Aleksander Marcin Osikowicz
Student number: 200130914
User name: aca20amo
Email: amosikowicz1@sheffield.ac.uk

COM2108 Functional Programing Solitaire Assignment

Design for Eight-off and Spider solitaire:

At the end of the report I attached a hierarchical diagram of functions used for playing and analysing both Solitaire's.

Top level functions:

When I was writing analyseSpider, I noticed that they are essentially the same function, the only difference is which dealing function needs to be called, sDeal or eODeal. So I wanted to make my code extensible for other solitaire games thus I created a general function called analyse. That both analyseSpider and analyseEO are going to use.

- ➢ analyseEO = analyse EO, where EO is of GameType data type
- ➢ analyseSpider = analyse Spider, where Spider GameType data type

- ➢ analyse - top level monad function that analyses performance of chosen solitaire game. Function starts by generating random seed for each of the games to be played and writes that to the channel. Then the function shares the workload among multiple threads to increase performance, forks threadPlayGame. After all working threads are done, sums up results and returns the number of won games and average score.
    - Parameters:
        - GameType - currently implemented for 8-off and Spider solitaire.
        - Int - initial seed for random number generation
        - Int - number of games to be played
    - Returns:
        - IO(Int,Float) - First element is number of won games, second is average score of all games played
    - Calls:
        - getWinningScore
        - threadPlayGame

- ➢ threadPlayGame - working thread created by analyse function. Gets GameType, variant to be played, channel of seed, and channel of results. Functions reads one seed from the channel and then deals the initial board using it. After that plays the game and writes its result to the scores channel.
    - Parameters
        - GameType

        - Chan Int - channel storing seeds for dealing
        - Chan Int - channel storing results
    - Returns:
        - IO()
    - Calls:
        - getDeal
        - sDeal or eODeal depending on the GameType
        - playSolitaire

Dealing initial board 8-off:

➢ eODeal - gets initial seed for shuffling cards, deals initial eight-off board according to game rules

Dealing initial board Spider:

➢ sDeal - gets initial seed for shuffling cards, deals initial board for Spider Solitaire, with empty foundations, stock with 50 cards facing down, and will columns according to the game rules.

Dealing building blocks:

➢ pack - constant function, creates sorted list of all cards using list comprehension.

➢ shuffle - shuffles (mixes) deck of cards, using random numbers and list comprehension

Playing Solitaire:

➢ playSolitaire - recursive function that each iteration plays a move, until there are no other moves that make sense
    - Parameters:
        - Board - current board, state of the game
    - Returns:
        - Score of the final board when run out of the moves
    - Calls:
        - Recursive call of playSolitaire if there is next move.
        - calculateScore if otherwise
        - chooseMove

➢ calculateScore - calculates the score of a final given board, uses foldr to go over foundations of the board, and gives one point for each card inserted.
    - Parameters:
        - Board - board of which score we are calculating
    - Returns:
        - Int - calculated score

- Calls:
    - Nothing

➢ haveWon - checks if a given board is a winning one, uses calculateScore to determine if the score is equal to 52 for 8-off or 104 for Spider .

➢ chooseMove - Takes current board and generates move for first step using findMoves, and for each of the moves finds another moves, then filters out moves that create loops (e.g current board appears in next 2 steps).
- Parameters:
    - Board, current board
- Returns:
    - Maybe Board, normally returns Just Board but, if there are not moves return Nothing
- Calls:
    - toFoundations
    - findMoves

To foundations:

➢ toFoundations - recursive function, that takes board and returns board where all cards that could be inserted into foundations, are in foundations. Uses list comprehension to determine all moves from reserve to foundation and from columns to foundations. For 8-off cards can be put to foundations one by one, but for Spider there needs to be a complete suit.
- Parameters:
    - Board, board from which we need to add card to foundations
- Returns:
    - Board, with all cards added toFoundations
- Calls:
    - toFoundations if there was a card inserted to foundations (we need to check if there is any other).
    - takeCardFromReserve (8-off)
    - takeCardFromColumn (8-off)
    - putCardToFoundations (8-off)
    - putCompleteSuitToFoundations (Spider)

FindMoves:

➢ findMoves - gets a list of all possible legal moves, and maps toFoundations on them.
- Parameters:
    - Board, current state of the game
- Returns:
    - [Board], all possible legal next states of the board
- Calls:
    - toFoundations
    - findMovesFromColumnsToColumns

- findMovesFromReservesToColumns (8-off)
- findMovesFromColumnsToReserves (8-off)
- dealStock (Spider)

➢ findMovesFromReservesToColumns (8-off) - uses list comprehension to get all possible moves, by moving one card from reserves to columns. Gets Board and returns a list of Boards

➢ findMovesFromColumnsToColumns (8-off and Spider) - uses list comprehension to get all possible moves, by moving cards from columns to columns. Gets Board and returns a list of Boards.

