Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report

on

“Tower of Hanoi ”

[Code No: COMP 202]

For partial fulfillment of Second Year/first Semester in Computer Engineering

<u>Submitted By:</u>

Sushan Luitel [28]

<u>Submitted To:</u>

Mr. Sagar Acharya

Department of Computer Science and Engineering

Submission Date: 2026-02-25

Bona fide Certificate

This is to certify that the project entitled

" Tower of Hanoi "

is a bonafide work carried out by the student in partial fulfillment of the requirements for the Second Year /First Semester of the Bachelor of Engineering in Computer Engineering, under the guidance of Mr. Sagar Acharya, Department of Computer Science and Engineering.

Name of Student :

Sushan Luitel

The project work has been carried out with sincerity, dedication, and to the satisfaction of the department, adhering to the academic guidelines.

Project Supervisor

____

Mr. Sagar Acharya

Department of Computer Science and Engineering

Date: 2026-02-25

Acknowledgement

I would like to express my sincere and heartfelt gratitude to my respected teacher, **Mr. Sagar Acharya**, for his continuous guidance, invaluable suggestions, and constant encouragement in the field of Data Structures and Algorithms. His excellent teaching, insightful explanations, and dedication to the subject inspired me to undertake and successfully complete this project. His support played a vital role in strengthening my understanding of the core concepts that formed the foundation of this work.

I would also like to extend my sincere appreciation to the Department of Computer Science and Engineering for providing me with the opportunity and platform to apply my theoretical knowledge in a practical and meaningful way. The academic environment, resources, and encouragement provided by the department were instrumental in helping me carry out and complete this project successfully.

Finally, I would like to express my gratitude to everyone who directly or indirectly supported and encouraged me during the course of this project. Their motivation and support contributed significantly to the successful completion of this work. I am truly thankful for their assistance and encouragement.

Thank You

Abstract

Data Structures and Algorithms are fundamental to the development of efficient and optimized software systems. This project presents the implementation of the classic Tower of Hanoi puzzle using C++ with a graphical user interface developed using the Qt Framework. The primary objective of this project is to demonstrate the practical application of core data structures such as Stack, Queue, and Recursion Tree within an interactive and user-friendly environment.

In this system, the three towers are implemented using Stack (std::stack<int>), which strictly follows the Last-In-First-Out (LIFO) principle and accurately represents the game rule that only the top disk can be moved at any time. The automatic solution feature is implemented using a recursive algorithm that forms a logical Recursion Tree to generate the complete sequence of moves required to solve the puzzle. These moves are stored in a Queue (std::queue<Move>) and executed sequentially to ensure proper order and smooth visualization. Additionally, an undo functionality is implemented using an auxiliary Stack to maintain move history, allowing users to reverse previous moves efficiently.

The graphical user interface is developed using Qt's QGraphicsScene to provide an interactive experience with features such as disk selection, movement, and animation. The animation mechanism is implemented using a QTimer-based approach to ensure smooth visual transitions. Furthermore, the project includes an informational dialog explaining the implementation and application of Stack, Queue, and Recursion Tree within the system. This project effectively demonstrates the integration of fundamental data structures in a real-world graphical application, enhancing both understanding and visualization of Data Structures and Algorithms concepts.

Keywords: Tower of Hanoi, Stack, Queue, Recursion Tree, C++, Qt Framework, Data Structures, Graphical User Interface, Animation.

Table of Contents

List of Figures

Acronyms/Abbreviations

The list of all abbreviations used in the documentation are as follows :

DSA: Data Structures and Algorithms

GUI: Graphical User Interface

Qt: Qt Framework

LIFO: Last In, First Out

FIFO: First In, First Out

OOP: Object-Oriented Programming

IDE: Integrated Development Environment

API: Application Programming Interface

FPS: Frames Per Second

UI: User Interface

UX: User Experience

Chapter 1 Introduction

1.1 Background

Data Structures and Algorithms (DSA) are essential components of computer science that form the foundation of efficient and optimized software development. The ability to apply theoretical data structures to practical, real-world problems is a crucial skill for software engineers. The Tower of Hanoi is a classical recursive puzzle widely used in computer science education to demonstrate problem-solving techniques and the application of fundamental data structures. Originally introduced by French mathematician Édouard Lucas in 1883, the puzzle serves as an excellent example for illustrating the practical use of concepts such as Stack, Queue, and Recursion Tree in an interactive environment.

Traditionally, the teaching and learning of DSA concepts are often limited to console-based programs, which provide minimal visual representation. Although students learn theoretical operations such as push and pop in Stack or First-In-First-Out (FIFO) operations in Queue, the lack of graphical visualization makes it difficult to fully understand how these data structures function in real-time applications. This gap between theoretical knowledge and practical visualization can reduce conceptual clarity and student engagement.

To bridge this gap, this project presents the development of a fully interactive Tower of Hanoi application using C++ and the Qt Framework. In this system, each tower is implemented using a Stack, the automatic solution sequence is managed using a Queue, and the recursive solving mechanism represents a Recursion Tree. The project incorporates a graphical user interface with animation, user interaction, and an undo feature, providing both functional gameplay and an educational platform. This implementation helps demonstrate how fundamental data structures can be effectively applied in a real-world graphical application while enhancing understanding through visualization.

1.2 Objectives

The primary objectives of the **Tower of Hanoi DSA Project** are as follows:

- To implement the Tower of Hanoi puzzle using C++, where each tower is represented using **std::stack<int>**, in order to demonstrate the practical application of the Last-In-First-Out (LIFO) principle in a real-world interactive environment.
- To develop an auto-solve feature using a recursive algorithm that generates the complete sequence of moves and stores them in a **std::queue<Move>**, thereby illustrating the First-In-First-Out (FIFO) behavior of the Queue data structure.
- To design and develop a professional graphical user interface using the Qt Framework that supports interactive features such as disk selection, drag-and-drop functionality, and animated disk movement to enhance user experience and visualization.
- To implement an undo functionality using an additional Stack to store move history, allowing efficient reversal of moves and demonstrating the practical use of Stack operations.
- To include an educational About/DSA Information dialog within the application that explains the implementation and working of Stack, Queue, and Recursion Tree using code examples, diagrams, and supporting material for learning and viva preparation.

1.3 Motivation and Significance

The motivation behind this project arises from the need to make Data Structures and Algorithms concepts more visual, interactive, and easier to understand. In traditional learning environments, Stacks and Queues are often explained using simple console-based examples such as push and pop operations, which do not fully demonstrate their real-world applications. This project addresses that limitation by presenting these data structures within a complete graphical application. Users can observe how the Queue executes the sequence of moves during the auto-solve feature and how each tower operates as a Stack following the Last-In-First-Out principle. This visual and interactive approach helps improve conceptual clarity and enhances the overall learning experience.

This project is also significant as it naturally integrates multiple data structures in a meaningful and practical context rather than using them in isolation. Stack serves as the core data structure for managing the towers, Queue is used to manage and execute the sequence of moves generated by the recursive solution, and the Recursion Tree represents the logical structure and complexity of the algorithm. This integration helps demonstrate the relationship between theoretical concepts and their practical implementation in solving real problems.

Furthermore, the use of the Qt Framework provides a professional graphical user interface with animation and interactive features, which improves usability and presentation quality. The project not only strengthens programming and problem-solving skills but also serves as an effective educational tool for learning and demonstrating DSA concepts. Its practical implementation, visual clarity, and professional design make it highly suitable for academic evaluation, project demonstration, and viva examinations.

Chapter 2 Related Works

Several research studies and educational tools have explored the visualization of data structures and algorithms, including the Tower of Hanoi problem. Online platforms such as VisuAlgo and Algorithm Visualizer provide interactive, web-based demonstrations of various data structures, including Stack and Queue operations. However, these platforms are designed as general-purpose visualization tools and do not specifically integrate the Tower of Hanoi puzzle with explicit implementation of multiple data structures within a standalone desktop application.

Traditional implementations of the Tower of Hanoi are commonly taught in Data Structures and Algorithms courses using console-based C++ programs. These implementations mainly focus on demonstrating the recursive algorithm and its mathematical properties, without incorporating graphical visualization or interactive features. Standard textbooks such as Introduction to Algorithms by Thomas H. Cormen and Data Structures, Algorithms, and Applications in C++ by Sartaj Sahni provide comprehensive theoretical explanations of Stack, Queue, and recursion, which form the conceptual foundation for this project. However, they do not provide practical GUI-based implementations for interactive learning.

In addition, some educational applications have been developed using the Qt Framework to visualize algorithms such as sorting and searching techniques. While these applications demonstrate individual algorithm behavior effectively, Tower of Hanoi implementations that explicitly connect game mechanics with Stack and Queue data structures, along with an integrated educational information module, are not widely documented. This project addresses that gap by combining game development, graphical visualization, and direct implementation of fundamental data structures in a single interactive system. Furthermore, the project incorporates a stable animation mechanism using Qt's graphics framework to enhance visualization, making it both an educational tool and a functional demonstration of core DSA concepts.

Chapter 3 Design and Implementation

3.1 Implementation

The system was implemented using an object-oriented programming approach in C++, with the graphical user interface developed using the Qt Framework. The project is organized into five main classes: Disk, Tower, Move, Game, and MainWindow, each designed with clearly defined roles and responsibilities to ensure modularity, maintainability, and clarity. This structured approach allows the separation of game logic, data structure implementation, and graphical interface. Special emphasis was placed on ensuring that each data structure—Stack, Queue, and Recursion Tree—was not only implemented internally but also reflected through visible system behavior and user interaction.

3.1.1 Implementation Planning

The implementation was carefully planned to demonstrate three fundamental DSA concepts: Stack, Queue, and Recursion Tree. The Stack data structure was used to represent the towers and manage disk movement according to the Last-In-First-Out (LIFO) principle. The Queue was used to store and execute the sequence of moves generated by the automatic solution feature, demonstrating the First-In-First-Out (FIFO) principle. The recursive algorithm used to solve the puzzle naturally formed a Recursion Tree, illustrating the hierarchical structure and execution flow of recursive function calls.

During the planning phase, Qt's QGraphicsScene was selected as the primary rendering component due to its flexibility and efficiency in handling graphical objects and animations. Additionally, a QTimer-based animation mechanism was chosen to control disk movement smoothly and safely, avoiding potential memory management and stability issues associated with other animation techniques. The development process was divided into several key tasks, including implementation of the core game logic (Tower, Game, and Move classes), graphical user interface design, mouse interaction features such as selection and drag-and-drop, animation handling, win condition detection, and the integration of an About/DSA Information dialog to enhance the educational value of the application.

3.1.2 Requirement Analysis

Functional Requirements

The system must provide a three-tower game board with disks that can be moved via click-to-select, click-to-place, or drag-and-drop. Move validation ensures larger disks cannot be placed on smaller ones. It includes an auto-solve feature using a Queue, an undo feature using a Stack, a move counter, game timer, win detection, and a congratulations dialog with statistics. An About/DSA Info dialog explains Stack, Queue, and Recursion Tree concepts.

Non-Functional Requirements

The application must run reliably without crashes, provide smooth animation (~60 FPS), and maintain a responsive UI that blocks input during animations. It should have professional styling and correctly implement Stack, Queue, and Recursion Tree to reflect their behavior accurately.

3.1.3 System Design

The system is designed following a layered architecture to separate game logic from graphical presentation. The Game class is responsible for all core logic, including the management of data structures, move validation, auto-solve, and undo functionality. The MainWindow class handles user interaction, such as disk selection, drag-and-drop movement, button clicks, and updates to GUI components. The QGraphicsScene provides the rendering environment for the visual representation of towers and disks, handling animations and real-time updates of the game state. Communication between layers is one-directional, with the GUI calling methods in the Game class while the Game layer does not directly modify GUI elements, ensuring modularity and maintainability.

