Command-Line Solitaire

Roman Bureacov

Table of Contents

[Motivation 3](#_Toc211526049)

[0. Specifications 4](#_Toc211526050)

[1. Use Cases 5](#_Toc211526051)

[Use 00: Start a game 5](#_Toc211526052)

[Use 01: Get help 5](#_Toc211526053)

[Use 02: Make a move 5](#_Toc211526054)

[Use 03: Invalid move 5](#_Toc211526055)

[Use 04: No More Moves 6](#_Toc211526056)

[2. Game Rules 7](#_Toc211526057)

[Drawing cards 7](#_Toc211526058)

[Playing cards 7](#_Toc211526059)

[Winning Solitaire 8](#_Toc211526060)

[Losing Solitaire 8](#_Toc211526061)

[3. Structure 8](#_Toc211526062)

[The model 8](#_Toc211526063)

[Commands 8](#_Toc211526064)

[NewGame 8](#_Toc211526065)

[MoveCard 8](#_Toc211526066)

[DrawCard 9](#_Toc211526067)

[NewHand 9](#_Toc211526068)

[AskMoves 9](#_Toc211526069)

[Exit 9](#_Toc211526070)

[The view 9](#_Toc211526071)

[The controller 10](#_Toc211526072)

# Motivation

Successive hours of thinking have led me to believe I need additional projects, so this shall suffice.

Prior to this project’s induction, there was an attempt to create this very same project (see: <https://github.com/RomaBureacov/stuff/blob/master/Solitaire%20(unfinished).java>). This minor project was a product of circa June 2022. During my stay at the community college, I was inspired to re-create solitaire after completing my Java II course. Although I had a burning motivation to create such a project, what I lacked was knowledge on how to create such a program.

Now, I exist with the ability to understand data structures and a few algorithms, but I primarily now share a brainpiece on some practices and principles in software development. This project will serve to exercise these further, given I have the time to provide and this does not become a dead repository. The idea is it does not become a dead repository.

This project should exercise testing, lexical and grammar analysis, object-oriented programming, and command-line interface interaction.

# Specifications

The program will be built in Java exclusively. The expected operating environment is described as follows:

* Java SE 21 or greater
* Windows 11 or greater
* Intel i3 or greater
* At least 10 MB free storage

The following environment description is a product of the limited resources and scope of the project; it is intended to provide what hardware this software will be run on:

* Windows 11
* Intel i7 processor
* Nvidia GTX1650

The following will be used to develop this software:

* OpenJDK 21 LTS
* Windows’s Command Line Interface
* IntelliJ IDEA Ultimate
* Windows 11

# Use Cases

## Start a game

The following describes the process of a user beginning a new game.

1. User opens the application.
2. The application prompts the user with options, on being to start the game.
3. User enters text to start the game.
4. The software will randomly shuffle the cards.
5. The software will place the cards in their respective stacks.
6. The software will place the rest of the cards in the player’s hand.
7. The software will write to the output text buffer the beginning of the game.
8. The software will clear the screen.
9. The software will display the game and prompt the user to make their move

## Get help

The following describes the process of a user getting help.

1. The user will forget how to do a certain action.
2. The user will notice a help hint on the command interface.
3. The user will type out the help keyword(s) into the prompt.
4. The software will append help hints and then move the cursor back into position of the prompt.

## Make a move

The following describes the process of a user making a move on the game board.

1. The user will enter a string of text specifying the moves to make.
2. The parser will ask for tokens.
3. The scanner will provide tokens.
4. The parser will perform commands based on the text.
5. Repeat 2-4 as necessary.
6. The software performs the move.
7. The software rewrites the game board into the buffer.
8. The cursor is repositioned to the start of the command line interface.
9. The software overwrites all text with the new text in the buffer to show the new being made.

## Invalid move

The following describes the process of a user making an invalid move command.

1. The user enters an invalid command that instructs the game to move the card to a position.
2. The software will read the input with a scanner and a parser.
3. The software will detect an error.
4. The software will write into the text buffer the state of the board and an error message below the user prompt.
5. The software will clear the screen.
6. The software will write the buffer to the output.

## No More Moves

The following describes the process of a user running out of possible moves in the game.

1. The user makes a move.
2. The software processes the move.
3. The software will write the output of the buffer to the output.
4. The software will begin to examine the board for possible moves.
5. The software will determine the game is over if all conditions are met simultaneously:
   1. No more cards from the user’s hand may be stacked onto the suite piles.
   2. No more cards from the board may be stacked onto the suite piles.
   3. No more cards from the user’s hand may be stacked onto existing stacks on the board.
   4. No spots are open for kings to go into from the user’s hand.
   5. No spots are open for kings to go into from the board.
   6. No moves of the cards on the board result in stacking cards onto the suite piles.

# Game Rules

Solitaire is a card game making use of the standard deck of 52 cards; additionally there are no jokers.

