The objective of this project is to practice and assess your understanding of functional programming and Haskell. You will write code to implement a logical deduction game.

The Game

Two players face each other, each with a complete standard deck of western playing cards (without jokers). One player will be the *answerer* and the other is the *guesser*. The answerer begins by selecting some number of cards from his or her deck without showing the guesser. These cards will form the *answer* for this game. The aim of the game is for the guesser to guess the answer.

Once the answerer has selected the answer, the guesser chooses the same number of cards from his or her deck to form the *guess* and shows them to the answerer. The answerer responds by telling the guesser these five numbers as *feedback* for the guess:

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

1. How many of the cards in the answer are also in the guess (*correct cards*).
2. How many cards in the answer have rank lower than the lowest rank in the guess (*lower ranks*). Ranks, in order from low to high, are 2–10, Jack, Queen, King, and Ace.
3. How many of the cards in the answer have the same rank as a card in the guess (*correct ranks*). For this, each card in the guess is only counted once. That is, if the answer has two queens and the guess has one, the correct ranks number would be 1, not 2. Likewise if there is one queen in the answer and two in the guess.
4. How many cards in the answer have rank higher than the highest rank in the guess (*higher ranks*).
5. How many of the cards in the answer have the same suit as a card in the guess, only counting a card in the guess once (*correct suits*). For example, if the answer has two clubs and the guess has one club, or vice versa, the correct suits number would be 1, not 2.

Note that the order of the cards in the answer and the guess is immaterial, and that, since they come from a single deck, **cards cannot be repeated in either answer or guess**.

The guesser then guesses again, and receives feedback for the new guess, repeating the process until the guesser guesses the answer correctly. The object of the game for the guesser is to guess the answer with the fewest possible guesses.

A few examples of the feedback for a given answer and guess (with clubs, diamonds, hearts, and spades shown as ♣, ♦, ♥, and ♠, respectively):

Answer	Guess	Feedback	Explanation
3♣,4♥	4♥,3♣	2,0,2,0,2	2 answer cards match guess, no answer rank less than 3, 2 answer ranks match guess, no answer rank greater than 4, 2 answer suits match guess.
3♣,4♥	3♣,3♥;	1,0,1,1,2	1 exact match (3♣), no answer rank less than 3, 1 answer rank matches guess, 1 answer rank greater than 3, 2 answer suits match guess.
3♦,3♠	3♣,3♥	0,0,2,0,0	No exact matches, no answer rank less than 3, 2 answer ranks match guess, no answer ranks greater than 3, no answer suits match guess.
3♣,4♥	2♥,3♥	0,0,1,1,1	No exact matches, no answer rank less than a 2, 1 answer rank matches guess, 1 answer rank greater than 3, 1 answer suit matches guess.
A♣,2♣	3♣,4♥	0,1,0,1,1	No exact matches, 1 answer rank less than 3 (Ace is high), no answer rank matches guess, 1 answer rank greater than 4, 1 answer suit matches guess (either ♣; you can only count 1 because there's only 1 in the guess).

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

The Program

For this assignment, you will write Haskell code to implement code for both the *answerer* and *guesser* parts of the game. This will require you to write a function to evaluate a guess to produce feedback, one to provide an initial guess, and one to use the feedback from the previous guess to determine the next guess. The latter function will be called repeatedly until it produces the correct guess. You will find it useful to keep information between guesses; since Haskell is a purely functional language, you cannot use a global or static variable to store this. Therefore, your initial guess function must return this game state information, and your next guess function must take the game state as input and return the updated game state as output. You may put any information you like in the game state, but you *must* define a type `GameState` to hold this information.

I will supply a `Card` module providing the `Card`, `Rank`, and `Suit` types and their constructors, similar to the types used in lectures. The suits are (in increasing order) `Club`, `Diamond`, `Heart`, and `Spade`, and the ranks are (in increasing order) `R2`, `R3`, `R4`, `R5`, `R6`, `R7`, `R8`, `R9`, `R10`, `Jack`, `Queen`, `King`, `Ace`, and cards are of the form `Card` *suit rank*.

Assignment Project Exam Help

The `Card` module has a few enhancements over the types presented in lectures. First, all three types are in the `Eq` and `Ord` classes. All are also in the `Bounded` and `Enum` classes, which means, for example, that `[minBound..maxBound]::[Card]` is the list of all cards in order, from 2 to A, and similarly `[minBound..maxBound]::[Rank]` is the list of ranks from 2 to Ace, and similarly for `Suit`. This also means that, for example, `succ (Card Club R5) == (Card Club R6)` and `succ (Card Heart Ace) == (Card Spade R2)`. Read the documentation for `Bounded` and `Enum` classes for more things you can do with them.

https://tutorcs.com

WeChat: cstutorcs

For convenience, all three types are also in the `Show` class so that ranks and suits are shown as a single character (10 is shown as `T`), and cards as two characters: rank followed by suit. All three are also in the Read class, which means that, for example, `(read "[2C,AH]")::[Card]` returns the list `[Card Club R2, Card Heart Ace]` (which would be printed as `[2C,AH]`).

You must define the following three functions, as well as the `GameState` type:

feedback :: [Card] -> [Card] -> (Int,Int,Int,Int,Int)

> takes a target and a guess (in that order), each represented as a list of `Card`s, and returns the five feedback numbers, as explained above, as a tuple.

initialGuess :: Int -> ([Card],GameState)

takes the number of cards in the answer as input and returns a pair of an initial guess, which should be a list of the specified number of cards, and a game state. The number of cards specified will be 2 for most of the test, and 3 or 4 for the remaining tests, as explained below.

nextGuess :: ([Card],GameState) -> (Int,Int,Int,Int,Int) -> ([Card],GameState)

takes as input a pair of the previous guess and game state, and the feedback to this guess as a quintuple of counts of correct cards, low ranks, correct ranks, high ranks, and correct suits, and returns a pair of the next guess and new game state.

