# Programming Assignment 4

## Assignment Details

Assigned: November 5th, 2017. Due: December 1st, 2017 by 11:59 p.m.

## Background

Solitaire is a game that was the most popular time-waster during the golden years of the personal computer in the 1990s. You will write your own Solitaire engine. This programming assignment will be in two parts. The first part uses the Standard Template Library to implement a Klondike Solitaire game without a graphical user interface. The second part uses X11 to provide a graphical user interface for the game. Part 1 is mandatory. Part 2 is optional.

## So, what are we doing?

We want to write a Klondike Solitaire game. We start with a reasonably solid class design to work with. We have to design the game in such a way as to maximize code reuse and simplify the interactions between objects.

## Requirements

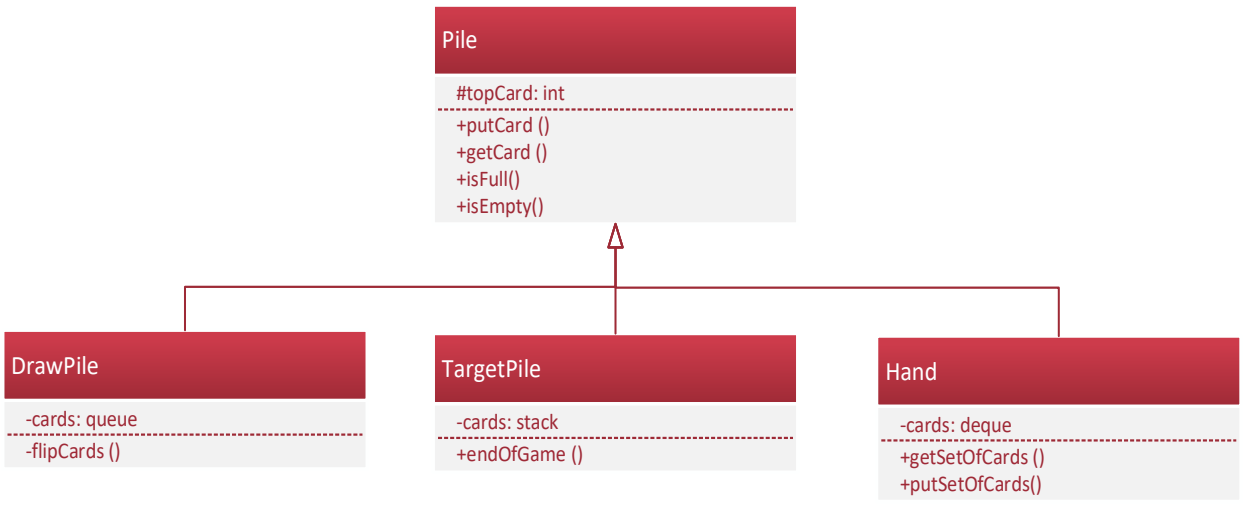
For the first part of the assignment, you will need to provide a header (.h) and implementation (.cpp) file for each of the four classes listed. All classes must follow the inheritance hierarchy specified, implement accessors and mutators for member variables, and member functions should be declared virtual.

Furthermore, all of the files must compile using separate compilation by the g++ compiler.

