Project #3 - Playing Games with Racket

Learning objectives

- Design and implement a complete program in a functional programming language.
- Employ sound principles of program design.

Overview

The object of this assignment is to develop a complete program in Racket using a sound functional design. To this end, you will write a program that can play the game BlackJack(21) with a single human player vs. the computer.

Grading

In order to help you design a clean program, I will specify a number of functions for you to implement. The majority (80%) of the grade for this project will be based on how well you implement these functions. The rest of the grade (20%) will be based on your user interface, and well commented code with good style following our code-guidelines for readability.

First, some basics...

For our card game, a "card" will consist of a dotted pair that contains the face value of the card followed by the suit. The face value will be a number from 2 to 10 or one of the symbols 'J 'Q 'K or 'A. The suit will be one of the symbols 'Clubs, 'Diamonds, 'Hearts, or 'Spades. For example, I could create the Jack of Diamonds and the 9 of Spades with the following two commands:

```
(define card1 (cons 'J 'Diamonds))
(define card2 (cons 9 'Spades))
```

A "hand" will simply be a list of cards. For example:

```
(define make-card
  (lambda (face suit)
    (cons face suit)
  )
)

(define hand1 (list (make-card 'A 'Clubs) (make-card 8 'Spades)
                    (make-card 2 'Diamonds)
              )
)
```

A "deck" is also a list of cards. To start with, the deck will contain all 52 cards. As each card is dealt, it should be removed from the deck.

Global variables

Your program should begin with the following global definitions:

```
(define faces '(2 3 4 5 6 7 8 9 10 J Q K A))
(define suits '(Clubs Diamonds Hearts Spades))
```

(You may, if you wish, choose to use images instead of strings for the suit names).

Required Functions

To obtain full credit, each of the following functions must be implemented *exactly* as described below, including the name of the function, the parameters expected (and their order), and the return value and type.

- make-deck (10 pts) -- Creates a new (unshuffled) 52 card deck
  - Parameters: none
  - Returns: The deck (a list of cards)
  - Notes: Depends on the global definitions of **faces** and **suits**.
  - Example: `(define thedeck (make-deck))`
- eval-hand (10 pts) -- Determine the *best* value of a hand, given that an Ace can be worth either 1 or 11 points.
  - Parameter: hand -- A list of cards to evaluate
  - Returns: The best possible value of the hand
  - Example: `(display (eval-hand playerhand))`
- deal! (10 pts) -- Create a two card hand, by removing the top two cards from the deck.
  - Parameter: deck -- The deck to deal from
  - Returns: A new two-card hand, consisting of the first two cards in the deck.
  - Side effects: The first two cards are **removed from the deck**. *(You may choose to use a macro to do this)*
  - Example: `(define playerhand (deal thedeck))`
- hit! (5 pts) -- Take the top card from the deck and add it to a hand.
  - Parameter 1: deck -- The deck to deal from
  - Parameter 2: hand -- The hand to deal into

    - Returns: Nothing of interest
    - Side effects: The first card in the deck is removed from the deck and added to the hand.
    - Example: `(hit! thedeck dealerhand)`
- show-hand (5 pts) -- Display a hand (or most of it) to the screen.
    - Parameter 1: hand -- The hand to display
    - Parameter 2: how -- Should be either 'Full or 'Part. If *how* is 'Full, then show the entire hand. If *how* is 'Part, then show only ***** for the first card, and then show the rest of the hand normally.
    - Parameter 3: description -- A string to be displayed to the screen before the actual hand.
    - Returns: Nothing of interest
    - Side effects: The hand is displayed to the screen
    - Example: `(show-hand dealerhand 'Part "The dealer has: ")`

The user interface and code style

Implementing the above functions correctly will result in a grade of 'B' (40/50 points) for the lab. The other 10 points will be based on your user interface. A user interface that plays blackjack with the user, allowing them to bet money from a starting quota, until they decide to quit, is sufficient to obtain a minimal 'A' (5 points). Use your creativity. A well commented code with good style and following our code-guidelines for readability, for the additional 5 points.

Header comments

Please include a header block at the beginning of your file that indicates what improvements (if any) you have made to the basic user interface to obtain more points.

Helper functions

Only functions used in your main routine should be defined globally using (define...). All other functions should be defined within the global functions using letrec or a named let. The *only exception* to this rule is the following: If you have a helper function that is used identically across multiple functions, you may define it globally.

Requirements

In order to obtain the target grade your code must be:

- Readable -- Your code should:
  - Use readable variable and parameter names;
  - Use inline comments for each major block;
  - Include header comments for each function describing the parameters (and their data types) any outputs or return values, any side effects, and the general purpose of the function; and
  - Be well organized, with *appropriate indentation, newlines, and whitespace*.
- Robust -- For this assignment it will not be necessary to check the *types* of the input parameters. In other words, if your function expects two integers, you do not have to worry about what will happen if it is passed two strings instead. However, you should be able to handle any input of the correct type. In other words, a function expecting an integer should function correctly for *any* integer. A function expecting a list should work correctly for *any* list, even the empty list.
- Correct -- Your functions must be named *exactly* as described above, and must expect *exactly* the parameters described above. *(Exception: If you are implementing a graphical interface, your show-hand function may differ.)*

A few final suggestions

Racket does include a drawing, graphics, and GUI libraries if you wish to use them. Choose Help/Help Desk from the DrRacket menu and look at the entries for *The Racket Drawing Toolkit* and *The Racket Graphical Interface Toolkit*

A common decision rule for the dealer in BlackJack is to stand on 17 or above, hit on 16 or less. Sometimes this can be modified when there is an Ace in the dealer's hand. I will leave improvements on this rule to you.

Submitting your code

Your entire blackjack game should be defined in a single Racket file -- function definitions first, main routine last. **Write your name (commented by ;) on the first line Immediately following the line #lang racket.** Place this code in a file called `Project3.rkt` and submit it via the Pilot dropbox.