Solitaire Klondike

Session 2024 - 2028

Submitted by:

Mueeza Akbar 2024-CS-34

Supervised by:

Professor Nazeef-ul-Haq

Course:

Data Structures and Algorithms

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Contents:

Table of Figures:

1. Solitaire Klondike

Solitaire Card Game is a modern, feature-rich implementation of the classic Klondike Solitaire, built entirely with vanilla JavaScript, HTML5, and CSS3. This project showcases advanced web development techniques while maintaining the timeless appeal of the world's most popular card game

Contribution:

This Solitaire game is designed to provide engaging and seamless card game experience for players of all skill levels. Built entirely with vanilla JavaScript, it demonstrates how classic games can be revitalized through modern web technologies without relying on external frameworks or libraries. The game automatically handles complex card game mechanics, including move validation, scoring, and win condition detection, allowing players to focus purely on strategy and enjoyment.

The system intelligently manages all aspects of gameplay - from drag-and-drop card movements and sequence validation to automatic scoring and progress tracking. Players can simply interact with the cards through intuitive clicks and drags, while the game handles the underlying rules, calculations, and visual feedback. This approach makes sophisticated card game mechanics accessible to casual players while maintaining depth for experienced enthusiasts.

Through its custom-built data structures and efficient algorithms, the game showcases how complex game logic can be implemented with clean, maintainable code.

Key Features:

• Custom Data Structures:
The game employs custom-built Stack, queue and Linked List implementations to efficiently manage card movements, tableau organization, and undo operations, ensuring optimal performance and data integrity throughout gameplay.

• Smart Drag & Drop:
Cards can be intuitively dragged and dropped with smooth animations and visual highlights, allowing players to easily identify valid moves and maintain an engaging, natural user experience.

• Three-Card Draw:
Implements the traditional Solitaire mechanic of drawing three cards from the stockpile, recycling them when exhausted, and reshuffling as per standard game regulations and any of the cards can be drawn if it is fixed in tableau or in foundation

position. The game user can drag and drop the card from the waste pile, or the user just clicks the card and it will automatically go to the required position.

• Undo/Redo System:
A robust move history feature records every action, allowing players to undo or redo moves without limitation, ensuring flexible and error-tolerant gameplay.

• Win Condition Detection:
The game continuously checks for victory conditions and displays an animated congratulatory message once all cards are successfully moved to the foundation piles.

• Comprehensive Scoring:
A dynamic scoring mechanism rewards efficient moves and promotes strategic play and skill improvement.

• Keyboard Shortcuts:
Players can perform frequent actions such as drawing cards, undoing moves, or auto-moving sequences using keyboard shortcuts, improving gameplay speed and accessibility.

• Responsive Layout:
The user interface adapts seamlessly to different screen sizes and orientations, ensuring consistent gameplay on both desktop and tablet devices.

• Instruction Menu:
When the game starts the instruction menu appears in which there is a complete guide for the user of the game how he can play with keyboard or mouse.

• Stock Recycling Limits:
Implements configurable limits for stock recycling cycles, tracking the number of resets and displaying out of moves message when the limit is reached.

• Out-of-Moves Detection:
The game intelligently detects stalemate situations where no valid moves remain and notifies the player with a helpful suggestion message.

• Cross-Browser Compatibility:
Developed using pure JavaScript, HTML5, and CSS3, ensuring consistent behavior and performance across all major modern browsers.

• No External Dependencies:
The entire project is implemented without using third-party frameworks or libraries, ensuring lightweight performance and complete code transparency.

• Real-Time Statistics:
Displays detailed metrics such as elapsed time, number of moves, and overall performance score, updating continuously as the game progresses.

2. Data Structure Implementation

• Stack:

The Stack data structure is used in multiple parts of the game where a Last-In-First- Out (LIFO) behavior is needed to simulate the way cards are drawn, discarded, or temporarily held.

❖ Stock Pile:

Code Line**:** this.stock = new Stack();
The Stack is used to represent the stock pile since it allows drawing cards from the top, maintaining the correct order of remaining cards. This ensures smooth gameplay and efficient card access without disrupting the underlying sequence.