➢ findMovesFromColumnsToReserves (8-off) - uses list comprehension to get all possible moves, by moving one card from columns to reserves. Gets Board and returns a list of Boards.

➢ dealStock (Spider) - given a board returns Just Board with cards dealt from stock if possible, otherwise returns Nothing

Building blocks used by multiple higher order functions:

Card putters:

➢ putCardToFoundations (8-off) - takes a pair of card and board, and returns Just board if it's possible to add card to the board's fundations, returns Nothing otherwise.

➢ putCompleteSuitToFoundations (Spider) - gets index of the column the board, and tries to put complete suit from that column to foundations, if it is possible returns Just board with applied transformation otherwise returns nothing

➢ putCardToReserve (8-off) - takes a pair of card and board, and returns Just board if it's possible to add card to the board's reserve (e.g there is an empty slot), returns Nothing otherwise.

➢ putCardToColumn (8-off) - takes index of column, a pair of card and board, and returns Just board if it's possible to add card to the board's column of a given index, returns Nothing otherwise.

➢ putCardsToColumn (Spider) - takes index of column, a pair of Deck and board, and returns Just board if it’s possible to add sequence of cards to the board’s column of a given index

Card takers:

- takeCardFromReserve (8-off) - takes index of reserves slot, board, and returns pair of taken card and board after taking card from reserves.

- takeCardFromColumn (8-off) - takes index of the column, board, and returns Just a pair of taken card and new board if it was possible to take card from the indexed column.

- takeCardsFromColumn (Spider) - takes index of the column, number of cards that we are about to take, the board, and returns Just a pair of taken cards and new board if it's possible.

Basic functionality:

- sCard
- pCard
- isAce
- isKing
- showHead
- showAllCards
- hideAllCards
- getWinningScore - gets GameType and returns number of points that need to be scored to win in that game variant
- getDeal - given GameType returns function to deal the board.

Data types:

- data Suit
- data Pip
- data Card - three arguments Pip Suit Bool, boolean indicates if card is visible.
- type Deck = [Card]
- Foundations, Columns, Stock, Reserves are of type Deck (which is list of cards)
- Data Board - two constructors EOBoard and SBoard
    - EOBoard takes three arguments: Foundations [Columns] Reserves
    - SBoard takes three arguments Foundations [Columns] Stock
- Data GameType, EO or Spider

Experimental results for eight-off solitaire.

I started with just finding all moves and then picking the first one. Back then implementation of my findMoves function was that moves found from columns to reserves were first on the list. This caused clogging up reserves very quickly, and therefore running out of moves very quickly, and losing a game. Average score was about 2.5.

Then I realised that a better approach would be to put moves from reserves to column first (to always free reserves if possible) then moves from columns to columns (to create sequences of cards if possible) and then at last move card to reserves if there are no other moves. This improved my average score to ~4 per game.

Lastly I tried some more advanced heuristics approaches. I wanted to evaluate each board from findMoves, and then sort them according to their value. Pick the best one, and repeat. I was giving points for overall score (number of cards added to foundations), empty slots in reserves, and type of card in reserves. For lower cards I was giving more points, except for the King which in my opinion is valuable because we can later put in an empty column place. Implemented all that my average score didn't improve. It was even worse, ~3 points per game on average. I was surprised that this approach didn't have better results. Possibly the arbitrary coefficients were set wrong. By trial and error I tried to tune them up correctly but the results were never higher than on the pick first approach. That's why in my final implementation I stick to the previous version described above that gives a consistent score of around 4 points.

Experimental results for Spider solitaire.

I didn't manage to successfully implement advanced heuristics for the Spider game, my final implementation as for 8-off involves picking the first move, from column to column. And if there aren't any deal cards from stock. Average score is about ~ 0 points.