Marist College

Nondeterministic Finite Automata

Solitaire

Andrew DiBella

Formal Languages CMPT440

Professor Rivas

15 May 2020

Abstract:

A Nondeterministic Finite Automata(NFA) is anything that can be described as having a defined number of states with a final accepting state as the end goal. Now any card game has an infinite number of possibilities depending on the shuffling of the alphabet in this game being the 52 cards in a full deck. Since this is the case I am going to abstract a NFA from the game of Solitaire and validate if the set of cards has a potential NFA based on the algorithm I created. Having an alphabet of 52 cards would make the NFA almost impossible to create; instead there will be three states and the alphabet will still be the cards but the diagram won't depict each individual card moving from state to state. Considering Solitaire is not easy to win, I have created my algorithm to follow basic strategy rules and if a game is won it will return the number of moves and the time it took to finish from a sample of differently shuffled decks. A shuffled deck will be considered a NFA if each suit is ordered from ace to king in the foundation state. Cards in the hand can only be placed if their preceding card is 1 greater than the previous value and the previous card is the opposite color of the card trying to be placed. Once the deal state and the hand state are both empty there is a possible NFA for that specific deck. I originally thought this system to be constructed as a DFA but after much thought it has proven to be an NFA due to its Non Deterministic nature and infinite number of possible moves.

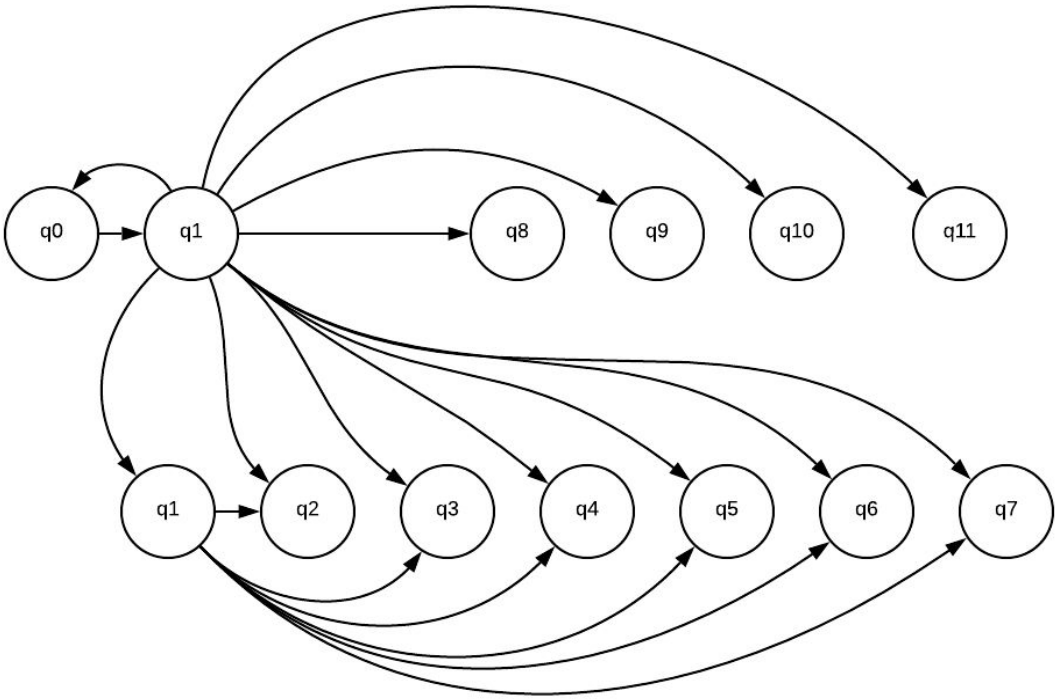
Introduction:

Solitaire is a famous card game designed for one player and a full deck of cards. Although many people find it boring and lengthy, it actually takes a fair amount of skill to win a game. Yes of course, online you are able to undo moves until you finally win, but where's the fun in that. Solitaire was one of the first card games I learned to play as a young kid, so I thought to create a NFA based algorithm that will perform the same actions to win a game of solitaire. Since it is not possible to win every game of Solitaire, I will be testing numerous differently shuffled decks on the algorithm I have created. The motivation for this project comes from my enjoyment from playing card games and I thought it would be interesting to combine that with my coding interests and implement a feasible single player game in Java.

Originally I was going to use stacks each different section i.e. dealer, hand, and foundation but my original program did not run correctly due to the visibility of the card objects in the hand stacks. Considering in solitaire you are also able to move entire sections of cards, I needed to implement an ArrayList of ArrayList<Card> to represent the hand. This way I am able to both print the entire stack even if some cards are not visible and also move full sections of the cards to different lists in the hand.

