John Hopkins University

Data Structures Lab 3 Analysis

By: Rafat Khandaker

Date: 11/17/2021

Abstract

A Linked-List Array data structure was implemented in a popular game of solitaire, known as “Scorpion.” The data structure was used in an algorithm to auto-play the game through a computer simulated program to obtain step by step test results of the game output, thoroughly detailing the decision-making strategy used to auto-play the game. The algorithm used to make decisions was based on a priority calculation function which calculates the priority of an optional move. The move option with the highest priority was taken by the algorithm at each cycle execution of the program simulated game. Test results were printed to an output file to obtain data for analysis.

Objective

The objective of this assignment was to produce an algorithm that can successfully play “Scorpion” until a successful win strategy was obtained. In this lab implementation, the algorithm decision making is solely based on 4 weights:

- Position of the card on column
- # of sequential matches in each column
- # of bottom cards carried over by the optional card move
- # of face down cards in each column

The weights used in priority calculations allow assignment to the strategy for the algorithm’s decision. Higher weight values placed on each of the 4 objects allow the algorithm to take more priority for those weights to calculate the best optimal move. In the default implementation of the algorithm, more weight was placed in the “# of face down cards” contained in each column; this allows the algorithm to make biased decisions to flip the face-down cards if the option exists as a higher priority than other weights. As a strategy of the game, it is recommended to flip the face down cards as much as possible so it can give more optional moves in the next sequential turns.

Compare Data Structure with Previous Lab

In our previous lab, we used an array data structure & stack data structure. This lab I am exploring a Linked List Array Data Structure, the Linked List implementation in Java allows us to access all methods associated to Stacks & Lists, as well as advantage of sequential access. This structure is able to efficiently traverse through data sets. The Array of the Linked Lists creates the columns of the game card layout. This is a good representation of the game, as the game contains 7 finite columns and may contain a variable length of data size contained within each column. Using a stack or array implementation for the List will make difficult to traverse through the data, as we would have to keep track of increments in array and only be able to access last element within each stack. The flexibility of this data-structure gives us a firm grip on creating functions within the game's algorithm.

Analysis

After executing the game several times, I decided the best default weights are best associated to # of face-down cards, as removing those cards early will also isolate the weights from 4 to 3 values, getting closer to a successful completion of the game. 1 default execution of the program was made on the given file: "ScorpionRequiredInput.txt" which printed 1927 lines before no more optional plays were found. Which is about 80 to 85 moves were played. Data was collected from 3 additional test files, scrambling the data contained from the original input file.

Approximate Number of Moves Played for Each File

File Name	# Of Moves Played By Program	# of face down cards flipped	# of Correctly placed
ScorpionRequiredInput.txt	80 to 85 moves	6	5/52
testfile1.txt	28 to 33 moves	0	1/52
testfile2.txt	33 to 38 moves	1	1/52
testfile3.txt	Infinite (stuck)	3	0/52

The randomness and order of cards in each shuffle associated to "Scorpion," cannot be controlled and no guarantee exists that a successful win can be obtained for each shuffle. The algorithm correctly swaps the card and child cards to the appropriate location after calculating the priority of each available option. If the optional choice involves moving a set of cards to reveal a face-down card, a higher priority is given to that move for the algorithm to make decision based on. On that level of knowledge, the algorithm successfully has the ability to flip face-down cards when the option is available.

Summary

In summary, in the current "Scorpion," algorithm, no successful endgame was achieved but the program successfully executed the lifecycle of the program with the desired strategy in mind. As a developer with limited understanding of the strategies involved in "Scorpion," I cannot formally say that a win strategy exists for each shuffle. I can say that the program reflects the best of my own understanding of the game

with the rules correctly placed. As I can randomly pick optional moves in each play, I can program an algorithm to make moves based on certain priorities of the game. If I were to enhance this algorithm, one possibility would be to assign a randomness choice on certain options within the priority weights, where the algorithm will allow a random decision to be taken outside the best priority, maybe within the top 50% priority of choices. By programming the algorithm with some randomness, I can go outside my own understanding of "Scorpion" and analyze the data collected by some small deviance of random choices taken by the algorithm. If I were to have more time to spare, I would chart statistics collected by the algorithm, in order to program a more efficient decision-making strategy collected by statistics of the game. This will allow me to enhance the program's ability to find a successful solution to the game. Which will allow the algorithm to update it's own priority weight suggestions, based on multiple executions and statistical data collected from the game.