Your source file must begin with the module declaration:

```
module Proj2 (feedback, initialGuess, nextGuess,
GameState) where
```

and must import the supplied `Card` module (you must not write your own).

Please keep all your code in this one module.

Assignment Project Exam Help

Assessment

https://tutorcs.com

Your project will be assessed on the following criteria:

WeChat: cstutorcs

30%
: Quality of your code and documentation;

10%
: Correctness of your implementation of `feedback`.

40%
: Correctness of your implementation of `initialGuess` and `nextGuess` for 2-card targets and quality of the guesses made by your implementation, as indicated by the number of guesses needed to find 2-card targets;

20%
: Correctness and quality of your guesses for 3- and 4-card targets.

Note that timeouts will be imposed on all tests. You will have at least 10 seconds to guess each answer, regardless of how many guesses are needed. Executions taking longer than that may be unceremoniously terminated, leading to that test being assessed as failing. Your programs will be compiled with `ghc -O2` (optimisation enabled) for testing, so 10 seconds per test is a very reasonable limit. The same 10 second timeout will apply to the 3 and 4 card test cases.

See the Project Coding Guidelines on the LMS for detailed recommendations for coding style. These guidelines will form the basis of the quality assessment of your code and documentation, which is worth 30% of the project mark, so read them carefully.

Be sure to submit your code for assessment by hitting the Mark button on the upper right side of the Grok window! It is not enough to hit **Save**.

Testing

You can test your code within Grok by hitting the **Run** button, which will test your code on the the sample answer Queen of Hearts and 4 of Spades. For more flexible testing, you can hit the **Terminal** button (which will run `ghci` and load your code) and use the `test` function, giving it *a single* string consisting of 2, 3, or 4 two-character card abbreviations, separated by spaces. This will call your `initialGuess` function and repeatedly call your `nextGuess` function to guess the target, using your `feedback` function to produce the feedback for each guess. For example:

Assignment Project Exam Help

```
test "3H TC"
```

will test your code on a target of 3♥ and 10♣. Note that this is case sensitive: letters must be upper case. Keep in mind that Grok imposes a 4 second timeout on testing from the **Run** button, which applies each time you hit the **Run** button, not each time you use the `test` function, so you may find you need to restart testing by hitting the **Run** button again to restart Grok's timer.

https://tutorcs.com

WeChat: cstutorcs

Important: if your `feedback` function is buggy, it is possible that this test will find the target, even if your code is incorrect. Therefore, it is important that you thoroughly test your `feedback` function.

Late Penalties

Late submissions will incur a penalty of 0.5% of the possible value of that submission per hour late, including evening and weekend hours. This means that a perfect project that is much more than 4 days late will receive less than half the marks for the project. If you have a medical or similar compelling reason for being late, you should contact the head tutor as early as possible to ask for an extension (preferably before the due date).

Hints

1. A very simple approach to this program is to simply guess every possible combination of different cards until you guess right. There are only 52×51/2 = 1326 possible answers, so on average it should only take about 663 guesses, making it perfectly feasible to do in 10 seconds. However, this will give a very poor score for guess quality.
2. A better approach would be to only make guesses that are consistent with the answers you have received for previous guesses. You can do this by computing the list of possible answers, and removing elements that are inconsistent with any answers you have received to previous guesses. A possible answer is inconsistent with an answer you have received for a previous guess if the answer you would receive for that guess and that (possible) answer is different from the answer you actually received for that guess. For two-card answers, the initial list of possible guesses is only 1326 elements, and rapidly shrinks with feedback, so this is quite feasible. This scales up to about 5 cards, when the initial number of possible guesses is 2,598,960. Beyond that, another strategy would need to be used, but you only need to handle 2 to 4 cards.

Assignment Project Exam Help

   You can use your `GameState` type to store the list of remaining possible answers, and pare it down each time you receive feedback for a guess.

https://tutorcs.com

3. The best results can be had by carefully choosing a guess that is likely to leave you the smallest remaining list of possible answers. You can do this by computing, for each remaining possible answer, the average number of possible answers it will leave if you guess it. Given a candidate guess G (which should be selected from the remaining possible answers), compute the feedback you will receive for each possible answer A if G is the guess and A is the answer. If you group all the As by the feedback they give you, your aim is to have many small groups, because that means if you make G your guess, that will probably leave few possible answers when you receive the feedback. Therefore the expected number of remaining possible answers for that guess is the average of the sizes of these groups, weighted by the sizes of the groups. That is, it is the sum of the squares of the group sizes divided by the sum of the group sizes.

WeChat: cstutorcs

4. For the first guess, when the list of possible answers is at its longest, you may want to use a different approach. Given the way the feedback works, the best first guess would be to choose two cards of different suits and with ranks about equally distant from each other and from the top and bottom ranks. In general, for an *n* card answer, you should choose ranks that are about 13/(*n*+1) ranks apart.

5. Note that these are just hints; you are welcome to use any approach you like to solve this, as long as it is correct and runs within the allowed time.
6. For a two card answer, with a good guessing strategy such as outlined above, 4 or 5 guesses is usually enough to guess it. Surprisingly, adding more cards does not increase the number of guesses needed very much.

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs