problem statement

The problem statement of the Algorithm Visualizer CPP project is that students in computer software engineering programs often face challenges in comprehending and visualizing algorithms. Traditional textual explanations and theoretical concepts may not adequately convey the practical aspects and visual nature of algorithms, leading to a gap between theory and practice. As a result, students may struggle to fully grasp the inner workings of sorting, searching, and graph algorithms and may find it difficult to develop efficient and scalable algorithm implementations. Therefore, there is a need for a user-friendly software application that provides a visual and interactive platform for students to observe and comprehend algorithms, bridging the gap between theory and practice and enhancing their algorithmic understanding and problem-solving skills

1.5. Scope of Project

The scope of the Algorithm Visualizer CPP project encompasses the development of a comprehensive software application that offers algorithm visualization capabilities to students in computer software engineering programs. The project aims to provide a user-friendly platform that supports the visualization and comprehension of various algorithms, including sorting, searching, and graph algorithms. The project will be implemented using the C++ programming language, leveraging its power and versatility for efficient algorithmic execution. The software application will allow step-by-step execution of algorithms, enabling students to observe and analyze the inner workings of these algorithms in a visual manner. Additionally, the project will emphasize code modularity, reusability, and object-oriented programming principles to promote good coding practices and scalable algorithm implementations. The scope also includes features such as a code editor and debugging environment to facilitate experimentation and refinement of algorithm implementations. The Algorithm Visualizer CPP project aims to bridge the gap between theory and practice, empowering students to enhance their algorithmic understanding and problem-solving skills through an engaging and intuitive software application.

2. LITERATURE REVIEW

Understanding data structures and algorithms is crucial for software engineers to develop efficient software solutions. Algorithm visualizers have proven to be effective tools in facilitating comprehension for users. Over the years, several research papers have highlighted the significance of algorithm visualization systems in educational settings. The 2008 study ”AlCoLab: Architecture of Algorithm Visualization System” focuses on educational script-supported algorithm visualization systems. It emphasizes the assistance and enhancements these systems provide in the instruction of abstract concepts like algorithms. The design of AlCoLab is discussed, including how script support, custom visualizations, step-by-step execution, and algorithm performance monitoring improve teaching and learning.[1]

In the paper ”Open Interactive Algorithm Visualization” from 2019, a project aimed at creating an open and interactive website for algorithm visualization is introduced. Users can study and engage with numerous algorithms on the platform to better their comprehension. It provides a library of algorithm visualizations. The website offers interactive elements, real-time visualization, and additional material while also promoting community participation and cooperation.[2]

The creation of AlgoAssist, a platform that integrates algorithm visualization, coding, and evaluation capabilities, is described in the paper ”AlgoAssist: Algorithm Visualizer and Coding Platform for Remote Classroom Learning” from 2021. It highlights the value of algorithm visualization for improved comprehension and provides students with step-by-step explanations and visualizations. The software is built to assist online instruction, encouraging participation and better understanding of algorithms.[3]

In the 2014 paper, ”Design and Evaluation of a Web-Based Dynamic Algorithm Visualization Environment for Novices,” the authors present a web tool that utilizes interactive visuals and animations to assist beginners in understanding algorithms. The paper evaluates the tool‘s effectiveness in improving beginners’ comprehension and provides insights into its user-friendliness. The web-based platform offers benefits such as making abstract concepts easier to understand and allowing practical exploration. The findings emphasize its relevance in introductory programming courses, demonstrating the potential of web-based dynamic algorithm visualization environments for novice learners.[4]

The ”Algorithm Visualizer” study from 2021 created a web-based platform with six pathfinding algorithms and four sorting algorithms. Pathfinding and sorting algorithms are available on the platform’s navigation bar. To improve student comprehension, the 4 pathfinding section included maze and pattern possibilities. The technology also lets users change the speed of the visualization. Four sorting techniques are offered in this section, and the visualizations used various colors to show sorted and unsorted bars for better understanding.[5]

The 2022 paper, ”Algorithm Visualizer,” describe an e-learning application that visualizes numerous algorithms, including Pathfinder, Prime Numbers, Sorting Algorithms, N Queen, Convex Hull, and Binary Search Game. An interactive UI and algorithm execution are made possible by the tool’s use of HTML5, CSS, and React JS. Its primary objective is to offer educators and students a supportive learning environment that effectively supports learning.[6]

3. METHODOLOGY

Algorithm Visualizer is an interactive tool developed using C++ SFML that aims to enhance understanding of various search and sort algorithms. The primary goal of AlgViz is to demonstrate the mechanics and functionality of algorithms, in a visually appealing manner, allowing users to observe the step-by-step execution and behavior of these algorithms in real-time. By combining a custom animation engine and a user friendly interface, the application provides an immersive learning experience, enabling users to explore different algorithms and gain insights into their mechanics and performance

3.1. Search Algorithm

In computer science, a search algorithm is an algorithm designed to efficiently navigate through the search space by considering possible actions or moves at each step. These actions lead to the exploration of new states, which are evaluated based on specific criteria, such as reaching the target state, minimizing cost, or maximizing a given objective function.

3.1.1. Types

Search algorithms can be categorized into different types based on their strategies and characteristics. Some can work with weighted graphs and some can't, some perform informed search and some perform uninformed search. Some search algorithms that will be implemented in AlgViz are depth-first search (DFS), breadth-first search (BFS), uniform-cost search (UCS), A* search.

Breadth First Search Breadth-First Search (BFS) is a graph traversal algorithm that explores all the vertices of a graph in breadth-first order. It starts at a given source vertex and systematically explores all its neighbors before moving on to the next level of neighbors.

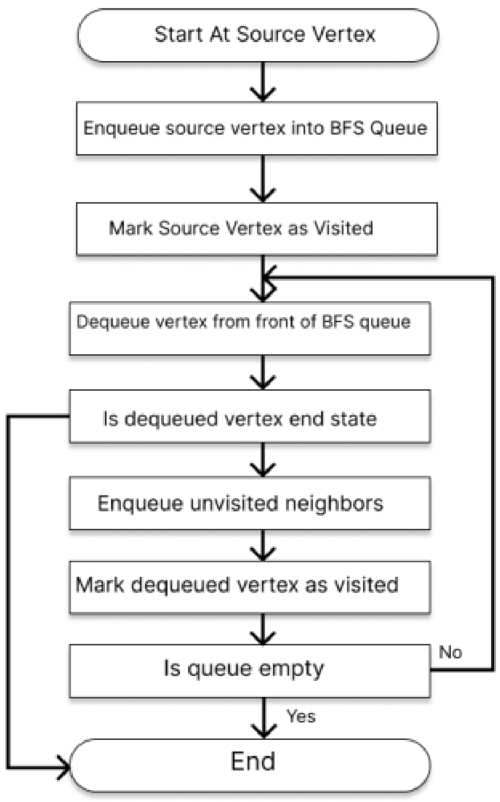
Start At Source Vertex
Enqueue source vertex into BFS Queue
Mark Source Vertex as Visited
Dequeue vertex from front of BFS queue
Is dequeued vertex end state
Enqueue unvisited neighbors
Mark dequeued vertex as visited
Is queue empty
No
Yes
End

Figure 3.1: Flowchart of Breadth First Search Algorith

3.1.2. Graph

Graphs are mathematical structures used to model and represent relationships between objects or entities. They consist of a set of vertices (also known as nodes) connected by edges. In a graph, vertices represent entities, while edges represent the connections or relationships between those entities. The connections between vertices can be directed (one-way) or undirected (two-way), depending on the nature of the relationship being modeled. Graphs play a crucial role in the context of search algorithms as they provide a structured representation of problem spaces. In search algorithms, graphs are used to model the relationships between states or configurations of a problem, allowing for systematic exploration and finding optimal solutions.