Stack Implementation

Each tower is implemented as a Tower class containing a std::stack<int>, where integers represent disk sizes. The LIFO behavior of the Stack enforces the game rule that only the top disk can be moved. Move validation uses stack::top() to compare the disk sizes and prevent invalid moves, ensuring that a larger disk cannot be placed on a smaller one.

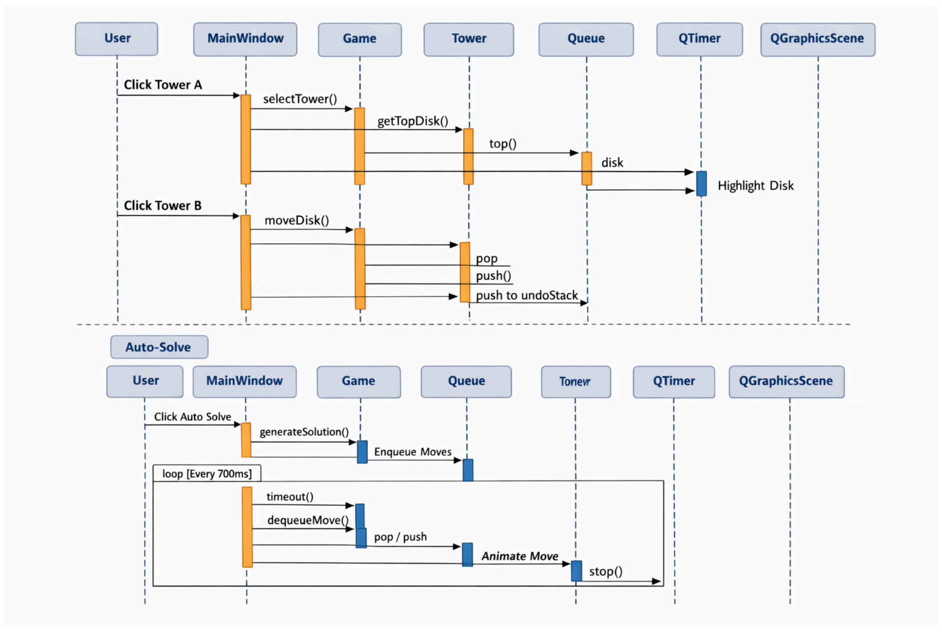


Fig 3.1 System's Sequence Diagram

This direct mapping of game rules to Stack behavior provides both correctness and an intuitive understanding of the data structure in action. Additionally, the About/DSA Info dialog visually displays the contents of each tower's stack, helping users understand LIFO operations in a practical context.

Queue Implementation

The auto-solve feature uses a std::queue<Move> called solutionQueue to store the sequence of moves generated by the recursive generateSolution() function. Each move contains the source tower, destination tower, and disk size. A QTimer dequeues one move at a time and executes it with smooth animation, providing a step-by-step visualization of the solution. This demonstrates the FIFO principle, as moves are executed in the exact order they were generated, ensuring correctness and giving the user a clear understanding of queue operations in practice.

Recursion Tree (generateSolution algorithm)

The recursive function Hanoi(n, src, aux, dst) forms a binary Recursion Tree, where each recursive call creates two child calls for n-1 disks, and the root represents moving the largest disk. The depth of the tree is n, and the total number of nodes is 2^n - 1, which corresponds to the minimum number of moves required to solve the puzzle. This structure allows students to visualize recursion in practice and understand how recursive calls expand and resolve. The algorithm has a time complexity of O(2^n) and a space complexity of O(n) due to the call stack, reflecting the inherent computational cost of recursive solutions.

Undo Stack

Every successful move is stored in a second Stack named undoStack, which contains a Move struct storing the from tower, to tower, and diskSize. When the Undo button is pressed, the last move is popped from the stack and reversed, restoring the disk to its original tower. This provides an O(1) undo operation, following LIFO ordering, and ensures that the most recent move is always undone first. The undo feature enhances interactivity and allows users to experiment and learn without penalty, reinforcing the educational objective of the project.

GUI Interaction and Animation

The graphical interface is implemented using Qt’s QGraphicsScene, providing an interactive canvas for rendering towers, disks, and animations. Users can select and move disks using click-to-select and drag-and-drop mechanisms. Smooth visual feedback is achieved through a QTimer-based arc animation system, which moves disks along a curved path to mimic realistic motion. The GUI also blocks additional user input during animation to prevent illegal moves, ensuring stability and correctness.

Educational Features

An integrated About/DSA Info dialog provides visual explanations of the Stack, Queue, and Recursion Tree implementations. It includes code snippets, diagrams of the recursion tree, and interactive examples of tower states. This feature bridges the gap between theoretical DSA concepts and practical implementation, making the project both a game and a learning tool. The GUI also displays the current move count and game timer, reinforcing concepts of algorithm efficiency and sequence tracking.

Data Structure Integration

- Stack: Towers (LIFO behavior), Undo feature.
- Queue: Auto-solve move sequence (FIFO behavior).
- Recursion Tree: Recursive solution visualization and algorithm complexity demonstration.

This structured implementation ensures that each data structure is not only functionally necessary but also educationally visible, providing an interactive and comprehensive demonstration of fundamental DSA concepts in a real-world application.

3.1.4 Development

The development of the project was carried out in phased stages to ensure modularity, correctness, and stability. Initially, the core logic of the game was implemented and tested in a console-based environment, focusing on the Tower, Game, and Move classes. This phase validated the correct functioning of the Stack, Queue, and recursive solution algorithm before integrating the graphical interface.

Subsequently, the Qt GUI was developed around the validated core logic. A major technical challenge encountered during this phase was a segmentation fault (SIGSEGV) caused by QPropertyAnimation holding raw pointers to QGraphicsRectItem objects, which were deleted when scene::clear() was called. This issue was resolved by replacing QPropertyAnimation with a QTimer-based animation system that fires every 16ms and computes the position of disks along a quadratic Bezier arc for each tick, ensuring both smooth animation and memory safety.

