COM2108 Assignment Report*

Jordan Pownall

December 8, 2021

1 Introduction

This is a report based on my COM2108 module assignment at the University of Sheffield. The objective was to design a program to play the game Eight-Off Solitaire using the functional programming language 'Haskell'.

Throughout this report I will be explaining the design of the program and assessing my results using my top-level function for eight-off solitaire, analyseEO.

2 Design

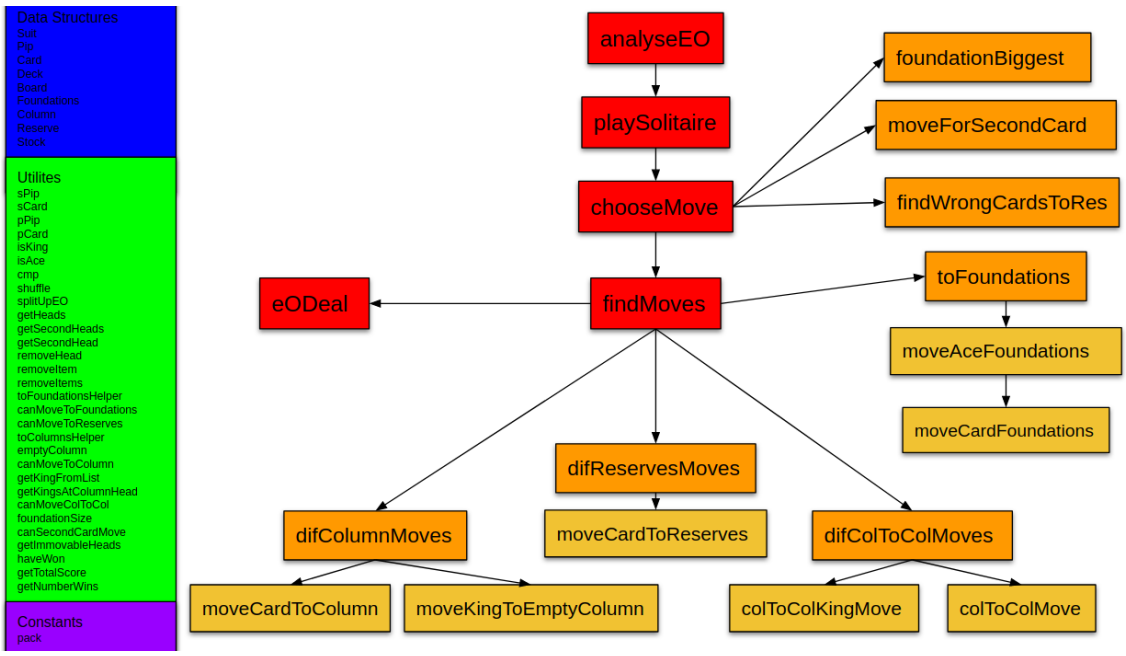

Figure 1: Top down design for my Eight-Off Solitaire program.

*https://gitlab.com/jord$_p$ownall/com2108 – *assignment*

3 Functions

3.1 eODeal

```
eODeal :: Int -> Board
eODeal n = EOBoard (foundation, column, reserve) where
    shuffledDeck = shuffle n                    --shuffle cards using seed n
    foundation = []                             --initialises empty array for foundation
    reserve = take 4 shuffledDeck               --takes the 4 first random cards from shuffled deck for reserve list
    column = splitUpEO (drop 4 shuffledDeck)      --splits the remaining cards into 8 equal lists for tableau
```

Figure 2: Code implementation for eODeal.

```
*Main> print (eODeal 123)
Foundations:
[]
Columns:
[[(Clubs,Ace),(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,E
ight)],[(Hearts,Three),(Diamonds,Ace),(Clubs,Ten),(Diamonds,Jack),(Spades,Nine),
(Hearts,Five)],[(Hearts,Seven),(Spades,Queen),(Clubs,Nine),(Clubs,Three),(Hearts
,King),(Clubs,Eight)],[(Hearts,Four),(Clubs,Two),(Spades,Four),(Diamonds,Three),
(Diamonds,Two),(Diamonds,Seven)],[(Diamonds,Nine),(Clubs,Five),(Spades,King),(He
arts,Six),(Spades,Ten),(Clubs,Jack)],[(Hearts,Queen),(Clubs,Four),(Spades,Eight)
,(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamonds,Four),(Spades,Ace),(Clu
bs,Seven),(Hearts,Ten),(Hearts,Nine),(Diamonds,Five)],[(Hearts,Jack),(Spades,Thr
ee),(Diamonds,Six),(Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Spades,Jack),(Hearts,Eight),(Spades,Six),(Hearts,Two)]
```

Figure 3: Printing the output of eODeal.

