Namal University, Mianwali

Data Structures Project Report

Project: Klondike Game

Course: CSC-200- Data Structure

Department: Computer Science

Semester: Fall 2025

Submitted To:
MR. Abdul Rafay Khan
Lecturer, Department of Computer Science

Submitted By:

Muhammad Shoaib	Reg No: Num-Bscs-2024-57
M Jamal Ahmed Khan	Reg No: Num-Bscs-2024-51
Naveed Abbasi	Reg No: Num-Bscs-2024-54
Muhammad Wasif	Reg No: Num-Bscs-2024-59
Bilal Ahmed Khan	Reg No: Num-Bscs-2024-21
Mubashir Hassan	Reg No: Num-Bscs-2024-38

Submission Date: 12 Jan 2026

Abstract

This project focuses on the implementation of the Klondike card game using different data structures. The main objective is to understand how data structures behave when applied to a real world problem. Three separate versions of the Klondike game were developed using stack, linked list, and queue. Each implementation highlights the strengths and limitations of the chosen data structure. A graphical user interface was also developed to allow user interaction. This project provides a clear comparison between the data structures based on functionality, flexibility, and performance.

Contents

1 Introduction

Data structures has a very important role in computer science field. This help in organizing and managing data efficiently. Choosing of data structure can help to improve the performance of a program. This project is part of the data structures course. In this project, we selected a real world Game as a problem and solved it using data structures. The selected project is a Klondike game, which is a popular single player card game. The game was implemented using three different data structures to analyze their behavior.

2 Literature Review

Klondike Solitaire, which is just known as "Solitaire," is a very popular game. It is not just for fun, it is also studied by experts in math and computer science. This section looks at the history of the game, the math behind it, and how difficult it is for computers to solve.

History and Evolution

No one is exactly sure when Solitaire started, but most people believe it started in Northern Europe in the late 1700s. The first written record of the game is in a German book from 1798. After that,it became very popular in France. We know this because we still use French words in the game today, like tableau and solitaire (which means "alone"). There is a famous story that Napoleon played it while he was in exile, but many historians think this might not be true.

The version called "Klondike" became famous in North America during the Gold Rush in late 1800s. but the game became globally famous because of computers. In 1990, a Microsoft intern named Wes Cherry created the digital version for Windows 3.0. It was designed to teach people how to use a mouse to "drag and drop" items, but it became one of the most played computer games ever.

Studies show that about 82% of all Klondike games can be won, but only if the player plays perfectly. A skilled human player will usually win about 43% of the time, while a casual player might only win 10–15% of games. This shows why game features like "Undo" or "Hints" are helpful, they help close the gap between human mistakes and the perfect solution.

Data Structures in Card Games

Computer science textbooks often use card games to explain data structures. The Stack is the most common example because it works just like a pile of cards.We can only take the top card off (Last-In, First-Out).

Now newer game development suggest that **Linked Lists**(Double link-list) are often better for games like Solitaire. While a Stack is good for a single pile, a Linked List is better for moving groups of cards from one pile to another. This project tests both methods to see which one works best for a real game.

3 Game Rules and Description

Klondike is a classical single player card game played with a deck of 52 cards. The game environment consists of four main areas.

1. **Tableau Piles:** Seven piles arranged in a row. The first pile containing one card, the second pile containing two cards, and like this seventh pile containing seven cards. Only the top cards of every pile is face-up and others are face down.

2. **Foundation Piles:** Four empty piles one for each, where all card are to be placed in ascending order from Ace to King.

3. **Stock Pile:** After distributing cards to the tableau else cards are placed in the stock pile. Player can draw cards from the stock.

4. **Waste Pile:** Drawn cards from the stock placed in the waste pile and player can move card from waste pile.

Gameplay Rules

- Player can move face up card or a sequence of a face up cards from one tableau to another in a descending order and a changing color.

- Only a king or a sequence starting with a king can be placed in an empty tableau pile.

- one Card at a time move from the waste pile or tableau pile to the foundation pile. Valid move If the picked card is one rank greater than the card currently placed on source foundation pile and of same suit .If oat first for empty Ace can be place on the foundation .
- Drawn cards from the stock pile placed in the waste pile. 1, 2 or 3 can be drawn based on the difficulty.
- The objective of the game is to move all cards to the foundation piles in ascending order of same suit.