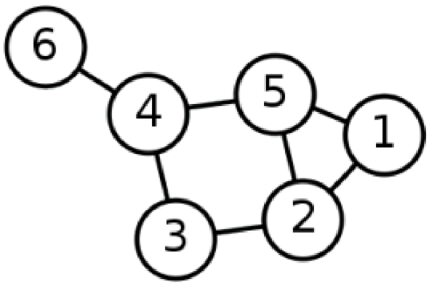

Figure 3.2: Graph

3.1.3. Maze

A maze is just a matrix of nodes where each node is bidirectionally connected to 4 of its surrounding node. In algorithm visualization, a maze represents a challenging puzzle or problem space that needs to be explored and solved. Mazes consist of interconnected passages, walls, and obstacles, creating a network of interconnected nodes and edges. The objective is to navigate through the maze from a starting point to a designated endpoint, following specific rules or constraints

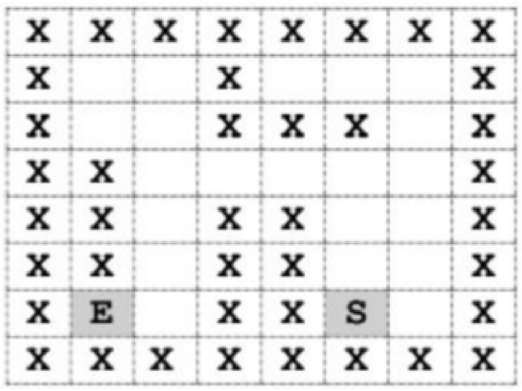

Figure 3.3: Maze

Maze Generation: Maze generation refers to the process of creating a maze, which is a complex network of interconnected paths, walls, and passages. The objective of maze generation algorithms is to produce mazes with certain properties, such as being solvable, having a specific level of complexity, or exhibiting particular patterns.

The recursive division is a maze generation algorithm that works by dividing a grid of nodes or cells into smaller subgrids using a recursive approach. The algorithm starts with an initial grid and randomly selects a division orientation (horizontal or vertical). It then chooses a random row or column to place a wall, creating two new subgrids. The algorithm recursively applies itself to each subgrid until the subgrid dimensions are too small to divide further.

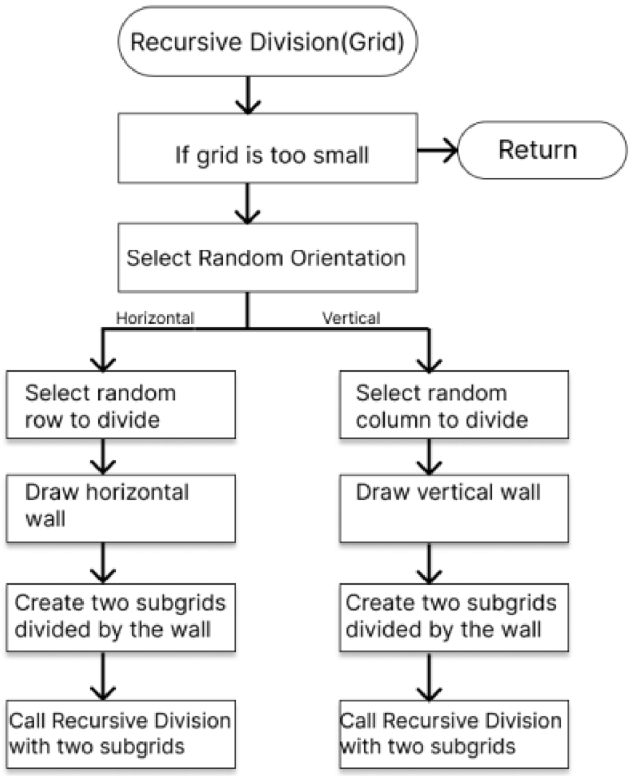
Recursive Division(Grid)
If grid is too small
Return
Select Random Orientation
Horizontal
Vertical
Select random row to divide
Select random column to divide
Draw horizontal wall
Draw vertical wall
Create two subgrids divided by the wall
Create two subgrids divided by the wall
Call Recursive Division with two subgrids
Call Recursive Division with two subgrids

Figure 3.4: Recursive Division Algorithm Flowchart

3.2. Sort Algorithm

In computer science, a sort algorithm is an algorithm designed to arrange elements in a specific order, typically ascending or descending. Sort algorithms work by comparing and rearranging elements based on specific comparison criteria, such as numerical value or lexicographical order. The goal is to efficiently organize the elements in a desired sequence.

3.2.1. Types

Sort algorithms can be categorized into different types based on their strategies and characteristics. Some algorithms have a complexity of O(n 2) and are suitable for small input sizes, while others have a complexity of O(n log n) and are more efficient for larger input sizes. Some search algorithms implemented in AlgViz are Bubble Sort, Insertion Sort, Merge Sort and Quick Sort.

Bubble Sort Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly compares adjacent elements in a list and swaps them if they are in the wrong order. This process is repeated until the entire list is sorted. The main idea behind Bubble Sort is to repeatedly pass through the list, comparing adjacent elements and swapping them if necessary. On each pass, the largest element ”bubbles” up to its correct position at the end of the list. The algorithm continues these passes until the list is completely sorted.

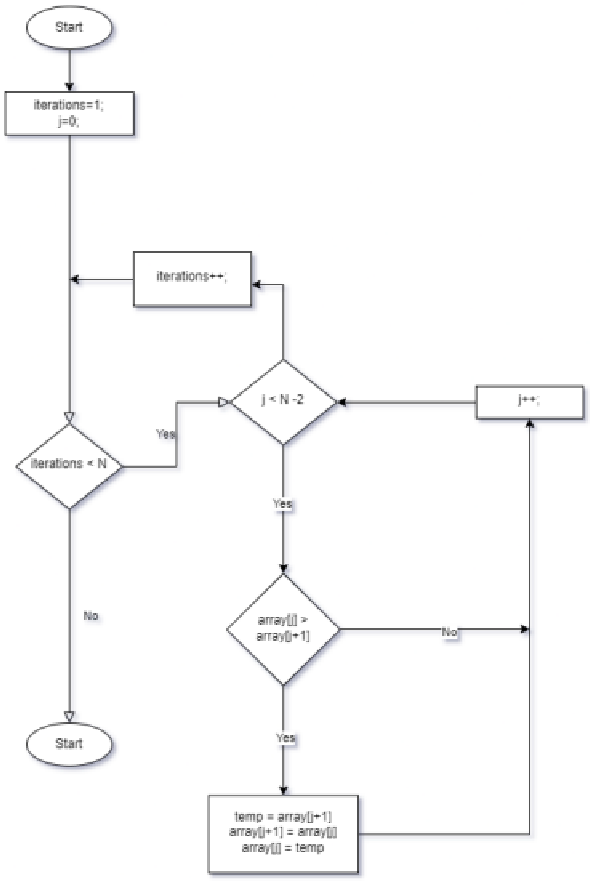

Figure 3.5: Flow Chart of Bubble Sort

3.3. User Interaction

3.3.1. Maze Customization

Maze customization refers to the ability of users to modify certain aspects of the maze, including size, layout, start and end points, and the option to incorporate weighted nodes. With this feature, users can create mazes tailored to their preferences or specific testing requirements. The inclusion of weighted nodes adds an extra dimension to the customization, allowing users to simulate real-world scenarios and analyze the performance of search algorithms that consider varying node weights.

3.3.2. Array Customization

Array customization for sort algorithms refers to the ability of users to customize the input arrays that are used for sorting operations within an application. Users can define the size of the array, choose the values of the elements, generate arrays with specific characteristics, and select the data type of the elements. This customization feature enables users to test sorting algorithms on different data distributions, sizes, and types. 13 They can evaluate the performance, scalability, and behavior of various sorting algorithms in different scenarios.

3.3.3. Algorithm Customization

Algorithm customization refers to the ability of users to modify the behavior and parameters of algorithms within an application. It allows users to have more control and flexibility in how the algorithms operate and can be a valuable feature for algorithm visualization applications. Some customization options are algorithm selection, algorithm parameters and speed and animation.

3.3.4. Optimization

Optimization techniques involve analyzing the algorithms and code base, identifying performance bottlenecks, and applying strategies to enhance efficiency. This includes algorithmic optimization and efficient data structure selection. By optimizing critical sections of the code base and employing efficient algorithms and data structures, the application can deliver faster and more responsive visualizations.

It is important to balance optimization efforts with code readability and maintainability. Over-optimization can lead to code complexity and decreased maintainability. Therefore, optimization should be performed in an iterative way, guided by thorough analysis and profiling results.

3.3.5. Concurrency

Concurrency in C++ is a powerful technique that allows multiple tasks or computations to be executed concurrently, enabling efficient utilization of system resources such as CPU cores. 14 By leveraging concurrent programming techniques, such as multi threading, the application can utilize the available computational resources more efficiently. Concurrency allows for the execution of multiple tasks simultaneously such as loading.

A ThreadPool Class is used to create and manage threads.

4. SYSTEM DESCRIPTION

The AlgViz program works on the object oriented procedure of programming. It uses libraries such as SFML to load and manipulate graphics and visualize various sort and search algorithm. The architecture of this program is mainly a App class that consists of all the other classes and can connect and manage the whole program. Similarly we have separate classes for the screens i.e Search , Home and Sort where we create an instance of App class and pass the main App object using constructor. By this way we have App class in all the screen and they can access App class and App class has all the other classes which allows us to access all the classes from each other. The main classes in the AlgViz application are given below.

(In the UML diagram below, 2d line represents inheritance)

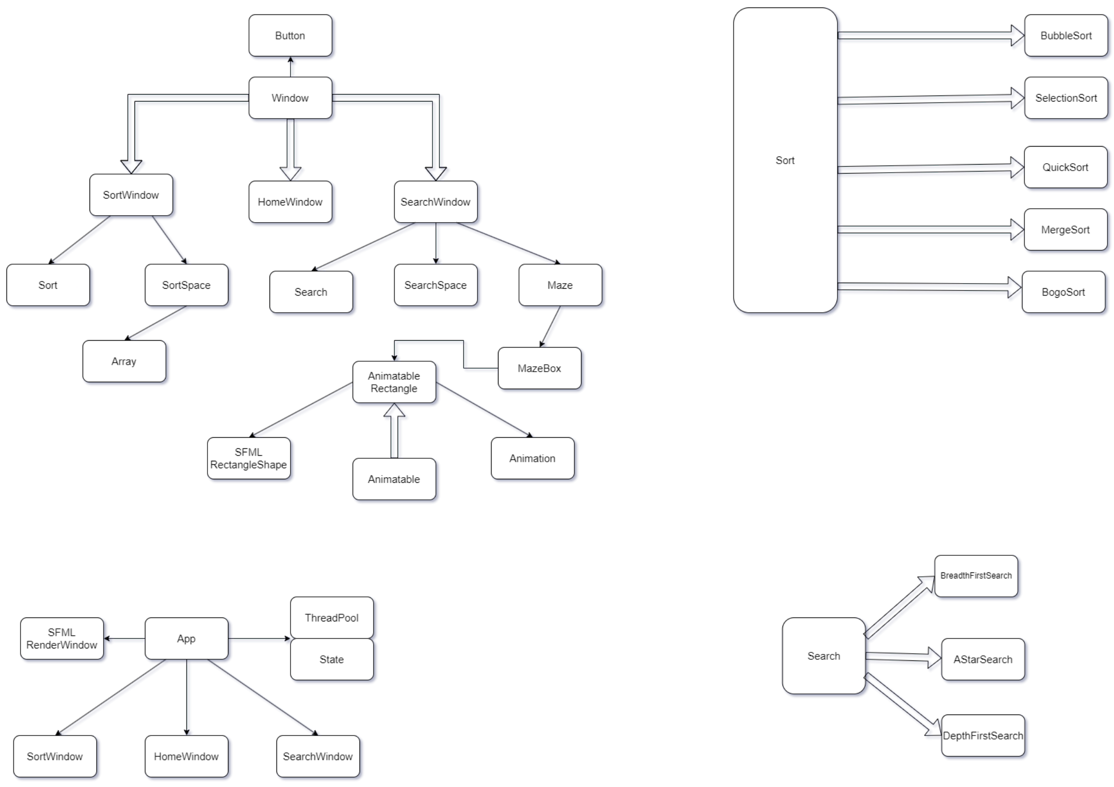

Figure 4.1: UML Diagram

4.1. Window Class

The Window class serves as a pivotal component within the application, representing an individual interface page. Its abstract nature lies in its foundational role, enabling the creation of diverse window types like Home, sorting, and search windows by mandating the implementation of essential methods—update, draw, and user input handling—by derived classes. The update method dynamically refreshes the Window and its internal components, including buttons, mazes, and object animations, ensuring real-time updates and responsiveness. In parallel, the draw method orchestrates the rendering of the Window onto the screen, maintaining a coherent visual representation of its current state. Simultaneously, the user input handling method interprets and manages various user interactions, such as mouse movements and clicks, facilitating a seamless and intuitive user experience by responding to these inputs.

4.2. SearchWindow Class

The SearchWindow class, inheriting from Window, manages a window dedicated to search algorithms in the application. It handles maze visualization, search space, and various algorithms. With methods for button setup, state changes, and drawing, it incorporates components like Maze, SearchSpace, and multiple Search algorithms. Users can select and visualize different algorithms via this window. This class includes functions for resetting, drawing content, updating interactions, and managing state changes. Its constructor defines the window's size, title, and a callback to return to the main menu.

4.3. SortWindow Class

The SortWindow class, extending Window, manages a window specifically for sorting algorithms in the application. It controls the visualization of sorting space, bars, and various sorting algorithms. The class handles button setup, state changes, and drawing on the window. It includes components like SortSpace and multiple Sort algorithms stored in a map. Users can select and visualize different sorting methods within this window. Additionally, it handles resetting, content drawing, interactions updating, and state changes. The constructor sets the window's size, title, and a callback to return to the main menu.

4.4. App Class

The App class acts as the core controller of the application, overseeing and coordinating different windows like Search, Home, and Sort. It maintains a RenderWindow for displaying graphics, manages time using Clock, and handles user inputs such as mouse clicks and movements through the handle state method. Within this class, specific instances for distinct windows—search_window, home_window, and sort_window—belonging to their respective classes (SearchWindow, HomeWindow, SortWindow) enable seamless transitioning between various functionalities or screens. The draw() function handles the visual representation of elements on the screen, ensuring their proper display, while update() oversees the necessary changes or movements within the app. The run() function serves as the primary initiator and executor of the application, orchestrating its entire functionality. By organizing different app components and managing their interactions, this class ensures a unified user experience while efficiently handling graphics, time, and user interactions across multiple windows.

4.5. Search Class

The Search class manages search algorithms, handling a search space and controlling step delays. It includes flags for running, backtracking, and tracks the current backtracked box. The constructor initializes these parameters. Methods execute single steps of the search algorithm and backtracking, start the search, and offer setters/getters for step delay and search status. Virtual functions enable algorithm updates and state resets for adaptability across different search algorithm implementations.

4.6. SearchSpace Class

The SearchSpace class manages search space properties for algorithms within the application. It holds dimensions and pointers to a maze, along with vectors for parent, visited, and explored statuses of boxes in the maze. The constructor initializes these properties. Getters retrieve parent information, visited status, maze dimensions, maze pointer, start and goal boxes. Setters modify parent relationships, set visited, explored, or path statuses of boxes. Additional methods fetch valid actions from a box and reset the search space.

4.7. Maze Class

The Maze class manages the visual representation of a maze for search visualization purposes in the application. It maintains box dimensions, maze boxes, start and goal box positions, along with their dimensions. The class handles box animation updates, drawing on the window, setting box types, event handling, and state changes. It provides methods for getting, setting, resetting, and clearing box attributes like start, goal, walls, and searched boxes. Operators [], (), and at() allow access to boxes within the maze. Additionally, the class includes an inline function for determining the box the mouse is hovering over within the maze.

4.8. MazeBox Class

The MazeBox class represents individual boxes within a maze for visualization in the application. It includes functionalities for defining box types, their colors, animations, and their appearance on the screen. The class maintains an animatable rectangle for the box, incorporating animations for searching and marking as found in the path. Static arrays hold colors for various box types. Constructors define box properties, while getters and setters manage box types, their drawable elements for screen display, and position setting. Methods enable animation updates and checks for ongoing animations.

4.9. Animatable Classes

The Animation class defines attributes for object animation, such as displacement, color, time, scale, and animation behavior, allowing customization of animations.

The Animatable class, acts as a base for animated objects, managing animation states, setting animations, starting animations, updating frames, and obtaining drawable objects. Derived classes can implement specific functionalities for their drawable objects.

The AnimatableRectangle class, derived from Animatable, handles animated rectangles (sf::RectangleShape). It stores initial rectangle properties like position, dimensions, and color, offering methods to control animation, adjust position, reset animation states, update frames, obtain drawable objects, start animation, and modify rectangle color.

4.10. Sort Class

The Sort class in the viz::sort namespace manages sorting methods for visualization. It contains functionalities for controlling sorting steps, such as initiating a single step, updating progress, checking completion status, modifying delay, resetting, and starting the sorting process. This class serves as a base for implementing various sorting algorithms, each handling its specific sorting logic.

4.11. SortSpace Class

The SortSpace class manages the properties and functionalities of the sorting visualization space. It encapsulates attributes like the starting position, minimum and maximum box heights, box width, the number of boxes, box heights vector, background box, and bar offset for rendering. Its constructor initializes these properties, while methods within the class handle adjustments to the number of boxes, randomization of box heights, and drawing of boxes on the window, serving as the space manager for sorting algorithm visualization.

4.12. Button Class

The Button class defines properties and functionalities for creating interactive buttons in a graphical user interface. Its constructor initializes the button attributes such as position, dimensions, text, font, fill colors for normal and hover states, and a callback function. The class includes a rectangle shape, text element, and various boolean flags to track button states like being pressed or hovered over. Methods within the class enable functionality to detect mouse presence over the button, handle mouse state changes, draw the button on a window, and update its behavior. This class provides an interface for creating interactive buttons in graphical applications.

4.13. State Class

The State class operates as a singleton, serving as a centralized storage entity responsible for managing the essential states within the application's visualizations. It encapsulates instances of search_state for search visualization states, Mouse to track mouse-related data, and sort_state for sorting visualization states. Utilizing the singleton pattern ensures the existence of only one instance of this class throughout the application's lifespan, allowing global accessibility and synchronized access to critical state information across diverse modules or components. Additionally, the State class provides a method, update_mouse, enabling the seamless updating of mouse positions, essential for handling mouse interactions across various visualization components, ensuring coherent state management across the entire application.

4.14. ThreadPool Class

The ThreadPool class is a Singleton system managing multiple threads for executing asynchronous tasks. It coordinates worker threads to process queued jobs, ensuring thread safety through synchronization mechanisms like mutexes and condition variables. By maintaining a single instance and providing methods to submit tasks (queue_job()), check workload status (busy()), and stop thread operations (stop()), this system enhances performance by facilitating parallel task execution, especially useful for concurrent processing or asynchronous operations within the application.

5. RESULTS AND ANALYSIS

5.1. Menu

The presented home page or menu page features three buttons, each serving a distinct purpose. The "Search Algorithm" button is designed to open a dedicated window for exploring search algorithms. Similarly, the "Sort Algorithm" button directs users to a specialized window for sorting algorithms. Lastly, the "Exit" button provides a straightforward means to close the window and exit the program. This user-friendly interface is intended to facilitate efficient navigation and enhance the overall user experience within the program.

