# Graded Assignment Report – Haskell Solitaire Implementation

Name : Vlad-Cristian Prisacariu

Registration Number : 200131014

Username: aca19vcp

## Design

Here is a list of the data structures defined in my solution.

* *Suit* – the suit of a card (Hearts, Clubs, etc)
* *Pip* – the number of a card (Ace, Two, .. ,King)
* *Card* – a *Suit, Pip* tuple
* *SCard* – initially thought to be a “spider solitaire” card, because it is defined as a card that can be either face up or face down, i.e *Card Bool*. It is used in Eight-off Solitaire, although all cards are face up.
* *Deck* – a list of *SCards*
* *Foundations ­*– a list of *SCards* representing the top card of the foundation piles
* *Columns* – a list of *Decks*, representing the columns on an Eight-off board
* *Reserve* – a list of *SCards*
* *Board* – *Foundations*, *Columns* and *Reserve* – that make up a tableau of a solitaire game

This is the design diagram of my implementation of eight-off solitaire in Haskell.

The diagram is not a comprehensive list of all of the functions used in my solution, but rather an overview of the most important ones, on a higher level, and it excludes the helper functions.

## Functions

toFoundationsReserve :: Board -> Board

Given a board, *toFoundationsReserve* places every card from reserve to foundations, and returns the resulting Board.

toFoundationsColumns :: Board -> Board

Given a board, *toFoundationsColumns* places every card from the column heads to foundations, and returns the resulting Board.

toFoundations :: Board -> Board

Given a board, toFoundations keeps using *toFoundationsReserve* and *toFoundationsColumns* untill no cards can be moved to foundations.

moveKingColToEmptyCol :: Board -> Board

Given a board, *moveKingColToEmptyCol* moves the first king it finds in the column heads, to an empty column. This is a good move when trying to declutter the columns, and when there are empty columns.

moveKingResToEmptyCol :: Board -> Board

Given a board, *moveKingResToEmptyCol* moves the first king it finds in the reserves, to an empty column. This is a good move because it empties a reserve slot, when there are any empty columns.

moveResCardToCols :: Board -> Board

Given a board, *moveResCardToCols* moves a card from the reserves to one of the columns, if there are any such movable cards. This move empties a reserve slot, when there are any moveable cards from reserve to columns.

moveColsCardToCols :: Board -> Board

Given a board, *moveColsCardToCols* moves a card from the columns to one of the other columns, if there are any such movable cards.

moveNthMvblColHeadRes :: Board -> Int -> Board

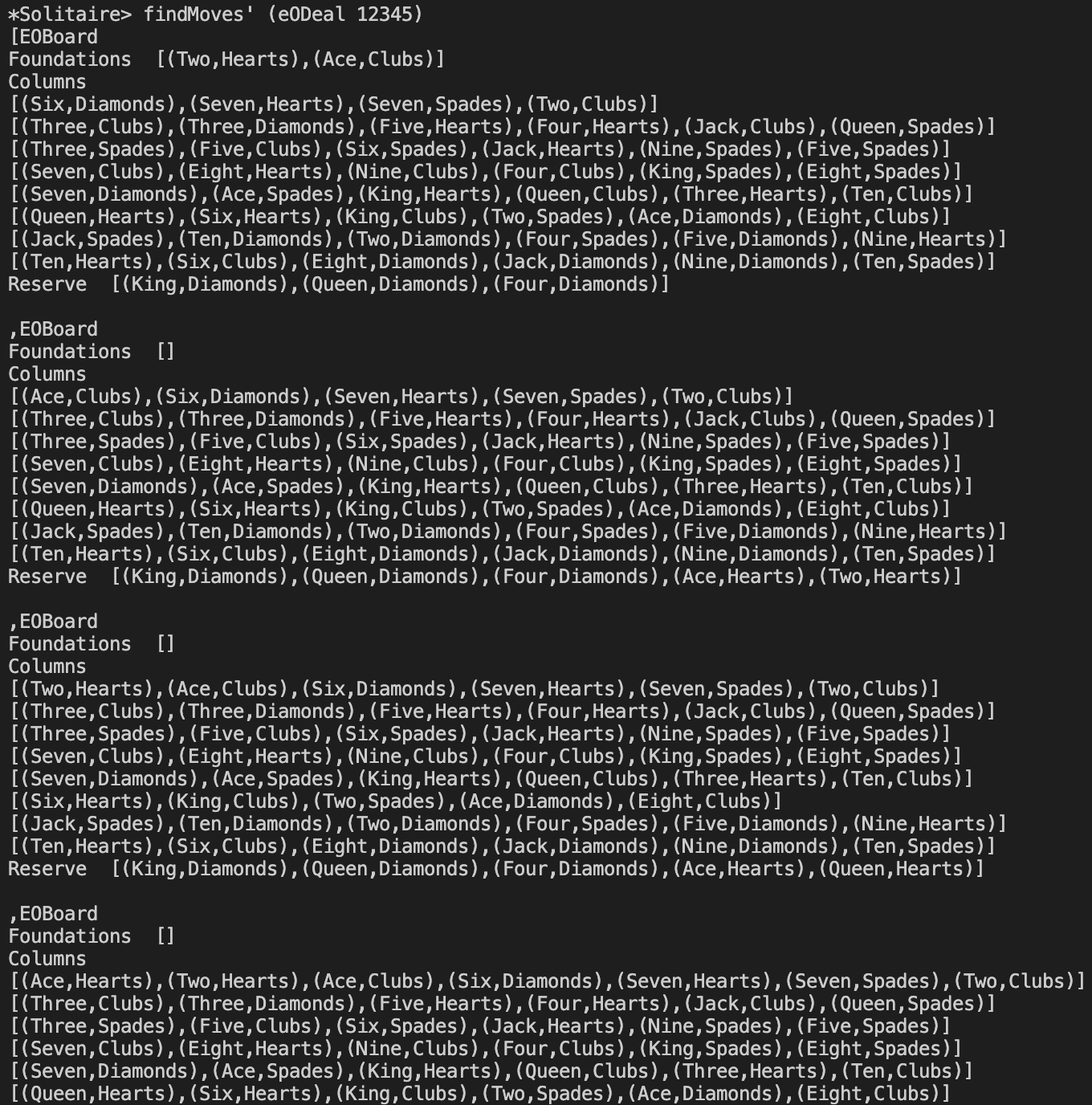
Given a board and an index, *moveNthMvblColHeadRes* moves the to reserve the head of a column whose nth card can be moved to foundations.

findMoves :: Board -> [Board]

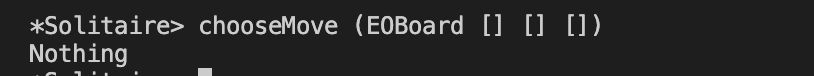
Given a board, *findMoves* returns all of the board states after a move has been played, ordered in terms of the move quality (i.e. “best” move is the first in the list). The best move is always *toFoundations*.

chooseMove :: Board -> Maybe Board

Given a board, *chooseMove* simply returns the first item in *findMoves* applied to that board, unless there are no available moves, in which case it returns Nothing.







score :: Board -> Int

Given a board, *score* counts how many cards there are in the foundations (it subtracts the number of cards in columns and reserves from 52, to be exact), and returns the result.

playSolitaire :: Board -> Int

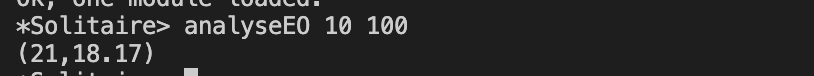
Given an initial, randomly dealt board, *playSolitaire* will exhaust all available moves and return the score of the final board.

analyseEO :: Int -> Int -> (Int, Float)

Given an initial seed and a number of games, *analyseEO* will play that number of eight-off games, and return the number of wins, and the average score across all of the games.

## Results and Performance

Here are a few of the results I managed to get using my implementation. Note that *analyseEO* returns a result of type (*number of won games*, *average score*).

This first one was using analyseEO, starting with seed 10, and incrementing it by 1 for every new game. I believe winning 21 games out of 100 is a satisfying result.

../../../../../../../../../../var/folders/k7/kl_mvv2x27bb1f87fkpyx8nw0000gn/T/TemporaryItems/NSIRD_screencaptureui_QfwTBi/ScreenshThis time, I have started with the seed on another number, and the seed is doubled for every new game. The performance is quite good, still.

../../../../../../../../../../var/folders/k7/kl_mvv2x27bb1f87fkpyx8nw0000gn/T/TemporaryItems/NSIRD_screencaptureui_TcwLO8/ScreenshJust out of curiosity, I let the program run for 1000 games, and the result is quite impressive.

## Spider Solitare - Beginnings

I have only started the implementation for the other type of solitaire, namely spider solitaire. The sDeal function has been in place since the first part of the assignment, but I have managed to add a few helper functions in the meantime.

moveStockToCols :: Board -> Board

Given a board, *moveStockToCols* deals one 10 card deck to the columns, assuming there are no empty columns (because this is checked in another function).

moveAceToKingFound :: Board -> Board

Given a board, *moveAceToKingFound* moves every Ace to King sequence in the columns, to the foundations. Here it is in practice:



