GRADING ASSIGNMENT (PART 2): SOLITAIRE

This is part 2 of your grading assignment. You should finish part 1 before you move on to this work: the datatypes and functions defined in part 1 are essential building blocks for part 2.

In this part, we are looking at code to play a game to completion, and analyse performance, first for eight-off solitaire, and then for an alternative solitaire game (such as Spider solitaire, as introduced in Part 1 (recommended), or the more traditional game of Klondike).

YOUR WRITTEN REPORT

You will make two submissions for this assignment: the code you have written (as a single .hs file) and a written report. In the written report you must include:

- A clear design for your code for eight-off solitaire. Your top-level function for eight-off solitaire is analyseEO (Step 4). Some of the other functions that you must write are given in earlier steps, but it is **expected** that you will need others. Remember: design top-down, implement bottom-up – you are strongly recommended to complete the design **before** you start implementing anything.
- A summary of your experimental results for eight-off solitaire. Did you try different strategies for the function to find the next move (Step 2)? If so, what were the results, and what do you think the best strategy is?
- If you attempt Step 5, you must also include:
    - A description of the game you are looking at (unless it is Spider Solitaire, as described in Part 1 of the assignment).
    - A design for your code for that form of solitaire
    - A summary of your experimental results for that form of solitaire.

YOUR CODE

STEP 1: A FUNCTION TO FIND **ALL POSSIBLE MOVES** FOR EIGHT-OFF SOLITAIRE

You must define a function

findMoves :: Board -> [Board]

This function must take the current board state and return a list of all the possible board states **after a single move.** The last step in any move should be a call to **toFoundations** (from Part 1).

STEP 2: A FUNCTION TO CHOOSE THE **NEXT MOVE**

You must define a function

chooseMove :: Board -> Maybe Board

This functions normally returns **Just Board**, but will return **Nothing** if there is no legal move from the given position (that is, **findMoves** returns an empty list and so the game is lost). For a win the final game will return a Board with empty columns and reserve.

The simplest choice is to take the first move returned by **findMoves** – but this is unlikely to be very successful. You will need to experiment to try to find successful strategies. Some tips include:

- It's good to move Aces to the Foundations if you can.
- It's good to move a King to a vacant column if you can.
- Try not to use up the Reserve. It's best to keep a minimum of 3 spare cells.
- Therefore, don't make a move from Column to Reserve unless you have to.
- If you do need to make a Column-to-Reserve move, then look for one which will allow you to make a move from Reserve to Column next time, thus recovering the reserve space.

Step 3: A function to play a game of Eight-Off solitaire

First you should define a function

haveWon :: Board -> Bool

This function should take a board and return True if it is in a winning state for eight-off solitaire (no cards in the reserve or columns), and False otherwise.

Next, you must define a function

playSolitaire :: Board -> Int

This function takes an initial Board as its argument and uses chooseEOMove to play the game to completion. The return value is the score, which is calculated by how many cards have been moved to the foundations – a successful game, in which all cards are moved to the foundations, will score 52.

Step 4: A function to analyse performance

To analyse the performance of your gameplay, you should define a function **analyseEO.** This function should take two arguments. The first should be an integer to seed the random number generator (for shuffling). The second should be the number of games to play. You will need to play that number of games, each with a different initial board, and return the number of wins (games in which all cards are moved to the foundations) and average score.

Step 5: A different game

By the end of step 4, you should have a good idea of how well you can play eight-off solitaire. For the extension work, you must attempt to do the same for an alternative game of solitaire. The recommended choice is spider solitaire, for which you did some preliminary work in Part 1, but if you wish, you can choose a different game of solitaire. If you choose a different game, you must explain its rules in your report.

For your chosen game, you should write equivalent cases of the functions as for eight-off solitaire above, and analyse your performance on this game as you did for eight-off solitaire. For all the functions above, except for **analyseEO**, this means writing a version of the function that works with a **SBoard** instance of a **Board** instead of an **EOBoard**. You will need to define a separate function for **analyseSpider.** (If you choose a form of solitaire other than Spider, use the appropriate Board constructor and name for your analyse function.)

DEMONSTRATING YOUR CODE

First and foremost, your code must compile and run without error. Code that does not compile will receive 0 marks for functionality and style. If you have code that you think is *almost* working, leave it out – better to get some marks for the parts that **are** working.

On Monday 15th November, a file called template.hs will be placed in the assignment area on Blackboard. It will provide the code that you need to demonstrate your work, and instructions (as comments) on how to adapt it for your code. Please read the instructions **carefully.** If you submit code that does not work because you did not follow the instructions, it will severely impact your mark.

The template and instructions will assume that you have implemented your functions as instructed (for Parts 1 and 2).

ASSESSMENT

Remember: you will **pass** this module by passing the two threshold quizzes – and despite the poor performance on the first attempt at threshold quiz 1, I expect that every one of you will pass. So before you even start your grading assignment, you have 40% for this module. Any marks you get for the grading assignment are added proportionately to this, so if you get 50% for the grading assignment, you will have 70% for the module. You must work hard to get a first!