4 Methodology

The project used an incremental development approach. Three versions of the Klondike game were created using stack, linked list, and queue. This allowed the team to compare the performance, limitations, and suitability of each data structure for the game. The development began by creating and organizing all cards, assigning their suit, rank, color and face up status. Then we implemented the game logic to move cards according to the Klondike rules. After this graphical user interface (GUI) was implemented so player can interact with the game visually, seeing moving of the card and flipping of the cards automatically. Finally, the game was tested to make sure that game ran properly and according to the real Klondike game.

Stack-Based Version

All card pile (Tableau, Foundation, Stock and Waste) were implemented using stacks. Each card store its suit, rank, color and face up status. The Game logic used main stack operations like (Push, Pop and Top) to perform valid moves. The GUI updates the display after each move, showing card movements and flipping cards when needed.

Linked List Based Version

A linked list data structure was used to implement all major card piles, including the Waste , Foundation , Stock, and Tableau piles. Each pile was represented as a dynamically managed linked list, allowing efficient insertion and removal of cards during game. Every card node has essential attributes such as suit, rank, color, and face-up status, making easy for rule validation and visual representation.

The linked list structure provided direct control over node traversal, making it suitable for tableau operations involving partial card sequences, where multiple consecutive cards must be moved together. Cards could be inserted or removed from any position into a pile without requiring a shifting operations, which improved flexibility compared to array-based or stack-only approaches.

The game logic is designed by traversing these linked lists to validate moves according to Klondike Solitaire rules, including alternating colors, descending rank order in the tableau, and ascending rank order within matching suits in the foundation piles. The Stock and Waste piles were also managed using linked lists, used for draw and recycle operations through node transfers.

For Graphical User Interface Raylib was used with the linked list state to render card positions, handle drag-and-drop interactions, and update animations. Upon successfully transferring all cards to the foundation piles, the system detected the winning condition through linked list state and displayed an related game completion message.

Queue Based Version

A queue is implemented in this game for all major card piles such as Tableau, foundation, stock and waste. Each pile is implemented in circular array based queue with front and rear that allowing the cards management during the game through enqueue and dequeue operations. Each card has attributes such as rank number, suit character and face up boolean status and makes game to check the rule validation and representation through methods such that check_red (), card_details () etc.

The queue structure gives FIFO access patterns with O(1) enqueue and dequeue operations that makes it for suitable for stock to waste, checking card sequences, through accessing top card tableau or foundation that needs queue complete traversal using a temporary queues to peek that last element without removing it permanent. Cards are added in pile in rear and dequeue through front but operations such that moving the multiple face up cards from tableau to tableau in systematic way like dequeue to temporary queue follow this to en queuing that maintains the game state consistence. Game logic is operated by traversing queues to valid moves according to game rules.

Stock and waste piles contains the 24 capacity queue that supports in card draw operations such that dequeue from stock and enqueue in waste with face up flipping and vice versa. In Foundation, there is simple enqueue is used that accepts the descending and dequeue for use temporary queues. Raylib is used to make GUI of game there is use of ray lib built in function to handle mouse click, render the card position.

5 Algorithms of All Applied Data Structures

5.1 Stack Based Klondike Game Algorithm

This section describes that how stack based Klondike game is implemented by using **Stack**. Each pile in the game is represented as stack. A graphical user interface (GUI) built using Raylib library provides real time visual interaction with the game.

5.1.1 1. Game Initialization

1. Create a deck of 52 cards. Each card stores:
    - Suit: Hearts, Diamonds, Clubs, Spades
    - Rank: Ace to King
    - Color: Red or Black
    - Face-up status: True or False
2. Push all cards into the stock stack.
3. Shuffle the stock stack randomly.
4. Deal cards to 7 tableau stacks:
    - Tableau 1 receives 1 card, tableau 2 receives 2 cards, ..., tableau 7 receives 7 cards.
    - Only the top card in each tableau is face-up; the rest remain face-down.
5. Initialize 4 empty foundation stacks and an empty waste stack.
6. Render all stacks on the GUI:

- Stock and waste piles at the top-left.
- Foundation piles at the top-right.
- Tableau piles arranged in a row below the stock and waste.
- Face down cards show a card back image face up cards show rank and suit.

5.1.2 2. Drawing Cards from Stock

- Clicking on the stock pile pops a card from the stock stack.
- The card's `faceup` property is set to true and it is pushed onto the waste stack.
- The GUI automatically updates to display the top card in the waste pile.
- If the stock is empty, all cards from the waste are moved back to the stock flipped face down, and the GUI is updated accordingly.

