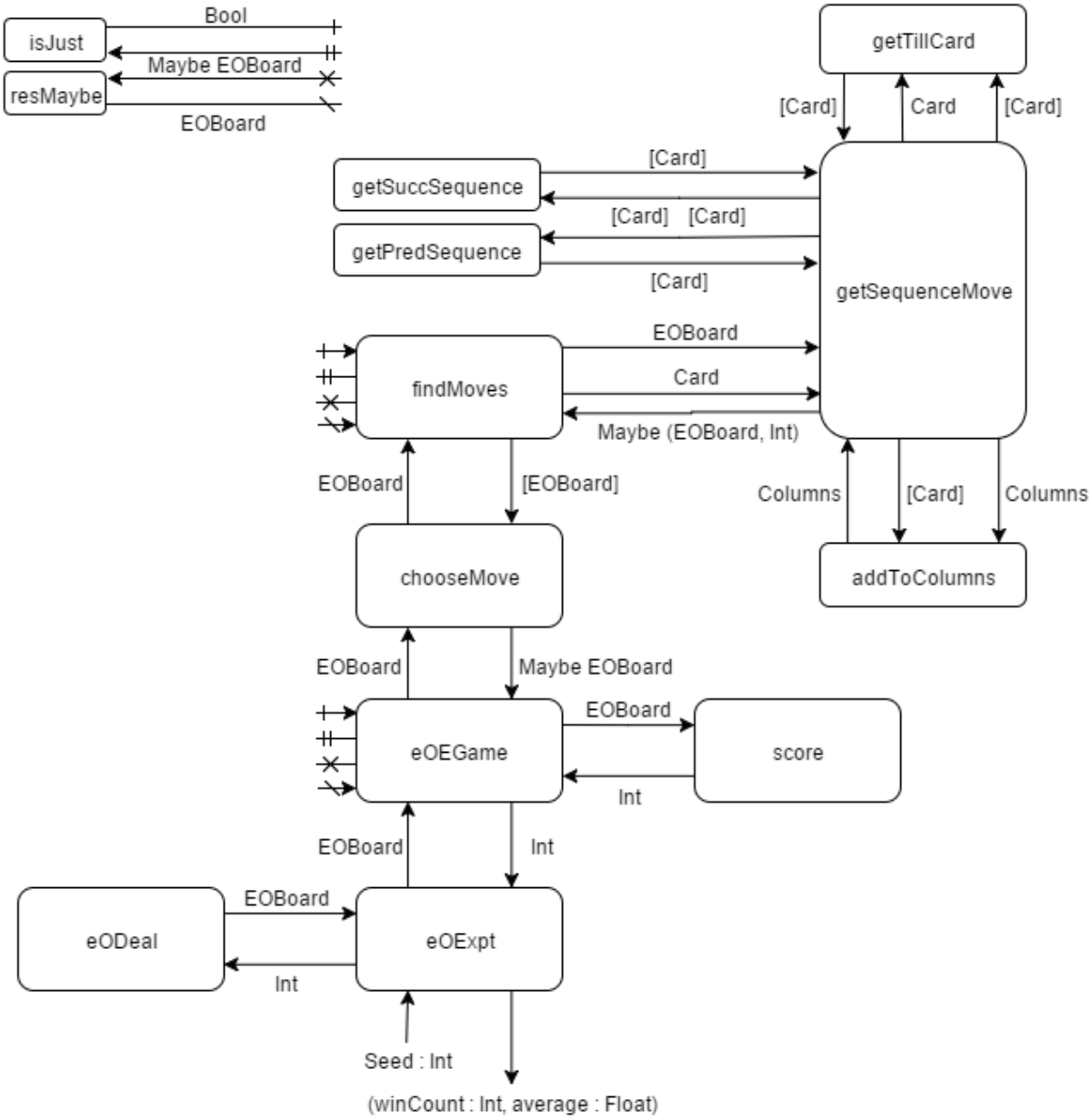
Blayze Millward

EightOff AI module implementation - COM2001 Assignment 3

Design Diagram



findMoves

Here, I test to make sure that appropriate moves are found:

Input	*Expected Result*	*Actual Result*	*Description*
[],[[(Three, Diamonds), (Nine, Hearts)], [(Jack, Clubs)]], [(Two, Diamonds)]	One move of the Two of Diamonds moved to the first column	[([],[[(Two,Diamonds), (Three,Diamonds), (Nine,Hearts)], [(Jack,Clubs)]], [])]	findMoves can successfully move a card from the reserves into the correct column

[],[[(Four, Diamonds)], [(Three, Diamonds)]], []	One move of the Three of Diamonds to the Four of Diamonds' column	[([], [[(Three,Diamonds), (Four,Diamonds)]], [])]	findMoves can successfully move a card between columns
[],[[(Four, Diamonds)], [(Two, Diamonds), (Three, Diamonds)]],[]	One move generated to place the third column into the second	[([],[[(Two,Diamonds), (Three,Diamonds), (Four,Diamonds)]],[])]	findMoves successfully moves one column's contents to another where there are enough reserve slots free
[],[[(Four, Diamonds)], [(Ace, Diamonds), (Two, Diamonds), (Three, Diamonds)]], [(Five, Hearts), (Five, Hearts), (Five, Hearts), (Five, Hearts), (Five, Hearts), (Five, Hearts), (Five, Hearts)]	No moves will be made as there are not enough reserve slots free to make the column shift	[]	findMoves successfully eliminates moves that cannot be performed when there are not enough slots left in the reserves
[],[[(Four, Diamonds)], [(Nine,Diamonds), (Three, Diamonds), (Two, Diamonds)]],[]	The Nine of Diamonds is moved out of the way	[([],[[(Four,Diamonds)], [(Three,Diamonds), (Two,Diamonds)]], [(Nine,Diamonds)])]	findMoves can successfully move a card covering a wanted card into the reserves
[],[[(Four, Diamonds)], [(Eight, Diamonds), (Nine,Diamonds), (Three, Diamonds), (Two, Diamonds)]],[]	The Eight and Nine of Diamonds will be moved out of the way	[([],[[(Four,Diamonds)], [(Three,Diamonds), (Two,Diamonds)]], [(Eight,Diamonds), (Nine,Diamonds)])]	findMoves can successfully perform a double-move to remove entire sequences that are covering needed cards
[],[[(Four, Diamonds)], [(Eight, Diamonds), (Nine,Diamonds), (Three, Diamonds), (Two, Diamonds)]], [(Five, Hearts), (Five, Hearts), (Five, Hearts), (Five, Hearts),	No moves will be available as there is no place to put every element of the blocking sequence into reserves	[]	findMoves successfully eliminates any uncovering moves that cannot be performed due to lack of space in reserves

