Solitaire Game

Session 2023 – 2027

Submitted by:

Anza Tamveel 2023-CS-87

Supervised by:

Prof. Nazeef ul Haq

Course:

CSC200 – Data Structure & Algorithms

Department of Computer Science

University of Engineering and Technology
Lahore Pakistan

Table of Contents:

Solitaire Game - Klondike Version:

This is a feature-rich Klondike Solitaire game project developed to demonstrate a comprehensive understanding of Data Structures and Algorithms (DSA) principles within an interactive, user-friendly application. The game is designed as part of a course project, integrating various data structures like linked lists, stacks, queues, arrays, and dictionaries.

Game Description:

The game provides a classic Solitaire experience with intuitive drag-and-drop functionality for moving cards. Players can engage with the traditional Solitaire layout, including tableau piles, a stockpile, and waste pile, following Klondike rules. A built-in timer tracks the game duration, and a scoring system updates as moves are made, enhancing the competitive aspect of gameplay. Additional features like undo and redo functions allow users to correct moves as needed.

Key Features:

- Drag-and-Drop Gameplay: Allows seamless movement of cards across the tableau, foundation, stockpile, and waste pile.
- Waste & Stockpile Mechanics: Players draw cards from the stockpile and can cycle through the waste pile, providing a realistic Solitaire experience.
- Undo Functionality: Players can undo moves to optimize strategies and correct mistakes.
- Win Condition: The game is won when all cards are sorted into the foundation piles according to the rules of Solitaire.
- Timer: Keeps track of the game duration, providing a sense of urgency and competitive engagement.
- Scoring System: Tracks points based on player moves, rewarding efficient gameplay and strategic moves.

Technical Highlights:

Data Structures:

❖ ***Stacks:***
- Used in the foundation piles and waste pile to follow Last-In-First-Out (LIFO) behavior, essential for managing card order in the game's logical structure.
- Also used for implementing undo functionality by storing previous game states.

❖ ***Queues:***
- Implemented in the stockpile to handle the draw mechanism where cards are processed in a First-In-First-Out (FIFO) order.

❖ ***Linked Lists:***
- Utilized in the tableau piles to allow dynamic addition and removal of cards at either end, facilitating smooth gameplay with efficient memory use.

❖ ***Dictionaries:***
- Used for card tracking to keep track of the card positions within different piles for easy lookups and game state management.

❖ ***Basic Algorithms:***
- Shuffling the deck, simulating the randomness of card order.
- Card movement validation using comparisons for rank and suit compatibility to maintain Solitaire game rules.

Class Breakdown:

❖ ***Card:***

- Represents an individual card with properties for suit, rank, and whether it is face-up or face-down.
- Has methods for flipping the card and loading card images for the GUI.

❖ ***Deck:***

- Creates a shuffled deck of 52 cards at the start of the game.

- Provides methods to draw cards and check if the deck is empty, which is essential for initializing the tableau piles and stockpile.

❖ ***Tableau:***

- Manages the seven tableau piles where cards are initially dealt.
- Each pile is a **LinkedList** that can be manipulated to add or move cards.
- Has methods to reveal the bottom face-down card when a card is moved and to transfer cards between tableau piles following Solitaire rules.

❖ ***Foundation:***

- Contains four foundation piles, each represented as a **Stack**, to build up each suit from Ace to King.
- Cards can only be moved here if they are in sequence and match the suit of the pile.

❖ ***WastePile:***

- Holds cards drawn from the stockpile, allowing one card to be accessible at a time.
- Allows cards to be moved to the tableau or foundation piles if a valid move exists.

❖ ***StockPile:***

- The reserve of remaining cards, where players draw cards to replenish the waste pile.
- Uses a **Queue** to ensure that the draw order is maintained and cards are transferred correctly between the stockpile and waste pile.

❖ ***Game:***

- Central controller for game logic and state management.
- Tracks moves for the undo functionality, manages all interactions between the piles, and checks for winning conditions.
- Handles the initialization of the tableau, shuffling of the deck, stockpile setup, and manages in-game hints.

Key methods include:

- **move_cards:** Validates and executes moves across tableau, foundation, and waste piles.
- **draw_from_stockpile:** Moves cards from the stockpile to the waste pile and handles recycling of waste pile cards.
- **find_hint:** Suggests possible moves if available.
- **check_win:** Checks the foundation piles to verify if the win condition (all 52 cards sorted by suit) is met.

