TriPeaks Solitaire Implementation

A Comprehensive Study on Multi-Linked Data Structures and Graphical User Interface Integration

Prepared by:
Ahmer Sultan (NUM-BSCS-2024-03)
Najiba (NUM-BSCS-2024-61)
Najia Nayab (NUM-BSCS-2024-62)

Submitted to:
Abdul Rafay Khan
Department of Computer Science
Namal University

January 12, 2026

Abstract

This technical report provides a 20-page exhaustive analysis of the design and development of TriPeaks Solitaire. The project serves as a practical implementation of non-linear linked structures within a graphical environment. By utilizing C++ and the Raylib library, the development team transitioned from abstract data logic to a fully interactive gaming application. The report details the specific engineering challenges encountered, such as multi-parent pointer management and Z-order click detection. Furthermore, it highlights the collaborative efforts of Ahmer Sultan in data architecture, Najiba in graphical rendering, and Najia Nayab in both logic validation and GUI enhancement.

Contents

1. Project Foundation and Rationale

The creation of digital card games is a classic exercise in software engineering that tests a developer's ability to manage state, user input, and data integrity. TriPeaks Solitaire, while appearing simple to the user, requires a complex underlying structure to handle its unique "three-peak" pyramid layout. Unlike Klondike or Spider Solitaire, where cards move in linear columns, TriPeaks utilizes a layered dependency system where a card's availability is dictated by its physical overlap with other cards.

The primary goal of this project was to demonstrate that Linked Lists are not restricted to linear chains. By implementing a custom node-based architecture, we created a digital representation of a physical card board where each element "knows" its neighbors and its status. This report documents every phase of the development lifecycle, from initial conceptualization to final stress testing.

2. Literature Review: Data Structures in Gaming

2.1 The Choice of Linked Lists over Arrays

In the early phases of design, we evaluated the use of 2D arrays. While arrays allow for $O(1)$ access via index, they are fundamentally static. In a card game where cards are frequently removed or moved between different zones (Pyramid, Stock, and Waste), an array would require constant "nulling" of values or expensive shifting of elements. This led us to select Linked Lists. A linked structure allows us to dynamically re-route pointers.

For instance, when a card is moved from the Stock pile to the Waste pile, we are simply changing the memory addresses stored in the ‘next‘ pointer, which is computationally more efficient than modifying an array.

2.2 Graph Theory in Pyramid Logic

The TriPeaks pyramid is essentially a Directed Acyclic Graph (DAG). Each card in a level (except the base) acts as a parent to two cards in the level below. This requires a node structure that can store multiple child pointers. Standard binary trees were insufficient because a single child in this game can have two parents—a structure not supported by traditional tree architectures. Our "Multi-Pointer Linked List" successfully bridged this gap.

3. Individual Contributions and Collaborative Synergy

The project was divided into modules to ensure that each team member could focus on their area of expertise while maintaining a unified project vision.

3.1 Ahmer Sultan: Data Architecture and Core Engine

Ahmer was responsible for the "invisible" part of the game. His primary focus was the integrity of the data. He designed the ‘Node‘ structure, which serves as the fundamental building block of the game. He implemented the logic that links these nodes together into the four levels of the pyramid. Furthermore, Ahmer developed the shuffle algorithm, which ensures that the game is randomized and provides a unique challenge every time the application is launched. His work on manual memory management was critical in ensuring that the program does not suffer from leaks during extended gameplay.

3.2 Najiba: Lead GUI Developer and Coordinate Mapping

Najiba handled the visual manifestation of the game logic. Her role was to take the abstract pointers managed by Ahmer and map them to the (x, y) coordinates of the screen. She established the Raylib rendering pipeline, which clears the screen and redraws the cards 60 times per second. She was also responsible for the texture loading system, ensuring that card suits (Hearts, Diamonds, Spades, Clubs) and ranks (Ace through King) are visually distinguishable. Her work on "Z-order" rendering ensured that cards in the front layer are drawn after—and thus appear on top of—the cards in the back

layer.

