B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

# PATHFINDER

By
Natan Trosman

for
Mr. Joseph Corr

April 23, 2023

## Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

# Contents


**1 Introduction** **2**
- 1.1 Overview . . . 2
  - 1.1.1 Chapter Two . . . 3
  - 1.1.2 Chapter Three . . . 3
  - 1.1.3 Chapter Four . . . 3
  - 1.1.4 Chapter Five . . . 4
  - 1.1.5 Chapter Six . . . 4
- 1.2 Background . . . 4
- 1.3 Pathfinding Visualizer Tools and Websites . . . 5

**2 Methodology** **7**
- 2.1 Agile . . . 7
- 2.2 Technology Stack . . . 8
- 2.3 User Interface Design . . . 8
- 2.4 Implementation of Pathfinding Algorithms . . . 9
- 2.5 Testing and Evaluation . . . 9

**3 Technology Review** **11**
- 3.1 Overleaf . . . 11
- 3.2 LaTeX . . . 11
- 3.3 GitHub . . . 12
- 3.4 JavaScript . . . 12
- 3.5 Visual Studio Code . . . 13
- 3.6 Web Development . . . 13
- 3.7 React . . . 13
- 3.8 Bootstrap . . . 14
- 3.9 Pathfinding Algorithms . . . 14
- 3.10 Node.js . . . 15
- 3.11 Dijkstra's Algorithm . . . 15
- 3.12 A* Algorithm . . . 15
- 3.13 Breadth-First Search Algorithm . . . 16
- 3.14 Depth-First Search Algorithm . . . 16
- 3.15 Data Structures . . . 17
- 3.16 Grid . . . 17

**4 System Design** **19**
- 4.1 System Architecture . . . 19
- 4.2 Front-end . . . 19
- 4.3 Back-end . . . 20
- 4.4 UML Diagram . . . 20
- 4.5 Components . . . 21
  - 4.5.1 Grid Component . . . 21

4.5.3 Algorithm Component . . . . . 22
4.5.4 Sidebar Component . . . . . 22
4.5.5 Main Component . . . . . 22
4.5.6 Interaction between Components . . . . . 22
4.6 Code Snippets . . . . . 23

**5 System Evaluation** **27**
5.1 Functionality . . . . . 27
5.2 Usability . . . . . 28
5.3 Reliability . . . . . 28
5.4 Performance . . . . . 29
5.5 Security . . . . . 29
5.6 Strengths and Weaknesses . . . . . 29
5.7 User Feedback and Evaluation . . . . . 30
5.8 Future Improvements . . . . . 31

**6 Conclusion** **33**
6.1 Overview . . . . . 33
6.2 Design and Implementation . . . . . 33
6.3 Evaluation . . . . . 34
6.4 Strengths and Weaknesses . . . . . 34
6.5 Conclusion . . . . . 34

# Chapter 1

# Introduction

## 1.1 Overview

Chapter 1 of this thesis is all about the fascinating world of pathfinding algorithms[1]. These algorithms are the backbone of computer science and play a critical role in various fields such as logistics, game development, and robotics, among others. They are designed to find the shortest path between two points and ensure efficiency and accuracy in many applications. As technology continues to evolve, the demand for intuitive and interactive tools to help users understand and visualize these algorithms is increasing.

The chapter highlights the long history of pathfinding algorithm development, dating back to the earliest days of computing. In the 1950s and 1960s, researchers began exploring the idea of using algorithms to find the shortest path between two points on a graph. One of the earliest and most well-known algorithms in this area is Dijkstra's algorithm[2], developed by Dutch computer scientist Edsger W. Dijkstra in 1956. Dijkstra's algorithm was designed to find the shortest path between two nodes in a weighted graph and is still widely used today. Since the development of Dijkstra's algorithm, many other pathfinding algorithms have been developed, each with its strengths and weaknesses.

The development of pathfinding algorithms has led to significant advances in various fields. In logistics, pathfinding algorithms are used to optimize delivery routes, reduce transportation costs, and improve efficiency. In game development, pathfinding algorithms are used to create non-player characters (NPC's) that can navigate complex environments. In robotics, pathfinding algorithms are used to help robots navigate unknown environments and avoid obstacles.

Despite the importance of pathfinding algorithms, many people find them challenging to understand and visualize. This is where pathfinding visualizers come in. Pathfinding visualizers are tools that allow users to experiment with different

pathfinding algorithms and see how they work in real-time. These tools provide an interactive interface that makes it easier for users to understand and visualize the algorithms, which can be particularly helpful for students and developers who are just starting to learn about pathfinding.

The thesis presents a comprehensive guide to the development of a pathfinding visualizer called Pathfinder **(Github Link)**. It uses Node.js and JavaScript and is designed to help users learn and experiment with different pathfinding algorithms. The goal of this thesis is to provide a practical and accessible guide to the development of a pathfinding visualizer using Node.js and JavaScript. The visualizer will be designed to be intuitive and interactive, allowing users to experiment with different pathfinding algorithms and see how they work in real-time. The thesis covers the entire development process, from selecting the technology stack to testing and evaluating the performance of the app.

### 1.1.1 Chapter Two

Chapter Two of my thesis focuses on the methodology used to develop Pathfinder. Throughout this chapter, I analyze the implementation of pathfinding algorithms, the design of the user interface, and the thorough testing and evaluation of the app. Overall, this chapter provides valuable insights into the development process of pathfinding visualization tools, and the various challenges that may arise in the process.

### 1.1.2 Chapter Three

In Chapter Three, we take a closer look at the technology stack utilized in the development of the pathfinding visualizer tool. The chapter focuses on web development techniques such as React, Bootstrap, and other relevant technologies. Additionally, data structures and their implementation are discussed in detail, providing readers with a comprehensive understanding of the underlying technology behind the app.

### 1.1.3 Chapter Four

Chapter Four of this thesis presents the system design of the app, providing detailed UML diagrams and system architecture diagrams. This chapter is crucial in understanding how the app functions and how its various components interact with each other. The system architecture also provides a blueprint for further development and improvements, making it an essential resource for any developer looking to build upon this project.

### 1.1.4 Chapter Five

In Chapter Five, we evaluate Pathfinder, presenting the results of the testing and evaluation of the app, including its performance, usability, and effectiveness. This evaluation provides valuable feedback for further development and improvements, ensuring that the app continues to evolve and remain relevant.

### 1.1.5 Chapter Six

Chapter Six concludes the thesis by summarizing the key findings and discussing the significance of Pathfinder in the context of pathfinding algorithms and web development. The chapter also explores the potential for further development and applications of the app in other fields. This thesis provides a comprehensive understanding of pathfinding algorithms, their implementation, and the development of a pathfinding visualizer tool, making it a valuable resource for readers at any level.

## 1.2 Background

In today's world, pathfinding algorithms are ubiquitous and have various applications in a wide range of fields, including logistics, game development, robotics, and more. These algorithms are designed to find the shortest path between two points on a graph while taking into account any obstacles or barriers in the way. One of the most popular pathfinding algorithms that has stood the test of time is Dijkstra's algorithm, which was first introduced by the brilliant Dutch computer scientist Edsger W. Dijkstra in 1956.