❖ Waste Pile:

Code Line: this.waste = new Stack();
The Stack structure is used to store cards drawn from the stock pile. The LIFO behavior guarantees that the most recently drawn card appears on top, allowing easy movement to tableau or foundation piles.

❖ Foundation Piles:

Code Lines:
```
this.foundations = {
   'hearts': new Stack(),
   'diamonds': new Stack(),
   'clubs': new Stack(),
   'spades': new Stack()
};
```
Each foundation pile is modeled as a Stack to build cards sequentially from Ace to King. Only the top card is accessible for validation and addition, which enforces Solitaire’s building rules.

❖ Move History (Undo/Redo):

Code Lines:

```
this.undoStack = new Stack();
this.redoStack = new Stack();
```

Stacks are used for managing the move history, providing efficient undo and redo operations. The most recent move remains on top, enabling players to reverse or reapply actions in the correct order.

- **Temporary Stack Operations:**

  Code Lines:

  ```
  const tempStack = new Stack();
  ```

  Temporary stacks are used for holding intermediate cards during transfer operations. They ensure card order and prevent data loss when cards are moved between piles.

• Linked List:

The **Linked List** data structure is used where flexible and dynamic handling of cards is required, particularly in the **tableau columns**.

❖ Tableau Columns:

Code Lines:

```
this.tableau = [
   new LinkedList(), new LinkedList(), new LinkedList(),
   new LinkedList(), new LinkedList(), new LinkedList(),
   new LinkedList()
];
```

Linked Lists manage cards in each tableau column dynamically. They support the addition and removal of cards at any position, enabling the game to efficiently handle sequences of variable lengths.

- **Linked List Operations:**

  Code Lines:

  ```
  this.tableau[col].append(card);
  this.tableau[col].getLast();
  this.tableau[col].removeLast();
  this.tableau[fromCol] = new LinkedList();
  ```

  The methods **append**, **getLast**, and **removeLast** are used to handle card movement and sequence manipulation efficiently. They ensure that the tableau structure remains consistent during gameplay.

• Array:

Arrays are used to handle fixed collections and enable fast, index-based operations in the game logic.

❖ **Game Configuration:**

Code Lines:
this.SUITS = ['hearts','diamonds','clubs','spades'];
this.RANKS = ['A','2','3','4','5','6','7','8','9','10','J','Q','K'];
Arrays are used for storing predefined suits and ranks, providing fast lookup and easy iteration during deck initialization.

❖ **Deck Shuffling:**

Code Line: const tempArray = [];
Arrays enable random access and element swapping for **deck shuffling**. This helps implement the **Fisher–Yates algorithm** to ensure fair randomness

❖ **Card Sequence Validation:**

Code Lines:
const movingCards = sourceArray.slice(cardIndex);
const sequence = columnArray.slice(cardIndex);
Arrays allow efficient slicing of sequences, which is essential when checking whether multiple cards can move together in tableau columns.

❖ **Waste Card Management:**

Code Line: const wasteArray = this.objectToArray(visibleWasteCards);
Arrays make it easier to iterate through waste cards and manage their visibility for display and movement validation

❖ **Game State Serialization:**

Code Lines:
const result = [];
const tableauState = [];
Arrays are used to collect and store serialized game data. This allows quick **saving and loading** of game states for resuming gameplay.

• Queue:

Queue works on the **FIFO** property. The **First-In-First-Out (FIFO)** property makes it suitable for managing processes that need to maintain their order of entry.

- **Add Recent Move:**

```
addRecentMove(moveDescription) {
this.recentMovesQueue.enqueue({
    timestamp: new Date().toLocaleTimeString(),
    description: moveDescription
});
```

The **enqueue** operation is used to add new moves to the recent moves queue because it efficiently appends items to the end of the queue while maintaining chronological order.

- **Removing Old Moves:**

```
    this.recentMovesQueue.enqueue(move);
if (this.recentMovesQueue.size() > 5) {
    this.recentMovesQueue.dequeue();
}
```

The **dequeue** operation is employed to automatically remove the oldest moves when the queue exceeds its five-item limit. This creates an efficient "sliding window" that maintains only the most relevant recent actions without requiring manual array management or complex indexing logic

• Object:

Objects are used to store and manage key-value data relationships that improve code readability and retrieval efficiency

- **Rank Values Mapping:**

Code Lines:

```
this.RANK_VALUES = {
    'A': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
    '8': 8, '9': 9, '10': 10, 'J': 11, 'Q': 12, 'K': 13
};
```

Objects are used to map each card rank to its corresponding numeric value. This allows instant access during **scoring, validation, and comparison** operations.

- **Drag Source Tracking:**

Code Lines:

```
this.dragSource = {
location: location,
col: colIndex,
index: visibleIndex,
sequence: location === 'tableau' && colIndex !== null?
this.getDraggableSequence(colIndex, visibleIndex) : null
};
```

Objects are used to track drag operation details. This bundles all drag-related information together for easy access during drop handling and sequence validation.

- **Foundations Organization:**

```
this.foundations = {
   'hearts': new Stack(),
   'diamonds': new Stack(),
   'clubs': new Stack(),
   'spades': new Stack()
};
```

Objects are used to map each suit to its corresponding foundation pile. This provides semantic access using suit names instead of numerical indices, improving code readability.

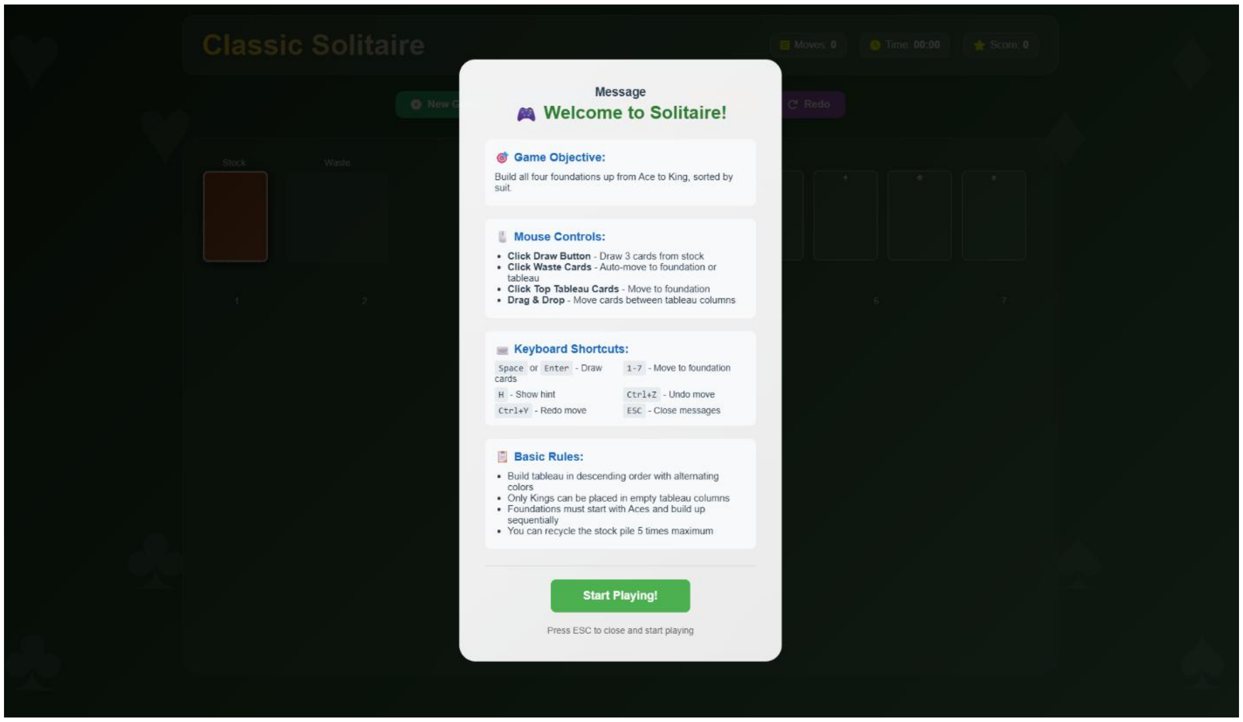
3. Game Logic

The game logic of **Solitaire Klondike** is designed to ensure smooth, responsive, and rule-based gameplay by integrating the functionalities of custom data structures and well-structured event-driven algorithms. The core logic continuously validates every move to ensure it complies with Solitaire's rules—such as placing alternating colors in descending order within tableau columns and building foundations from Ace to King. Each user interaction, including card movement, drawing from the stockpile, or transferring cards between piles, is handled through efficient algorithms that maintain the game state and update the score, move count, and timer in real time.