Before any of the other functions can be used a Board needs to be dealt. The method has one parameter, and this is the seed used for shuffling the deck before separating it out into its foundation, column and reserve.

For Eight-Off Solitaire, the game starts with an empty foundation, 4 cards in the reserve and 8 columns with 6 cards each. These columns form the tableau.

This function takes the first 4 cards from the shuffled deck, and then uses the splitUpEO utility to split up the rest of the cards into equal piles of 6.

3.2 findMoves

```
findMoves :: Board -> [Board]
findMoves (EOBoard board@(foundation,column,reserves))
    | null (removeItem (EOBoard board) (map toFoundations allMoves)) = []
    | otherwise = removeItem (EOBoard board) (map toFoundations allMoves)      -- all the different moves, with toFoundations applied on them all
    where
        allMoves = [EOBoard board] ++ difColtoColMoves (EOBoard board) ++ difColumnMoves (EOBoard board) ++ difReservesMoves (EOBoard board)
```

Figure 4: Code implementation for findMoves.

```
*Main> print (findMoves (eODeal 123))
[Foundations:
[[(Clubs,Ace)]]
Columns:
[[(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,Eight)],[(Hearts,Three),(Diamonds,Ace),(Clubs,T
bs,Three),(Hearts,King),(Clubs,Eight)],[(Hearts,Four),(Clubs,Two),(Spades,Four),(Diamonds,Three),(Diamonds,Two),(D
)],[(Hearts,Queen),(Clubs,Four),(Spades,Eight),(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamonds,Four),(Spad
Diamonds,Six),(Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Spades,Jack),(Hearts,Eight),(Spades,Six),(Hearts,Two)]
,Foundations:
[[(Clubs,Ace)],[(Diamonds,Ace)]]
Columns:
[[(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,Eight)],[(Clubs,Ten),(Diamonds,Jack),(Spades,N
,Eight)],[(Hearts,Three),(Hearts,Four),(Clubs,Two),(Spades,Four),(Diamonds,Three),(Diamonds,Two),(Diamonds,Seven)
en),(Clubs,Four),(Spades,Eight),(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamonds,Four),(Spades,Ace),(Clubs
Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Spades,Jack),(Hearts,Eight),(Spades,Six),(Hearts,Two)]
,Foundations:
[[(Clubs,Ace)]]
Columns:
[[(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,Eight)],[(Hearts,Three),(Diamonds,Ace),(Clubs,T
bs,Three),(Hearts,King),(Clubs,Eight)],[(Hearts,Four),(Clubs,Two),(Spades,Four),(Diamonds,Three),(Diamonds,Two),(D
)],[(Hearts,Jack),(Hearts,Queen),(Clubs,Four),(Spades,Eight),(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamon
Diamonds,Six),(Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Spades,Jack),(Hearts,Eight),(Spades,Six),(Hearts,Two)]
,Foundations:
[[(Clubs,Ace)]]
Columns:
[[(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,Eight)],[(Hearts,Two),(Hearts,Three),(Diamonds
bs,Nine),(Clubs,Three),(Hearts,King),(Clubs,Eight)],[(Hearts,Four),(Clubs,Two),(Spades,Four),(Diamonds,Three),(Di
),(Clubs,Jack)],[(Hearts,Queen),(Clubs,Four),(Spades,Eight),(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamond
ades,Three),(Diamonds,Six),(Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Spades,Jack),(Hearts,Eight),(Spades,Six)]
,Foundations:
[[(Clubs,Ace)]]
Columns:
[[(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,Eight)],[(Hearts,Three),(Diamonds,Ace),(Clubs,T
bs,Three),(Hearts,King),(Clubs,Eight)],[(Hearts,Four),(Clubs,Two),(Spades,Four),(Diamonds,Three),(Diamonds,Two),(D
)],[(Hearts,Queen),(Clubs,Four),(Spades,Eight),(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamonds,Four),(Spad
Diamonds,Six),(Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Spades,Jack),(Hearts,Eight),(Spades,Six),(Hearts,Two)]
,Foundations:
```

Figure 5: The partial output of printing findMoves.

findMoves is a method that takes an initial board and returns all of the possible moves that can be made from that board. findMoves uses a combination of 3 methods to find these moves: difColToColMoves, difColumnMoves and difReservesMoves.