5.1.3 3. Moving Cards Between Stacks

Waste to Tableau/Foundation:

- Validate moves based on Klondike rules:
  - Tableau: top card must be one rank lower and opposite color, or a King if the tableau is empty.
  - Foundation: same suit and next rank in sequence, or Ace if empty.
- If valid, pop from waste and push onto the target stack.
- The GUI updates the card movement visually.

Tableau to Tableau:

- Select a sequence of face up cards from the source tableau using click and drag.
- Validate the sequence is in descending order with alternating colors.

- Pop cards from the source stack and push them onto the destination stack.
- Animate the movement in the GUI for smooth visuals.

Tableau to Foundation:

- The top card of a tableau can be moved to the foundation stack if it is of same suit and color and one rank greater than the top card of the foundation pile or if foundation pile is empty only ace can place on the foundation pile.
- Pop from tableau and push to foundation, GUI updates accordingly.

Foundation to Tableau:

- The top card of a foundation stack can be moved back to a tableau if it is os different color and one rank lower than top card of tableau pile or only king can be moved to the tableau pile.
- The stack is updated and GUI reflects the change.

5.1.4 4. Flipping Tableau Cards

- When a face up card is moved from a tableau the new top card is flipped face up automatically.
- GUI shows the flip animation instantly.

5.1.5 5. Checking Win Condition

- The player wins the game when each of the four foundation piles has all 13 cards of its suit, arranged in ascending order from Ace to King.
- GUI displays a winning message.

5.1.6 6. User Interaction

1. Click on stock to draw a card.
2. Drag and drop cards or sequences between tableau piles.
3. Drag cards to foundation piles.
4. GUI updates are instantaneous and animated.

5.2 Linked List Based Klondike Game Algorithm

Each pile in the game is represented as a linked list, allowing dynamic memory allocation, efficient traversal, and flexible manipulation of card sequences. A graphical user interface (GUI) built using the **Raylib** library provides real-time visual interaction with the game.

5.2.1 1. Game Initialization

1. A standard deck as per Klondike of 52 cards is created. Each card node stores:
    - Suit (0–3 representing Hearts, Diamonds, Clubs, Spades)
    - Rank (1–13 representing Ace to King)
    - Color (Red or Black)
    - Face-up status (True or False)
2. All cards are inserted into the **stock pile**, implemented as a linked list using head insertion.
3. Then stock pile is shuffled for random card distribution with the elp of array.
4. Cards are distributed to seven tableau linked lists:
    - Tableau 1 receives 1 card, tableau 2 receives 2 cards, up to tableau 7 receiving 7 cards.
    - Only the last card in each tableau is face-up; all others remain face-down.
5. Four empty foundation linked lists and an empty waste linked list are initialized.
6. The GUI renders all piles dynamically:
    - Stock and waste piles at the top-left.
    - Foundation piles at the top-right.
    - Tableau piles arranged horizontally below.
    - Face-down cards display the card back image, while face-up cards display their rank and suit.

5.2.2 2. Drawing Cards from Stock

- Clicking the stock pile removes a card node from the head of the stock linked list.
- The card's face-up status is set to true and it is inserted at the head of the waste linked list.
- If the stock becomes empty, all cards from the waste pile are recycled back into the stock:
  - Cards are transferred through a temporary linked list.
  - Each card is flipped face-down before reinsertion.
- The GUI updates immediately to reflect these changes.

5.2.3 3. Moving Cards Between Linked Lists

Waste to Tableau / Foundation:

- The top card of the waste linked list is selected.
- Moves are as per Klondike rules:
  - Tableau piles accept a card of opposite color and one rank lower, or a King if empty.
  - Foundation piles accept a card of the same suit and one rank higher, or an Ace if empty.
- If valid, the card node is removed from the waste list and inserted into the destination linked list.

Tableau to Tableau:

- A sequence of consecutive face-up cards is selected by traversing the tableau linked list.
- The sequence is validated for descending rank order with alternating colors.
- Valid sequences are removed node-by-node from the source tableau and inserted into the destination tableau.

- This linked list traversal allows efficient handling of multi-card moves.

Tableau to Foundation:

- Only the head (top card) of a tableau linked list can be moved.
- The card must match the suit of the foundation and be one rank higher than the foundation's top card.
- If valid, the node is transferred to the relative foundation linked list.