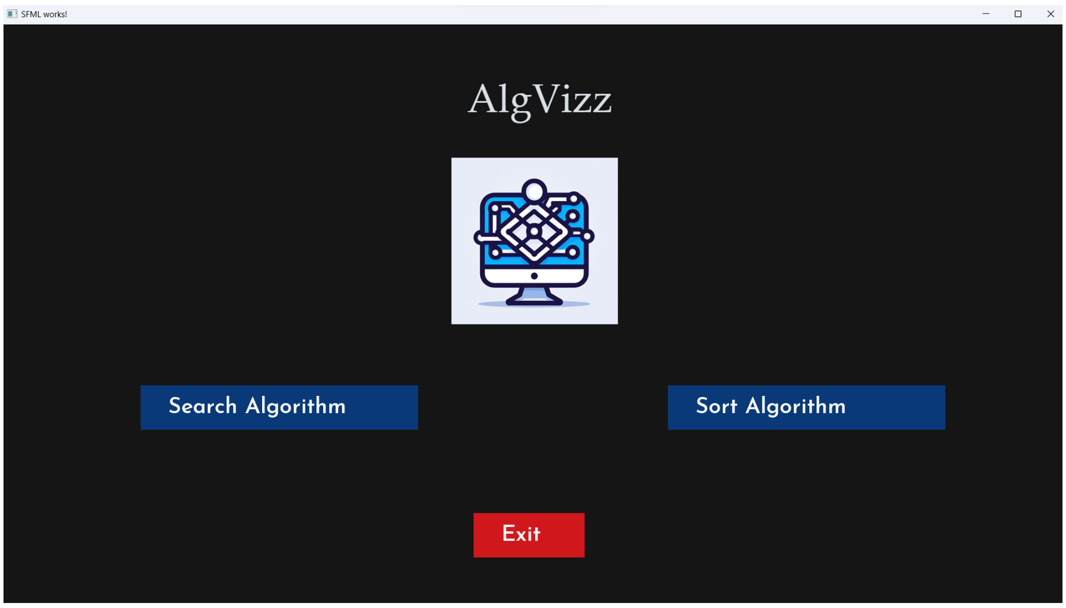

Figure 5.1: Start Page

5.2. Search Algorithm

The Search Algorithm window is equipped with various essential features to facilitate a comprehensive exploration of search algorithms. It includes a "Search" button to initiate the algorithm, a "Maze" button enabling users to create a maze, and options to specify the "Start Point" and "Destination" for the search. The "Clear" button is available for efficiently clearing the screen, while the "Generate Maze" option allows users to generate a random maze. Additionally, users can choose from different algorithm options through specific buttons tailored for each algorithm they wish to visualize. To streamline navigation, there is a convenient "Back" button, providing users with the option to return to the home page or menu page.