Interactive features were implemented using Qt's event handling system. Drag-and-drop functionality was achieved via an eventFilter on QGraphicsView, capturing MouseButtonPress, MouseMove, and MouseButtonRelease events. A temporary "ghost" disk rectangle follows the cursor during dragging and is safely removed using scene::removeItem() before each redraw. The click-select system tracks the selected tower using an integer variable (-1 representing none selected), enabling users to pick up and place disks with consecutive clicks.

The major development phases of the project can be summarized as follows:

1. **Core class design:** Disk, Tower, Move, Game.
2. **Console validation:** Testing Stack, Queue, and recursive algorithm correctness.
3. **Qt GUI layout:** Implementing QGraphicsScene, QGraphicsView, and control panel.
4. **Mouse interaction:** Click-select and drag-and-drop using eventFilter.

5. **Animation system:** QTimer-based Bezier arc animation with drawDisksExcept() to handle ghost disks.
6. **Win dialog:** Displaying game statistics, ratings, and replay options.
7. **About/DSA Info dialog:** Four tabs covering Stack, Queue, Recursion Tree, and game information.
8. **Testing and crash resolution:** Fixing SIGSEGV and ensuring stable execution.

This structured development approach ensured that the project was robust, interactive, and educational, successfully integrating core DSA concepts with a professional graphical interface.

3.1.5 Testing and Debugging

The project underwent extensive testing to ensure stability, correctness, and smooth user interaction. During development, three major segmentation faults (SIGSEGV) were identified and resolved.

The first crash occurred when a subclassed QGraphicsScene (HanoiScene) attempted to delete a ghost disk pointer within a mouse event while the scene's event loop still held references to it. This was resolved by removing the subclass and implementing an eventFilter on the QGraphicsView to handle mouse interactions safely.

The second crash was caused by view->setInteractive(false), which inadvertently blocked all mouse events, preventing user interaction. The third crash arose from QPropertyAnimation animating a raw QGraphicsRectItem pointer that was subsequently deleted via scene::clear(). All three issues were resolved in the final implementation through careful pointer management and a QTimer-based animation system, which ensured both safe memory handling and smooth animation.

In addition to these critical issues, several minor bugs were addressed:

- Duplicate win dialogs appearing due to simultaneous triggers from onAutoSolveStep and checkWin.
  The visual representation of the Queue section was too small (height fixed at 110px) and previously cut off content.
- The Queue diagram displayed an incorrect move sequence, which was corrected to show the complete set of moves (e.g., 7 moves for n = 3).

Through iterative testing and debugging, the system now performs reliably, providing a crash-free, interactive, and visually accurate experience that demonstrates the intended Stack, Queue, and Recursion Tree concepts.

3.2 System Requirement Specifications

The system was developed and tested with specific software and hardware requirements to ensure proper functionality, performance, and compatibility. The specifications are detailed below.

3.2.1 Software Specifications

The following software components are required for the development and execution of the system:

- **Programming Language:** C++17
- **GUI Framework:** Qt 5.15 or Qt 6.x (Qt Widgets module)
- **Integrated Development Environment (IDE):** Qt Creator
- **Compiler:** MinGW (Windows) or GCC (Linux/Mac)
- **Build System:** qmake (.pro project file)

These components provide a stable foundation for implementing both the backend logic and the graphical frontend, enabling efficient development, debugging, and testing of the Tower of Hanoi application.

3.2.2 Hardware Specifications

The system requires the following hardware to operate efficiently:

- **Processor:** Intel Core i3 or equivalent (minimum); Intel Core i5 or higher recommended
- **Memory (RAM):** 4 GB minimum; 8 GB recommended
- **Storage:** 500 MB minimum for Qt installation and project files
- **Display:** 1366×768 resolution minimum; 1920×1080 recommended
- **Input Devices:** Standard keyboard and mouse

These specifications ensure smooth execution of the system, support real-time animations, and provide a responsive and user-friendly experience while interacting with the game and visualizing data structure operations.

Chapter 4 Discussion on the achievements

4.1 Project Overview

The Tower of Hanoi DSA project is a C++ application developed using the Qt Framework, presenting the classic puzzle as an interactive game while explicitly demonstrating three core data structures: Stack, Queue, and Recursion Tree. The application features a dark-themed graphical interface with three towers rendered on a QGraphicsScene, visually distinct disks of varying sizes, and smooth arc animations for disk movement. The game supports 2 to 8 disks, selectable prior to each game session.

The right-hand control panel displays a move counter, game timer, setup options, and three action buttons: Auto Solve (Queue), Undo Last Move (Stack), and Reset/New Game. A chronological Move History log displays every move in queue order. Additionally, an About/DSA Info button opens a comprehensive educational dialog with four tabs that explain the underlying data structures and game logic.

4.2 Implemented Features

4.2.1 DSA Implementations

Stack (Towers A, B, C)

Each tower is implemented as an instance of the Tower class containing a std::stack<int>. Disk placement invokes the push() method, lifting a disk invokes pop(), and top() is used for move validation without removing the disk. The LIFO property of the stack naturally enforces the game rule that only the top disk of a tower may be moved.

Queue (Auto-Solve)

The Auto Solve feature calls generateSolution(numDisks, 'A', 'B', 'C'), which recursively populates a solutionQueue with all 2^n - 1 moves. A QTimer fires every 700 ms to execute onAutoSolveStep(), dequeuing the front move and animating it. The user can pause the auto-solve by toggling the button. The FIFO behavior of the queue ensures that moves are executed in the correct sequence.

Undo Stack

Every successful move is recorded in an undoStack as a Move struct containing from, to, and diskSize. Pressing the Undo button pops the last move and reverses it, returning the disk to its original tower. Undo is disabled during auto-solve to avoid conflicts. The LIFO property ensures that the most recent move is always undone first.

Recursion Tree

The recursive function generateSolution(n, src, aux, dst) forms a binary recursion tree, with each call spawning two child calls for n-1 disks and the root representing the movement of the largest disk. For n=3, the function generates 7 moves (2^3 - 1). The Tree tab in the About/DSA Info dialog displays an SVG-style diagram illustrating the recursion tree, including in-order traversal and complexity analysis.

4.2.2 GUI and Interaction

Click-to-Select / Click-to-Place

Clicking a tower with disks selects it; the selected rod is highlighted in blue, the top disk glows orange, and a “SELECTED” badge appears above. Clicking another tower attempts to place the disk; invalid moves trigger an error message and cancel the selection. Clicking the same tower cancels the selection.

Drag-and-Drop

Clicking and holding a tower creates a semi-transparent “ghost” disk that follows the cursor. Releasing over a different tower executes the move with animation. This system is implemented via an eventFilter on QGraphicsView, ensuring safe removal of the ghost rectangle using scene::removeItem() before redraw.

Arc Animation

Each disk move executes a three-phase animation: lift, horizontal slide, and drop. The motion follows a quadratic Bezier curve, computed every 16 ms (~60 FPS) using a QTimer. Input is temporarily blocked during animation to maintain consistency.

Win Dialog

Upon completion, a styled congratulations dialog displays the total moves used, minimum possible moves (2^n - 1), elapsed time, and a rating (PERFECT / Excellent / Good job). A Play Again button resets the game with the same disk count.

About / DSA Info Dialog

The dialog (780×620) contains four tabs:

- **Stack Tab:** Visual tower diagram, push/pop operations, and corresponding C++ code.
- **Queue Tab:** Step-by-step queue diagram for n=3 and associated code.
- **Tree Tab:** Recursion tree diagram, complexity analysis, and code examples.
- **About Game Tab:** Game rules, controls, and quick viva-style Q&A.

Each tab includes working C++ code with syntax highlighting and visual illustrations to enhance learning and reinforce the connection between theory and implementation.

4.3 Achievements

During the development of the Tower of Hanoi DSA project, the team acquired and honed a range of technical and problem-solving skills. A complete object-oriented class hierarchy was designed and implemented, comprising the Disk, Tower, Move, Game, and MainWindow classes, with a clear separation of responsibilities between game logic, data structures, and graphical interface. The project explicitly demonstrates the use of std::stack for towers, std::queue for the auto-solve feature, and a recursive algorithm for the Recursion Tree, integrating all three data structures into a single, coherent application.

The team encountered and resolved three critical SIGSEGV crashes, which developed advanced debugging skills, particularly in the context of Qt's graphics and event systems. Implementing a QTimer-based Bezier arc animation system enabled smooth, crash-free disk movement, providing a more reliable and memory-safe alternative to Qt's built-in QPropertyAnimation for this application.

Additionally, the development of the About/DSA Info dialog—with interactive tabs, visual diagrams, code examples, and viva-style questions—highlighted the educational value of the project, transforming it into both a functional game and a learning tool. Throughout the project, the team strengthened their understanding of Qt's signal-slot architecture, C++ memory management, recursive algorithm design, and the development of interactive educational software, making this project a comprehensive learning experience in both theory and practical application.

Chapter 5 Conclusion and Recommendation

The project successfully accomplished its primary objective of implementing the Tower of Hanoi puzzle as a C++ Qt application that explicitly demonstrates the Stack, Queue, and Recursion Tree data structures within an interactive game environment. The three towers are implemented as std::stack<int> instances, the auto-solve feature utilizes a std::queue<Move> for sequential move execution, and the recursive algorithm effectively illustrates the O(2^n) binary recursion tree structure. The Undo feature employs a second stack, allowing O(1) reversal of moves.

The graphical user interface provides an engaging user experience, incorporating click-to-select, drag-and-drop, and arc animation for disk movement. The win dialog displays game statistics and performance ratings, while the About/DSA Info dialog with four educational tabs enhances the project's value as a teaching tool for DSA concepts. All three SIGSEGV crashes encountered during development were resolved, resulting in a stable and crash-free application.

5.1 Limitations

Despite its successful implementation, the project has certain limitations:

1. Fixed animation speed: Auto-solve moves occur at a fixed interval of 600 ms, and users cannot adjust the speed in the current version.
2. Platform limitation: The application runs only on desktop platforms (Windows, Linux, Mac) and does not support Android or iOS due to the complexity of Qt Android build configuration.
3. Code display in About dialog: The About dialog uses Qt widgets for code representation rather than a fully featured code editor with syntax highlighting, which may reduce readability for longer examples.

4. Static recursion tree visualization: The recursion tree diagram in the About dialog is fixed for n=3 and does not dynamically update to reflect the number of disks selected by the user.

These limitations provide opportunities for future enhancement, including dynamic recursion tree generation, adjustable animation speed, mobile support, and improved code visualization.

5.2 Future Enhancement

The current implementation of the Tower of Hanoi DSA project provides a stable and educational platform, but several enhancements can further improve functionality, interactivity, and learning value:

1. **Adjustable Animation Speed:** Incorporate a slider to allow users to slow down or speed up disk movements during auto-solve, providing a more customizable learning and gameplay experience.
2. **Dynamic Recursion Tree Visualizer:** Develop a recursion tree that dynamically builds and highlights nodes in real time as the recursive algorithm executes, enabling users to better visualize the recursive process for any number of disks.
3. **Sound Effects:** Add audio feedback for disk placement, invalid move warnings, and puzzle completion to enhance user engagement.
4. **Mobile Platform Support:** Package the application for Android using Qt Android deployment, extending accessibility to mobile devices.
5. **Hint System:** Introduce a hint feature that suggests the next optimal move based on the Queue-based solution algorithm, assisting users when they are uncertain about the next step.