Foundation to Tableau:

- The top card of a foundation linked list can be moved back to a tableau.
- The destination tableau must either be empty (King only) or satisfy alternating color and descending rank rules.

5.2.4 4. Flipping Tableau Cards

- After a card or sequence is removed from a tableau, the new head node is checked.
- If the card is face-down, it is removed, its face-up status is updated, and it is reinserted.
- The GUI immediately displays the flip animation.

5.2.5 5. Checking Win and Loss Conditions

- The game is won when all four foundation linked lists contain exactly 13 cards each.
- A loss condition is detected when:
  - Stock and waste piles are empty, or time up
  - No legal moves exist across tableau and foundation piles.
- The GUI displays appropriate win or game-over messages.

5.2.6 6. User Interaction and GUI Synchronization

1. Mouse clicks are used to draw cards from the stock.
2. Drag-and-drop functionality allows moving single cards or sequences.
3. Undo functionality is implemented using snapshots of linked list states.
4. The GUI continuously synchronizes with linked list data to render cards, animations, hints, timers, and sound effects.

5.3 Queue Based Klondike Game Algorithm

This section describes the implementation of the Klondike Solitaire game using a **queue** data structure. All card piles are represented using queues to demonstrate how a First-In-First-Out (FIFO) structure behaves in a game that naturally follows a Last-In-First-Out pattern. A graphical user interface (GUI) is used to visually represent card movements and game progress.

5.3.1 1. Game Initialization

1. A standard deck of 52 cards is created. Each card stores:
   - Suit (Hearts, Diamonds, Clubs, Spades)
   - Rank (Ace to King)
   - Color (Red or Black)
   - Face-up status (True or False)
2. All cards are enqueued into the **stock queue**.
3. The stock queue is shuffled by converting it into an array, applying shuffle logic, and converting it back into a queue.
4. Cards are dealt into seven tableau queues:
   - Tableau 1 receives 1 card, tableau 2 receives 2 cards, up to tableau 7 receiving 7 cards.
   - Only the rear card of each tableau queue is marked face-up.
5. Four empty foundation queues and an empty waste queue are initialized.

6. The GUI renders all piles:
    - Stock and waste piles at the top-left.
    - Foundation piles at the top-right.
    - Tableau piles arranged horizontally below.

5.3.2 2. Drawing Cards from Stock

- Clicking the stock dequeues a card from the front of the stock queue.
- The card is flipped face-up and enqueued into the waste queue.
- If the stock queue becomes empty, waste cards are recycled:
    - All waste cards are dequeued one by one.
    - Cards are flipped face-down.
    - Cards are enqueued back into the stock queue in reverse order.
- The GUI updates immediately after each action.

5.3.3 3. Moving Cards Between Queues

Waste to Tableau / Foundation:

- The rear card of the waste queue is selected.
- Moves are validated using Klondike rules:
    - Tableau: opposite color and one rank lower, or King if empty.
    - Foundation: same suit and one rank higher, or Ace if empty.
- If valid, the card is dequeued from waste and enqueued into the destination queue.

Tableau to Tableau:

- A sequence of face-up cards is identified by traversing the queue.
- The sequence is validated for descending rank and alternating colors.
- Cards are moved one by one using dequeue and enqueue operations.

Tableau to Foundation:

- Only the rear card of a tableau queue can be moved.
- The card must match the foundation suit and be one rank higher.

5.3.4 4. Flipping Tableau Cards

- After removing cards from a tableau queue, the new rear card is checked.
- If it is face-down, its face-up status is updated.
- The GUI displays the flip visually.

5.3.5 5. Checking Win Condition

- The game is won when all four foundation queues contain exactly 13 cards.
- This check is performed after each successful move to the foundation.
- The GUI displays a winning message when the condition is met.

5.3.6 6. User Interaction and GUI Updates

1. Mouse clicks are used to draw cards and select piles.
2. Drag-and-drop allows moving cards between piles.
3. Double-click enables automatic movement to foundation when possible.
4. The GUI continuously synchronizes with queue data to render cards, animations, and messages.

