This problem involves a solitaire card game invented just for this question. You will write a program that tracks the progress of a game. You only need to complete part (g), but parts (a)–(f) are provided as hints. A game is played with a card-list and a goal. The player has a list of held cards, initially empty. The player makes a move by either drawing, which means removing the first card in the card-list from the card-list and adding it to the held-cards, or discarding, which means choosing one of the held-cards to remove. The game ends either when the player chooses to make no more moves or when the sum of the values of the held-cards is greater than the goal. The objective is to end the game with a low score (0 is best). Scoring works as follows: Let `sum` be the sum of the values of the held-cards. If `sum` is greater than `goal`, the preliminary score is three times `(sum-goal)`, else the preliminary score is `(goal-sum)`. The score is the preliminary score unless all the held-cards are the same color, in which case the score is the preliminary score divided by 2 (and rounded down as usual with integer division; use ML's `div` operator).

(a) Write a function `card_color`, which takes a card and returns its color (spades and clubs are black, diamonds and hearts are red). Note: One case-expression is enough.

(b) Write a function `card_value`, which takes a card and returns its value (numbered cards have their number as the value, aces are 11, everything else is 10). Note: One case-expression is enough.

(c) Write a function `remove_card`, which takes a list of cards `cs`, a card `c`, and an exception `e`. It returns a list that has all the elements of `cs` except `c`. If `c` is in the list more than once, remove only the first one. If `c` is not in the list, raise the exception `e.` You can compare cards with =.

(d) Write a function `all_same_color`, which takes a list of cards and returns `true` if all the cards in the list are the same color.

(e) Write a function `sum_cards`, which takes a list of cards and returns the sum of their values. You may use a locally defined helper function that is tail recursive, but this is not a requirement.

(f) Write a function `score`, which takes a card list (the held-cards) and an int (the goal) and computes the score as described above.

(g) Write a function `officiate`, which "runs a game." It takes a card list (the card-list) a move list (what the player "does" at each point), and an int (the goal) and returns the score at the end of the game after processing (some or all of) the moves in the move list in order. You can use a locally defined recursive helper function that takes several arguments that together represent the current state of the game. As described above:

- The game starts with the held-cards being the empty list.

- The game ends if there are no more moves. (The player chose to stop since the move list is empty.)

- If the player discards some card c, play continues (i.e., make a recursive call) with the held cards not having c and the card-list unchanged. If c is not in the held-cards, raise the `IllegalMove` exception.
- If the player draws and the card-list is (already) empty, the game is over. Else if drawing causes the sum of the held-cards to exceed the goal, the game is over (after drawing). Else play continues with a larger held-cards and a smaller card-list.  Note: For the problem, the following datatypes have been defined:

```
datatype suit = Clubs | Diamonds | Hearts | Spades
datatype rank = Jack | Queen | King | Ace | Num of int
type card = suit * rank
datatype color = Red | Black
datatype move = Discard of card | Draw

exception IllegalMove
```

You do not need to copy these code into your solution.