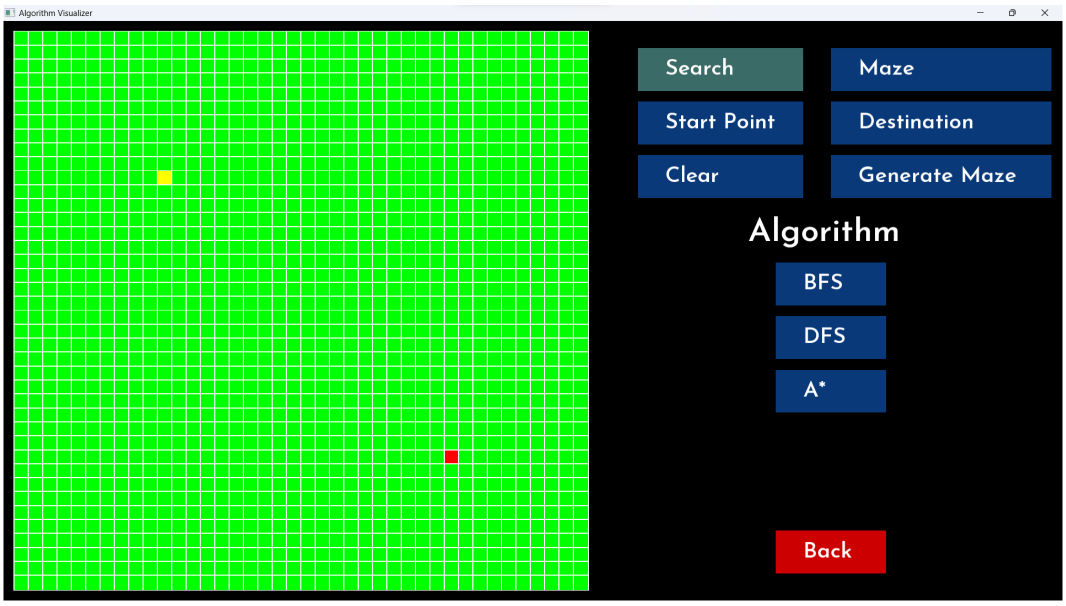

Figure 5.2: Search Page

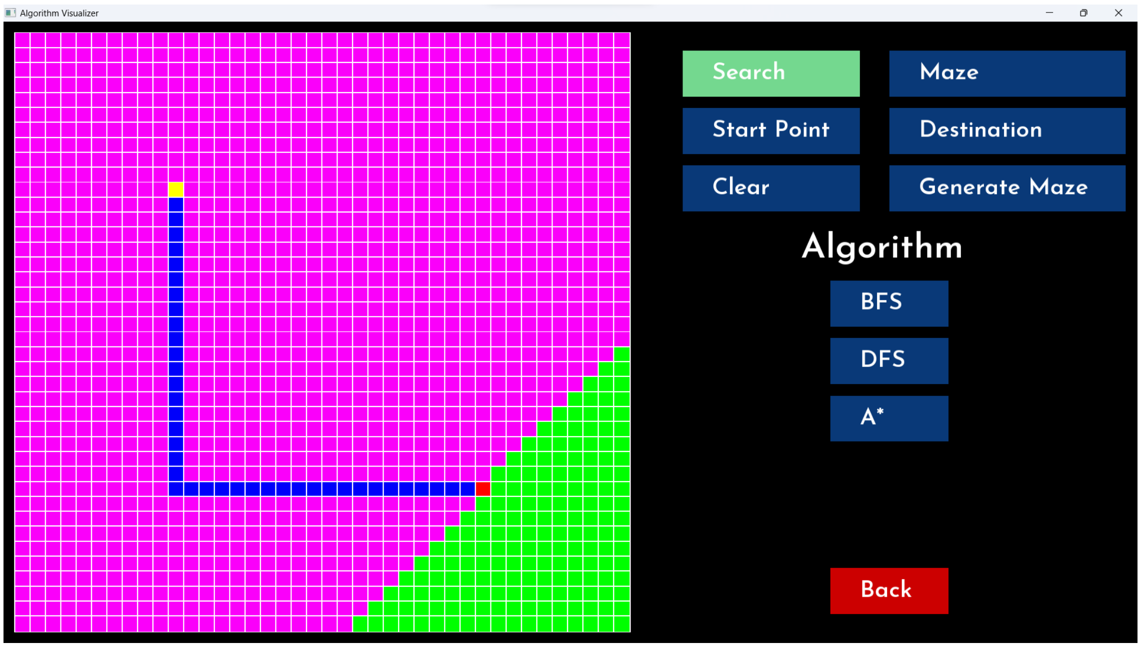

Figure 5.3: Searching Complete with BFS

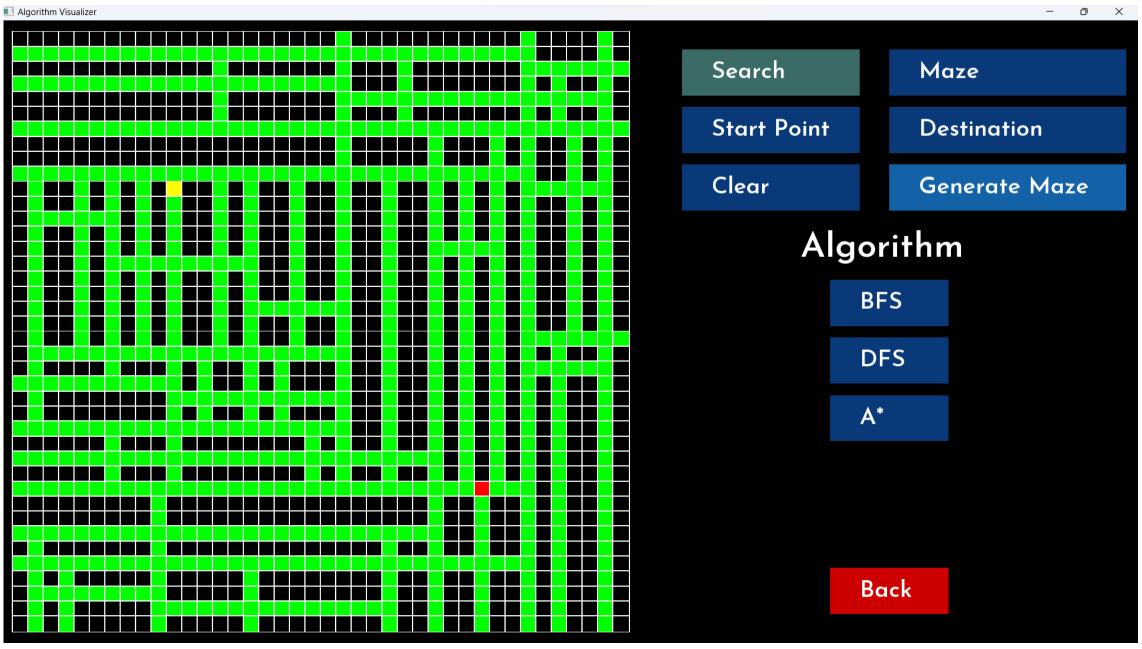

Figure 5.4: Maze Creation

5.3. Sort Page

The Sort Page serves as a specialized interface for showcasing sorting algorithms, equipped with a ”Start” button that initiates the sorting process and a ”Reset” button for efficiently resetting the array to its initial state. Users benefit from the flexibility to choose from a range of sorting algorithms, including bubble, insertion, quick, merge, and radix.