5.3.7 Game Setup and Initialization

5.3.8 Deck Creation and Shuffling

1. Create a standard deck of 52 cards.
    - Four suits: Hearts, Diamonds, Spades, Clubs
    - Thirteen ranks per suit: Ace to King

    - All cards are initially face-down

2. Shuffle the deck:
    - Convert the queue into an array
    - Apply the Fisher–Yates shuffle algorithm
    - Repeat the shuffle process three times for better randomness
    - Convert the shuffled array back into a queue

5.3.9 Dealing Cards

1. Deal cards into seven tableau columns:
    - Column 1 gets 1 card, column 2 gets 2 cards, up to column 7
    - Only the top card of each column is face-up
    - A total of 28 cards are dealt
2. Place the remaining 24 cards into the stock pile, face-down
3. Initialize game state:
    - Set score and move counter to zero
    - Clear all four foundation piles
    - Clear the waste pile
    - Create an initial snapshot for undo functionality
    - Reset recycle counter for Hard mode

5.4 Main Game Loop

The game runs continuously and consists of three phases:

5.4.1 Update Phase

- Update card animations and particle effects
- Detect mouse hover areas
- Update timers for double-clicks and visual effects

5.4.2 Input Phase

- Handle keyboard input:
  - H: Help screen
  - U: Undo move
  - S: Statistics
  - N: New game
  - ESC: Close overlays
- Handle mouse clicks, drags, and releases

5.4.3 Render Phase

- Draw background and UI elements
- Display score, moves, difficulty, and completion percentage
- Render stock, waste, foundations, and tableau piles
- Show animations, highlights, and overlays

5.5 Card Movement Rules

5.5.1 Tableau Movement

1. Identify source and destination columns
2. Validate move:
   - Empty column accepts only a King
   - Cards must alternate colors
   - Rank must be exactly one lower
3. Allow moving multiple face-up cards together
4. Flip the new top card if it was face-down
5. Award 5 points for flipping a card

5.5.2 Foundation Movement

- Each foundation is suit-specific
- Cards must be placed from Ace to King in order
- Only an Ace can start an empty foundation
- Award 10 points for valid moves

5.5.3 Auto-Move to Foundation

- Double-click attempts automatic move to foundation
- Works for both waste and tableau cards

5.6 Drawing from Stock and Recycling Waste

5.6.1 Drawing Cards

- Easy mode: draw 1 card
- Medium/Hard mode: draw up to 3 cards
- Only the top waste card is playable

5.6.2 Recycling Waste

- Triggered when stock is empty
- Cards are flipped face-down and reversed
- Easy/Medium: unlimited recycles
- Hard: maximum 3 recycles with penalty
- Deduct 100 points per recycle

5.7 Undo System

- Save game state before every valid move
- Store up to 50 snapshots in a queue
- Undo restores the most recent snapshot
- No penalty for using undo

5.8 Animations and Particle Effects

- Card movement animations last 0.3 seconds
- Smooth easing and slight arc movement
- Particle effects trigger on foundation moves
- Win celebration uses continuous particle effects

5.9 Winning and Scoring

5.9.1 Win Condition

- All four foundation piles contain 13 cards
- All cards are arranged from Ace to King

5.9.2 Scoring System

- +10 points for moving to foundation
- +5 points for flipping tableau cards
- -100 points for recycling waste

5.10 Difficulty Levels

- Easy: draw 1 card, unlimited recycles
- Medium: draw 3 cards, unlimited recycles
- Hard: draw 3 cards, maximum 3 recycles

5.11 Queue-Based Data Structure Design

- All piles are implemented using circular queues
- Supported operations:
  - enqueue()
  - dequeue()
  - peekRear()
  - valueAt(index) as to cover limitation
  - size()

5.12 Card Representation

Each card stores:

- Rank (1–13)
- Suit (H, D, S, C)
- Face-up status

6 Time Complexity Analysis

6.1 Stack Based Klondike Game Time Complexity

Operation	**Worst Case Time Complexity**	**Explanation**
Push / Pop / Top	O(1)	Standard stack operations are constant time.
Deal Cards to Tableau	O(1) per card $\rightarrow$ O(28) total	Each card is dealt one by one to the 7 tableau piles. Since there are 28 cards in total, it takes constant time per card, adding up to O(28) overall.
Draw from Stock	O(1)	Pop from stock and push to waste takes O(1).
Recycle Waste to Stock	O(n)	Move all cards from the waste pile back to the stock. Since there are at most 24 cards remaining after dealing 28 to the tableau, $n \leq 24$.
Move Card to Tableau/Foundation	O(1) per move	Only the top card of the source stack is checked and moved. Comparing the top cards and performing a push or pop operation takes constant time.
Move Sequence of Cards	O(f)	f = number of face-up cards in the sequence. Each card in the sequence is checked and moved one by one, so the time depends on how many cards are being moved.
Flip Tableau Top	O(1)	Only the top card of a tableau pile is flipped face-up when needed. This takes constant time because we only change the status of one card, no matter how many cards are in the pile.
Check Win Condition	O(1)	Verify if each of the four foundation stacks has all 13 cards. This involves checking the size or top index of each stack, which is a fixed number of operations and does not depend on the total number of cards.