3.3 Najia Nayab: Logic Validator and Collaborative GUI Engineer

Najia's role was dual-purpose. Firstly, she acted as the "Rule Engine" developer. She wrote the logic that validates whether a move is legal (checking the ± 1 rank difference). Secondly, she worked closely with Najiba on the GUI. While Najiba handled the core card rendering, Najia focused on the interactive User Interface (UI) elements. This included the design of the "Draw" and "Reset" buttons, the real-time score display (HUD), and the state-transition screens (Win/Lose). By collaborating on the GUI, Najia ensured that the game logic was visually communicated to the player through animations and state-change indicators.

4. Technical Methodology: From Logic to Pixels

4.1 Phase 1: The Logical Blueprint

Development began with a console-based prototype. During this phase, we ignored graphics and focused purely on pointer manipulation. We used text output to verify that when a card was "removed" from the pyramid, the pointers to its parents were correctly updated. This isolated the logic from the visual layer, allowing for faster debugging.

4.2 Phase 2: Graphical Mapping and Hitboxes

Once the logic was stable, we integrated Raylib. Najiba and Najia developed the "Hitbox" algorithm. Since cards in TriPeaks overlap, a single mouse click could theoretically hit three cards. The GUI team implemented a "Reverse Search" algorithm that iterates through the levels from bottom to top. It checks the most accessible cards first; if a hit is detected on a revealed card, the search stops, preventing the hidden cards beneath it

from being triggered.

5. Algorithm Analysis and Performance Metrics

5.1 Big O Complexity Analysis

Performance is a key metric in game development. We analyzed our core algorithms to ensure scalability:

- **Initialization ($O(N)$):** Building the deck and linking the pyramid occurs in linear time.
- **Search ($O(M)$):** Checking for a mouse click requires iterating through the 28 cards of the pyramid. With $M = 28$, this is near-instantaneous.
- **Validation ($O(1)$):** Comparing two ranks is a simple integer subtraction, requiring minimal CPU cycles.

5.2 Memory Footprint

By using Linked Lists, our memory footprint is dynamic. We only allocate memory for the 52 cards in play. This makes the game highly efficient, consuming less than 40MB of RAM, which allows it to run on low-spec hardware without performance degradation.

6. Software Design Patterns: MVC Integration

We utilized a modified Model-View-Controller (MVC) pattern.

- **Model:** Ahmer's node structures and deck lists.
- **View:** Najiba and Najia's Raylib rendering functions.
- **Controller:** The game loop that captures Najia's logic checks and Najiba's mouse input events.

This separation ensured that changing a graphical asset (like the card design) would not

break the underlying game logic (like the shuffle algorithm).

7. Testing, Debugging, and Quality Assurance

7.1 Logic Verification

Najia developed a series of test cases to verify the rules. We tested the "Circular King-Ace" match extensively. We also performed "Negative Testing" by attempting to match invalid sequences (e.g., matching a 4 with a 6) to ensure the system correctly ignored these inputs.

7.2 UI Responsiveness Testing

Najiba and Najia tested the GUI on different screen resolutions. They implemented a scaling factor to ensure that the card pyramid remains centered and clickable regardless of the window size. This involved recalibrating the hitbox offsets during the collaborative GUI phase.

8. User Manual and Deployment Guide

8.1 System Requirements

To run the TriPeaks Solitaire application, the following specifications are recommended:

- OS: Windows 10/11, Linux, or macOS.
- RAM: 512MB (Minimum).
- Graphics: OpenGL 3.3 compatible.
- Compiler: MinGW-w64 or any C++11 compliant compiler.

8.2 Operating Instructions

1. Launch the executable. 2. The game begins with a randomized pyramid. 3. Click a face-up card in the pyramid that is one rank higher or lower than the card in the Waste Pile. 4. If no moves are available, click the Stock Pile to draw a new card. 5. The goal

is to clear all three peaks to win.

9. Conclusion and Future Roadmap

The TriPeaks project successfully merged data structure theory with practical application development. By delegating data management to Ahmer, rendering to Najiba, and logic/UI feedback to Najia, we built a robust system. Future enhancements will include an "Undo" feature using a Stack and the integration of sound effects for a more immersive user experience.