

**Session 2023-2027**

**Submitted by:**

Huzaifa Arshad 2023-CS-86

**Supervised By:**

Mr. Nazeef ul Haq

**Course:**

Data Structures and Algorithms

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Project Report**

* **Description:**

Solitaire, also known as Patience in some parts of the world, is a classic game played by a single player. The game objective is to sort and move cards into specific piles following a set of rules, ultimately achieving a completed and sorted deck.

* **Objective:**

The aim of this project is to develop a command-line version of a solitaire game using C++. The game involves building a standard solitaire having functionalities of moving cards from tableau pile to foundation pile and from stock pile to foundation pile and other functionalities.

* **Game Design and Rules:**

This solitaire game is based on the proper rules:

* A desk of 52 cards in dealt into 7 tableau piles with incremental cards per pile.
* The player must move cards to foundation piles starting from Ace up to King of each suit.
* The game ends when all cards go to foundation pile in correct order starting from Ace to King.
* **Classes and Modules:**
* **Card Class:**

The card class represents a single card in the deck of cards used in the game. It contains essential attributes such as the rank (e-g Ace, 2, King) and suit (e-g Diamond, Club, Heart, Spade) of the card. It also have boolian indicating whether the card is face-up or not.

* **Attributes:**

Card class have following attributes:

* **suit:** The suit of card (Hearts, Diamonds, Clubs, Spades)
* **rank:** The rank of card (Ace, 1,2,3,4,5,6,7,8,9,10,J,Q,K)
* **isFaceup:** Boolean indicating whether the card is face up or not.
* **Deck Class:**

The Deck class in the solitaire game project is responsible for managing a deck of 52 cards, including initialization, shuffling, and dealing cards to other components of the game. Its main purpose is to initialize and manage cards in the game.

* **Attributes:**

Following are the attributes of the deck class.

* **Card\* cards[52]:** An array of pointers to Card objects, each representing one of the unique 52 cards in a standard deck.
* **int currentCard:** An integer that tracks the index of the next card. It is initialized to 0, indicating the first card in the shuffled deck.
* **Functions:**

Following are the functions of the deck class.

* **Shuffle card:** It shuffles random 52 cards. It uses rand() function to give random cards.
* **DealCard:** It deals one card at a time from the deck, allowing other classes(like Game and stack) to access cards sequentially.
* **Stack Class:**

The stack is a custom data structure implemented using linked lists. It has following functions like push, pop, peek and isEmpty functions. This class is well suited for stack operations within the solitaire game.

* **Inner Node class:**

The node class is designed to be a internal class that represents each element in the stack. The node contains two things. First is the data or value and other thing is the reference or pointer to the next node.

* **Attributes:**

Following are the attributes of the stack class.

* **T data:** Holds the actual data of the node.
* **Node<T>\* next:** A pointer to the next node in the stack.
* **Functions:**

Following are the functions of the stack class.

* **isEmpty:** Check if the stack is empty. It returns true.
* **Push(T data):** Insert a new element at the top.
* **Pop:** It removes the element from the top.
* **Peek:** It returns the top element
* **Inner Iterator Class:**

It is the inner class within the stack class, allowing traversal through the stack.

* **Attributes:**

Following are the attributes.

**Node<T> \* current:** A pointer that keeps track of the current node.

* **Foundation Pile Class:**

It represents foundation pile in the solitaire game, where stack of same suits are stacked in same order starting from ace to king. This class handles operations such as adding cards, removing cards, displaying the top card, and validating if a card can be added to the foundation pile.

* **Attributes:**

Following are the attributes of the Foundation pile class.

**Stack<Card> pile:** A stack that holds Card objects in the foundation pile. Cards are added to the stack as long as they follow the required sequence like same suit and ascending order from ace to king.

* **Functions:**

Following are the functions of the Foundation pile class.