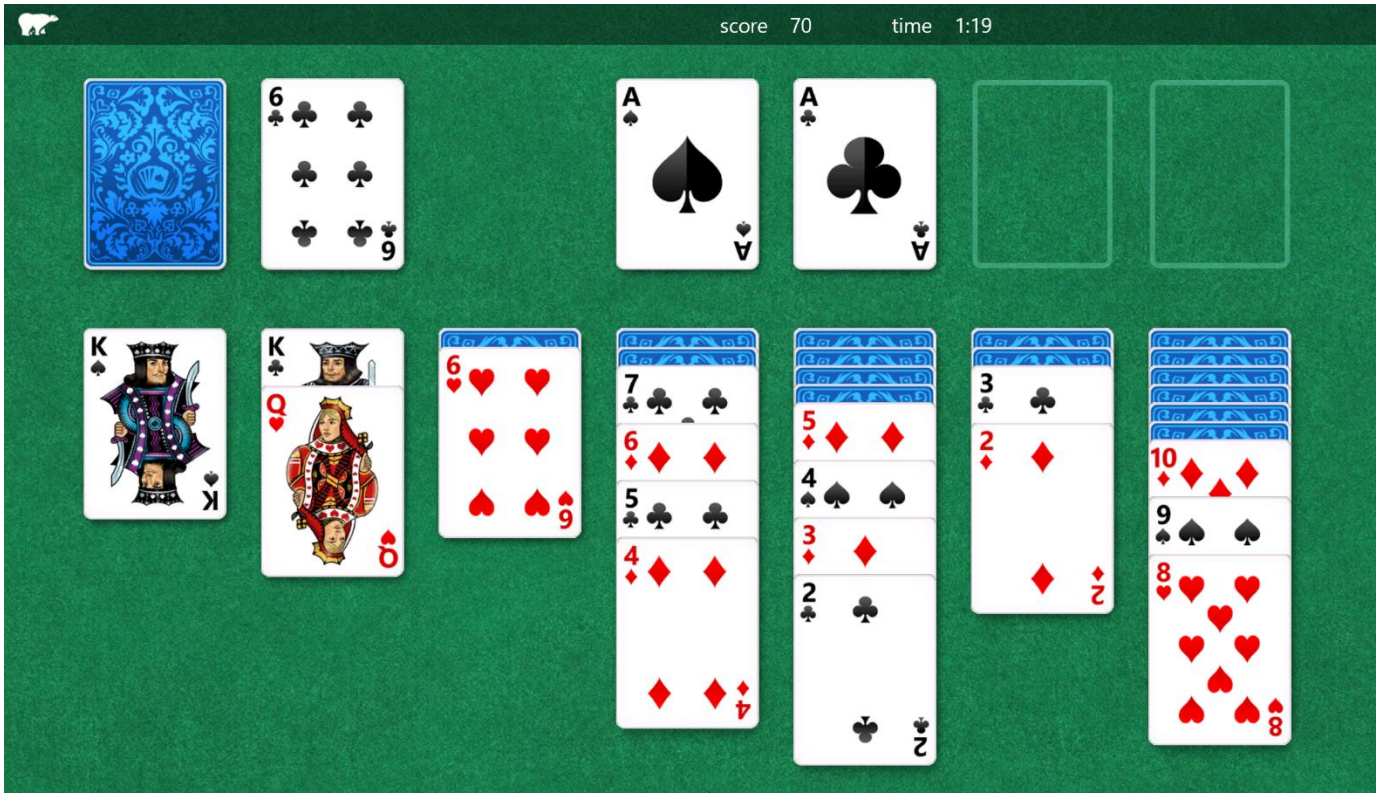
## Deliverables

Your source code enable me to grade your project.

## UML Diagram



## What is a classic solitaire game?




This is a snapshot of a Solitaire game session. The main features to note are drawing pile (top left), target piles (four slots where the aces are located), and seven hands piles at the bottom.

A class will be needed to represent each of the types of piles.

## Initial setup specification user story

A user will want a screen output of an initial Solitaire game as follows:

- There are 52 cards
- The hand piles will have 1, 2, 3, 4, 5, 6, 7 cards from left to right respectively
- The target piles will be empty
- The top card of each pile will be displayed

## Game playing user story

A user will want to play the game as follows:

- A displayed card can be moved.
- A card can be moved to a target pile of the same suit in ascending order. Only ace can be moved to empty target slot
- A card can be moved to a hand pile of a different color in descending order. Only K (13) can be moved to an empty hand pile.
- When a card is moved from a pile, the next undisplayed card should be flipped

## Program specification

**Data structures**

1. Hand piles: create seven deques
   - Randomly generate 28 different numbers (in range 1 to 2) and place one of them in first (left most) hand pile, two numbers in second, three of them in third and so on.
   - Show the user the top most of each of these piles
2. Drawing pile: create a queue
   - Randomly generate 28 different numbers that are different from those generated in (1) above and place them in the drawing queue.
   - Show the user the first card that was inserted in the queue
3. Target piles: create four empty stacks

**Playing**

1. Card designation
   - You may assign a unique number to each card. For example, you can say 1 is ace of spades, 2 is 2 of spades, …, 13 is K of spades, 14 is ace of clubs, 15 is 2 of clubs, …, 26 is K of clubs, 27 is ace of hearts, …, 39 is K of hearts, 40 is ace of diamonds, …, 52 is K of diamonds.
   - The user will input the card number to move, from which pile to which pile
   - If the move is legal, the card will be moved. The next card in the source pile will be shown.
   - All cards moved TO the hand piles will be visible
2. When all cards have been flipped from the drawing pile, the first card that was flipped will be shown
   - In other words, the drawing pile is circular
   - Cards may be moved as a block from one hand pile to another provided the move is legal
3. The game is won when all cards have been moved to the target piles
4. The game is lost if no card can be moved from the drawing pile and no card can be moved to the target piles.

## Randomness

Use STL's Mersenne Twister engine.

You will need to include the `<random>` header file. The class name is `mt19937`, and you'll need to construct an object, seed it with something (`time(NULL)` or C++11 **chrono time** is a reasonable seed), and then you get random numbers by calling the `()` operator on the object, as below:

```
mt19937 mt;
mt.seed( time(NULL) );
// This generates a random number, chosen from
//all possible unsigned ints unsigned int
iRandom = mt();
// This generates a random number in the range 0-99. int
iLimitedRandom = iRandom % 100;
```

## Submission and Grading

Submit by attaching all project files to the assignment page on Blackboard. Please include a **makefile**.

This project will be graded out of 100 points. External documentation is waived for this project.