These enhancements will not only increase the usability and interactivity of the project but also strengthen its educational value, making it a more comprehensive tool for learning Data Structures and Algorithms in an engaging, interactive environment.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

2. Sahni, S. (2005). *Data structures, algorithms, and applications in C++* (2nd ed.). Universities Press.

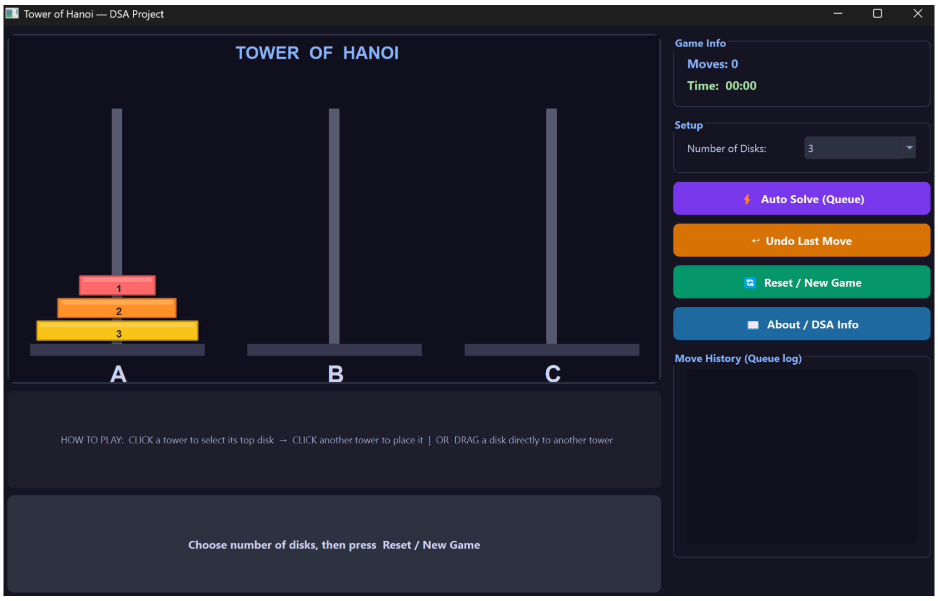
3. Qt Company. (2023). *Qt documentation: QGraphicsScene, QGraphicsView, and QTimer*. Retrieved from https://doc.qt.io

4. VisuAlgo. (n.d.). *Data structure and algorithm visualizations*. Retrieved February 2026, from https://visualgo.net/en

5. Algorithm Visualizer. (n.d.). *Interactive algorithm animations*. Retrieved February 2026, from https://algorithm-visualizer.org

6. Lucas, É. (1883). *Recreations in mathematics and natural philosophy* (original Tower of Hanoi puzzle).

APPENDIX



Moves: 0

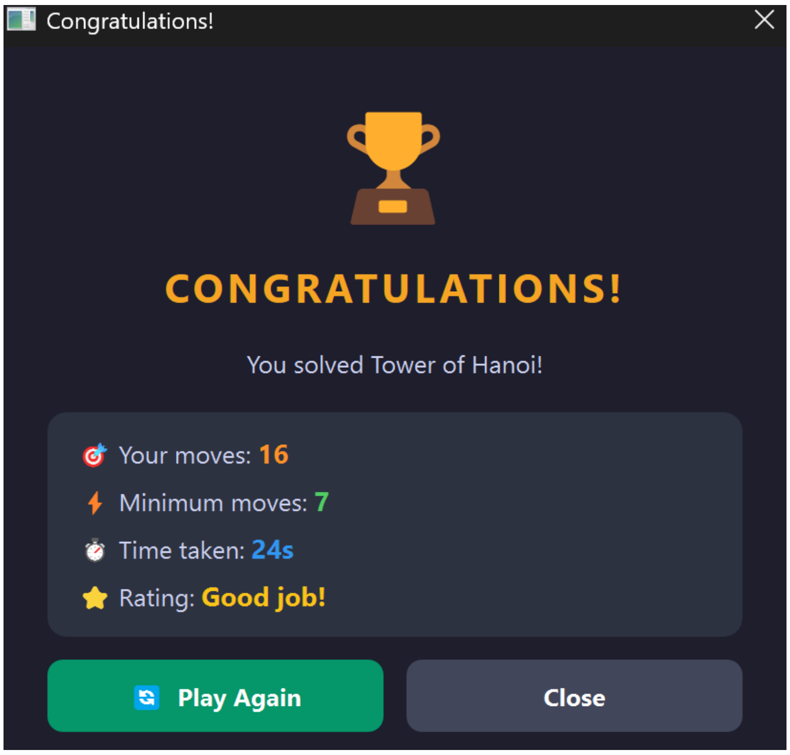
Time: 00:00

Number of Disks: 3

HOW TO PLAY: CLICK a tower to select its top disk → CLICK another tower to place it | OR DRAG a disk directly to another tower

Choose number of disks, then press Reset / New Game

Fig A.1: System Interface



CONGRATULATIONS!

You solved Tower of Hanoi!