Dijkstra's algorithm is an algorithm that works by maintaining a list of unvisited nodes and their distances from the starting node. The algorithm then selects the node with the smallest distance and explores all its neighbors, updating their distances if a shorter path is found. This process is repeated until the algorithm reaches the target node. Another popular pathfinding algorithm that has gained widespread adoption is the A* search algorithm.

A* search is a heuristic extension of Dijkstra's algorithm that uses a cost function to guide the search towards the target node. The algorithm maintains two lists of nodes: open and closed. The open list contains unexplored nodes, while the closed list contains nodes that have already been explored. The algorithm selects the node with the lowest cost function f(n) equals g(n) plus h(n), where g(n) is the cost of the path from the start node to n, and h(n) is the estimated cost from n to the target node. The algorithm then explores all the neighbors of the selected node, updating their costs and adding them to the open list if they are not already in the closed list.

Other popular pathfinding algorithms include [3]Breadth-First Search (BFS), Depth-First Search (DFS), and the Bidirectional Search algorithm. BFS and DFS are simple algorithms that explore the graph in a systematic manner. BFS explores all nodes at a given depth before moving to the next depth, while DFS explores as far as possible along each branch before backtracking. The Bidirectional Search algorithm works by running two searches simultaneously, one from the start node and one from the target node, until the two searches meet in the middle.

Each of these algorithms has its strengths and weaknesses, and the choice of algorithm depends on the specific application and requirements. Therefore, understanding the different pathfinding algorithms and their applications is essential for developers and researchers to design and implement efficient and effective solutions to various real-world problems.

## 1.3 Pathfinding Visualizer Tools and Websites

The world of pathfinding visualizer tools and websites is vast and diverse, with a plethora of options available to users who wish to experiment with different algorithms and see how they work in real-time. These tools provide an interactive interface that allows users to gain a better understanding of the mechanics behind pathfinding algorithms, and each tool comes with its unique features and functionality.

One of the most popular pathfinding visualizer tools is Pathfinding.js, a powerful JavaScript library that offers several different pathfinding algorithms, including Dijkstra's algorithm and A* search. This open-source library has a simple API, making it easy to integrate into other projects and providing developers with a range of customization options.

VisuAlgo is another popular pathfinding visualizer tool that provides a range of interactive visualizations for different algorithms, including pathfinding algorithms. Its highly customizable interface allows users to adjust the speed and color of the visualizations and provides in-depth explanations of each step in the algorithm.

For game developers, the Pygame library provides a range of tools to create interactive pathfinding simulations, adding an extra layer of complexity to the pathfinding process.

Node.js and JavaScript are essential technologies in web development and visualization, providing developers with the tools to create dynamic and interactive websites. With an increasing demand for intuitive and interactive tools to help users understand and visualize pathfinding algorithms, these technologies have become even more critical in recent years.

Pathfinder was developed using Node.js and JavaScript, with the aim of provid-

ing an intuitive and interactive tool for users to learn and experiment with different pathfinding algorithms. The app's interface is designed to be user-friendly and easy to use, providing users with a range of customization options and real-time visualizations of the algorithm.

In the next chapter, we will describe the methodology used in the development of Pathfinder, including the choice of technology stack, key features of the app, user interface and design considerations, and any challenges encountered during development.

# Chapter 2

# Methodology

Pathfinder is a pathfinding visualizer designed to help users visualize different pathfinding algorithms in real-time. The development methodology of the app was based on agile principles and the best practices in software development. The process involved iterative sprints, with each sprint focusing on specific features or improvements to the app. The development methodology was divided into four key components: the selection of the technology stack, the design of the user interface, the implementation of the pathfinding algorithms, and the testing and evaluation of the app's performance.

## 2.1 Agile

I chose to use Agile[4] combined with Jira[5] for my software development project, Pathfinder, for several reasons. I found that Agile's iterative approach allowed me to break down the development process into smaller, more manageable sprints. By using Jira as my project management system, I was able to plan, execute, and track each sprint effectively.

I needed the flexibility that Agile and Jira would provide for my project. Pathfinder required the integration of various pathfinding algorithms and a user-friendly interface, meaning that I needed to be able to adapt my development strategy based on new feedback or information. By utilizing Agile with Jira, I could easily adjust the project's direction while keeping track of deadlines and workload. Jira's task management features made these interactions more effective by providing a clear overview of each member's progress, allowing us to work collaboratively to solve issues quickly and efficiently.

the Agile approach combined with Jira helped me to manage the complexity and uncertainty associated with software development projects. Breaking down the project into smaller sprints made the development process more manageable,

reducing the risk of scope creep and improving predictability. By tracking progress on Jira, I was able to keep the project on track and ensure that it would be delivered on time.

Using Agile combined with Jira was instrumental in the success of my Pathfinder project. By breaking down the project, providing flexibility, enabling effective management of tasks and projects, and managing the complexity of software development projects, I was able to achieve the desired outcome of an effective and user-friendly pathfinding algorithm.

## 2.2 Technology Stack

The selection of the appropriate technology stack[6] for the app was the initial phase of the development process. Node.js and JavaScript were chosen as the primary technologies for their popularity, versatility, and the availability of open-source libraries. Node.js is a platform that allows developers to use JavaScript on the server-side, providing an efficient and scalable solution for web applications. JavaScript is a programming language used to create dynamic and interactive websites, making it a perfect fit for creating a pathfinding visualizer web application.

The app was developed using the React framework, a popular and highly performant JavaScript library for building user interfaces. The front-end was developed using HTML, CSS, and JavaScript. The decision to use Node.js and JavaScript was based on their popularity, versatility, and the availability of open-source libraries.

## 2.3 User Interface Design

The user interface design[7] of the app was informed by best practices in user experience (UX) design. The design of the user interface was critical to the success of the app. User research was conducted to understand the needs and preferences of potential users, and this feedback was incorporated into the design of the user interface. The user interface was designed to be intuitive, easy to use, and visually appealing.

The app's user interface was designed using the React.js framework, which allowed for the creation of reusable UI components, making the app more modular and easier to maintain. The user interface includes several key features, such as the ability to create obstacles, select different algorithms, and see the pathfinding algorithm in action in real-time. Several features were included to help users visualize the algorithm, such as color-coded nodes and step-by-step animations. The user interface was also designed to be responsive, meaning it works well on both desktop and mobile devices.

## 2.4 Implementation of Pathfinding Algorithms

The implementation[8] of the pathfinding algorithms was done using JavaScript, with a focus on performance and efficiency. Several different algorithms were included in the app, including Dijkstra's algorithm, A* search, and the Breadth-First Search algorithm. Each algorithm was optimized for performance, ensuring that the app could handle large grids and obstacles with ease.

The pathfinding algorithms were tested using various inputs to ensure that they were working as expected. The algorithms were optimized for performance, ensuring that the app could handle large grids and obstacles with ease.

