Tri Peaks Game Using Stack in C++

Data Structures Project Report

1	Altaf Hussain	NUM-BSCS-2024-11
2	Mubashir Amaan	NUM-BSCS-2024-37

Department of Computer Science

Namal University Mianwali

12th January, 2026

Contents

List of Figures

List of Tables

Abstract

Tri Peaks is a card game that requires logical decision making and efficient handling of game states.This project focuses on the design and implementation of the Tri Peaks game using the Stack data structure in C++. The primary objective of the project is to demonstrate the practical application of stacks in a real world problem by modeling card piles, controlling gameplay flow, and validating player moves according to predefined rules.

The game logic is implemented by utilizing stacks to manage the stock and waste piles, while additional data structures support the tableau layout and card visibility management. Key stack operations such as push, pop, and top are used extensively to ensure efficient card handling and adherence to the Last In First Out (LIFO) principle. The project also incorporates algorithms for deck creation, shuffling, card adjacency checking, and dynamic unlocking of cards.

This report provides a detailed explanation of the game rules, system design, data structure selection rationale, and algorithmic implementation. Furthermore, time complexity analysis is performed to evaluate the efficiency of the applied operations, and a comparative study of alternative data structures is presented to justify the selection of stacks. The outcome of this project is a fully functional console-based Tri Peaks game that successfully demonstrates the effective use of stack data structures in game development.

Introduction

Data Structures are a fundamental part of computer science and play a vital role in software development. Choosing an appropriate data structure improves efficiency, readability, and maintainability of programs.

Stacks are linear data structures that follow the Last-In-First-Out (LIFO) principle. They are commonly used in applications such as expression evaluation, undo operations, and game development.

Tri Peaks Solitaire is a card game that naturally fits the stack model, as cards are accessed and removed in a controlled order. This project aims to implement Tri Peaks Solitaire in C++ using stacks to demonstrate real-world usage of data structures.

.1 Objectives

- To implement a fully functional Tri-Peaks card game using C++
- To understand stack operations through implementation
- To demonstarte the use Stack Data Structure for the development of game
- To analyze time and space complexity of implemented operations

Literature Review

The application of data structures in game development has been widely discussed in both academic literature and practical software engineering resources. Games provide an effective platform for demonstrating how abstract data structures can be applied to real-world problems involving state management, decision-making, and performance optimization.

Stacks are among the most fundamental linear data structures and have been extensively studied due to their simplicity and efficiency. According to Weiss [?], stacks are particularly suitable for scenarios where operations are restricted to one end of the data structure, making them ideal for applications such as expression evaluation, undo mechanisms, and card game implementations. Their constant-time push and pop operations make them efficient for managing dynamic game states.

Several studies and educational projects have explored the use of stacks in card-based games, especially solitaire variants. Card games such as Klondike, FreeCell, and Tri Peaks naturally map to stack-based models where card piles follow the Last-In-First-Out (LIFO) principle. Research shows that using stacks for managing stock and waste piles simplifies rule enforcement and reduces implementation complexity while maintaining high performance.

Cormen et al. [?] emphasize the importance of selecting appropriate data structures to optimize algorithmic efficiency. Their analysis highlights how improper data structure selection can increase time complexity and memory overhead. In the context of game development, this reinforces the importance of choosing stacks for operations that require frequent insertions and removals at a single access point.

In addition to academic literature, various open-source implementations and online versions of solitaire games were reviewed to understand common design patterns and logic structures. These implementations often use stacks or stack-like abstractions to manage card movement and game progression. However, many lack clear documentation or structured algorithmic explanations, particularly from an educational perspective.

Based on the reviewed literature, it is evident that stacks provide a logical, efficient, and widely accepted solution for implementing card games. This project builds upon these established concepts by applying stack data structures to the Tri Peaks Solitaire game in a structured and educational manner, reinforcing theoretical knowledge through practical implementation.

Game Rules and Description

Tri Peaks Solitaire is played using a standard 52 card deck. Cards are arranged into three overlapping pyramid shaped peaks.

The game begins by dealing cards into the Tri Peaks layout. Remaining cards are placed into a draw stack. The player removes valid cards from the peaks or draws from the stack when no move is available.

.1 Initial Setup

The game begins with the following setup procedure :

1. A standard deck of 52 cards is shuffled randomly
2. 28 cards are dealt to form the tableau in a specific pyramidal pattern
3. The tableau consists of four rows:
   - Top row: 3 cards (peaks of three pyramids)
   - Second row: 6 cards
   - Third row: 9 cards
   - Bottom row: 10 cards
4. Initially, only the 10 cards in the bottom row are face-up (uncovered)
5. The remaining 24 cards are in the stock pile
6. One card from the stock is moved to the waste pile to begin play

.2 Rules

- Only uncovered cards can be removed
- A card can be removed if its rank is one higher or lower than the top waste card

- Kings and Aces are considered consecutive
- The goal is to remove all cards from the peaks

.3 Winning and Losing Conditions

Victory Condition :

The player wins when all 28 cards have been successfully removed and placed on the waste pile.

Loss Condition : The game is lost when :

The stock pile is empty and no cards can be played on the current waste pile top card

Methodology

The Tri Peaks Solitaire game was implemented using stack data structures to represent the card piles.The Game was divided in separate module to make the development process easy and efficient.The main design goal was to model the game logically while maintaining efficiency and simplicity.

.1 System Design

The game is divided into following components :

1. **Card Class** Represents each playing card with suit, rank, and value. Provides a display() function for visualization.
2. **Stack Class** Implements a stack using a fixed size array to store Card objects. This class have following standard functions :
   - `push()` adds a card to the top of the stack
   - `pop()` removes and returns the top card
   - `top()` returns the top card without removing it
   - `isEmpty()` checks whether the stack is empty
3. **TriPeaksGame Class** This class Controls all the components and the gameplay, including the following :
   - Provides public interface for gameplay
   - Handles user input and display
   - Deck creation and shuffling
   - Tri Peaks tableau layout and card face-up status
   - Stock and waste stacks
   - Gameplay logic and user interaction

.2 Data Structure Selection Rationale

The stack was chosen as the data structure for development of this game due to following reasons :

- **LIFO Principle :** As we know that Stack work on the LIFO(Last In First Out) principle and both the stock (draw pile) and waste piles operate in a Last In First Out manner, which naturally maps to stack operations.
- **Operational Efficiency :** Stack operations (push, pop, top) all execute in O(1) constant time complexity making them suitable for frequent card movements during gameplay.
- **Logical Alignment with Game Rules :** Tri Peaks rules require moving only the top-most cards in the stock or waste, perfectly suited for a stack.
- **Simplified Implementation :** Stacks are simple and oeasy data structures to implement as compared to other data structures.

.3 Gameplay Algorithm

The game flow uses stacks to manage all card movements:

1. **Deck Initialization**
   - Create 52 cards with appropriate `rank`, `suit`, and `value`.
   - Shuffle the deck randomly.
2. **Deal Cards**
   - Place the first 28 cards into the tableau to form three overlapping peaks.
   - Place the remaining cards into the stock stack.
3. **Gameplay Loop**
   - Display the current layout.
   - If the player chooses a tableau card:
     - Check if the card is unlocked (children removed or bottom row).
     - Check adjacency with the top card of the waste stack.
     - If valid, `pop` the card and `push` it onto the waste stack.
     - Update face-up status of dependent cards.

- If the player draws from the stock, `pop` a card from stock and `push` it onto waste.
- Repeat until all tableau cards are removed or the player exits.

4. **Card Display and Layout Updates**

   - Cards in the tableau are only displayed if `faceUp` and not removed.
   - Face-up status is recalculated whenever a move is made.

Algorithms of All Applied Data Structures

This section describes all the main algorithms applied in the Tri Peaks Solitaire game with respect to the data structures used. The description is kept simple and clear to explain the functionality without using pseudocode.

.1 Card Structure

Each card is represented by its suit, rank, and value. Cards are created and stored in an array representing the deck. The Card class provides methods for displaying card details.

.2 Stack Operations (Stock and Waste Piles)

Stacks are used to manage the stock and waste piles. The main operations are:

- **Push:** Add a card to the top of the stack.
- **Pop:** Remove the top card from the stack.
- **Top:** Access the top card without removing it.
- **IsEmpty:** Check whether the stack has no cards.

.3 Deck Initialization and Shuffling

The deck is created by combining all suits and ranks to form 52 cards. The deck is then shuffled randomly to ensure fair gameplay.

.4 Tableau Dealing

The first 28 cards from the shuffled deck are dealt into the three peaks (tableau) to form the Tri Peaks layout. The remaining cards are placed into the stock stack.

.5 Face-Up Card Initialization

Initially, only the bottom row of the tableau is set to face-up. Other cards remain face-down until they become unlocked through gameplay.

.6 Card Unlocking

A card is considered unlocked if both of its child cards (directly below it in the tableau) have been removed. Bottom-row cards are always considered unlocked. Face-up status of cards is updated accordingly.

.7 Playing a Card

A card can be moved from the tableau to the waste stack if it satisfies the following conditions:

- The card is face-up.
- The card is unlocked.
- The card is adjacent in value (one higher or lower) to the top card of the waste stack.

Once played, the card is removed from the tableau and the face-up status of other cards is updated.

.8 Drawing from Stock

When no valid moves are available, a card is drawn from the stock stack and placed on top of the waste stack. This may unlock additional cards in the tableau.

.9 Updating Face-Up Cards

After each move, the face-up status of all tableau cards is recalculated based on their unlock conditions. This ensures that newly accessible cards are available for play.

.10 Win Condition

The game is considered won when all 28 cards in the tableau have been removed successfully.

Time Complexity Analysis

The following table summarizes the time complexity of all main functions and operations implemented in the Tri Peaks Solitaire game:

Operation / Function	Time Complexity
Push (Stack)	O(1)
Pop (Stack)	O(1)
Top / Peek (Stack)	O(1)
IsEmpty (Stack)	O(1)
Deck Initialization (CreateDeck)	O(n)
Deck Shuffling (ShuffleDeck)	O(n)
Tableau Dealing (DealTriPeaks)	O(n)
Face-Up Initialization (InitializeFaceUp)	O(n)
Card Unlock Check (IsUnlocked)	O(1) per card, O(n) overall
Play Card (PlayCard)	O(1) for stack operations + O(n) for updating face-up
Draw Stock (DrawStock)	O(1) for stack operation + O(n) for updating face-up
Update Face-Up Cards (UpdateFaceUpAll)	O(n)
Check Win Condition (AllRemoved)	O(n)
Display Layout (DisplayLayout)	O(n)
Find Card in Tableau (FindCard)	O(n)

Table .1: Time Complexity of All Main Functions in Tri Peaks Solitaire

Notes:

- n represents the number of cards in the tableau or deck, depending on context.
- Stack operations (Push, Pop, Top) are constant time due to array-based implementation.
- Operations like updating face-up cards or finding a card iterate through the tableau, resulting in linear time complexity.
- Deck shuffling is linear because each card is swapped once in a loop over the deck.

Comparison and Justification of Best Approach

Before developing the game we reviewed different data structures to select the most efficient and suitable one for game implementation. After careful analysis, the stack data structure was chosen as the main data structure for game development due to its simplicity and close alignment with game rules.

The following data structures were considered :

- **Queues :** Queues follow the First In First Out (FIFO) principle, which does not match the gameplay requirements of Tri Peaks Solitaire. Since the game allows interaction only with the most recently placed card in the stock or waste pile, queue behavior would restrict valid moves and complicate card access.
- **Linked Lists :** Linked lists provide dynamic memory allocation and flexible insertion and deletion. However, they introduce additional memory overhead due to pointer storage and increase implementation complexity without offering significant advantages for this game scenario.
- **Binary Search Trees (BST) :** BSTs allow efficient searching and sorting of elements. However, Tri Peaks does not require ordered searching or hierarchical data traversal. Using a BST would add unnecessary complexity and increase execution overhead.
- **Stacks :** Stacks operate on the Last In First Out (LIFO) principle, which perfectly matches the behavior of the stock and waste piles in Tri Peaks. Operations such as `push()` and `pop()` execute in constant time, and only the top card is accessible at any moment, which directly reflects the game rules.