Element	Weighting
Core functionality (data types and functions)	**20%**
Part 1	10%
Part 2 (up to and including Step 4)	10%
Written report (1000 words max up to an including Step 4, 1500 if including Step 5)	**50%**
Design of eight-off solitaire code	20%
Description of results and experimentation for eight-off solitaire	10%
Design of code for alternative solitaire	10%
Description of game, results and experimentation for alternative solitaire	10%
Style – including layout, choice of function and parameter names, logical presentation (incl. ordering) of function definitions, code comments. These marks will be awarded in proportion to the amount of work that you complete. If you only write one function, you will not score highly here no matter how well that one function is written.	**20%**
Extension functionality (Step 5)	**10%**

Your work must be submitted by 3pm Thursday 9th December 2021. You are encouraged to seek feedback as you develop your solution, asking the demonstrators or other staff. Bear in mind that there are many students in this module: if you all seek feedback in the week before submission, it will not be possible!

SUBMISSION

There are two submission points: one for your report, which must be submitted either as a Word document or a PDF (recommended), and one for your code, which must be submitted as a single Haskell source code file.

There will be separate clearly labelled submission points for the two parts.

There are no requirements about the names of the files.

INDIVIDUAL WORK

This assessment is intended to be individual work. Feel free to discuss ideas with your peers but **what you submit should be written by yourself.** You should have completed the unfair means tutorial when you were in first year, but if you need a refresher you can find it at the top of this page in the student handbook (along with further information about unfair means).
https://sites.google.com/sheffield.ac.uk/comughandbook/general-information/assessment/unfair-means

In previous years we have used similar assignments, and we know some students have published their solutions on the web. If you use their code, this is **not your work** and we will take disciplinary action, as set out in the handbook page linked above.

Please do not publish your solutions on the web.

NEED HELP?

The best place for help is in the lab classes – with online support available for those who cannot attend in person.

If you are seeking help, be prepared to explain the design of your functions. Sometimes the help you need will just be understanding what an error message means, or a small misunderstanding, but at other times you will be advised to reconsider the design – sometimes the difficulties arise because you are trying to solve the problem from the wrong angle.

You can also use the discussion board for general questions (but do not post too much detail about your solution – posts are moderated and those with too much detail will be rejected) or you can email Professor Green. If you email Prof. Green, remember:

> email: p.green@sheffield.ac.uk
> subject: COM2108
> body: **No screen shots.** Include text of code (all the code, so that it can be run) and/or error messages rather than screen shots.

As above, you are **encouraged** to seek feedback about your work, particularly in relation to design and style. We will not give out solutions but we want you to have every chance to learn and improve.

APPENDIX A: 8-OFF SOLITAIRE

Eight-off solitaire Is played with a normal deck of playing cards. The layout at the start of a game is below:

The object of the game is to move all the cards to the foundations. The board and rules are as follows:

Foundations (4 piles: complete these piles to win the game)

- Build up each foundation in suit from Ace to King (for example, a 2♥ can be played on an Ace♥).
- You must start a Foundation with an Ace

Tableau (8 columns, initially of 6 cards each)

- Build down in suit sequence (for example, a 10♠ can be played only on a Jack♠).
- The top card of each column (the **head**) is available for play to another column, the foundations or the cells.
- In addition, you can move groups of cards from the head of a column to the head of another column if they are in sequence and if there are enough free cells so that the cards could be moved individually. (e.g. if one column is 6♥, 7♥, Jack♣… another column is 8♥… and there is at least one space left in the reserve cells you can move the 6♥, 7♥, to leave Jack♣… and 6♥, 7♥, 8♥…)
- If you have <8 non-empty columns, and the head of one of these columns is a King or a King-sequence you can move this King or King-sequence in a new column provided you have sufficient space left in the reserve cells.
- You can move a King from the reserve to an empty column.

Cells (or Reserves; 8 cells)

- These are the "cells". These cells are storage locations for cards being played to the foundations and the tableau.
- The game starts with 4 filled cells.
- There is a maximum of 8 cells.
- Cards in the cells can be moved to the foundations and the tableau.
- Cells can only hold one card.

You should play a few games of 8-off to get the idea. You'll find several web sites where you can play online.

Note that we aren't going to do a simulation with graphics, so the order of the foundations, columns and reserve cells is unimportant.

Appendix B: Spider solitaire

Spider solitaire is played with *two* decks of playing cards (104 cards). The object of the game is to move all cards to the foundations. The board and rules are as follows:

Foundations (8 piles, complete these to win the game).

- As in eight-off solitaire, each pile has thirteen cards, from Ace to King, of the same suit.
- In spider solitaire, you can only move cards to the foundation when you have a **complete** set for that pile.

Tableau (10 columns)

- Initially, six cards are dealt to the first four piles and five to the remaining ones. Cards are dealt face down, with only the top card in each pile turned face up.
- The top card in each column is available for play to another column, it can be moved onto the successor card of *any* suit. (So 2♦ could be moved onto 3♠ or 3♦.)
- Where there are groups of cards in sequence of the same suit at the top of a column, these groups may be moved as one unit. (So 3♦2♦ could be moved in one go, but 3♠2♦ would require two moves.)
- When a pile is empty, you can fill the space with any card or group of cards.
- When you can make no further useful moves, you must ensure you have no empty columns (even if this means splitting a group), then deal the next 10 cards from the stock.

Stock

- Initially, this is the remaining 50 cards after the tableau has been dealt.
- You can deal the next 10 cards from the stock at any time, provided there are no empty columns.
- When there are no cards remaining in the stock and you can make no useful moves, the game is over.