The game is dealt with 7 columns of cards, with the cards initially face-down. The number of cards in each stack is 1 on the left, and 7 on the right, increasing incrementally to the right (the number of cards per column, from left-to-right, is 1, 2, 3, 4, 5, 6, and 7). The remaining cards are given to the player face-down and as a single stack.

When the cards are dealt, the bottom-end of each column is flipped to reveal the faces of those cards.

### Drawing cards

There are two ways one might choose to play solitaire:

* First variation: Cards drawn from the hand one-by-one.
* Second variation: Cards are drawn from the hand in threes.

In the first variation, you draw from the hand and may play as you draw if you choose. Otherwise you can set the card aside and continue drawing.

In the second variation, you must draw three cards at a time from the hand and must play the top-most card. For example, if the hand had—from top-to-bottom—a jack, queen, and king, then you would see—from top-to-bottom—a king, queen, and jack. You must play the king first before being able to play the two under it, the queen and then the jack. You play the top-most cards first.

In either variation, the drawn cards are placed into their own pile. We will call this the “offhand.” The offhand contains all the previously drawn cards on top of it, face-up. The player may choose to play the cards as they appear on the top of the offhand.

Once all the cards in the player’s hand are drawn into the offhand, then the player simply picks up the offhand and uses it as their new hand and repeats the drawing process.

### Playing cards

To play cards the player has two choices:

* Stack cards onto the board’s columns.
* Stack cards onto the suite piles.

The board’s columns have the two rules to stacking:

* Each card must alternate in color.
* The cards must descend incrementally, going down. The order is king, queen, jack, followed by 10-2, and finally ace.

Note that if the board has open spots, it is only the kings that may take those open spots. Additionally, note that there is no suite restriction on the columns.

The suite piles have three restrictions to stacking:

* The pile must be available.
* The pile can only contain its respective suite.
* The pile must stack incrementally.

Suite piles are made available with aces. Aces begin their own separate stack on the side of the columns. When the player has an available ace, they can play that ace to begin the suite pile.

#### Winning Solitaire

The condition to win solitaire is simple: stack every card onto the suite piles. This winning condition may be shortcut with turning every hidden card in the start of the game in each pile.

#### Losing Solitaire

The condition to lose solitaire is when you have no more cards you can play, and ultimately it is impossible to otherwise reveal the hidden cards and stack the suite piles.

# Structure

The software will be split into the model, view, and controller.

## The model

This component will consist of the game board itself. It will feature actions that may be taken to modify the state of the game and algorithms to check for remaining moves.

### Commands

The model has a command input consisting of three integer arguments to modify the state of the game described in the following sections. The commands may or may not write back an output object. Unless otherwise specified, the command will output null.

#### NewGame

Creates a new game and writes out to the screen the new game.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 0 | 0 | 0 |

#### MoveCard

Moves an applicable card from one pile in Arg1 to the pile in Arg2.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 1 | Pile# | Pile# |

There are 12 pile numbers in total and they are organized as follows:

|  |  |
| --- | --- |
| Pile# | Pile description |
| 0-6 | Board columns 0-6 |
| 7 | Aces suite pile |
| 8 | Diamonds suite pile |
| 9 | Clubs suite pile |
| 10 | Hearts suite pile |
| 11 | Hand |

#### DrawCard

Draws a set of cards from the hand into the offhand.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 2 | DrawMode | 0 |

The draw mode is one of two numbers:

|  |  |
| --- | --- |
| DrawMode | Description |
| 1 | Draw 1 |
| 3 | Draw 3 |

Returns a Boolean true if it successfully drew a new set of cards, otherwise false if it failed to draw cards (there are no more cards in the hand and all cards are in the offhand).

#### NewHand

Moves all the cards from the offhand back into the hand.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 3 | 0 | 0 |

#### AskMoves

Examines the board to see if it is possible to make a move that results in either:

* A card moving to one of the suite piles from the board.
* A card moving to one of the suite piles from the hand.
* A card moving to one of the board columns from the hand

This command will return a Boolean true if there is still a remaining move that can be made. Otherwise it will return a Boolean false.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 4 | 0 | 0 |

#### GetPile

Returns card arrays that correspond to specific objects.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 5 | Pile# | 0 |

Pile numbers may be referenced in the command reference for MoveCard.

#### Exit

Exits the game and the software.

|  |  |  |
| --- | --- | --- |
| *Opcode* | *Arg1* | *Arg2* |
| 9 | 0 | 0 |

## The view

This component will consist of the algorithms to write to the console output. It features the ability to query the game board and attain a buffer to write out to the console the new game state. Additionally, it will contain the means to send user input over to the controller.

## The controller

This component will consist of algorithms to parse user input and send a corresponding instruction to the game board. It will consist of the scanner to look for tokens, and the parser to translate that into commands.

Upon recognizing a string as a command, the controller will turn it into a command that it will pass off to the model to change the state of the game.