6.2 Linked List Based Klondike Game Time Complexity

Major Function	**Worst Case Time Complexity**	**Explanation**
createdeck()	O(n)	All 52 cards are generated and inserted into the stock linked list. Each insertion is constant time.
shuffledeck()	O(n)	The entire stock pile is traversed to move cards into an array, shuffled, and then reinserted. Each card is processed a constant number of times.
dealtableau()	O(1)	A fixed total of 28 cards are dealt to tableau piles. Each card is removed from stock and inserted into a tableau in constant time.
drawtowastee()	O(1)	A single card is removed from the stock and inserted into the waste pile. Both operations involve only head pointer updates.
recyclewastetostock()	O(n)	All cards in the waste pile are transferred back to the stock pile. Each card is removed and reinserted once, giving linear complexity.
move_tableau_to_foundation()	O(1)	Only the top card of the tableau pile is examined and moved if the rules allow. No traversal is required.
move_waste_to_foundation()	O(1)	The function checks and moves only the top card of the waste pile, making it a constant-time operation.

Major Function	**Worst Case Time Complexity**	**Explanation**
hasLegalMoves()	O(n)	All tableau piles are scanned, and each face-up card sequence is checked for valid moves. In the worst case, all cards may be face-up.
provideHint()	O(n)	The hint system searches through waste and tableau piles to identify all legal moves, requiring traversal of card structures.
saveGameState() / loadGameState()	O(n)	Each pile is serialized or reconstructed by traversing its linked list. Since all cards may be processed, the operation is linear.
performUndo()	O(n)	Undo restores a previously saved state by copying all card piles. Each linked list copy involves full traversal.

6.3 Queue Based Klondike Game Time Complexity

Operation	**Worst Case Time Complexity**	**Explanation**
`shuffle_cards()`	O(n)	The shuffle function performs multiple passes over the queue. In each pass, cards are dequeued, distributed into temporary queues, and then re-enqueued. Since all cards are processed, the overall time complexity is linear.

Operation	**Worst Case Time Complexity**	**Explanation**
`tab_to_tab(int fcol, int tocol)`	O(n)	The source tableau column is traversed to locate movable face-up cards. The destination column is also checked for a valid move. After moving the cards, the new top card of the source column is flipped if needed.
`tab_to_foundation(int n)`	O(n)	The tableau column is traversed to access the top card and the foundation pile is checked for a valid move. The last card is removed and the next card is flipped, which requires full traversal.
`waste_to_tab(int n)`	O(n)	The waste pile is traversed to obtain the top card. The destination tableau column is then checked to validate the move, which requires traversal due to queue-based access.
`waste_to_foundation()`	O(n)	Both the waste pile and the foundation pile are traversed to compare their top cards before performing a valid move.
`recycle_waste_to_stock()`	O(n)	All cards present in the waste pile are moved back to the stock pile. Each card is dequeued and enqueued once.
`check_win()`	O(1)	Checks whether all four foundation piles contain 13 cards each. Since the number of foundations is fixed, this operation takes constant time.
`empty_placement()`	O(1)	Checks only the rank of a card to determine whether it can be placed in an empty tableau or foundation pile. No traversal is required.

Operation	Worst Case Time Complexity	Explanation
`count_fdcard() / count_fucard()`	O(n)	Traverses the entire column queue to count face-down or face-up cards. Cards are temporarily dequeued and then re-enqueued.
`get_top_card(int& faceup_count)`	O(n)	The column is fully traversed to reach the last card while counting face-up cards. All cards are restored back into the queue afterward.
`move_faceup_cards_to()`	O(n)	The source column is traversed to separate face-down and face-up cards. The face-up cards are then moved to the destination column.
`remove_lastcard()`	O(n)	Since queues do not support direct access to the last element, the entire column must be traversed to remove the last card and rebuild the queue.
`flip_topp()`	O(n)	The full column is traversed to locate the last card, flip its face-up status, and then re-enqueue all cards back.
`display_column(int max_height)`	O(n)	All cards are dequeued for display and then re-enqueued back into the column, resulting in linear time complexity.