The logic also supports a comprehensive **Undo/Redo** mechanism that allows players to reverse or reapply previous actions using **Stacks**, ensuring flexibility and a forgiving gameplay experience. Additionally, the system includes intelligent **move detection** and **automatic placement**, allowing valid cards to move directly to the foundation piles with a single click or keypress.
To enhance accessibility and user control, several **keyboard shortcuts** have been implemented. Players can draw a card from the stockpile using the **Spacebar** or **Enter** key, while pressing **H** provides an automatic **hint** for the next valid move. The **Ctrl + Z** (or **Cmd + Z** on macOS) shortcut performs an **Undo** action, whereas **Ctrl + Y** or **Ctrl + Shift + Z** executes a **Redo** operation. Pressing the **Escape** key closes any open hint or victory messages, maintaining a clean interface. Moreover, the number keys **1 to 7** correspond to tableau columns, allowing players to automatically move the top card from a chosen column to its appropriate foundation pile.
Overall, game logic ensures a balanced integration of intelligent automation and player control, creating an efficient, responsive, and user-friendly Solitaire experience that closely mirrors the mechanics of traditional gameplay while leveraging modern web technologies for enhanced interaction.

4. User Interface Design

The **User Interface (UI)** of the Solitaire Klondike game is designed to provide a clean, intuitive, and responsive gameplay environment. The layout follows the traditional Solitaire setup, with clear separation between the **Stock**, **Waste**, **Tableau**, and **Foundation** areas. Cards are visually rendered with realistic suits, ranks, and smooth animations to provide an engaging, authentic playing experience.
The UI employs **HTML5** for structure, **CSS3** for styling and animations, and **JavaScript** for interactivity. The design adapts seamlessly to different screen sizes, ensuring compatibility across both desktop and tablet devices. Elements such as buttons, messages, and card movements use consistent color schemes and transitions to maintain visual balance and readability.
A central **instruction menu** is displayed at the beginning of the game, guiding new players on how to play using either a mouse or keyboard shortcuts. Additionally, interactive feedback such as card highlighting, move validation effects, and dynamic score updates help players track their progress in real time. All the instructions are written on the instruction menu. The user can also play the with the help of the keyboard. The shortcut keys are displayed in the instruction menu so that the user can play game with the help of the keyboard.



Message

Welcome to Solitaire!

Game Objective:

Build all four foundations up from Ace to King, sorted by suit.

Mouse Controls:

- **Click Draw Button** - Draw 3 cards from stock
- **Click Waste Cards** - Auto-move to foundation or tableau
- **Click Top Tableau Cards** - Move to foundation
- **Drag & Drop** - Move cards between tableau columns

Keyboard Shortcuts:

`Space` or `Enter` - Draw cards
`1-7` - Move to foundation
`H` - Show hint
`Ctrl+Z` - Undo move
`Ctrl+Y` - Redo move
`ESC` - Close messages

Basic Rules:

- Build tableau in descending order with alternating colors
- Only Kings can be placed in empty tableau columns
- Foundations must start with Aces and build up sequentially
- You can recycle the stock pile 5 times maximum