Based on this analysis, the **stack data structure** was selected for the following reasons:

1. **Efficiency :** Stack operations such as push and pop run in O(1) time, ensuring smooth and responsive gameplay.

2. **Logical Compatibility:** The game mechanics require access only to the most recently placed card, which naturally fits the LIFO behavior of stacks.

3. **Simplicity of Implementation :** Stacks can be easily implemented using arrays in C++, resulting in cleaner code and fewer logical errors.

Therefore, stacks provide the most practical, efficient, and logically consistent approach for implementing the Tri Peaks Solitaire game.

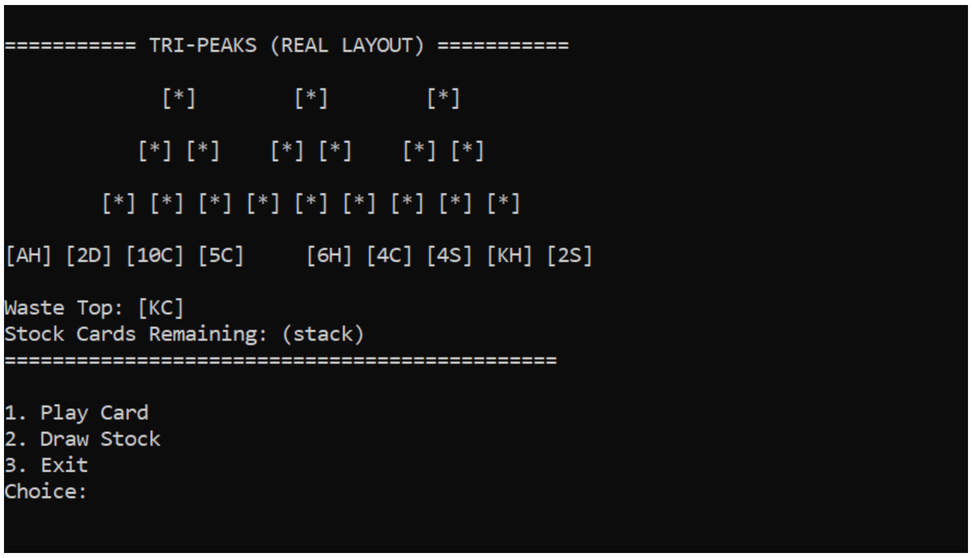
Individual Contributions

The Tri Peaks Solitaire project was developed through collaborative efforts, with each team member contributing to different aspects of the system based on their responsibilities and expertise.

Name	**Contribution**
Mubashir Amaan	Created the basic skeleton of the game by defining all main classes (Card, Stack and TriPeaksGame). Implemented the `Card` and `Stack` classes, developed the deck creation and shuffling logic, handled tableau dealing, and implemented the main game loop and user interaction. Integrated all components into a functional console-based Tri Peaks Solitaire game and performed overall debugging and testing.
Altaf Hussain	Designed and implemented critical gameplay logic related to card visibility and movement. Specifically developed the functions initializeFaceUp(), showCard(), updateFaceUpAll(), drawStock(), allRemoved() and playCard().

