SOLITAIRE

C + +

Sheak Motin Abdul, Shoeb Iqbal, David Kendal & Mahir Shah

1 THE GAME

2 AIM AND OBJECTIVE

3 SOLUTIONS

4 FLOWCHART

T H E G A M E

- Solitaire is a game that can by played by a single player, requiring just a standard deck of 52 cards. The objective of the game is to organise a shuffled deck of cards into four separate piles, with each pile representing a suite and organised in ascending order (Ace to King).

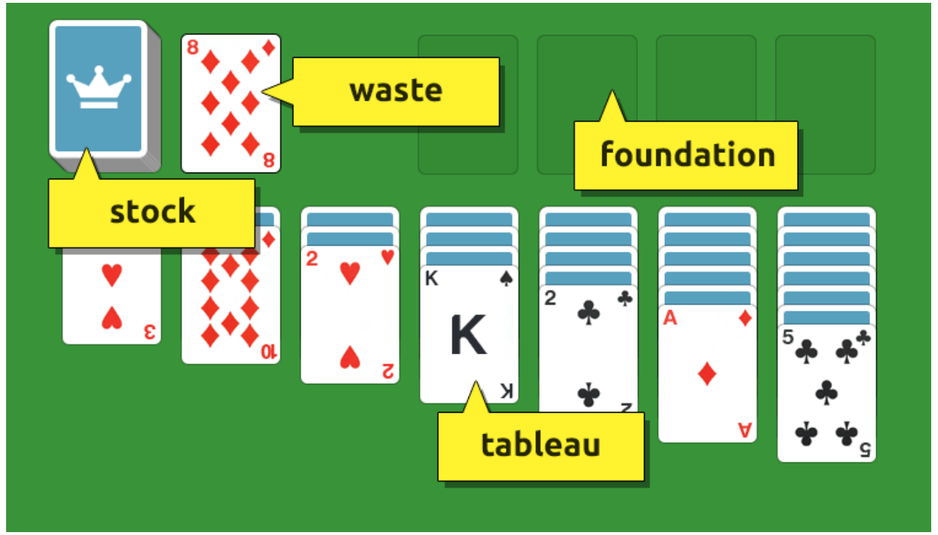
Set-Up
In the game, there are four different piles - the tableau, the stock, the waste and the foundations.

The stock

The remaining card after setting the Tableau

Tableau

7 piles of 7 sizes with random cards that only has the top card face up



The waste

Space where 1 or 3 card will be placed from the stock

The foundation

4 piles, one for each suit that will be placed in ascneding order

A I M & O B J E C T I V E

The aim of the project is to recreate a working and playable version of the the game in c++ with the use of different data structures.

The complete project should initialise the board with the shuffle card and ask the an input through the command line for the next step. The user should only be able to make allowed move:
- move a card from the waste or from a tableau to another tableau, only if is descending order and different suit colour
- flip a card from the deck to the waste
- move card from the waste or the tableau to the foundation only if the correct suit and in the correct order.

Once the deck is empty the waste should be placed back as deck and the game should keep on going till the user can complete the foundation.

S O L U T I O N

In our solution we start by creating all the 52 cards in a list that has their number and suit.
We shuffle the card by placing it in a vector and copying it back in the list.
We than proceed to set up the tableau and their columns with the use of splicing.

```
class Solitaire
{private:
    int cardNumber;
    char suit;
public:
    Solitaire(int cardNumber, char suit);
    int getNumber();
    char getSuit();
    bool getColor(); // returns true if red
};
```

```
std::vector<Solitaire> randDeck;
std::list<Solitaire> deck;
std::vector<std::list<Solitaire>> lists(12);

for (int i = 1; i <= 13; ++i) {
    Solitaire card(i, 'S');
    randDeck.push_back(card);
}

for (int i = 1; i <= 13; ++i) {
    Solitaire card(i, 'D');
    randDeck.push_back(card);
}
for (int i = 1; i <= 13; ++i) {
    Solitaire card(i, 'C');
    randDeck.push_back(card);
}
for (int i = 1; i <= 13; ++i) {
    Solitaire card(i, 'H');
    randDeck.push_back(card);
}
```

```
std::random_device rd;
std::mt19937 g(rd());
shuffle(randDeck.begin(), randDeck.end(), g);
for (auto card : randDeck) {
    deck.push_back(card);
}
```

```
auto itDeck = deck.begin();
for (int i = 1; i < 8; ++i) {
    advance(itDeck, i + 1);
    lists[i].splice(lists[i].begin(), deck, deck.begin(), itDeck);
}
```

Now the user has the option to cycle a card from the waste, move a column to another column or move a single card from the waste to the columns or to the foundation.

```
void moveCardtoCol(std::list<Solitaire>& to, std::list<Solitaire>& from) {
    if (from.size() == 0) {
        std::cout << "Illegal move\n";
        return;
    }
    if (to.size() != 0 && compareCol(to.front(), from.begin())) {
        to.push_front(from.front());
        from.pop_front();
        to.front().setVisible(true);
        if (from.size() > 0) from.front().setVisible(true);

    }
    else if (to.size() == 0 && from.front().getNumber() == 13) {
        to.push_front(from.front());
        from.pop_front();
        to.front().setVisible(true);
        if (from.size() > 0) from.front().setVisible(true);
    }
}
```