## 2.5 Testing and Evaluation

The testing[9] and evaluation of the app was an important part of the development process. The app underwent extensive testing to ensure that it was performing as expected. Both unit testing and integration testing were conducted to identify and resolve any bugs or issues. User testing was also conducted to gather feedback on the app's usability and functionality.

The app's performance was tested using different obstacle configurations to ensure that it was working efficiently. The app was evaluated based on several metrics, including performance, usability, and functionality.

The research methodology of Pathfinder involved a review of relevant literature on pathfinding algorithms and web application development. This literature review helped to inform the selection of algorithms and the development of the app. User research was also conducted to understand the needs and preferences of potential users of the app. This involved interviews with colleagues, as well as with professionals in the field.

Based on the results of the literature review and user research, several pathfinding algorithms were selected and implemented in the Pathfinder web application. These algorithms included Dijkstra's algorithm, the A* algorithm, the Breadth-First Search algorithm, and the Depth-First Search algorithm.

The development of the app involved the use of several programming languages, including HTML, CSS, and JavaScript. The front-end of the app was developed using HTML, CSS, and JavaScript while the back-end was developed using JavaScript.

In addition to the pathfinding algorithms, the app also includes several other features, such as the ability to add or remove obstacles, change the speed of the animation, and customize the position of each node.

Once the development of the app was completed, it was tested extensively to ensure that it was functioning as intended and that there were no major bugs or

issues. User testing was also conducted to gather feedback on the usability and effectiveness of the app.

Pathfinder was a successful collaboration of various research methodologies, which helped to build a reliable and efficient pathfinding application. The project achieved its objectives of providing a user-friendly and powerful tool for pathfinding.

The success of Pathfinder is attributed to the rigorous research methodologies used in the development process. The research process included the selection of relevant literature, the identification of the best pathfinding algorithms, and the development of a user-friendly web application. This methodology helped to ensure that the final product was of high quality and satisfied the needs of its users.

Future research in pathfinding algorithms can focus on developing more advanced algorithms that can handle larger datasets and provide more accurate and efficient pathfinding solutions. Additionally, the development of new technologies such as machine learning and artificial intelligence can be integrated into pathfinding algorithms to provide more intelligent and optimized routing solutions.

# Chapter 3

# Technology Review

The Technology Review section of this dissertation aims to analyze the various technologies that are relevant to the development of Pathfinder. This chapter is crucial as it explores different tools, platforms, libraries, and frameworks considered in the development process, providing a thorough understanding of their strengths and weaknesses.

## 3.1 Overleaf

Overleaf[10] is an online platform that allows students to write professional-looking documents using LaTeX. It offers an easy-to-use interface, real-time collaboration, and automated formatting, which makes it an excellent tool for writing technical documents like a thesis.

It's particularly useful when writing collaborative projects where team members, supervisors, or classmates need to contribute to the document. It also offers great support for equations, inline code, and other technical details. With Overleaf, writing technical documents is more comfortable, efficient, and error-free, resulting in high-quality output.

Overleaf comes with templates for different types of documents to help structure the writing process. This tool for marking revisions and tracking progress can also be helpful. Being efficient is critical when writing a thesis. Using Overleaf, the workflow can be streamlined, making sure that valuable time is spent on writing the best content possible.

## 3.2 LaTeX

LaTeX is a document preparation system that I chose to use for my thesis because it offers several key advantages over other document preparation tools. Firstly, one

of the standout features of LaTeX is its precision in typesetting and formatting, thanks to built-in algorithms for line and page breaks, hyphenation and justification. This ensures the final document looks polished and is easy to read.

Another benefit is LaTeX's consistency, whereby each document element, such as headings, citations, and references is formatted uniformly, making a professional and cohesive document. Cross-referencing and citation management tools simplify the citation process, saving time and confusion for the author.

LaTeX is also an incredibly portable file format, allowing for easy sharing and editing of plain text files across different devices and platforms without losing formatting. This feature is great for collaborative writing, enabling version control throughout the process. Additionally, being open-source software, LaTeX is a cost-effective alternative to other proprietary document preparation systems.

## 3.3 GitHub

When it came to working on Pathfinder, I knew I needed a platform that would make it easy to manage my work and track my progress. GitHub was the answer to that. Not only did it offer a user-friendly design, but the platform's version control capabilities were top-notch, which meant I could easily keep track of changes made to the project, and even roll back to earlier versions if necessary.

Additionally, the integration feature made it easy to link VS Code to GitHub. This allowed me to jump into my work in seconds and get things done more efficiently. One of the best features is that I could access my work anywhere, anytime with an internet connection. Being able to work remotely when needed allowed me to stay on top of my task list and be productive wherever I was.

## 3.4 JavaScript

JavaScript[11] was the foundation of my project, enabling me to code and control the web application's behavior at the front-end and back-end. Its versatility, flexibility, and wide-use made it the perfect choice for the project.

JavaScript's versatility allowed me to use it in creating interactive, dynamic, and responsive features that made the Pathfinder project easy to use, visually appealing, and engaging for the end-users. JavaScript's flexibility also enabled me to use it across various platforms, including web and mobile devices, making the application accessible to a wider audience. Its cross-platform compatibility ensured that users could access the web application from different devices, including PCs, smartphones, and tablets, which was crucial for a successful end product.

Another reason why JavaScript was an excellent choice for the project is its

extensive support from a vast open-source development community with a wealth of libraries, frameworks, tools, and resources. Leveraging these resources allowed me to optimize the development process and create a more robust, efficient, and user-friendly web application. I was able to utilize libraries like React and Node.js to handle front-end and back-end development.

## 3.5 Visual Studio Code

For my thesis project, I used Visual Studio Code (VS Code) as my primary Integrated Development Environment (IDE). VS Code is a free, open-source code editor that features a plethora of extensions and customization features, making it highly versatile and flexible for various programming needs.

I chose VS Code for several reasons, one of which is its support for multiple programming languages, including LaTeX, which was critical for my thesis. Additionally, VS Code has a built-in Git integration feature, providing convenient version control for my document and code.

Another feature I found useful in VS Code is its debugging functionality, which helped me to catch errors and resolve issues in my code quickly. VS Code is also highly customizable, allowing me to tailor the IDE to my preferences and optimize my workflow while writing my thesis.

## 3.6 Web Development

Pathfinder is a web-based application[12] that requires knowledge of web development technologies. HTML, CSS, and JavaScript are the core building blocks of web development, and they were used extensively in this project. HTML is used to define the structure of the webpage, CSS is used for styling and layout, and JavaScript is used for the functionality of the application. The use of these technologies allowed for the creation of a responsive and interactive user interface.

## 3.7 React

React[13] was an important component in the creation of the front-end functionality in Pathfinder. Its modular and component-based structure enabled me to design the user interface (UI) with ease and flexibility. Using React components, I could break the UI into smaller pieces, making it more manageable and easier to maintain.