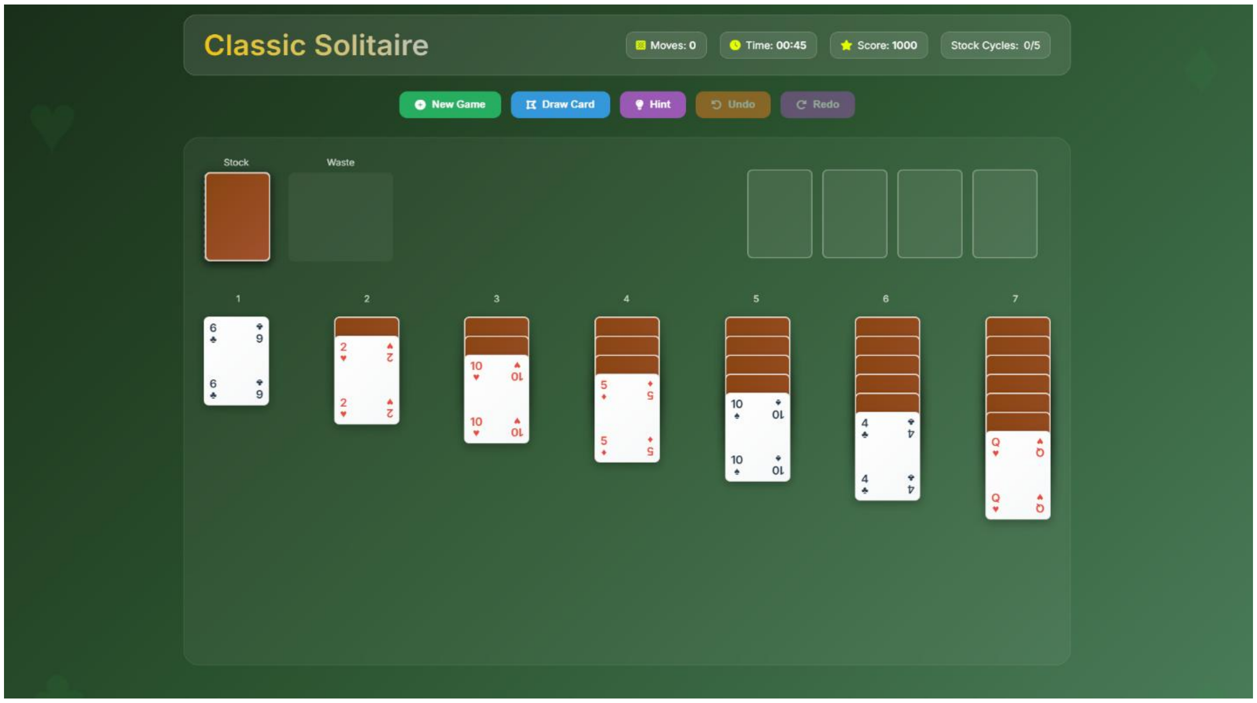
Start Playing!

Press ESC to close and start playing

Instruction Menu

The interface also includes a **scoreboard**, **move counter**, and **timer** at the top of the screen for live tracking of player performance. Visual messages such as "No More Moves" or "Congratulations!" are displayed using modal overlays that blend smoothly with the background, maintaining immersion without disrupting gameplay.
Overall, the user interface design combines **aesthetic simplicity**, **functional clarity**, and **responsive behavior**, ensuring that players experience an enjoyable and accessible Solitaire session across all modern browsers.

Furthermore, the UI provides clear visual feedback for valid and invalid moves, ensuring rule transparency throughout gameplay. Smooth transition effects enhance user engagement, while color contrast and readable typography improve accessibility. The interface is designed to minimize distractions and maintain focus on gameplay. Additionally, responsive layout adjustments automatically reposition elements, allowing the game to function effectively on different screen orientations and resolutions. The interface of the game is user friendly. Every time the user has to click the draw card button to draw the card. Glimpse of the UI design of the Solitaire Design is given below



User Interface of the game

This is the user interface of solitaire Klondike game which is responsive for all the types of the desktops and even for the tablets.

5. Conclusion

The **Solitaire Klondike Game** successfully combines the principles of **Data Structures and Algorithms** with modern web development practices to recreate the classic Solitaire experience. By utilizing custom implementations of **Stack**, **Queue**, and **Linked List**, the project demonstrates how abstract data structures can be effectively applied in real-world scenarios. The efficient handling of game state, move validation, and scoring logic reflects a strong understanding of algorithmic thinking and structured programming.
The game's **user interface** provides a visually appealing and responsive experience, ensuring smooth performance across different devices and browsers. Additionally, features such as **drag-and-drop mechanics**, **keyboard shortcuts**, **undo/redo operations**, and **real-time statistics** make gameplay both interactive and engaging.
Overall, the project achieves its objective of demonstrating the integration of theoretical computer science concepts into an enjoyable, functional, and user-friendly application. It not only strengthens programming skills but also highlights the importance of optimized logic, modular design, and clean code in software development.