These 3 moves are concatenated in this order which ranks the boards in an order of which I think is the optimal move to make. First being column to column moves, then reserves to column, then column to reserve, as it is a good idea to keep the reserves as empty as possible.

Once all of these boards are in order, toFoundations is then applied onto each of these moves. This is to automatically move cards to the foundations at any time possible.

3.3 chooseMove

```haskell
chooseMove :: Board -> Maybe Board
chooseMove (EOBoard board@(foundation, column, reserve))
    | null allMoves = Nothing
    | canSecondCardMove (EOBoard board) && moveForSecondCard (EOBoard board) /= EOBoard board = Just (moveForSecondCard (EOBoard board))
    | bestAllMoves == EOBoard board = Nothing                                   -- if there is no move without toRemove, return nothing
    | otherwise = Just bestAllMoves                                             -- choose best moved, ranked from findMoves, with the b
    where
        allMoves = removeItem (EOBoard board) (findMoves (EOBoard board))       -- all of the moves
        toRemove = findWrongCardsToRes (EOBoard board)                          -- the moves to remove that are from columns to reserve
        bestAllMoves = if null (removeItems toRemove allMoves) then EOBoard board else foundationBiggest (removeItems toRemove allMoves)
```

Figure 6: Code implementation for chooseMove.

```
*Main> print (chooseMove (eODeal 123))
Just Foundations:
[[(Diamonds,Ace)]]
Columns:
[[(Clubs,Ace),(Diamonds,Ten),(Clubs,Queen),(Spades,Two),(Clubs,King),(Diamonds,Eight)],[(
Clubs,Ten),(Diamonds,Jack),(Spades,Nine),(Hearts,Five)],[(Hearts,Seven),(Spades,Queen),(C
lubs,Nine),(Clubs,Three),(Hearts,King),(Clubs,Eight)],[(Hearts,Four),(Clubs,Two),(Spades,
Four),(Diamonds,Three),(Diamonds,Two),(Diamonds,Seven)],[(Diamonds,Nine),(Clubs,Five),(Sp
ades,King),(Hearts,Six),(Spades,Ten),(Clubs,Jack)],[(Hearts,Queen),(Clubs,Four),(Spades,E
ight),(Spades,Five),(Hearts,Ace),(Diamonds,Queen)],[(Diamonds,Four),(Spades,Ace),(Clubs,S
even),(Hearts,Ten),(Hearts,Nine),(Diamonds,Five)],[(Hearts,Jack),(Spades,Three),(Diamonds
,Six),(Spades,Seven),(Diamonds,King),(Clubs,Six)]]
Reserve:
[(Hearts,Three),(Spades,Jack),(Hearts,Eight),(Spades,Six),(Hearts,Two)]
```

Figure 7: The output of printing chooseMove.

This method takes a board as a parameter, and returns Nothing if there are no more moves or a board it has chosen for its next move from this board.

First, this method checks if there is a card behind the head of any columns that can move to the foundations. If there is, it chooses the head card to either another column or the reserves, then moves that card to foundations.

If this is not the case, it modifies the boards from findMoves by removing the moves from the column to the reserves if they are in a good position, i.e. in front of their successor on a column or if it is a king on an empty column. From the remaining selection, it simplifies it down to the cards what have the joint highest number of cards in foundations. Since these cards are already in order from findMoves, it chooses the head of this filtered list.

3.4 playSolitaire

```haskell
playSolitaire :: Board -> Int
playSolitaire (EOBoard board@(foundation, column, reserve))
    | haveWon (EOBoard board) = 52
    | isNothing (chooseMove (EOBoard board)) = foundationSize (EOBoard board)
    | otherwise = playSolitaire (fromJust (chooseMove (EOBoard board)))
```

Figure 8: Code implementation for playSolitaire.

```
*Main> print (playSolitaire (eODeal 123))
4
```

Figure 9: The output of printing playSolitaire.