Advantages of Using These Concepts:

The application of specific Data Structures and Algorithms (DSA) concepts enhances the Solitaire game's performance, memory management, and user experience. Here's a breakdown of the advantages, along with the time complexities of key operations:

Efficiency in Game Operations:

- **Stacks** and **Queues** allow **O(1)** time complexity for push and pop operations, making them ideal for handling card moves, drawing cards, and managing undo operations, as these actions are performed instantly regardless of the number of cards.

Dynamic Memory Management:

- **Linked Lists** enable **O(1)** time complexity for adding or removing cards at any position (beginning or end), which is crucial for managing the tableau piles without shifting elements. This ensures efficient memory usage and fast updates when adding or moving cards.

Fast Data Access and Retrieval:

- **Dictionaries** provide an average **O(1)** time complexity for access and update operations, making them highly efficient for tracking card locations, validating legal moves, and monitoring the game's progress. This minimizes the time needed to retrieve and verify information during each turn.

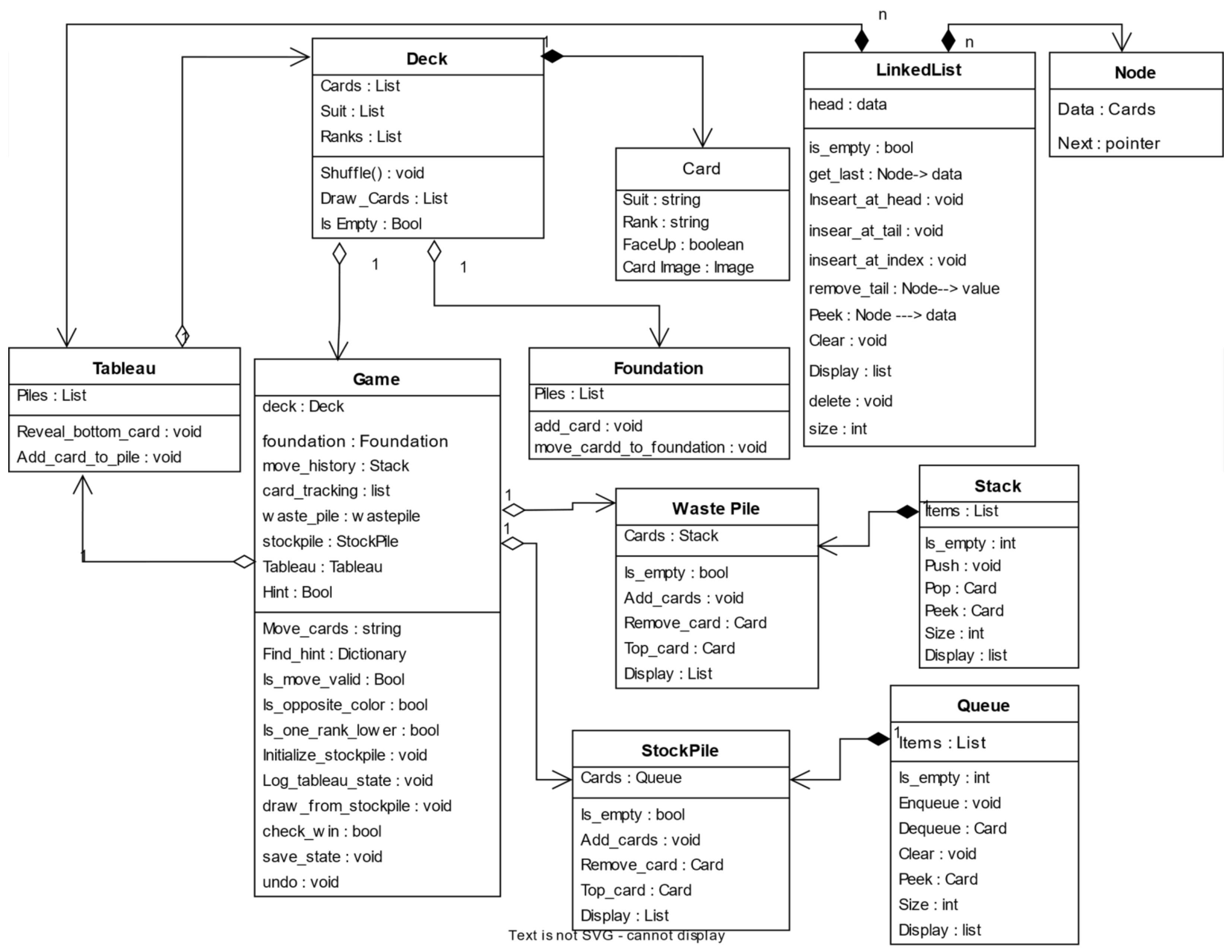
Improved User Experience:

- Optimized **shuffling algorithms** generally operate in **O(n)** time complexity, ensuring that card shuffling happens quickly, enhancing the speed and smoothness of gameplay. Additionally, constant-time operations (O(1)) in data structures ensure instant feedback to user actions, creating a seamless user experience.