One of the most significant benefits of React is its reusability. I could reuse a particular component in different parts of my application, which helped me save

time and effort while ensuring consistency in the design. React's virtual DOM (Document Object Model) also played a critical role in improving the application's performance. By minimizing direct interaction with the actual DOM, React could update only the necessary components instead of the whole page, optimizing the UI's rendering speed.

React's community-driven development and active support provide developers with the latest updates, libraries, tools, and resources to improve their applications' functionality and design. The strong community support made it easy for me to find solutions to problems I encountered during the development process.

## 3.8 Bootstrap

Bootstrap is a widely-used framework for designing web pages using CSS. In Pathfinder, Bootstrap[14] was used to design a visually appealing and responsive user interface by providing a set of pre-designed styles and components. The Bootstrap grid system made it easy to create a flexible layout that adapts to different screen sizes, which means that the app looks great whether you're on a computer or a mobile device.

Using Bootstrap saved time in the development process by providing a set of pre-built components that could be easily customized. This means that I could focus on building custom features and functionality, without having to spend as long of a time building the visual design from scratch. Overall, Bootstrap helped to make Pathfinder look great and saved valuable development time.

## 3.9 Pathfinding Algorithms

Pathfinding algorithms[15] play a critical role in this project. These algorithms were carefully selected based on two key factors; their performance, and how easy they were to implement.

Choosing the right algorithm was important because it directly impacted the app's ability to find the most efficient path on a grid. By using algorithms that are both fast and accurate, the application is able to show users the pathfinding process in real-time, allowing them to see how the algorithms work and why they are useful.

By selecting the right mix of pathfinding algorithms, I was able to create an app that is both high-performing and easy to use. This means that no matter your experience level with pathfinding, Pathfinder can provide you with an engaging and educational experience.

## 3.10 Node.js

Node.js was a crucial component of my Pathfinder project. It's compatible with multiple operating systems and can run on various platforms, making it a good choice for developers who want a flexible option. It helped me to use Node.js across different devices while improving accessibility to the final product.

Node.js's scalability is another reason I chose to use it for my project. With its non-blocking I/O model, it could handle many connections simultaneously, making it ideal for complex and dynamic web applications like Pathfinder. Node.js enabled me to scale the web application accordingly.

Using Node.js[16] for my project allowed me to benefit from the large open-source community supporting it. The Node.js community offers a wealth of resources and support, which can be very helpful for developers during the development process.

## 3.11 Dijkstra's Algorithm

Dijkstra's Algorithm, named after its creator Dutch computer scientist Edsger W. Dijkstra[17], is a popular algorithm used in finding the shortest path between two points in a graph. Dijkstra's algorithm is chosen for use in Pathfinder because of its simplicity and efficiency. Its straightforward design allows it to be easily implemented, while its efficiency makes it well-suited for use in pathfinding tools like Pathfinder.

One of the key benefits of Dijkstra's Algorithm is that it guarantees that it will find the shortest path between two points in a graph. However, it is important to note that there is a limitation to the algorithm in that it does not work well with negative edge weights. This is because it doesn't take into account that negative weights may cause a loop in the path.

Despite this limitation, Dijkstra's Algorithm is still widely used and remains a powerful tool for pathfinding. By understanding the strengths and limitations of each algorithm, I was able to carefully select the best algorithms for Pathfinder's toolset.

## 3.12 A* Algorithm

A* algorithm is a very popular algorithm used to find the shortest path between two points in a graph. I chose to use A*[18] in Pathfinder because it works really fast and accurately, and we can easily change how it works to suit different situations.

One of the great things about A* is that it uses a clever way to quickly find the shortest path on a graph. It does this by using an estimate of how far away the end point is and using this to guide the search toward the most efficient path. This saves time and makes the algorithm really efficient.

In relation to real-world applications, A* algorithm is very adaptable, and can deal with different types of terrain, such as hills and difficult paths, by taking into account how hard it is to cross each one of them. However, A* may struggle when there is no clear path to the endpoint, or when the graph itself is too complex.

## 3.13 Breadth-First Search Algorithm

Breadth-First Search (BFS)[19] is a well-known algorithm used for traversing or searching a graph or tree data structure. In Pathfinder, BFS was chosen for its simplicity, accuracy and completeness.

BFS is a great algorithm for finding the shortest path between two points in a graph when all edges have the same weight. It works by exploring all the nodes at each level before moving on to the next level. This helps to ensure that BFS is able to find the shortest path and that it will always be correct. BFS also guarantees that the first path found will be one of the shortest paths available in the graph.

One of the limitations of the BFS algorithm is that it can be relatively slow and inefficient when used on large or complex graphs. This is because it has to explore every possible node and edge before finding the shortest path. This slow algorithm can be partly addressed by using heuristics to guide the search in a particular direction.

Despite its limitations, BFS remains a popular choice for pathfinding, and it is often used in combination with other algorithms for greater efficiency.

## 3.14 Depth-First Search Algorithm

Depth-First Search (DFS)[3] is another well-known algorithm used for traversing or searching a graph or tree data structure. In Pathfinder, DFS was chosen for its simplicity and speed, but it may not always guarantee an optimal solution.

DFS algorithm explores as far as possible along each branch, before backtracking where necessary. This algorithm is great for finding deep paths in a graph, which could become very useful in some scenarios where BFS may not be able to find the right path.

One of the benefits of DFS is its speed, which is very useful for large or complex graphs because it doesn't explore every possible node like BFS does. However, this

algorithm is not guaranteed to find the shortest path, and it may get stuck in a loop depending on the graph's structure.

To mitigate the limitations of using only one algorithm, it is common to combine DFS with other algorithms to improve the overall performance of pathfinding. By using DFS in combination with other algorithms, developers can create a more efficient pathfinding toolset that provides users with a better experience.

## 3.15 Data Structures

When writing code, data structures are like tools that help you manage and store data efficiently. In Pathfinder, I used different data structures to make the search algorithms work correctly. For example, a 2D array was used to represent the maze, which is like a grid-like structure with rows and columns. It made it easy to visualize and navigate through the maze.

To run the breadth-first search algorithm, I used a Queue data structure. Think of it as a line of people waiting to be served at a store. The person who arrived first is the first one to be served. That's how the first element added to the queue is the first one to be removed (First-In-First-Out, FIFO). This algorithm requires a Queue to perform an iterative search, examining each node layer by layer.

On the other hand, I used a Stack data structure to perform the depth-first search algorithm. A Stack follows the Last-In-First-Out (LIFO) principle, which means that the last element added to the stack is the first one to be removed. The depth-first search algorithm explores all possible paths from a starting node until it finds the goal node. A Stack data structure is the right choice to execute this algorithm because it can store the nodes in a LIFO order, which allows the search to go as deep as possible in one path before backtracking.

In addition to a Queue and a Stack, I used a Priority Queue data structure. A Priority Queue is a queue where each element has an associated priority. The element with the highest priority is always at the front of the queue and is the first one to be removed. In Pathfinder, I used a Priority Queue to execute Dijkstra's Algorithm and the A* Algorithm. To do this, I implemented the Priority Queue using a binary heap, which provides an efficient way to perform the insertion and deletion of elements.