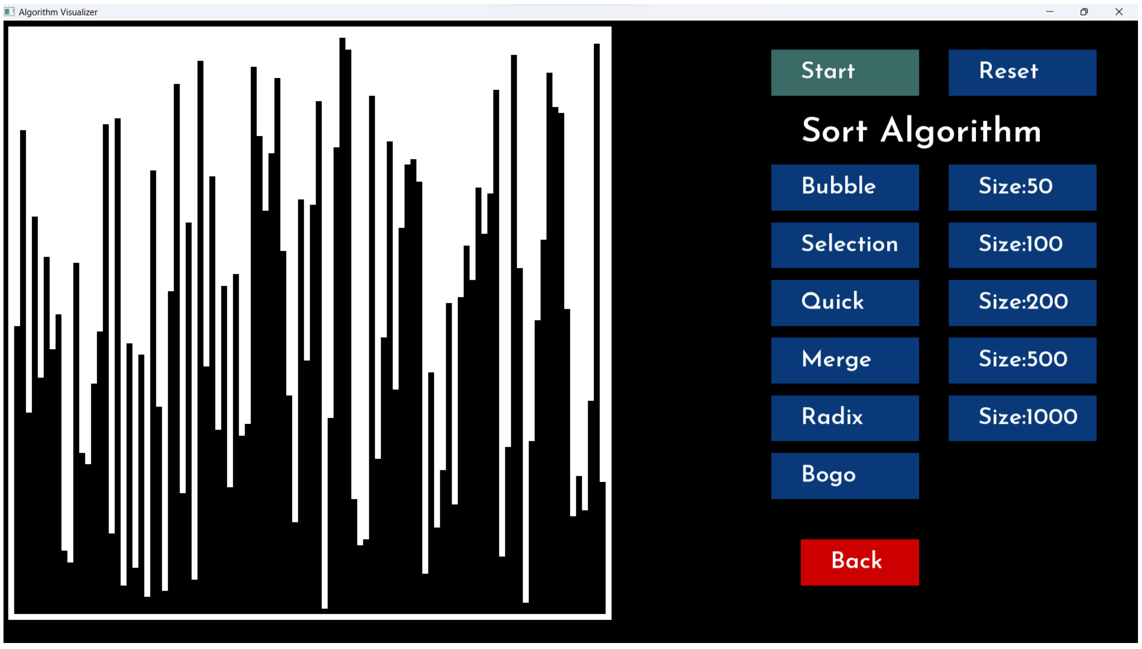

Figure 5.5: Sorting Page

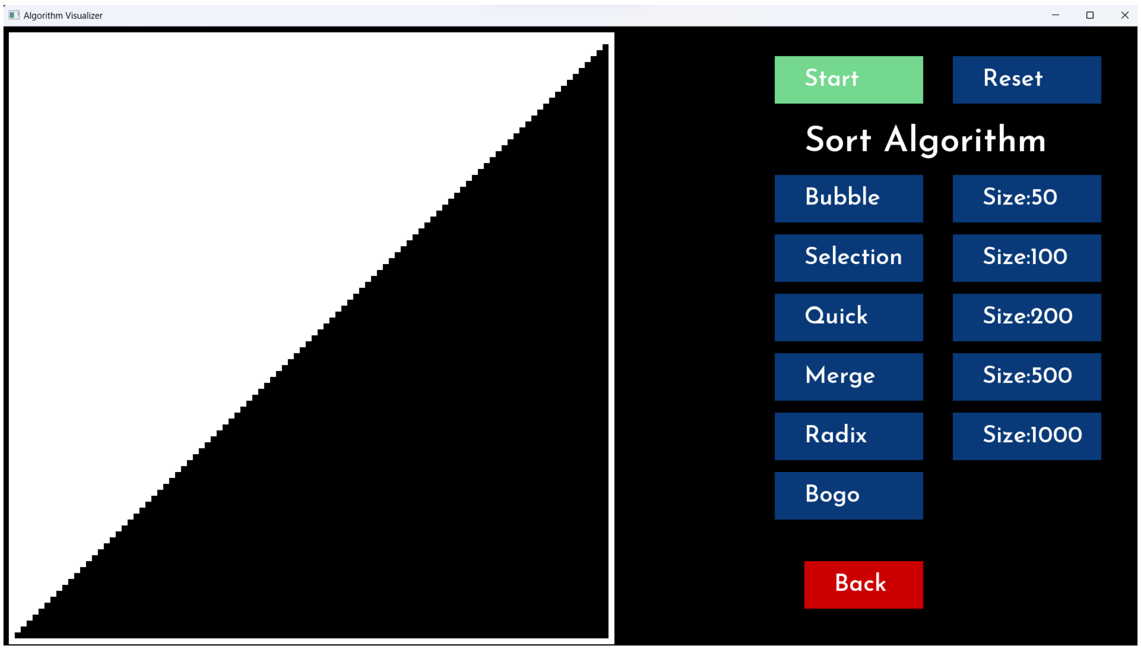

Figure 5.6: Complete Sorting

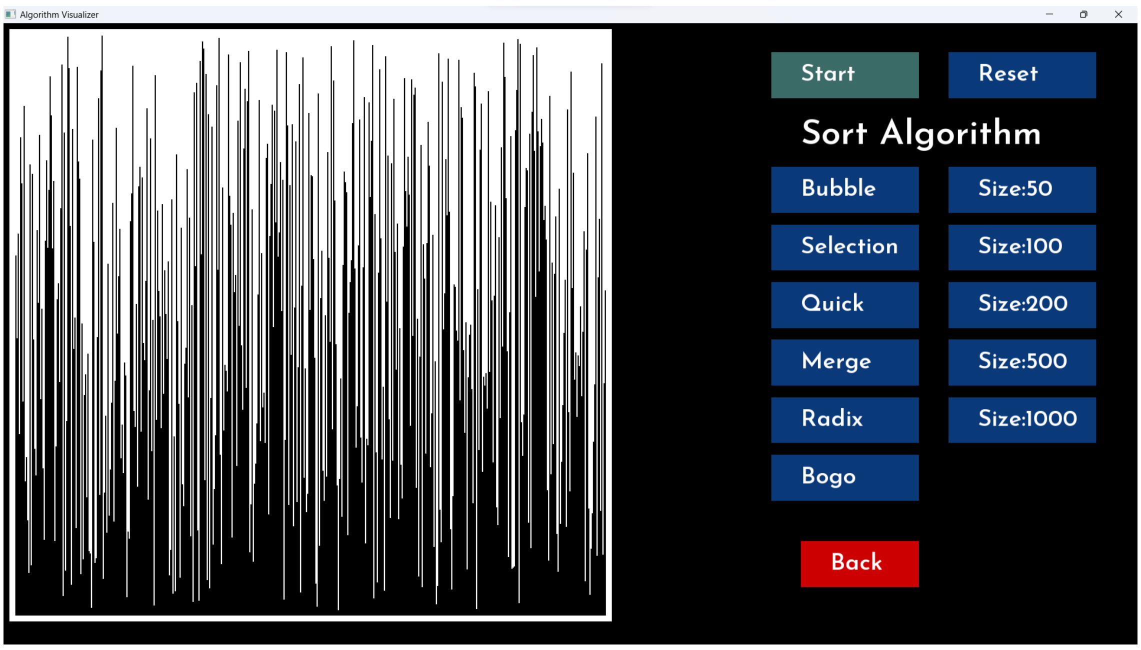

Figure 5.7: Array Customization

6. CONCLUSION AND FUTURE ENHANCEMENT

6.1. Conclusion

In conclusion, our project successfully navigated the realms of searching and sorting algorithms, yielding a potent algorithm visualizer with SFML and C++. Along with algorithmic insights, this project increased our understanding of Object-Oriented Programming (OOP) fundamentals greatly. Encapsulating algorithms within classes exemplified the practical application of OOP ideas, promoting code modularity and reusability. Simultaneously, hands-on experience with SFML not only aided in the production of visually appealing displays, but also demonstrated the adaptability of our newly gained skills.

The effect of this research goes beyond algorithmic exploration, stressing the integration of OOP and SFML in software development. The visuals not only allowed for a more detailed comparative examination of methods, but they also improved our ability to create scalable and maintainable software solutions. Looking ahead, the experience gained via this project prepares us well for handling different software development difficulties and making important contributions to future projects.

6.2. Limitations

1. It's challenging to comprehensively cover all algorithms, potentially excluding niche or advanced ones.
2. Visualizations may struggle to capture dynamic changes, particularly for algorithms with frequent data rearrangements or real-time updates.
3. The project's functionality can be influenced by the capabilities and limitations of external libraries, such as SFML.
4. The visualizer might consume significant system resources, limiting its usage on machines with lower specifications.
5. Visualizing large datasets or complex algorithms may result in performance issues affecting the application's responsiveness.
6. Compatibility and platform-specific issues may arise, affecting the portability and usability across different operating systems.
7. Users may have constraints in exploring a wide range of algorithm parameters, restricting the exploration of algorithmic behavior under different conditions.

6.3. Future Enhancement

1. Allow users to input custom data and choose different input sizes for algorithms.
2. Display real-time statistics such as comparisons, swaps, and execution time during visualization.
3. Implement user controls for pausing, resuming, and stepping through algorithm execution with speed adjustments.
4. Enhance visual appeal with color-coded elements and animations for better engagement.
5. Expand algorithm options to include bidirectional search, Dijkstra's algorithm etc.
6. Introduce weighted cells in maze

7. APPENDIX

7.1. User Manual

7.1.1. Home Page

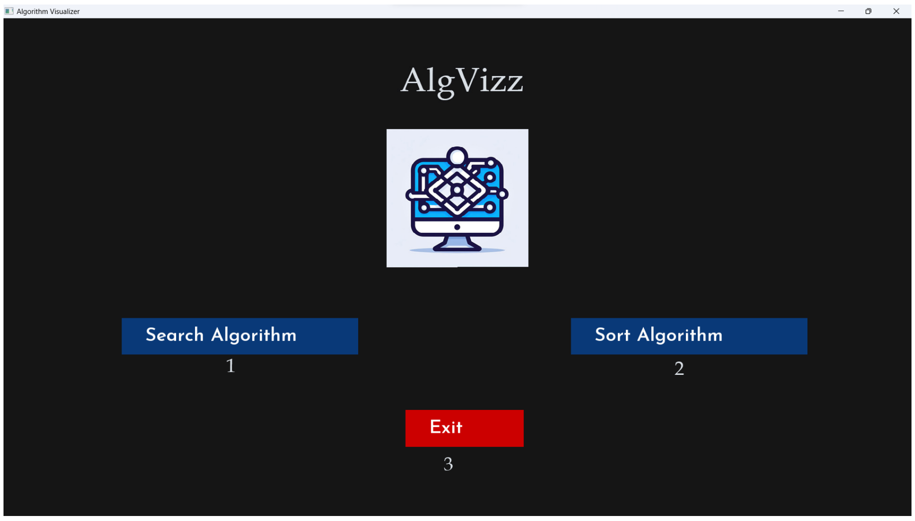

Figure 7.1: Home Page Description

1. Click it to go to Searching Algorithm page.
2. Click it to go to Sorting Algorithm page.
3. Click it to exit the program.