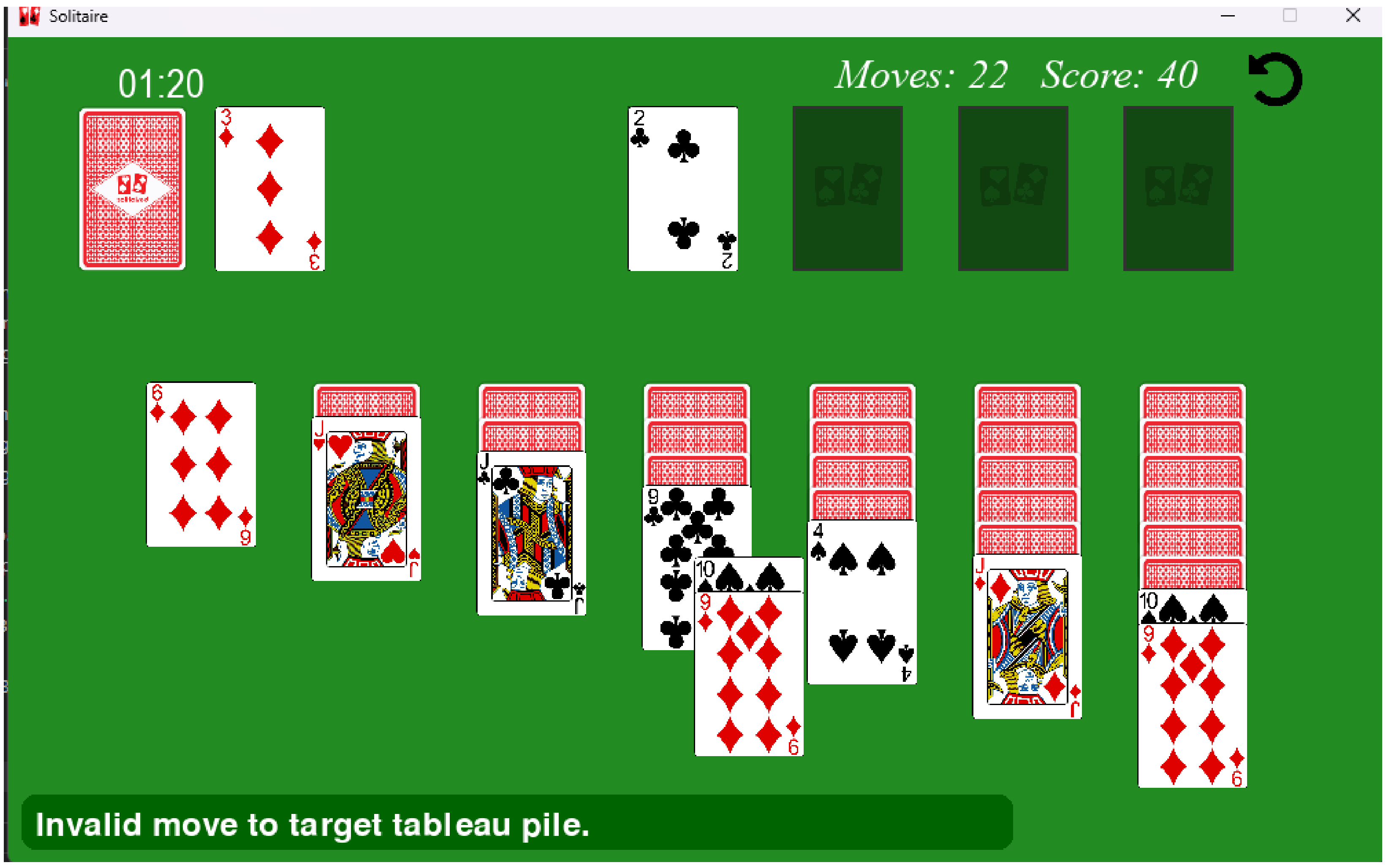
Scalability for Enhanced Features:

- The use of efficient DSA concepts supports the game's scalability, allowing future enhancements like multi-level undo. Efficient data structures help keep time complexities manageable, even as new features are added.