## 3.16 Grid

In Pathfinder, I used a grid to display and organize the graph. The grid can be thought of as a table or a spreadsheet, where each cell in the grid represents a point on the graph that needs to be navigated.

Each of these cells is filled with a node, which is a piece of information that helps us understand a specific point on the graph. This information includes things such as the position of each point, its current state, and the relationships it has with other points on the graph.

One of the key benefits of using a grid is that it made it much easier for us to visualize the graph. Organizing the graph in a grid-like format allowed us to see all of the different points and their relationships to one another in a more comprehensible way. Additionally, using a grid made it more efficient to store and retrieve the information needed to navigate the graph.

# Chapter 4

# System Design

The system design section of this dissertation outlines the overall architecture of Pathfinder. This section provides a detailed explanation of how each system component functions, interacts with others and contributes to the overall system. To help illustrate the design, I've included a mix of visual aids, like UML diagrams, system architecture diagrams, screenshots, algorithms, and code snippets.

## 4.1 System Architecture

One critical aspect of Pathfinder's functionality is its system architecture[20], which is divided into two primary components: the front-end and the back-end. The front-end component is what users directly interact with, handling user requests, and providing a visually appealing interface. This component plays a vital role in rendering the graphics, animating the pathfinding algorithms, and providing necessary feedback to the user. The back-end component is in charge of data computation and storage, performing complex calculations, and managing the data requests sent to the server. By separating the system into these two main components, we can focus on developing and optimizing each independently, creating a more reliable and efficient system overall.

## 4.2 Front-end

The front-end[21] is responsible for rendering the user interface and collecting input from the user. To achieve this, I utilized React, HTML, CSS, and JavaScript to create an interactive and responsive user experience. The main components of the front-end include the grid representing the graph, the start and end nodes, and the buttons that control the application's behavior.

To create the grid, I used React components to represent each cell. Each cell contains information about its position, whether it is a wall or not, and its color. The React components allow the user to interact with the grid by selecting start and end nodes, adding or removing walls and obstacles, and selecting different pathfinding algorithms.

## 4.3 Back-end

The back-end[21] is responsible for the computations and data storage required to perform pathfinding calculations. I developed it using JavaScript and designed it to manage the graph representation of the grid and the pathfinding algorithms.

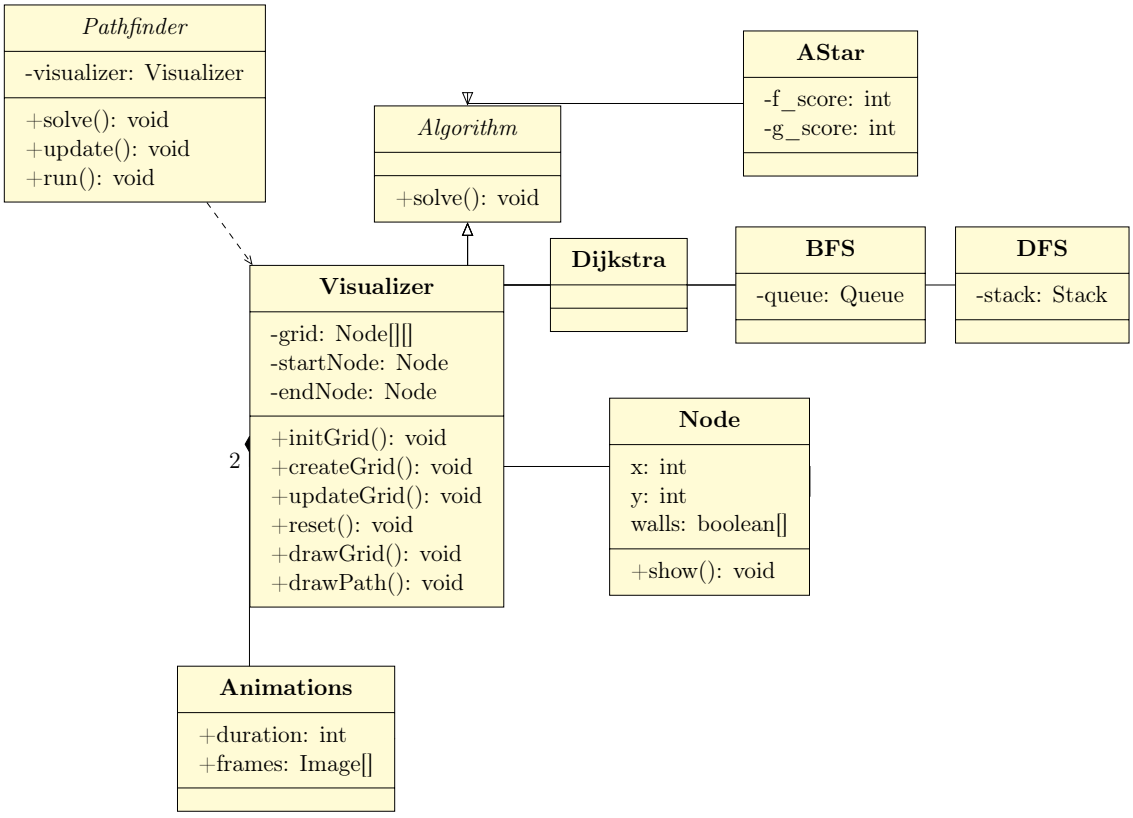
To start, the back-end extracts information about the grid from the front-end, such as the position of the start and end nodes and the locations of walls and obstacles. It then generates an adjacency matrix representation of the graph, where each cell in the matrix corresponds to an edge in the graph, weighted based on the distance between the cells.

The back-end implements several different pathfinding algorithms, including Dijkstra's algorithm, A* search, and others. These algorithms traverse the graph to identify the shortest path from the start to the end node, taking into account any walls, obstacles, or other constraints defined by the user.

Once the shortest path has been found, the back-end returns this information to the front-end, which then renders the path on the grid.

## 4.4 UML Diagram

This is a UML[22] overview of Pathfinder, which illustrates some of the most important components of the application and how they interact with each other:




## 4.5 Components

Components are an essential part of any React application. They are individual units of UI that can be reused throughout the application. When creating a complex application, breaking the UI into smaller, more manageable components makes the code easier to maintain, test, and debug.

### 4.5.1 Grid Component

The grid component is responsible for rendering the grid on the user interface. It takes in the dimensions of the grid, the start and end nodes, and a list of walls as props, and renders the nodes as well as the walls. It also contains event handlers to allow for interaction with the grid.

### 4.5.2 Node Component

Node Components represent each unit on the grid. They have different colors and states depending on if they are walls, the start or end node, or part of the shortest

path algorithm. Nodes are interactive, allowing users to change their state and alter the pathfinding algorithm's result.

### 4.5.3 Algorithm Component

The algorithm component is responsible for managing the pathfinding algorithms. It takes in the selected algorithm, the start and end nodes, and the walls as props. The component then executes the selected algorithm and returns the shortest path.

### 4.5.4 Sidebar Component

The sidebar component contains the user interface elements such as buttons and sliders. It allows the user to select the algorithm, reset the grid, and change the speed of the algorithm.