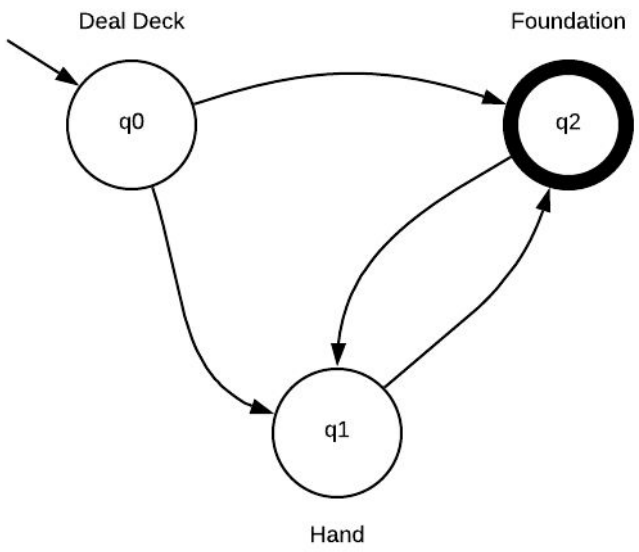
Detailed System Description:

My initial idea was to make a DFA for all of the possibilities for all 11 states and 52 cards. Looking back, I see now that this was a naive approach. The mere number of states and a large deck of cards as the alphabet would make the DFA extremely convoluted and impossible to comprehend.



This DFA is only the start of my initial plan and doesn't even consist of half of the transitions it should in the diagram. Each of the bottom states need transitions to and from each of the other states which you can see would be nearly impossible to implement as a DFA. Because of this I decided to divert my original plan and create an abstraction of this original DFA and make it something easier for a person to understand.

The new NFA consists of three states and the only transitions necessary from state to state is the mere absence of cards from the deal state, the initial state(q0), and its transfer to the player hand(q1), and the foundation being the accepting state (q2).



Even though this is a simple diagram with only 4 transitions, the algorithm I have created will still validate each card movement from state to state as described in the more intricate diagram. The program will take a randomly shuffled deck of cards and attempt to win a game of solitaire. If the program reaches an accepting state then each of the cards moved from the initial deal state and transferred either directly to the foundation, or from the hand to the foundation. Once a card has been moved from the deal state it will only be able to transfer back and forth from the hand and the foundation. Considering it is possible to lose a game of solitaire depending on the moves you make and the deck given, there will not always be a corresponding NFA for each game.

My implementation of solitaire will include 4 java classes

Card

CardStack

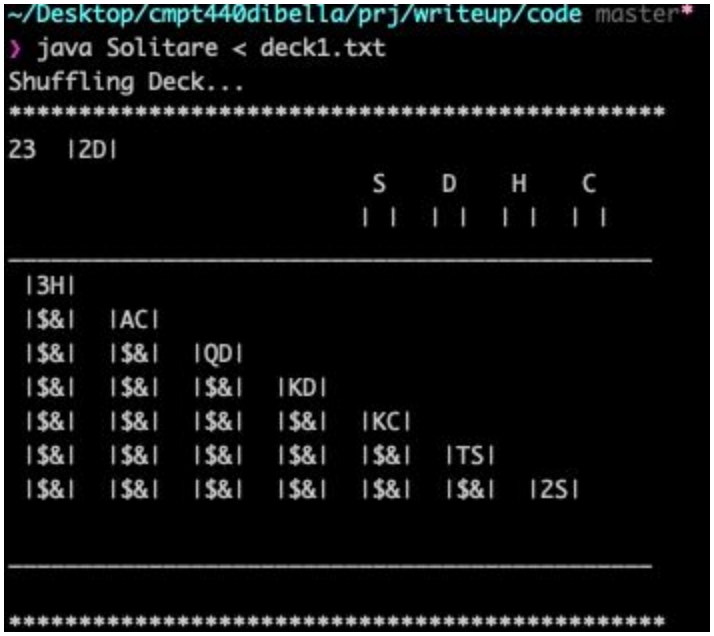
CardStackNode

Solitaire: Driver

The program will utilize several different CardStack objects for each state shown in diagram1 which represents each CardStack. I chose to use a stack data structure to organize the data due to its First In Last Out to represent a physical deck of cards. This will make it easy to only access the top of each stack which is what is used in the game of Solitaire.