7 Comparison and Justification of Best Approach

In this project Klondike game was implemented using three different data structures. Stack, Linked List and Queue. Each approach was evaluated based on time complexity, ease of implementation and suitability with the game.

Stack Based Approach

The stack based implementation is closely matches the real world behavior of card piles. Operations such as drawing from stock, moving to waste and placing card on foundations naturally follow the Last-In, First-Out (LIFO) principle.

- Operations such as push, pop , top and a single card moves execute in O(1) time.
- Moving a sequence of face up cards requires an additional temporary stack to preserve the order. however, the overall time complexity remains O(f), where f is the number of face up cards.
- The logic is simple, efficient and easy.

Limitation

Stacks are naturally suited for Klondike because they follow LIFO behavior, matching physical card piles. However, there are some limitations: Moving a sequence of face-up cards from one tableau column to another requires an extra step. Since popping cards from a stack reverses their order, the sequence must be copied into an intermediate stack to restore the original order before placing it in the destination column. Operations like moving multiple cards, checking sequences, or accessing deeper cards are not direct and need temporary storage, which slightly complicates the logic. While single-card operations are efficient (O(1)), handling sequences or implementing undo/hints requires careful management of multiple stacks.

Linked List Based Approach

The Linked list approach provides the high flexibility mainly for moving sequence from one tableau to another tableau.

- Linked list allow moving whole sequences easily by changing pointers.
- Linked list is efficient for sequence moves (O(f)).
- Linked list gives more flexibility and control.

Limitation

The primary challenge is memory management and the risk of memory leaks if nodes are not deleted properly. Additionally, accessing a specific index requires O(n) traversal as there is no random access.

Queue Based Approach

The Queue based implementation was the least suitable for Klondike game.

- Since queues follow a First-In, First-Out (FIFO) so accessing the top card requires full rotation.
- Many basic operations such as moving cards, flipping top cards and validating moves require O(n) time.
- This increases execution cost for frequent game actions.

Limitation

Game using a queue faces a limitation due to fundamental mismatch between queue and games needs a LIFO requirements. This choice made performance inefficiencies and every operation needs a traversal of complete queue to access last element in tableau column, waste piles and foundation piles. It transforming code $O(1)$ stack operation into $O(n)$ queue. Game code become complex with methods like `get_top_card` and `move_faceup_cards_to()` that constantly reconstructing entire queue through by temporary storages. This approach impact performance as well as game natural card movement into enforced repetitive dequeue and requeue patterns. This implementation make a **violation** in term of easy we can create logic data in stack such as tableau columns, waste piles and foundation but forcing queues to work as stack. This implementation of queue sacrifices efficiency and code clarity.

Final Justification

Based on performance analysis and practical implementation:

- Stacks provide the lowest overall time complexity and are the most efficient for core gameplay.

- Linked Lists offer superior flexibility, particularly for tableau sequence movements.
- Queues are functionally correct but inefficient for this problem.

Therefore, the Stack-based approach is the best overall choice for implementing Klondike Solitaire due to its efficiency, simplicity, and natural alignment with card game mechanics. The Linked List approach is a good alternative where flexibility is prioritized, while the Queue-based approach is not to be suggested.

8 Individual Contributions

- **Team Collaboration:** Determined the game development schedule, designed the Graphical User Interface (GUI), defined the initial game structure, and managed the early-stage implementation.
- **Group Leader:** Responsible for overall coordination, report structuring, cross-implementation comparison, and final technical review.
- **Muhammad Shoaib &Naveed Abbasi:** Developed the Stack-based Klondike implementation and logic.
- **M Jamal Ahmad Khan & Muhammad Wasif:** Developed the Linked List-based Klondike implementation and node management.
- **Bilal Ahmed Khan & Mubashir Hassan:** Developed the Queue-based Klondike implementation and FIFO logic adaptation.