This is a scoring method. It takes a board and recursively calls chooseMove on an eOBoard until the program cannot find any more moves. This is indicated by chooseMove returning nothing.

Once chooseMove indicates the game is over, playSolitaire evaluates the score of the game by counting the number of cards in the foundations and returns this figure.

3.5 analyseEO

```
analyseEO :: Int -> Int -> (Int,Int)
analyseEO _ 0 = error "No amount of games specified"
analyseEO seed numGames = (wins, averageScore)
    where
        wins = getNumberWins seed numGames
        totalScore = getTotalScore seed numGames
        averageScore = totalScore `div` numGames
```

Figure 10: Code implementation for analyseEO.

```
*Main> print (analyseEO 123 10)
(0,6)
```

Figure 11: The output of printing analyseEO.

This is a function to analyse the performance of my program. It takes 2 integers, one being the seed used for shuffling and one being the number of games to play.

The function recursively calls playSolitaire until no more games need to be played and records the scores to return a tuple containing the number of wins and the average score. In figure 11, the number of wins was 0 and the average score was 6 in 10 games.

4 Results

Test Number	Initial Seed Value	Number of Games	Number of Wins	Average Score
1	1234	1	0	17
2	1235	1	0	2
3	1233	1	0	0
4	1000	1000	0	6
5	1000	100	0	8
6	1100	100	0	7
7	1200	100	0	6
8	1300	100	0	7

Table 1: The result of using different inputs in analyseEO.

5 Discussion

The following two figures show the initial and resulting board for test 1:

```
*Main> print (eODeal 1234)
Foundations:
[]
Columns:
[[(Clubs,Ten),(Diamonds,Queen),(Hearts,Ace),(Hearts,Eight),(Clubs,Ace),(Hearts,T
wo)],[(Clubs,Queen),(Hearts,Seven),(Diamonds,Ten),(Diamonds,Five),(Spades,Ten),(
Spades,Four)],[(Clubs,Five),(Clubs,Four),(Hearts,Three),(Hearts,Nine),(Clubs,Jac
k),(Spades,Eight)],[(Clubs,Eight),(Spades,Six),(Spades,Five),(Diamonds,Seven),(H
earts,Four),(Clubs,Six)],[(Clubs,Nine),(Diamonds,Three),(Spades,Nine),(Hearts,Te
n),(Spades,King),(Clubs,Two)],[(Spades,Three),(Hearts,King),(Spades,Two),(Spades
,Jack),(Diamonds,Nine),(Diamonds,Four)],[(Clubs,Seven),(Spades,Seven),(Hearts,Qu
een),(Diamonds,Jack),(Diamonds,Ace),(Spades,Queen)],[(Hearts,Jack),(Clubs,King),
(Spades,Ace),(Clubs,Three),(Diamonds,Eight),(Diamonds,Six)]]
Reserve:
[(Hearts,Six),(Hearts,Five),(Diamonds,King),(Diamonds,Two)]
```

Figure 12: The initial dealt board of test 1.

```
Move 37: Foundations:
[[(Spades,Seven),(Spades,Six),(Spades,Five),(Spades,Four),(Spades,Three),(Spades
,Two),(Spades,Ace)],[(Clubs,Ace)],[(Hearts,Nine),(Hearts,Eight),(Hearts,Seven),(
Hearts,Six),(Hearts,Five),(Hearts,Four),(Hearts,Three),(Hearts,Two),(Hearts,Ace)
]]
Columns:
[[(Diamonds,Queen),(Diamonds,King)],[(Hearts,King)],[(Clubs,Ten),(Clubs,Jack),(S
pades,Eight)],[(Clubs,Queen),(Clubs,King)],[(Clubs,Three),(Clubs,Four),(Clubs,Fi
ve),(Clubs,Six),(Clubs,Seven),(Clubs,Eight),(Clubs,Nine),(Diamonds,Three),(Spade
s,Nine),(Hearts,Ten),(Spades,King),(Clubs,Two)],[(Spades,Ten),(Spades,Jack),(Dia
monds,Nine),(Diamonds,Four)],[(Hearts,Jack),(Hearts,Queen),(Diamonds,Jack),(Diam
onds,Ace),(Spades,Queen)],[(Diamonds,Seven),(Diamonds,Eight),(Diamonds,Six)]]
Reserve:
[(Clubs,Six),(Diamonds,Five),(Diamonds,Ten),(Diamonds,Two)]
Score: 17
and if I'd used playSolitaire, I would get score: 17
```

Figure 13: The resulting board of test 1.