* **addCard:** This function adds cards in the foundation pile if it follows the rule like start from Ace and ends at the king.
* **removeCard:** Removes the top card from the foundation pile.
* **getTopCard:** It returns top card from the foundation pile without removing it.
* **Display:** Displays the top card of the foundation pile. If pile is empty, it returns XXX but if the pile is not empty it shows the top card on the foundation pile.
* **CanAddCard:** It determines that if a given card can be added in the foundation pile or not because according to rules only Ace card can start foundation pile and it ends at king which number is 13.
* **Stock Pile Class:**

This class represents a stock of cards that contains 24 random cards that is not present in the tableau piles that contains 28 cards in 7 columns. So total cards become 52. The stock pile is used as a reserve of cards that can be drawn and moved to tableau pile and foundation pile according to the requirement and rules.

* **Attributes:**

Following are the attributes of the Stock pile class.

**List<Card> pile:** A linked list containing Card objects representing the stock pile. It includes the insertion and removal operations as needed when cards are moved from the stock pile to other pile.

* **Constructor:**

**StockPile(Deck& deck):** It initializes the stock pile by drawing 24 cards from the provided deck object.

* **Functions:**

Following are the functions of the Stock pile class.

* **GetCardAt:** It retrieves the card from the stock pile at a specific index. If index is less than zero and greater than the size of pile it will give invalid card index. Otherwise it will get card.
* **RemoveCardAt:** Removes or returns a card from the stock pile at the specific index.
* **Display:** It displays all cards currently in the stock pile, each having its index.
* **TableauPile Class:**

This class tableau pile in the solitaire game, allowing players to manage cards in a stack format which means that insertion and removal from the top end. This class facilitates initialization of the pile from the deck, adding cards, removing the top card and displaying all cards in the pile. It allows only the last or top card to be faced up otherwise every card below them is hidden. It follows the game rules that only top card is faced up.

* **Attributes:**

Following are the attributes of the Tableau pile class.

* **stack<Card> pile:** A stack of card objects representing the tableau pile.

Constructor:

* **TableauPile(Deck& deck, int numcards):**Initializes the tableau pile with a specific cards from the Deck class. It also sets only the last card in the pile as face-up.
* **Functions:**

Following are the functions of the Tableau pile class.

* **Addcard:** Insert or add card at the top of the tableau pile.
* **removeCard:** It removes the top card from the tableau pile and returns it.
* **getTopcard:** It gives the top card of the pile.
* **Display:** It displays all card in the tableau pile like 2 of Hearts, 3 of Clubs, 4 of Diamonds etc.
* **Game Class:**

The game class is the central manager of this solitaire game. It uses the classes like foundation pile, stock pile, tableau pile. It provides functions to initialize the game, displaying piles, and moving cards between different piles according to the rules of the solitaire game.

* **Attributes:**

Following are the attributes of the Game class.

* **Deck deck:** A Deck object that provides cards.
* **TableauPile\* tableaupile[7]:** An array of seven piles pointers each representing a tableau pile in the game.
* **FoundationPile\* foundationPile[4]:** An array of four piles pointers representing a foundation pile.
* **StockPile\* stockpile:** A pointer to a StockPile object representing the stock of cards.
* **Constructor:**

**Game():** It initializes the game by setting up the tableau pile, foundation pile and stock pile.

* **Private Helper Functions:**

Following are the private functions of the Game class.

* **ismoveAllowed:** It determines if a move between tableau cards is valid or not. It means that the next card that has to be inserted must be one number smaller and different color because diamond and heart have same color and club and diamond have same color.
* **isSameColor:** It checks that the card that has to be inserted in tableau pile has same color with top card like diamond and heart have same color and club and spade has same color.
* **RankInInt:** It converts rank like J,Q,K and A to its respective numbers like 11,12,13 and 1. It simply converts rank to an integer for easy comparison.
* **Functions:**

Following are the functions of the Game class.

* **DisplayTableauPiles:** It displays each tableau pile and its cards in a format.
* **DisplayFoundationPiles:** It displays each foundation pile and its current cards in a format.
* **DisplayStockPile:** It displays each stock pile.
* **moveTableauToTableau:** Moves the top card from the one pile to another pile if the move is valid.
* **moveTableauToFoundation:** Moves the top card from the tableau pile to foundation pile if the move is valid.
* **moveStockToTableau:** Moves a stock card from the stock pile to tableau pile if the move is valid.
* **moveStockToFoundation:** Moves a card from the stock pile to the foundation pile if the move is valid.
* **Main Function:**