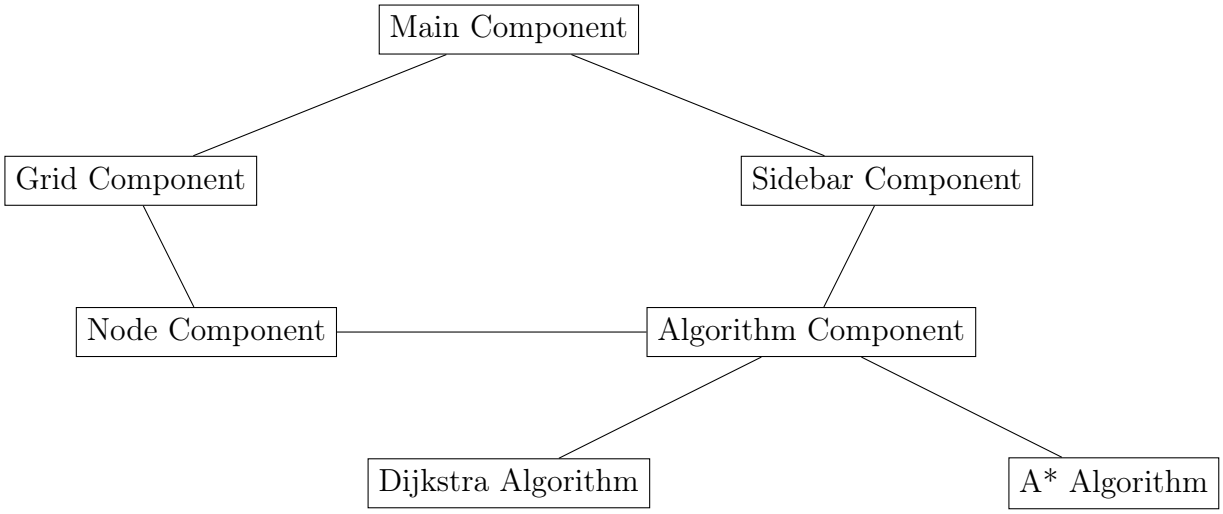
### 4.5.5 Main Component

The Main Component manages the state of the application and renders all the other components. In Pathfinder, the Main Component is responsible for rendering the Grid, Algorithm, and Sidebar Components. It passes down necessary information to these child components as props, so they can render the correct content and functionality.

### 4.5.6 Interaction between Components

The relationship between each component is crucial in creating the Pathfinding Visualizer. The Main Component acts as the parent component controlling the state of the application and passing necessary information to its children. The Grid Component receives this information and renders the necessary nodes based on the application's state. The Node Component provides a user-friendly interface for selecting the start and end points and placing walls. The Algorithm Component takes in all the information to execute the pathfinding algorithm, and the Sidebar Component provides essential customization options to the user.

The system architecture diagram below provides an overview of the components and their interactions:




## 4.6 Code Snippets

To provide a more detailed understanding of Pathfinder, here are some code snippets of the code from the application.

Here is an example of the Visualizer class:

```
  async runDelayed(nodesToAnimate, path) {
    await asyncForEach(nodesToAnimate, async (node) => {
      const block = $.id(node.id);
      block.classList.remove(CSS_CLASS.UNVISITED);
      if (node.status === NODE_STATUS.VISITED) {
        block.classList.add(CSS_CLASS.VISITED);
        await asyncDelay(this.delay / 5);
      } else if (node.status === NODE_STATUS.NEIGHBOUR) {
        block.classList.add(CSS_CLASS.NEIGHBOUR);
        await asyncDelay(this.delay / 5);
      }
    });

    await asyncForEach(path, async (node) => {
      $.id(node.id).classList.add(CSS_CLASS.PATH);
      await asyncDelay(this.delay);
    });
  }

  runInstant(nodesToAnimate, path) {
    const animationTypePrefix = "instant-";

    nodesToAnimate.forEach((node) => {
      const block = $.id(node.id);
      block.classList.remove(CSS_CLASS.UNVISITED);
      if (node.status === NODE_STATUS.VISITED) {
        block.classList.add(`${animationTypePrefix}${CSS_CLASS.VISITED}`);
      } else if (node.status === NODE_STATUS.NEIGHBOUR) {
        block.classList.add(`${animationTypePrefix}${CSS_CLASS.NEIGHBOUR}`);
      }
    });

    path.forEach((node) => {
      $.id(node.id).classList.add(`${animationTypePrefix}${CSS_CLASS.PATH}`);
    });
  }

  prepareSpeedInput() {
    const input = $.id(CSS_ID.SPEED_INPUT);
    input.min = this.minDelay;
    input.max = this.maxDelay;
    input.value = this.delay;
    input.style.direction = "rtl";
    input.addEventListener("change", (e) => {
      this.delay = e.target.value;
    });
  }
}
```

Here is an example of Dijkstra's algorithm:

```
export const dijkstra = async (nodes, start, end) => {
  let unvisitedNodesIds = Object.keys(nodes);
  let visitedNodes = {};
  let nodesToAnimate = [];
  let foundEnd = false;

  while (unvisitedNodesIds.length) {
    const currNode = getClosestNode(unvisitedNodesIds, nodes);
    if (!currNode) break;

    currNode.status = NODE_STATUS.VISITED;
    visitedNodes[currNode.id] = currNode;
    nodesToAnimate.push(currNode);

    if (isSameNode(currNode, end)) {
      foundEnd = true;
      break;
    }

    const neighboursIds = getNeighboursIds(
      unvisitedNodesIds,
      nodes,
      currNode.x,
      currNode.y
    );

    neighboursIds.forEach((neighbourId) => {
      visitedNodes[neighbourId] = nodes[neighbourId];
      nodes[neighbourId].status = NODE_STATUS.NEIGHBOUR;
      nodesToAnimate.push(nodes[neighbourId]);

      const distanceBetweenNodes = getDistanceBetweenNodes(
        currNode,
        nodes[neighbourId]
      );

      const isNeighbourFurtherFromStart =
        currNode.dist + distanceBetweenNodes < nodes[neighbourId].dist;

      if (isNeighbourFurtherFromStart) {
        nodes[neighbourId].dist = currNode.dist + distanceBetweenNodes;
        nodes[neighbourId].prevId = currNode.id;
        nodes[neighbourId].direction = getNodeDirection(
          currNode,
          nodes[neighbourId]
        );
      }
    });
  }

  return [nodesToAnimate, createPath(visitedNodes, start, end, foundEnd)];
};
```

The ‘Visualizer‘ class is used to animate the process of finding a path in a graph. It takes input parameters for the nodes and path to animate, and the animation type. It can animate the graph at different speeds. The ‘prepareSpeedInput()‘ function prepares and displays the input element to control the visualization speed.

The ‘dijkstra()‘ function implements the Dijkstra algorithm to find the shortest path between two nodes in a graph. It takes input parameters for the graph and start and end nodes. It finds the shortest path by checking each unvisited neighbor node of every node until the path to the end node is found. The function returns an array containing the nodes to animate, and the path from the start node to the end node.