9 Game Screenshots

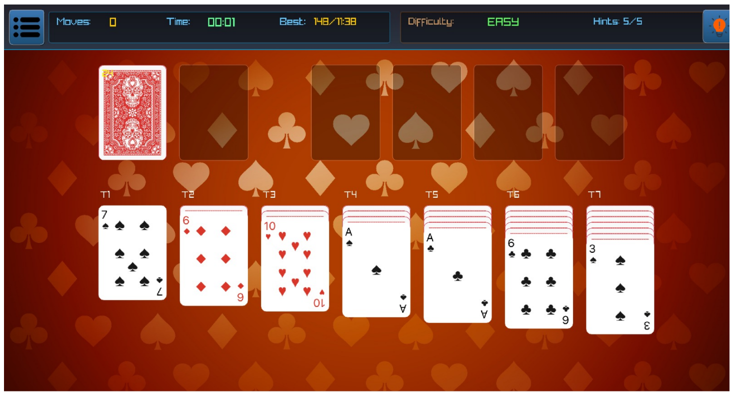

Figure 1: Stack Based Klondike Game GUI

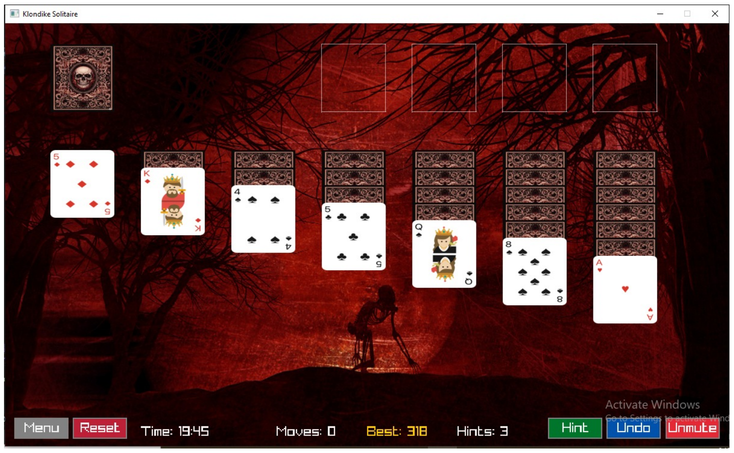

Figure 2: Linked List based Klondike Game GUI

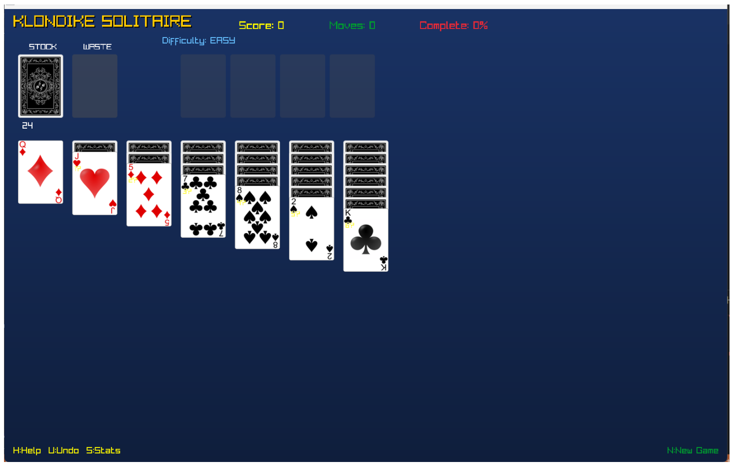

Figure 3: Queue based Klondike Game GUI

10 Conclusion

The successful implementation of the Klondike Solitaire game using three distinct data structures-Stack, Linked List, and Queue provides valuable insights into their practical applications.

Through this project, we observed that while all three data structures can be used in the game but their efficiency varies significantly:

- **Stacks** are the most relavent for card games because they naturally follow the Last-In, First-Out (LIFO) behavior of physical card piles.
- **Linked Lists** offered the highest flexibility, particularly for the Tableau, where moving entire sequences of cards is a core mechanic.
- **Queues**, while functional, were the least efficient for this specific game because accessing the ”top” card required traversing or rotating the entire structure, leading to higher time complexity for simple moves.

In conclusion, this project effectively demonstrated that the choice of data structure is not just about making a program work, but about optimizing performance and simplifying the logic. By integrating these structures with the Raylib GUI, we successfully bridged the gap between theoretical computer science concepts and real-world software development.

11 References

References

[1] Cherry, W. (1990). *Microsoft Solitaire History*. Microsoft Corporation.

[2] Yan, X., et al. (2004). *Solitaire: Man vs. Machine*. University of Alberta.

[3] Longpré, L., & McKenzie, P. (2009). *The Complexity of Solitaire*. Journal of Computer and System Sciences.