▪ Class Diagram:

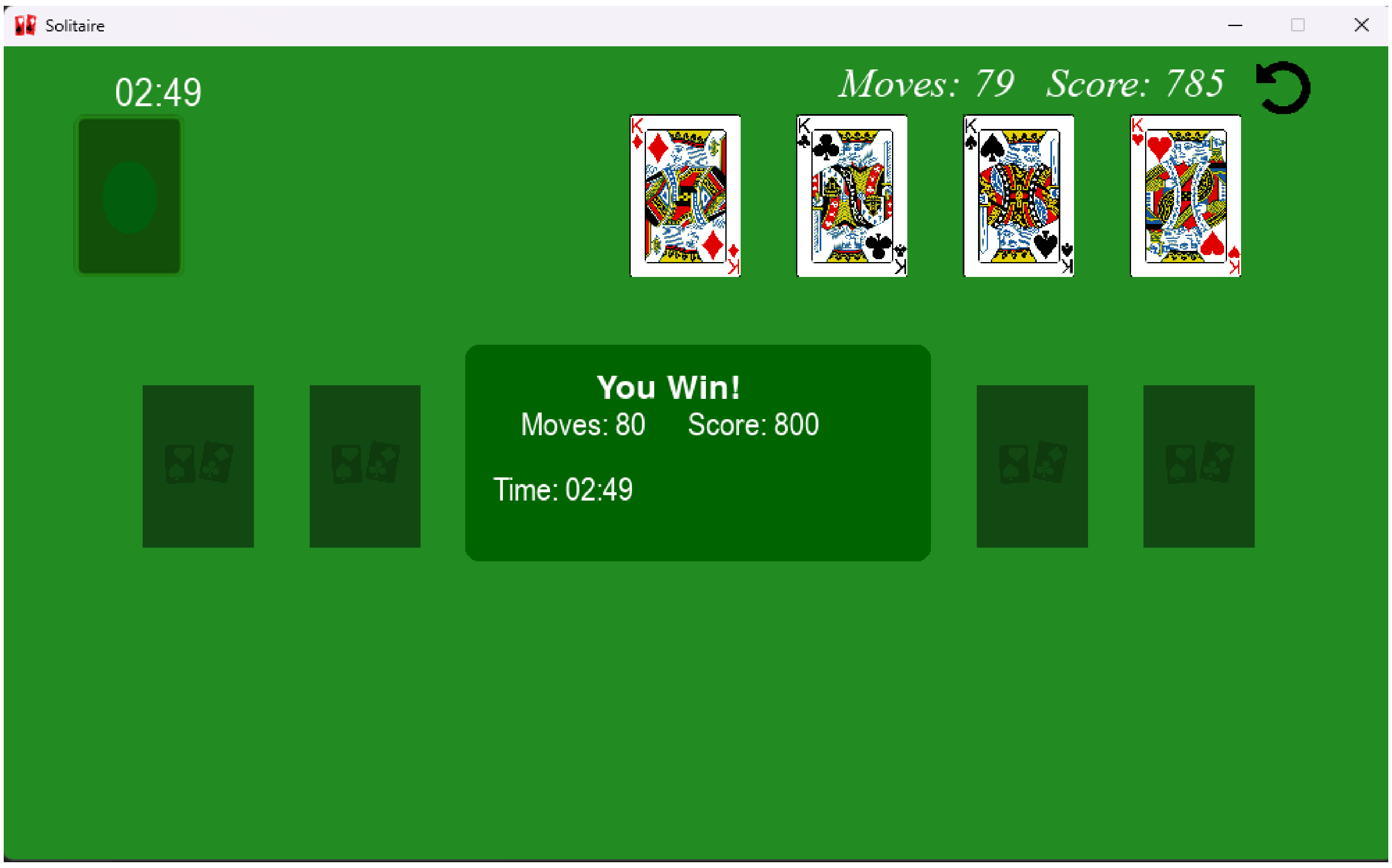


Wire Frames:



Invalid move to target tableau pile.

WireFrame-01



You Win!
Moves: 80 Score: 800
Time: 02:49

WireFrame-02