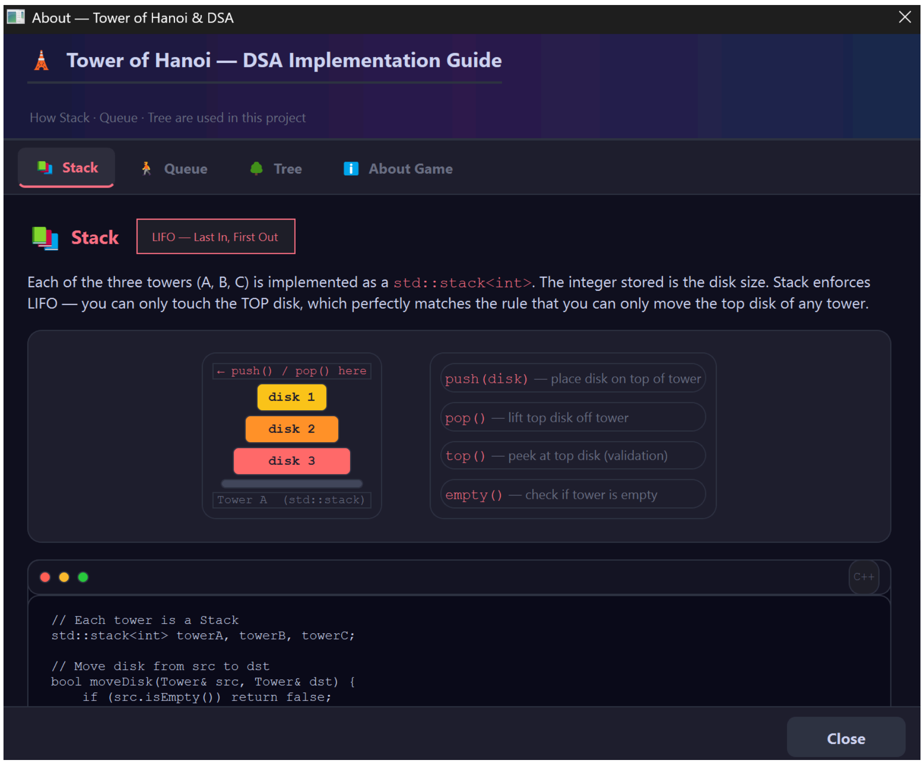
Your moves: 16

Minimum moves: 7

Time taken: 24s

Rating: Good job!

Fig A.2: Game Completion Interface



Tower of Hanoi — DSA Implementation Guide

How Stack · Queue · Tree are used in this project

Stack LIFO — Last In, First Out

Each of the three towers (A, B, C) is implemented as a `std::stack<int>`. The integer stored is the disk size. Stack enforces LIFO — you can only touch the TOP disk, which perfectly matches the rule that you can only move the top disk of any tower.

`push(disk)` — place disk on top of tower

`pop()` — lift top disk off tower

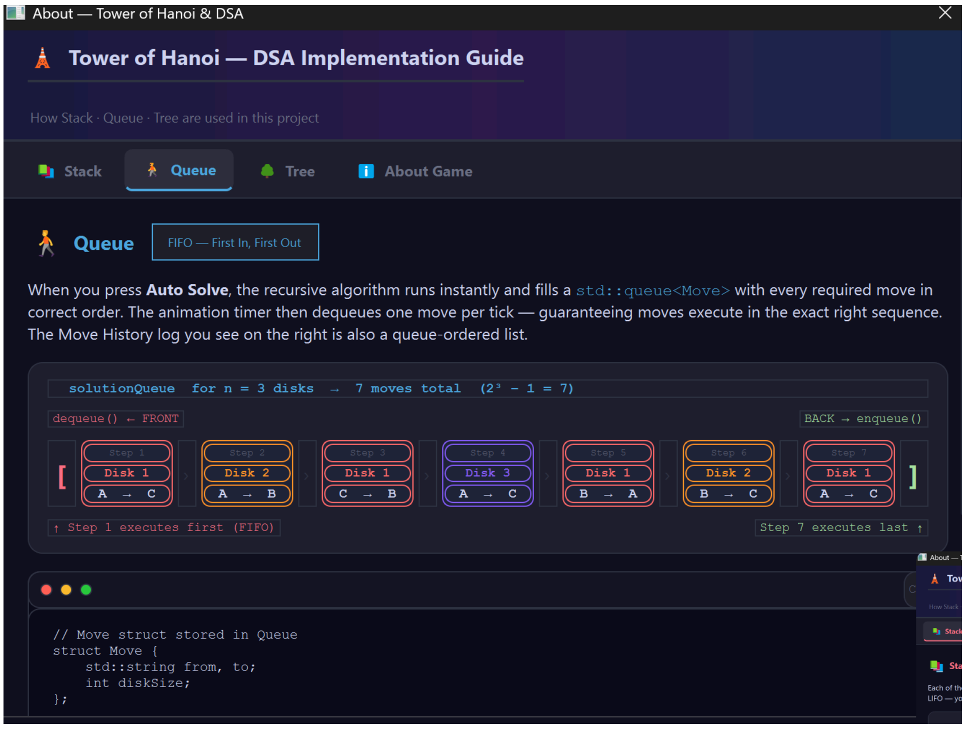
`top()` — peek at top disk (validation)

`empty()` — check if tower is empty

```
// Each tower is a Stack
std::stack<int> towerA, towerB, towerC;

// Move disk from src to dst
bool moveDisk(Tower& src, Tower& dst) {
    if (src.isEmpty()) return false;
```