The game operates in a command line interface where the user is presented with a menu of actions. Before accepting input from the player, the game displays the state of tableau piles which has 7 columns having total cards 28, foundation pile which has 4 piles and at last stock pile which contains the remaining deck of cards which is about 24 cards. The game is presented with a number of choices and each choice represent movement. These movements include:

* **Move tableau to tableau:** It moves a card from tableau pile to another tableau pile.
* **Move tableau to foundation:** It moves a card from tableau pile to foundation pile.
* **Move stock to tableau:** It moves a card from the stock pile to tableau pile.
* **Move stock to foundation:** It moves a card from the stock pile to the foundation pile.
* **Quit:** Exit the game.

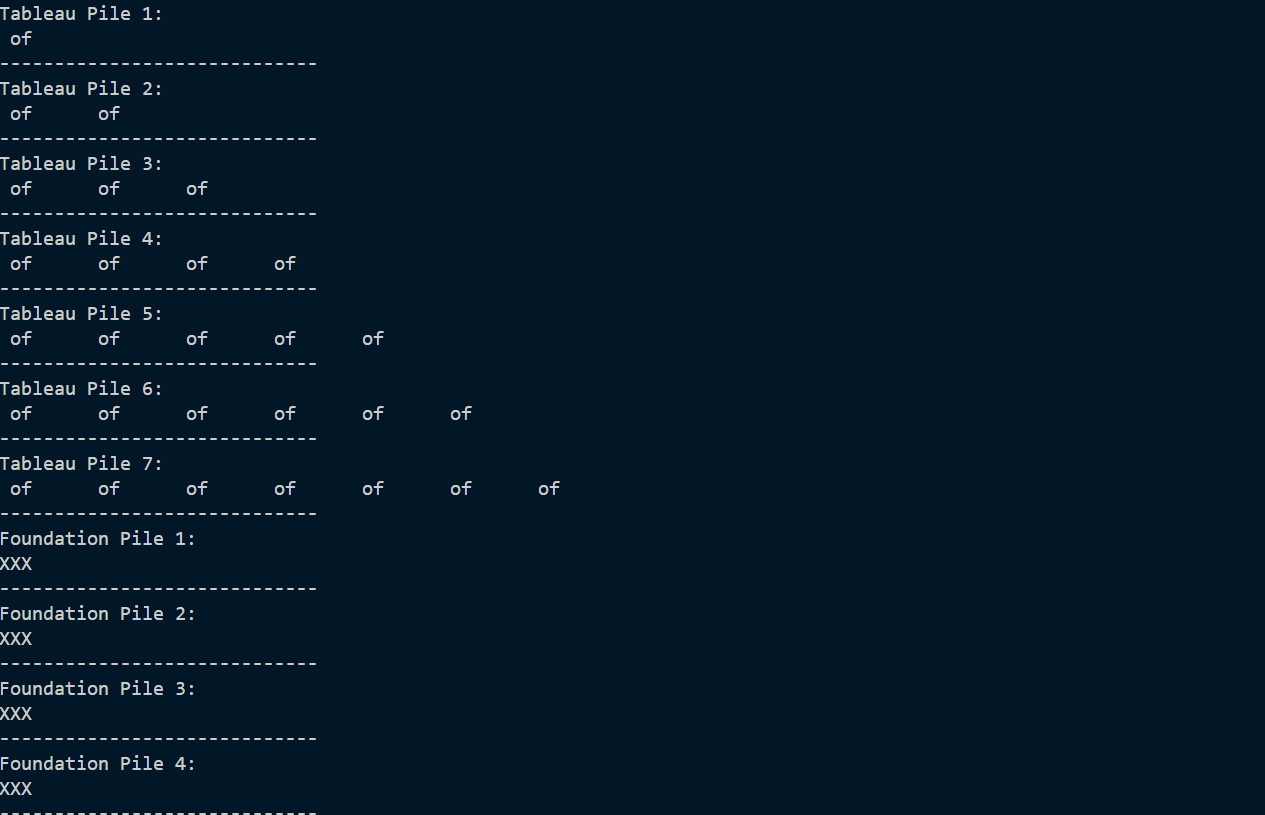
Figure:1

Figure:2