My program dealt very well with this particular dealt board in comparison to other boards. This board shows how a board dealt with the ability to shuffle cards around easily produces a higher score. This reflects a solitaire player at a basic level.

The following two figures show the initial and resulting board for test 2:

```
Foundations:
[]
Columns:
[[(Hearts,Eight),(Diamonds,Nine),(Hearts,Jack),(Diamonds,Seven),(Spades,Queen),(
Spades,Two)],[(Spades,Ace),(Spades,Jack),(Spades,Ten),(Clubs,Seven),(Hearts,Quee
n),(Diamonds,Three)],[(Hearts,Ten),(Hearts,Five),(Diamonds,Four),(Spades,Six),(S
pades,Eight),(Clubs,Two)],[(Hearts,Two),(Clubs,Six),(Clubs,Four),(Diamonds,Jack)
,(Hearts,Three),(Clubs,Eight)],[(Diamonds,Five),(Clubs,Jack),(Spades,Three),(Clu
bs,Queen),(Diamonds,Ace),(Hearts,Six)],[(Spades,Four),(Clubs,Three),(Diamonds,Si
x),(Hearts,Four),(Spades,Nine),(Diamonds,King)],[(Diamonds,Two),(Clubs,Ten),(Spa
des,Seven),(Hearts,King),(Spades,Five),(Clubs,Nine)],[(Spades,King),(Diamonds,Te
n),(Clubs,Five),(Diamonds,Queen),(Hearts,Ace),(Hearts,Seven)]]
Reserve:
[(Diamonds,Eight),(Clubs,Ace),(Hearts,Nine),(Clubs,King)]
```

Figure 14: The initial dealt board of test 2.

```
Move 12: Foundations:
[[(Spades,Ace)],[(Clubs,Ace)]]
Columns:
[[(Diamonds,Eight),(Diamonds,Nine),(Hearts,Jack),(Diamonds,Seven),(Spades,Queen)
,(Spades,Two)],[(Spades,King)],[(Hearts,Eight),(Hearts,Nine),(Hearts,Ten),(Heart
s,Five),(Diamonds,Four),(Spades,Six),(Spades,Eight),(Clubs,Two)],[(Clubs,Four),(
Diamonds,Jack),(Hearts,Three),(Clubs,Eight)],[(Diamonds,Five),(Clubs,Jack),(Spad
es,Three),(Clubs,Queen),(Diamonds,Ace),(Hearts,Six)],[(Spades,Four),(Clubs,Three
),(Diamonds,Six),(Hearts,Four),(Spades,Nine),(Diamonds,King)],[(Diamonds,Two),(C
lubs,Ten),(Spades,Seven),(Hearts,King),(Spades,Five),(Clubs,Nine)],[(Diamonds,Te
n),(Clubs,Five),(Diamonds,Queen),(Hearts,Ace),(Hearts,Seven)]]
Reserve:
[(Clubs,Six),(Hearts,Two),(Diamonds,Three),(Hearts,Queen),(Clubs,Seven),(Spades,
Ten),(Spades,Jack),(Clubs,King)]
Score: 2
and if I'd used playSolitaire, I would get score: 2
```

Figure 15: The resulting board of test 2.

As this board shows, although it mustered 12 moves, it only returned a score of 2. This suggests my programs random approach to shuffling cards around as much as it can until it finds a solution is not ideal for a lot of boards. This is why with boards like this a lower score is returned, as this kind of board needs a more calculated approach of an experienced player.

As my results show, the average score acquired from 1000 games in test 4 is 6, which is relatively low.

6 Summary

In conclusion, my program for playing Eight-Off Solitaire uses an approach that is only calculated up to the first few cards, and then uses a more reckless searching approach which is moving things around in the columns to their successor cards hoping a move possible to the foundations will be uncovered. This therefore produces an average score for some boards dealt to suit this technique.

When the board is dealt in a more difficult way which might require more calculated moves, my program fails to reach an adequate score.

It appears that my program reflects a solitaire player of a very basic skill level.