Fig A.3: Stack Implementation Description Interface



Tower of Hanoi — DSA Implementation Guide

How Stack · Queue · Tree are used in this project

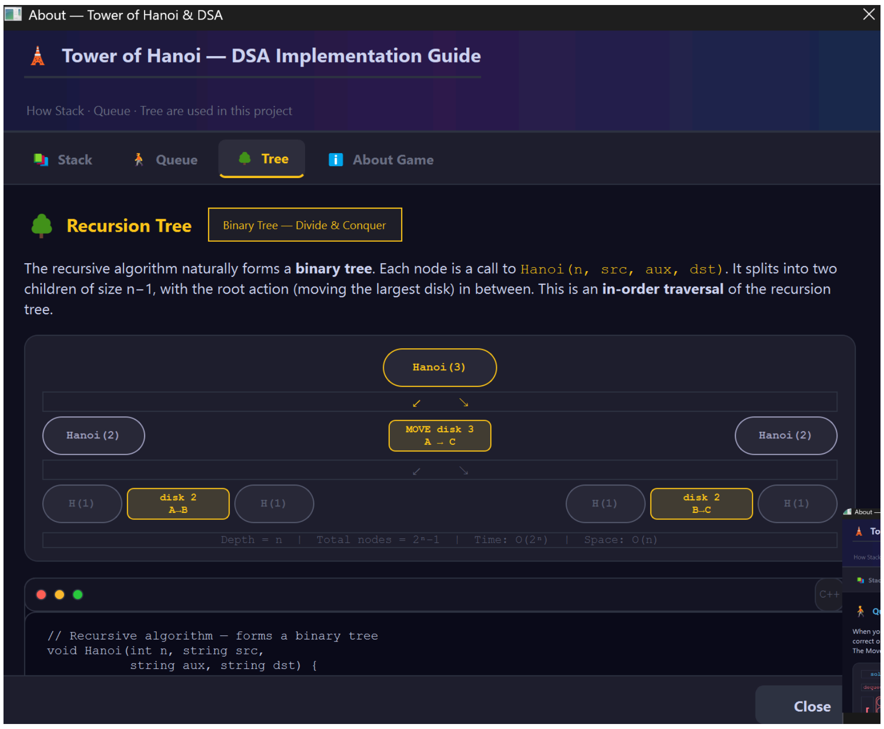
Queue FIFO — First In, First Out

When you press **Auto Solve**, the recursive algorithm runs instantly and fills a `std::queue<Move>` with every required move in correct order. The animation timer then dequeues one move per tick — guaranteeing moves execute in the exact right sequence. The Move History log you see on the right is also a queue-ordered list.

```
solutionQueue  for n = 3 disks  →  7 moves total  (2³ − 1 = 7)
dequeue() ← FRONT                                  BACK → enqueue()
[ Step 1  | Step 2  | Step 3  | Step 4  | Step 5  | Step 6  | Step 7  ]
  Disk 1    Disk 2    Disk 1    Disk 3    Disk 1    Disk 2    Disk 1
  A → C     A → B     C → B     A → C     B → A     B → C     A → C
↑ Step 1 executes first (FIFO)                     Step 7 executes last ↑
```

```
// Move struct stored in Queue
struct Move {
    std::string from, to;
    int diskSize;
};
```

Fig A.4: Queue Implementation Description Interface


About — Tower of Hanoi & DSA

Tower of Hanoi — DSA Implementation Guide

How Stack · Queue · Tree are used in this project

Stack | Queue | Tree | About Game

Recursion Tree — Binary Tree — Divide & Conquer

The recursive algorithm naturally forms a **binary tree**. Each node is a call to `Hanoi(n, src, aux, dst)`. It splits into two children of size n−1, with the root action (moving the largest disk) in between. This is an **in-order traversal** of the recursion tree.

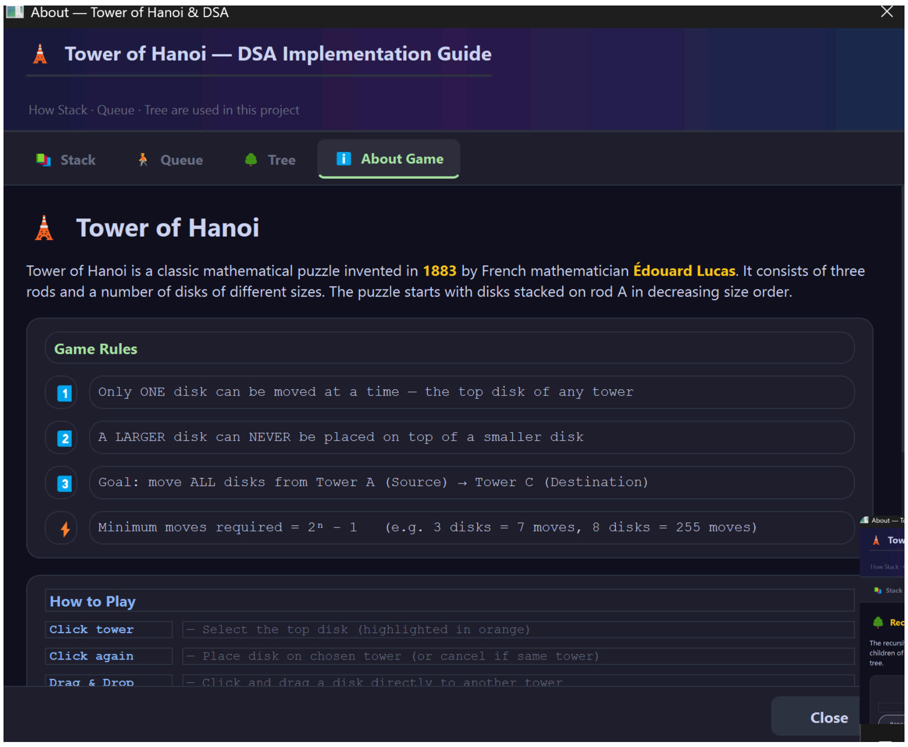
Depth = n | Total nodes = 2^n-1 | Time: $O(2^n)$ | Space: $O(n)$

C++

```
// Recursive algorithm — forms a binary tree
void Hanoi(int n, string src,
           string aux, string dst) {
```

Close

Fig A.5: Tree Implementation Description Interface


About — Tower of Hanoi & DSA

Tower of Hanoi — DSA Implementation Guide

How Stack · Queue · Tree are used in this project

Stack | Queue | Tree | About Game

Tower of Hanoi

Tower of Hanoi is a classic mathematical puzzle invented in **1883** by French mathematician **Édouard Lucas**. It consists of three rods and a number of disks of different sizes. The puzzle starts with disks stacked on rod A in decreasing size order.

Game Rules

1. Only ONE disk can be moved at a time — the top disk of any tower
2. A LARGER disk can NEVER be placed on top of a smaller disk
3. Goal: move ALL disks from Tower A (Source) → Tower C (Destination)

⚡ Minimum moves required = $2^n - 1$ (e.g. 3 disks = 7 moves, 8 disks = 255 moves)

How to Play

Click tower	— Select the top disk (highlighted in orange)
Click again	— Place disk on chosen tower (or cancel if same tower)
Drag & Drop	— Click and drag a disk directly to another tower

Close

Fig A.6: About Game Interface