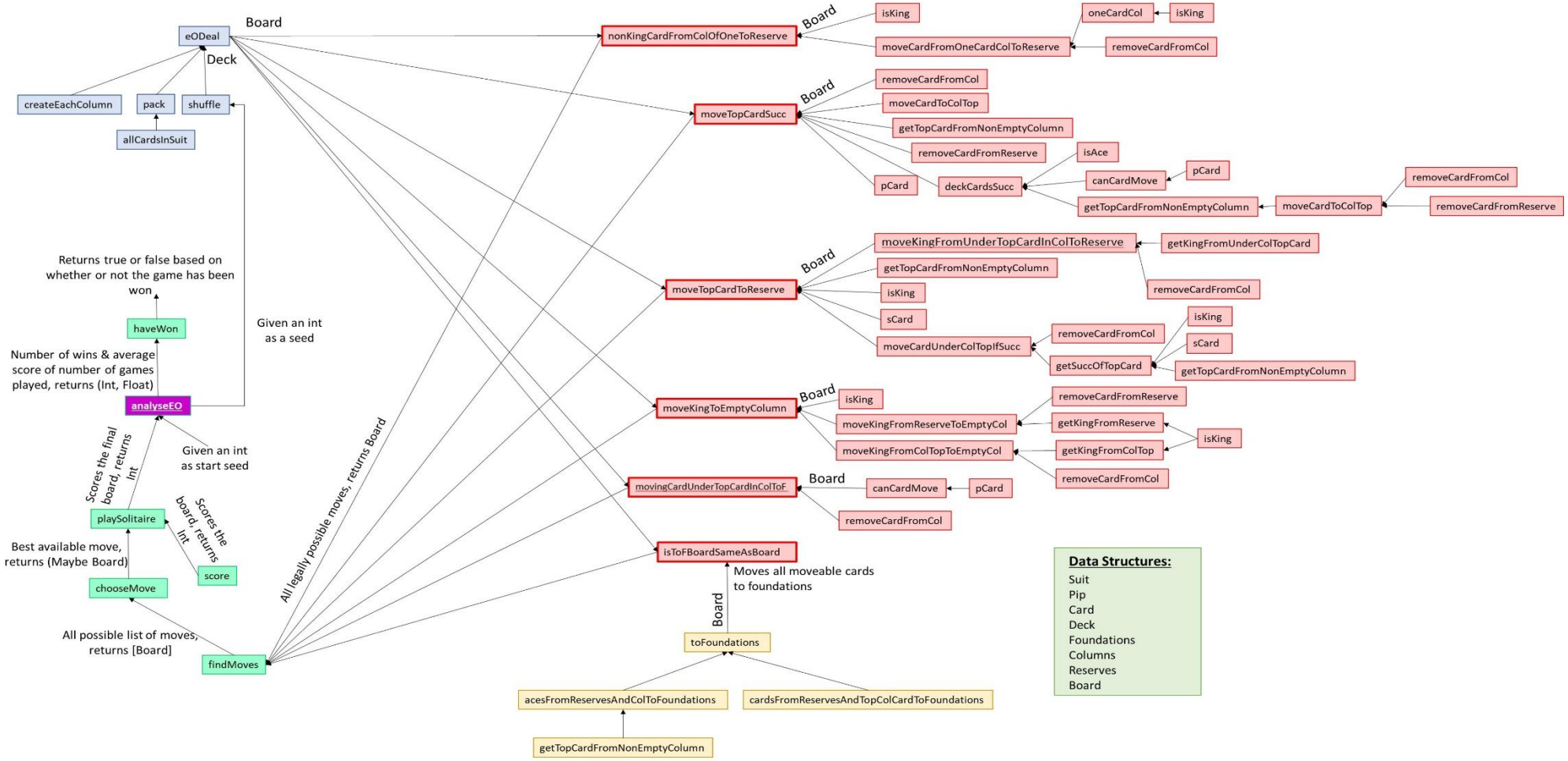
Samiha Fansur, aca20sf

Assignment report

Eight-Off Solitaire design

The below diagram shows the top-down design implementation for eight-off solitaire.



Samiha Fansur, aca20sf

My eight-off implementation uses the following data structures: Suit, Pip, Card, Deck, Foundations, Columns, Reserves, Board. The top-level function **analyseEO** calls various other functions to work. It uses the **findMoves** function which returns a list of possible boards after applying all legal moves by applying the functions written in within the textbox with a bold red outline (nonKingCardFromColOfOneToReserve, …, isToFBoardSameAsBoard). Each of these functions makes use of smaller helper functions.

Most function names describe what the function does.

A few important functions within my eight-off implementation:

findMoves :: Board -> [Board]

findMoves takes in an EOBoard as its input and returns a list of of board after applying the best the best legal move. The list is formed of moves which return a board that is different to the input board. If any legal moves do produce a board which is the same as the input board it will not be added to the list. The best board being at the top.

chooseMove :: Board -> Maybe Board

chooseMove takes in a board which is fed into the findMoves function to find all possible legal move. The function then chooses the topmost board result since it is a result of the best possible legal moves. If findMoves returns Nothing then chooseMove will also return Nothing

score :: Board -> Int

This scores the input board based on the number of cards in foundations

playSolitaire :: Board -> Int

playSolitaire plays a full game of eight off and returns a score of the final board before no more legal moves can be performed.