7.1.2. Search Page

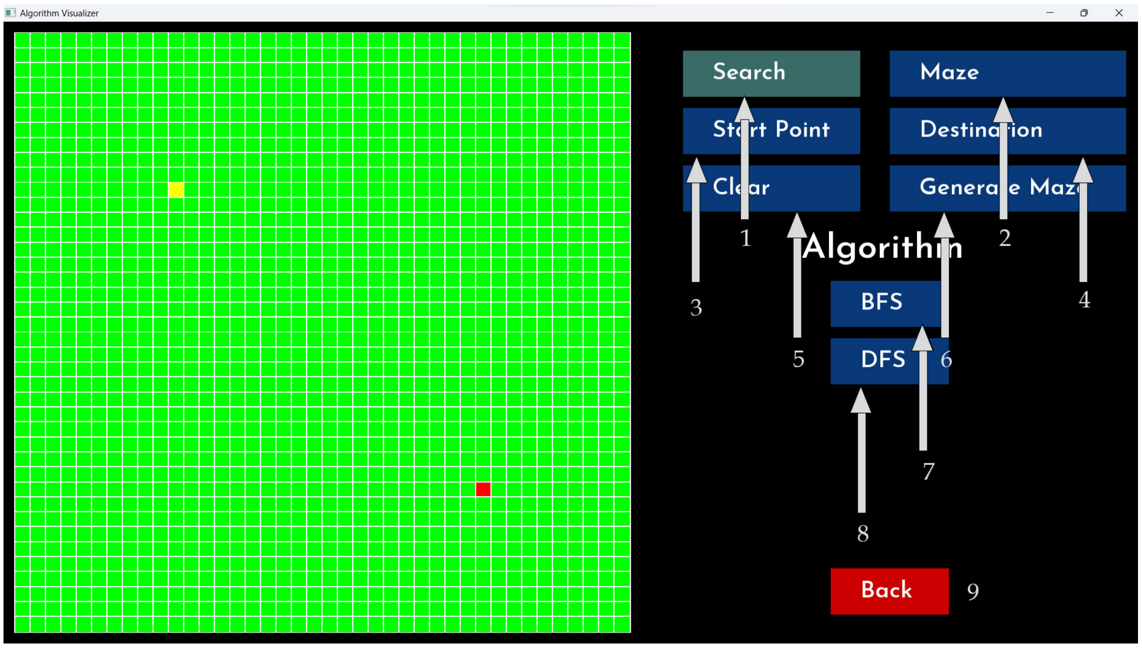

Figure 7.2: Search Page Description

1. Use it to start the Searching algorithm.
2. Select it and draw the maze in the region.
3. Select it and click in any box to set it as start point.
4. Select it and click in any box to set it as Destination Point.
5. Click it to clear the screen and clear the maze.
6. Click it to Generate a random maze.
7. Click it to choose Breadth-for-search Algorithm.
8. Click it to choose Depth-for-search Algorithm.
9. Click it to go back to menu page.

7.1.3. Sort Page

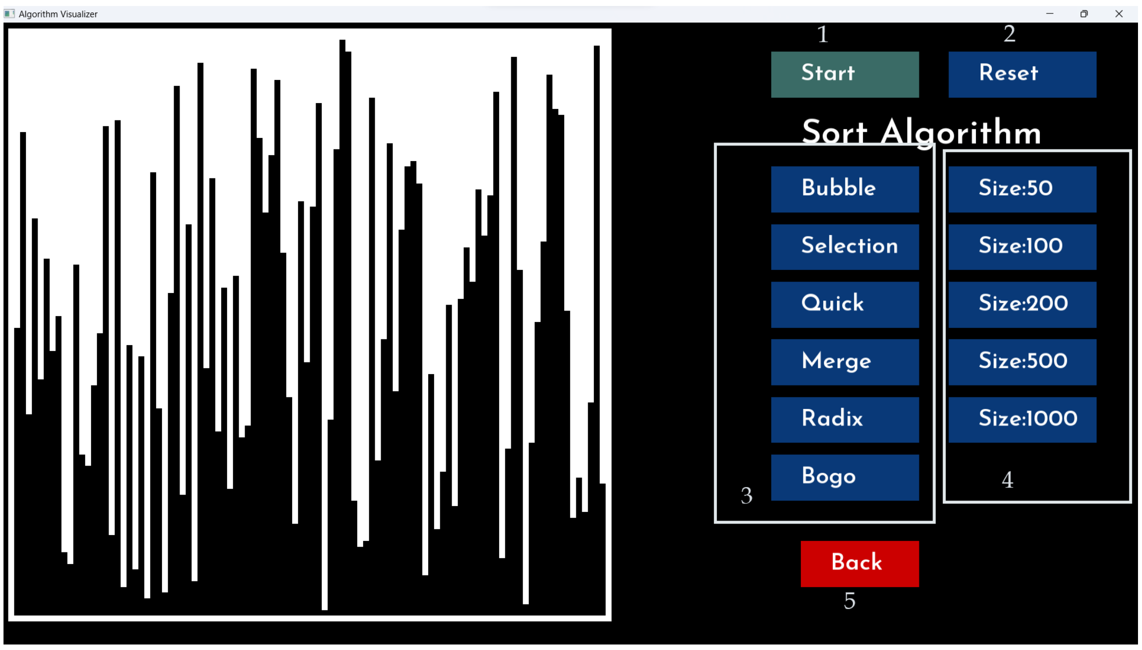

Figure 7.3: Search Page Description

1. Click it to start the Sorting algorithm.
2. Click it to reset the array of bars.
3. Click any to your choice of algorithm.
4. Click any of your choice for array length.
5. Click it to return back to the menu.

References

[1] C. Foutsitzis and S. Demetriadis, “Alcolab: Architecture of algorithm visualization system,” in *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, 2008, pp. 172–174. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4561658

[2] F. Lin, A. Dewan, and V. Voytenko, “Open interactive algorithm visualization,” in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, 2019, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/9465503

[3] A. B. Ghandge, B. P. Udhane, H. R. Yadav, P. S. Thakare, V. G. Kottawar, and P. B. Deshmukh, “Algoassist: Algorithm visualizer and coding platform for remote classroom learning,” in *2021 5th International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 2021, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9465503

[4] E. Vrachnos and A. Jimoyiannis, “Design and evaluation of a web-based dynamic algorithm visualization environment for novices,” *Procedia Computer Science*, vol. 27, pp. 229–239, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050914000283

[5] B. Goswami, A. Dhar, A. Gupta, and A. Gupta, “Algorithm visualizer: Its features and working,” in *2021 IEEE 8th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2021, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9667586

[6] “Algorithm visualizer,” in *International Research Journal of Modernization in Engineering Technology and Science.*, IRJMETS Journal, 2022, pp. 2–5. [Online]. Available: https://www.irjmets.com/uploadedfiles/paper/issue4april2022/20840/final/finirjmets1650114363.pdf