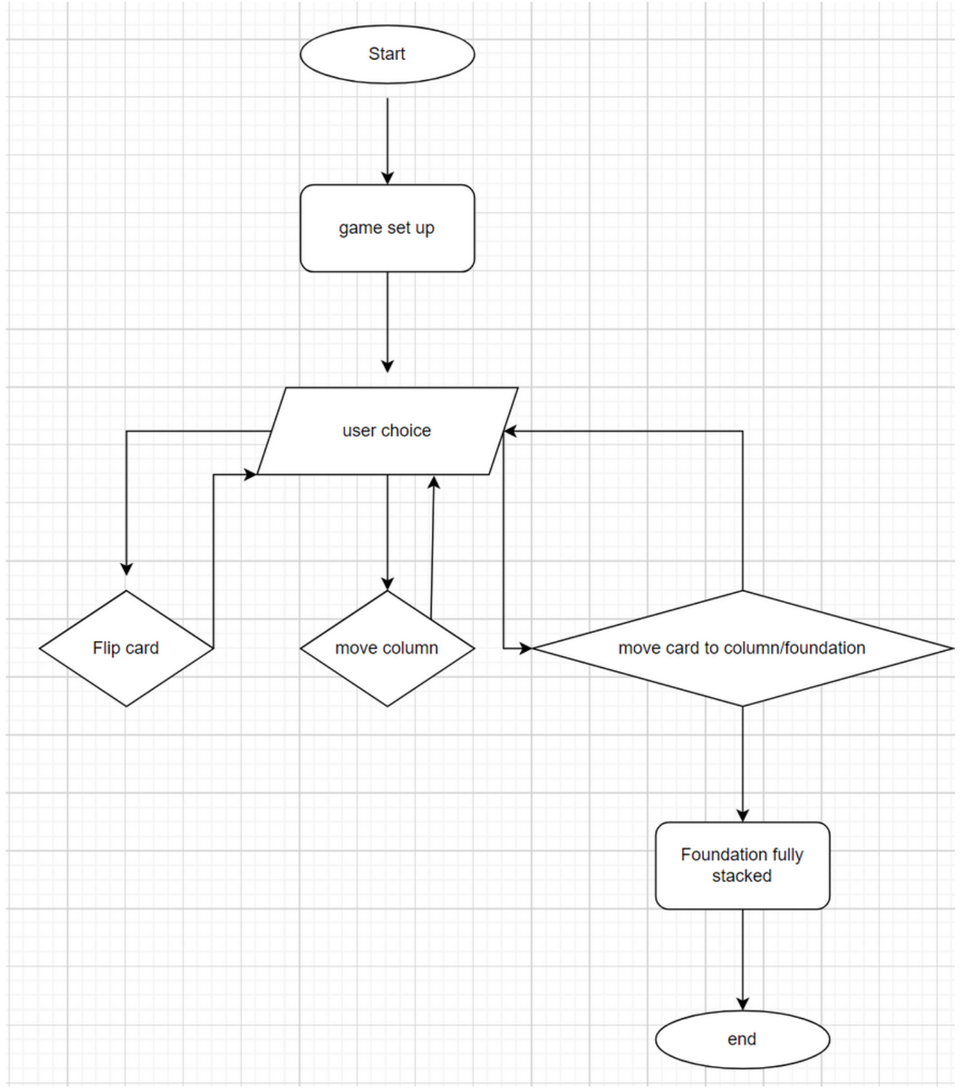
```
void cycleWaste(std::list<Solitaire>& deck, std::list<Solitaire>& waste) {
    if (deck.size() > 0) {
        waste.push_front(deck.front());
        deck.pop_front();
    }
    else {
        waste.reverse();
        deck.splice(deck.begin(), waste);
    }
}
```

```
void moveCol(std::list<Solitaire>& to, std::list<Solitaire>& from) {
    auto itFrom = from.begin();
    if (from.size() == 0) {
        std::cout << "Illegal move\n";
        return;
    }
    while (itFrom != from.end() && itFrom->getVisible()) {
        if ((to.size() != 0 && compareCol(to.front(), itFrom))) {
            advance(itFrom, 1);
            to.splice(to.begin(), from, from.begin(), itFrom);
            to.front().setVisible(true);
            if (from.size() > 0) from.front().setVisible(true);
            break;
        }
        else if (to.size() == 0 && (itFrom->getNumber() == 13)) {
            advance(itFrom, 1);
            to.splice(to.begin(), from, from.begin(), itFrom);
            to.front().setVisible(true);
            if (from.size() > 0) from.front().setVisible(true);
            break;
        }
        std::advance(itFrom, 1);
    }
}
```

This will be looped until the user can put all the cards in ascending order in the foundation pile. Once that is done the user will win the game.

```
    if (lists.at(8).size() == 13 && lists.at(9).size() == 13
        && lists.at(10).size() == 13 && lists.at(11).size() == 13) {
        std::cout << "You win!\n";
        won = true;
    }
```

F L O W C H A R T


Start
game set up
user choice
Flip card
move column
move card to column/foundation
Foundation fully stacked
end