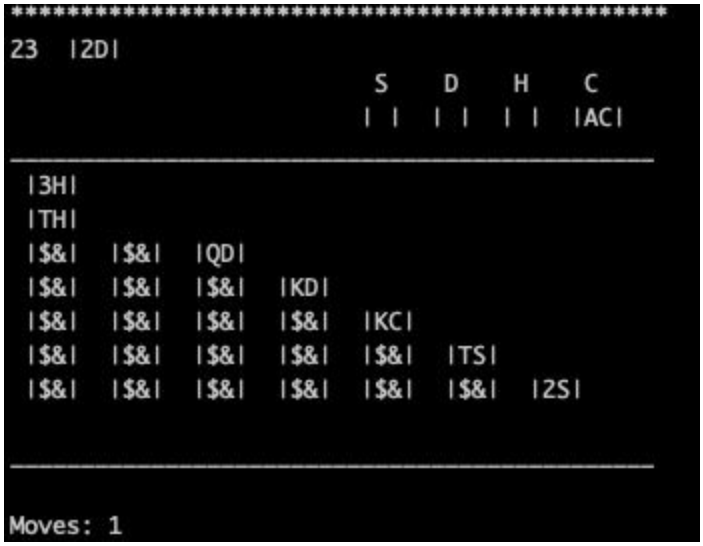
As for the Hand portion of the program I utilize a ArrayList of ArrayLists holding card objects to utilizes the flexibility of each stack of cards while also being able to utilize full sections of cards for example you are able to move an entire section from King-2 if they are connected by the intermediate values in between. Moving an entire section of cards is a very common move in solitaire but difficult to implement using a stack. Using an arraylist proved to be effective so you are able to move entire sections from king and so on to empty sections of the hand. The hardest challenge of the implementation was deciding when to utilize cards that have already been put in the foundation to your advantage in the hand. This concept is very difficult to implement in code and ended up being the demise of my project. After many different versions I've finally come to a version that works(almost).

Example of initial screen:



Shows the all visible cards and not visible face down cards for each arrayList of cards in the hand.

Example after a single move of Ace card to foundation:



Requirements:

Java installed

Both a JRE and JDK are required to run the project.

Command Line

Can be run on any operating system as long as Java is installed

Github account

Required to clone the repository to have access to the files

Literature Survey:

There are many different versions of klondike solitaire that involve the user playing the game. I have not been able to find a version that actually plays the game for you. This is what makes my project unique but also very difficult.

User Manual:

If you are interested on testing this DFA yourself on different decks follow these instructions:

1. Open a command line
2. Clone the Repository-

   $ git clone https://github.com/andrewdibs/cmpt440dibella.git
3. Change directory path to -

   $ cd cmpt440dibella/writeup/code
4. Compile all java files -

$ javac <filename>.java

5. To run program -

$ java Solitare < <inputFile>.txt

Input files should follow the structure of "value suit" -

1 H

5 S

11 D

12 C

This input should consist of a total of 52 cards in any order with no duplicates.Notice the number inputs for 11 - 14 represent Jack through King. In most card games Ace is represented as either 1 or 15; in the game of Solitaire Ace is always defined as a value of 1.

Results:

Considering there are many different strategies and ways to play the game, making an algorithm that plays for you is very unlikely to be successful. I have made several different versions of the game but actually have not been able to make the algorithm win a single game. I have also played many games on my own and I win about 15% of the games I play without undoing any moves. The main issue of creating an algorithm to solve a game of solitaire is using the cards that have already been put in the foundation to your advantage and there are infinite possibilities of moves for every different game. Another issue I ran into many different decks are actually impossible to win from the start.

Conclusion:

I just want to say before I conclude that I put a ton of hours into this project and haven't been very successful. I think creating a self playing game is extremely difficult and I wish I had more time to make it win 100% of the time, but as for now that is unfeasible. I should have taken professor Rivas's advice when he said I wouldn't enjoy making a solitaire game, but here we are. Also, the mere unfortunate circumstances of the pandemic has caused issues with the production of this project.

That being said, this project demonstrates how the game of Klondike Solitaire is in fact a working Nondeterministic Finite Automata in its structure considering the infinite number of moves that are possible when playing the game. The NFA demonstrated by the abstract representation shown above in figure two shows the progression of card objects from the deal stack to the hand or foundation with the possibility of moving from different positions in the hand, and finally all cards will eventually be in the foundation of each suit from Ace to King. Once this is achieved the NFA has reached an accepting state, and the program returns a congratulation message as well as the number of moves taken by the program and the amount of time it took in nanoseconds.

References:

Webber, Adam Brooks. “Formal Language a Practical Introduction”. Franklin, Beedle & Associates, Inc. 8536 SW St. Helens Drive, Ste. D Wilsonville, Oregon 97070. www.fbeedle.com.