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

Function returns a tuple (number of wins, average score of games played). The number of games is specified in the 2nd input parameter. The 1st input parameter is the initial seed for the 1st game to be played.

Eight-Off Solitaire experimental results

After defining the sub functions for findMoves, I tried different combinations in which to perform those functions and chose the best combination based on the results produced. The results are shown in the table below:

f1 = isToFBoardSameAsBoard

f2 = movingCardUnderTopCardInCol

f3 = moveKingToEmptyColumn

f4 = moveTopCardToReserve

f5 = moveTopCardSucc

f6 = nonKingCardFromColOfOneToReserve

Samiha Fansur, aca20sf

Function combinations	Board given	Average score	Average score for 100 games (analyseEO)
f1 + f2 + f3 + f4 + f5 + f6	boardA	14	-
	eODeal 21	17	21.02
	eODeal 2678	17	17.08
	eODeal 2	5	22.06
	eODeal 5	52	22.14
f2 + f1 + f3 + f4 + f5 + f6	boardA	14	-
	eODeal 21	17	22.19
	eODeal 2678	17	16.3
	eODeal 2	5	22.91
	eODeal 5	52	22.99
f3 + f2 + f1 + f4 + f5 + f6	boardA	14	-
	eODeal 21	17	22.19
	eODeal 2678	17	16.3
	eODeal 2	5	22.91
	eODeal 5	52	22.99
f2 + f3 + f4 + f5 + f6 + f1	boardA	Infinite Loop	-
	eODeal 21	7	16.71
	eODeal 2678	52	17.56
	eODeal 2	5	16.61
	eODeal 5	52	17.26
f1 + f2 + f5 + f6 + f3 + f4	boardA	30	-
	eODeal 21	10	Infinite loop
	eODeal 2678	19	Infinite loop
	eODeal 2	5	Infinite loop
	eODeal 5	52	Infinite loop
f3 + f4 + f1 + f2 + f5 + f6	boardA	30	-
	eODeal 21	17	18.55
	eODeal 2678	8	16.2
	eODeal 2	5	19.36
	eODeal 5	5	19.26
f3 + f4 + f5 + f6 + f1 + f2	boardA	Infinite loop	-
	eODeal 21	7	Infinite loop
	eODeal 2678	8	Infinite loop
	eODeal 2	4	Infinite loop
	eODeal 5	52	Infinite loop
f5 + f6 + f3 + f4 + f1 + f2	boardA	Infinite loop	-
	eODeal 21	Infinite loop	Infinite loop
	eODeal 2678	3	Infinite loop
	eODeal 2	4	Infinite loop
	eODeal 5	Infinite loop	Infinite loop
f5 + f6 + f1 + f2 + f3 + f4	boardA	Infinite loop	-
	eODeal 21	Infinite loop	18.55
	eODeal 2678	19	16.2
	eODeal 2	Infinite loop	19.36
	eODeal 5	Infinite loop	19.26

As shown by the table of results the combination f1 + f2 + f5 + f6 + f3 + f4 or f2 + f1 + f3 + f4 + f5 + f6 produced the best average score per game as well as the best average score for 100 games, hence I used that combination of functions in findMoves. If I applied f1 after f2, f3, f4, f5 and f6 the scores remained the same as when I applied f1 after f3 or after f6, thus showing that it doesn't have much of an impact on the function. The same process was applied to other function, and hence by testing various other permutations the most efficient one is being used within the code; f2 + f1 + f3 + f4 + f5 + f6.

Infinite loop is mostly caused by the same board being produced over and over again after applying different functions.

Spider implementation

I think the most efficient way to play Spider solitaire would be to only move cards of they are the same suit on top of a predecessor.

This would require the use of the following functions:

flipTopFaceDownCard :: Card -> Card

This would be used to flip a face down card to face up if it was at the head of a column

isCardSameSuit :: Card -> Card -> Bool

This function would compare two face up cards and return true if they were the same suit, else false.

moveCardFromColToCol :: Board -> Board

This would call the isCardSameSuit function and the flipTopFaceDownCard. If the card can be moved onto its predecessor in another column, then it should update the 1st column by removing the card moved and adding it to the 2nd column. It should then check if the head of column 1 is a face down card, if yes then should flip it to face up.

dealCards:: Board -> Board

If no more cards can be moved between columns and stock is not empty, then the function should deal 10 cards and place 1 card on top of each column face up.

ToFoundationsS :: Board -> Board

This would be applied after no more legal moves can be made on the board, so it moves all moveable cards to foundations.

In this implementation of solitaire, findMoves, chooseMove and analyses would mostly work the same way and produce a score or a tuple of number of wins and average score of games played