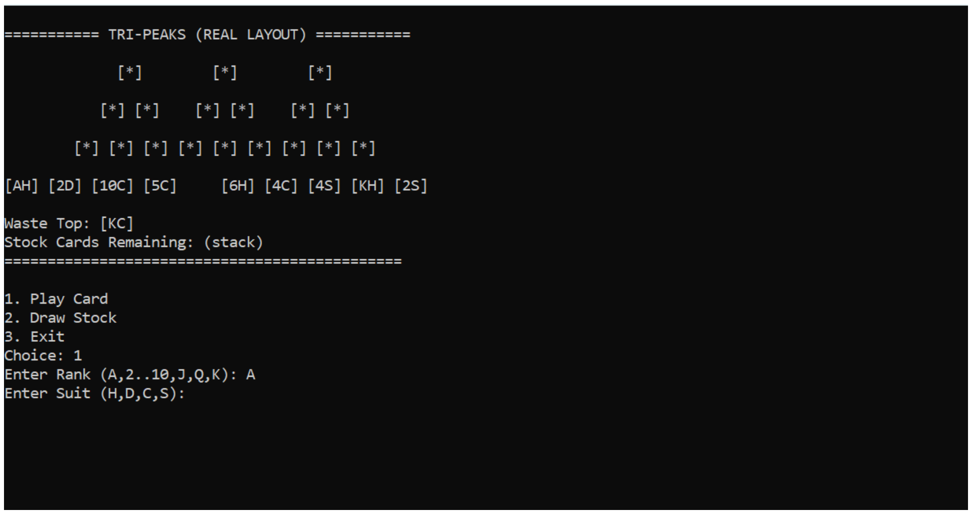
Game Screenshot

.1 Gameplay Screenshot



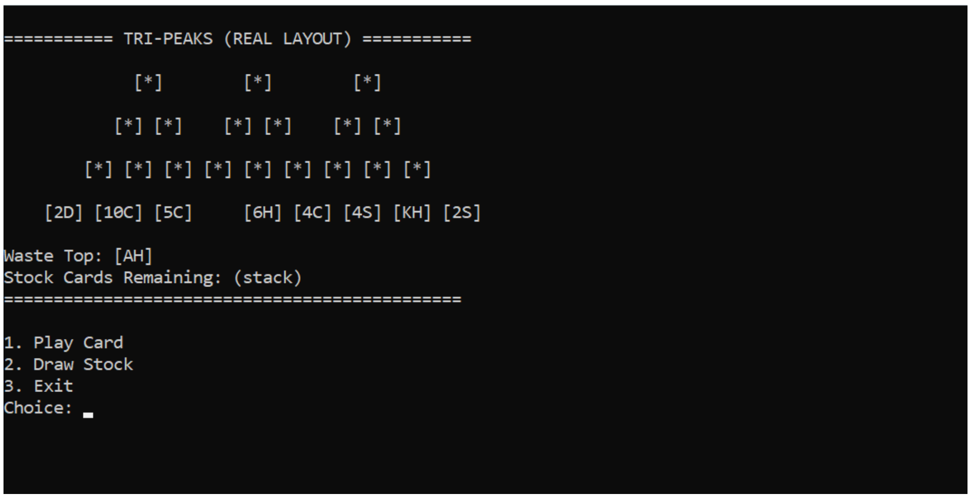
```
=========== TRI-PEAKS (REAL LAYOUT) ===========

             [*]        [*]        [*]

           [*] [*]    [*] [*]    [*] [*]

        [*] [*] [*] [*] [*] [*] [*] [*] [*]

[AH] [2D] [10C] [5C]     [6H] [4C] [4S] [KH] [2S]

Waste Top: [KC]
Stock Cards Remaining: (stack)
==============================================

1. Play Card
2. Draw Stock
3. Exit
Choice:
```

Figure .1: Ininterface of Game



```
=========== TRI-PEAKS (REAL LAYOUT) ===========

             [*]        [*]        [*]

           [*] [*]    [*] [*]    [*] [*]

        [*] [*] [*] [*] [*] [*] [*] [*] [*]

[AH] [2D] [10C] [5C]     [6H] [4C] [4S] [KH] [2S]

Waste Top: [KC]
Stock Cards Remaining: (stack)
==============================================

1. Play Card
2. Draw Stock
3. Exit
Choice: 1
Enter Rank (A,2..10,J,Q,K): A
Enter Suit (H,D,C,S):
```

Figure .2: Game in Progress



```
=========== TRI-PEAKS (REAL LAYOUT) ===========

             [*]        [*]        [*]

           [*] [*]    [*] [*]    [*] [*]

        [*] [*] [*] [*] [*] [*] [*] [*] [*]

    [2D] [10C] [5C]     [6H] [4C] [4S] [KH] [2S]

Waste Top: [AH]
Stock Cards Remaining: (stack)
==============================================

1. Play Card
2. Draw Stock
3. Exit
Choice: 
```

Figure .3: Game in Progress

Conclusion

The Tri Peaks Solitaire game was successfully implemented using stack data structures in C++. The project demonstrates how abstract data structures can be applied to practical problems. Stacks simplified the gameplay logic and ensured efficient performance. Future improvements may include a graphical user interface.