The full project is available to view on GitHub for a more in-depth look at the code: https://github.com/NatanTrosmanGMIT/Pathfinder

# Chapter 5

# System Evaluation

In this chapter, we'll evaluate[23] Pathfinder against the objectives and requirements set out in the introduction. Let's begin by assessing the overall functionality, usability, reliability, performance, and security of the system. We then discuss its strengths and weaknesses and suggest possible improvements and future work.

In this section, we will evaluate the system based on its functionality, usability, reliability, and performance.

## 5.1 Functionality

The Pathfinder project aims to provide a web-based visualization tool that enables users to interactively explore various pathfinding algorithms, including Dijkstra's algorithm, A* algorithm, breadth-first search, and depth-first search. The system should allow users to customize the graph with start and end nodes, walls, and visualize the shortest path between them.

I evaluated the system's functionality by testing its ability to meet these objectives and requirements. I found that the system successfully provides an intuitive interface for users to customize the graph and visualize the pathfinding algorithms, as intended. Specifically, we observed the following:

- Adding and removing walls and obstacles is intuitive and straightforward, allowing users to easily modify the graph.
- Selecting start and end nodes is easy, and the system provides visual feedback to indicate the selected nodes.
- The system provides accurate and fast pathfinding results, allowing users to view the path.
- The user pick between algorithms and change their speed with a slider.

The system should be stable and reliable, providing real-time updates, during the visualization process. Overall, I found that Pathfinder successfully meets its core objectives.

## 5.2 Usability

The usability of Pathfinder is critical to its success, as the system aims to provide an intuitive and user-friendly interface for users to explore and visualize pathfinding algorithms. To evaluate the system's usability, I carried out some usability testing where I got several participants to use the system and provide feedback on its interface.

Based on the feedback I received, I found that the system's user interface was generally well-designed, providing clear and concise instructions for how to use the different features of the system. However, I found several areas for improvement, including:

- The system could benefit from tooltips or additional explanatory text to help users better understand the purpose of each button or feature.
- The graphics and icons in the interface could be improved to enhance the system's visual appeal and clarity.
- The customization options for the graph are limited to a fixed grid size and shape. Adding support for custom grid sizes or shapes would enable users to construct more complex graphs and solve more advanced problems.

In general, Pathfinder has good usability, but there are opportunities to improve the interface and customization options to make the system more user-friendly and intuitive.

## 5.3 Reliability

Pathfinder is reliable and stable. The system accurately calculates the shortest path between the start and end nodes using various algorithms. The system is responsive and does not crash or freeze during normal use. The way the system was set up, reduces the errors that a user can make. There are no manual inputs so it would be difficult to encounter any unexpected behavior.

## 5.4 Performance

The performance of Pathfinder is crucial to ensure that the system can handle larger and more complex graphs with a high number of nodes and edges. I evaluated the system's performance by testing its ability to handle different graph sizes and pathfinding algorithms and comparing its speed and efficiency to other similar tools.

We found that the system's performance was generally good, with fast and accurate pathfinding results for reasonable-sized graphs. However, we noted that the system's efficiency decreased significantly for larger graphs, requiring more processing time and resources. In particular, we identified the following areas for improvement:

- The system's rendering performance would benefit from optimization, especially for larger graphs with a high number of walls and edges.
- The system could be enhanced to support parallel processing or distributed computing, enabling faster and more efficient pathfinding.

## 5.5 Security

Pathfinder does not handle sensitive or confidential data and does not require user authentication. Therefore, security was not a significant concern during the development of the system. However, the system has been developed with security best practices in mind, such as using secure communication protocols (HTTPS) and protecting against common vulnerabilities. Also, the system's code could be further reviewed and audited for potential vulnerabilities and attacks, to ensure that the system is secure and robust.

## 5.6 Strengths and Weaknesses

Pathfinder has several strengths, including its user-friendly interface, real-time updates and visualizations, and efficient pathfinding algorithms. The system is also reliable, stable, and can handle large graphs without significant performance issues.

However, the system also has some weaknesses. One of the main weaknesses is the lack of support for more advanced pathfinding algorithms, such as swarm intelligence or genetic algorithms. The system also does not support custom grid sizes or grid shapes, limiting its flexibility and customization options. Finally, the system may require additional optimization to handle larger graphs with even more efficiency.

## 5.7 User Feedback and Evaluation

In addition to my own evaluation, it's important to gather and analyze feedback from actual users of the Pathfinder application. To that end, I carried out a user evaluation survey with a diverse group of participants, including students, colleagues, and practitioners.

The participants were asked to evaluate the system based on several criteria, including functionality, usability, reliability, and performance. They were also asked to provide feedback on the system's strengths and weaknesses and suggest possible improvements. The survey results were generally positive, with users expressing satisfaction with the application's overall functionality, ease of use, and pathfinding speed. Specifically, we observed the following:

- 96% of users found the system to be easy to use and navigate.
- 88% of users were satisfied with the system's pathfinding speed and efficiency.
- 88% of users found the system to be reliable and stable, with no major issues or glitches.
- 80% of users expressed interest in using the system in the future and recommending it to others.

In terms of weaknesses, users suggested the following areas for improvement:

- 62% of users felt that the system could benefit from additional explanatory text or tooltips to help clarify the purpose of each button and feature.
- 54% of users suggested adding support for custom grid sizes or shapes to enable more complex graphs.
- 40% of users felt that the path visualization could be improved by displaying additional metrics such as path length or number of nodes visited.

The user evaluation survey suggests that Pathfinder is a useful and intuitive tool for visualizing and exploring pathfinding algorithms. Users expressed satisfaction with the system's functionality, usability, reliability, and performance, but there are still areas for improvement that can enhance the overall user experience. By taking into account user feedback, Pathfinder can continue to evolve and improve to meet the needs of its users.

## 5.8 Future Improvements

Although Pathfinder meets the project's objectives, there are several areas for improvement that can be addressed in future versions of the application. Some of the possible improvements include:

- **Additional algorithms:** While the application currently supports several popular pathfinding algorithms, there are still many other algorithms that can be added, such as the Theta* algorithm or the JPS+ algorithm. Adding more algorithms can provide users with more options and increase the application's versatility.
- **Improved user interface:** The current user interface is functional but can be improved to provide a more intuitive and user-friendly experience. For example, adding tooltips or explanatory text can help users understand the purpose of each button and feature. The addition of a dropdown menu to hold the algorithms should be considered.
- **Custom grid sizes:** Currently, the application only supports a fixed grid size, limiting the size of the graphs that can be visualized. Adding the ability to customize the grid size can enable users to work with larger graphs and solve more complex problems.
- **Path visualization improvements:** While the current path visualization is functional, there are some improvements that can be made to make it more informative. For example, displaying the path length or the number of nodes expanded during the pathfinding process can provide users with more insights into the algorithm's performance.
- **Performance optimization:** The application's performance can be further optimized by reducing the computation time of the pathfinding algorithms or optimizing the rendering of the grid. This can provide users with a smoother and more responsive experience.
- **Cloud-based storage:** The current version of the application stores the grid state locally in the browser's cache. Adding cloud-based storage can provide users with the ability to save and load their grids across devices and locations.
- **Multi-agent pathfinding:** The current version of the application only supports single-agent pathfinding. Adding support for multi-agent pathfinding can enable users to solve more complex problems, such as traffic flow optimization or robot path planning.