(Five, Hearts), (Five, Hearts), (Five, Hearts)]			
[], [[(King, Spades), (Ace, Spades)]], []	A new column should be made for the King of Spades	[([],[[(King,Spades)], [(Ace,Spades)]],[])]	findMoves successfully moves kings into their own columns
[], [[(King, Spades), (Ace, Spades)]], [(King, Hearts)]	Two moves; one creating a new column for each king. The King of Hearts move should be first	[([],[[(King,Hearts)], [(King,Spades), (Ace,Spades)]],[]), ([],[[(King,Spades)], [(Ace,Spades)]], [(King,Hearts)])]	findMoves successfully priorities moving a card from the reserves over inter-column moves

The function acts as expected.

chooseMove

As chooseMove simply takes the head of the list generated by findMoves then runs toFoundations on it, testing it is trivial:

Input	*Expected Result*	*Actual Result*	*Description*
[],[],[]	Nothing	Nothing	When no moves are returned, chooseMove returns Nothing
[],[[(Two, Spades)]], [(Ace, Spades)]	The first move that can be made; moving the ace onto the two, then with toFoundations applied	Just ([(Two,Spades)],[[]],[])	chooseMove succesffully chooses the top move from a ranked set of moves and then applies toFoundations to it.

eOGame

In order to test eOGame, I first show that it outputs the correct data, then I run it 10000 times, each with different seeds, to show that eOGame runs successfully on a sample of random seeds, indicating that it will not fail on any given random seed:

Input	*Expected Result*	*Actual Result*	*Description*
eODeal 0	A score between 0 and 52	5	eOGame seems to run okay
length $ filter (\n ->	10000	10000	eOGame returns a

n>=0 && n<=52) [eOGame (eODeal x) | x <- [1..10000]]			score between 0 and 52 on 10000 seed samples

eOExpt

In order to test eOExpt, I run similar tests to those in eOGame, first showing that it returns the appropriate information, then showing that it consistently runs by running it 100 times without problem.

Input	*Expected Result*	*Actual Result*	*Description*
0	A count for the number of wins, and an average score	(39,24.47)	eOExpt seems to run okay
[eOExpt (x*100)| x <- [1..100]]	The system to perform the tests one after another without hanging or taking too long.	A list of 100 different results formed in 2 minutes and 27 seconds	eOExpt returns appropriately on 100 different seed samples

Experimentation

My code has gone through numerous iterations. My original design was limited to only making single-card moves. This limitation meant that it played extremely badly, rarely ever winning a game and almost always getting caught in an infinite loop, passing cards between the reserves and the columns. It became apparent that in order to uncover hidden cards that it had identified as desirable, it was breaking sequences formed at the heads of the columns those cards resided it, meaning that on the next iteration of move selection, it would simply re-construct that sequence by replacing the detached sequence head.

My first attempt to work around this problem was to try to implement some kind of look-ahead, whereby if, for each suggested move, future moves returned to the same point, it should not be done. However, this -- along with a history for a look-back -- was difficult to implement in my system.

My second method of working around this problem was to implement a function ‘properlyPlaced’ that looked at the card at the head of the column that contained a hidden card, and if the head was part of a sequence, simply ignored the move.

Note: In older iterations of my code, eODeal took a shuffled deck, rather than an int

Input	*Result*	*Observation*
[eOGame (eODeal (shuffle x)) | x <- [0..100]]	… 2,5,5,6,2,2,11,3,11, 3,0,3,6,5,3,0,5,1,2,18,	The system no longer loops, however, it scores very badly.

	3,1,9,5,5,2,8,2,2,0,17, 0,7 ..	

After that, my third approach saw better success, in this I realised that if I moved away from single-card based moves altogether, and limited my choice of moves to only those that move sequences, I would avoid any loops altogether. So in my third system, I only perform sequence-wise moves or single-card moves when a sequence is of length 1.

Input	*Result*	*Observation*
length $ filter (== 52) [eOGame (eODeal (shuffle x)) | x <- [0..100]]	6	The system no longer loops, and now scores above 1% win rate

I then implemented a check to look for 'bonus cards' - cards that, if chosen, will lead to a more successful toFoundations call

Input	*Result*	*Observation*
length $ filter (== 52) [eOGame (eODeal x) | x <- [0..100]]	14	The system no longer loops, and now doubles the win-rate

My final alteration was to change the way un-covering of desired cards was handled; up until this point, only one card was ever moved out of the way at a time, however, I altered this system to remove sequences of cards under the desired card and put them into the reserves, this increased it's performance significantly. I also re-weighted king moves to be less important, which seemed to add a few percent on:

Input	*Result*	*Observation*
eOExpt 0	(39,24.47)	The system now works at around 40% success