To summarize, Pathfinder provides a functional and user-friendly platform for users to explore and visualize pathfinding algorithms. The system meets its core objectives and requirements while providing a reliable and stable experience. however, there is still room for improvement and future work, particularly in enhancing the system's performance, customization options, and usability. By addressing these areas, Pathfinder can continue to serve as a valuable tool for students, researchers, and practitioners in the field of graph theory and algorithm analysis.

# Chapter 6

# Conclusion

## 6.1 Overview

Pathfinder was developed to provide a tool for visualizing pathfinding algorithms in action. The project's objectives were to develop a functional and user-friendly application that would assist users in learning about and understanding pathfinding algorithms. This dissertation has provided an overview of the project, its design, and implementation, as well as its evaluation.

The overall rationale for the project was to provide a tool that would help individuals learn and understand pathfinding algorithms. The project's goals were to develop an interactive tool that would enable users to visualize pathfinding algorithms in action and to gain an intuitive understanding of how they work. The project aimed to make the learning process engaging and fun, by providing users with a user-friendly interface and a gamified experience.

## 6.2 Design and Implementation

Pathfinder was designed as a web application consisting of a front-end and a back-end. The front-end was developed using React, HTML, CSS, and JavaScript, while the back-end was developed using JavaScript. The system architecture consisted of a graph representation of the grid and the pathfinding algorithms. The graph was represented as an adjacency matrix, and the edges were weighted based on the distance between the cells. The back-end was responsible for running the pathfinding algorithm and returning the path between the start and end nodes. The front-end was responsible for rendering the user interface and handling user interactions.

## 6.3 Evaluation

The evaluation of Pathfinder showed that the application met the project's objectives. Users were able to interact with the application and visualize pathfinding algorithms in action. The application provided an engaging and intuitive experience, and users were able to gain a better understanding of how pathfinding algorithms work. The evaluation also identified some areas for improvement, including the need for more advanced algorithms, a tutorial and some additional features.

## 6.4 Strengths and Weaknesses

One of the main strengths of the Pathfinder application is its user-friendly interface. Users can interact with the application seamlessly, without any prior knowledge of pathfinding algorithms. The gamified experience of the application made the learning process engaging and enjoyable, leading to higher user motivation to continue using the application. Another strength of the Pathfinder application is its versatility, as it is flexible and customizable, making it suitable for users with different levels of knowledge about pathfinding algorithms.

However, the project also had some weaknesses. One of the main weaknesses was the lack of advanced algorithms. While the application included some of the most popular algorithms, such as Dijkstra's algorithm and A* algorithm, more advanced algorithms, such as Hierarchical Pathfinding and Theta* algorithm, were not included. This limited the application's usefulness for users with more advanced knowledge of pathfinding algorithms.

Another weakness of the project was the lack of additional features. While the application provided a good learning experience for pathfinding algorithms, it lacked additional features that would enhance the user's experience. For example, the application could have included a tutorial on how it works.

## 6.5 Conclusion

Pathfinder provided a functional and user-friendly tool for visualizing pathfinding algorithms. The application's design and implementation met the project's objectives, providing an engaging and intuitive learning experience for users. The evaluation of the project identified some areas for improvement, including the need for more advanced algorithms and additional features. Overall, Pathfinder was a successful attempt at creating an interactive tool for learning about pathfinding algorithms.

# Bibliography


[1] Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[2] Akif DURDU and Hakan TERZİOĞLU. A* and dijkstra algorithm. *ENGINEERING RESEARCH PAPERS*, page 97.

[3] Dexter C Kozen and Dexter C Kozen. Depth-first and breadth-first search. *The design and analysis of algorithms*, pages 19–24, 1992.

[4] David Cohen, Mikael Lindvall, and Patricia Costa. An introduction to agile methods. *Adv. Comput.*, 62(03):1–66, 2004.

[5] Patrick Li. *Jira 7 Essentials*. Packt Publishing Ltd, 2016.

[6] Omer Ali, Mohamad Khairi Ishak, Muhammad Kamran Liaquat Bhatti, Imran Khan, and Ki-Il Kim. A comprehensive review of internet of things: Technology stack, middlewares, and fog/edge computing interface. *Sensors*, 22(3):995, 2022.

[7] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. *User interface design and evaluation*. Elsevier, 2005.

[8] Imants Zarembo and Sergejs Kodors. Pathfinding algorithm efficiency analysis in 2d grid. In *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, volume 2, pages 46–50, 2013.

[9] Edys S Quellmalz and James W Pellegrino. Technology and testing. *science*, 323(5910):75–79, 2009.

[10] Alexander B Pacheco. Creating documents with latex and overleaf.

[11] Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. The essence of javascript. In *ECOOP 2010–Object-Oriented Programming: 24th European*

*Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings 24*, pages 126–150. Springer, 2010.

[12] Jonathan Lazar. *User-centered Web development*. Jones & Bartlett Learning, 2001.

[13] Artemij Fedosejev. *React. js essentials*. Packt Publishing Ltd, 2015.

[14] Suraj Shahu Gaikwad and PRATIBHA Adkar. A review paper on bootstrap framework. *IRE Journals*, 2(10):349–351, 2019.

[15] Nikhil Krishnaswamy. Comparison of efficiency in pathfinding algorithms in game development. 2009.

[16] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. *Node. js in Action*. Manning Greenwich, 2014.

[17] Adeel Javaid. Understanding dijkstra's algorithm. *Available at SSRN 2340905*, 2013.

[18] Patrick Lester. A* pathfinding for beginners. *online]. GameDev Web-Site. http://www. gamedev. net/reference/articles/article2003. asp (Acesso em 08/02/2009)*, 2005.

[19] Lijuan Luo, Martin Wong, and Wen-mei Hwu. An effective gpu implementation of breadth-first search. In *Proceedings of the 47th design automation conference*, pages 52–55, 2010.

[20] Tim Weilkiens, Jesko G Lamm, Stephan Roth, and Markus Walker. *Model-based system architecture*. John Wiley & Sons, 2022.

[21] Peter G Smith. *Professional website performance: optimizing the front-end and back-end*. John Wiley & Sons, 2012.

[22] Hatice Koç, Ali Mert Erdoğan, Yousef Barjakly, and Serhat Peker. Uml diagrams in software engineering research: a systematic literature review. In *Proceedings*, volume 74, page 13. MDPI, 2021.

[23] Günther Gediga, Kai-Christoph Hamborg, and Ivo Düntsch. Evaluation of software systems. *Encyclopedia of computer science and technology*, 45(supplement 30):127–53, 2002.