
Programmieren für Geisteswissenschaftler

Blatt 9 (geändert am 11.1.2002, 10:00 Uhr)

Abgabe: 24.1.2002

In einer Abwandlung des Kartenspiels „Herzblatt“ sind die Karten folgendermaßen geordnet: Zahlen 7 bis 9, Bube, Dame, König, Zehn, Ass. Die Werte der Karten sind: 7 bis 9 null Punkte, Bube 2, Dame 3, König 4, Zehn 10 und Ass 11 Punkte. Weiterhin hat jede Karte eine Farbe: Herz, Pik, Kreuz, Karo. Herz ist immer Trumpf, das bedeutet, eine Herz-Karte sticht jede Karte einer anderen Farbe.

Zu Beginn des Spiels erhält jeder Spieler zehn Karten. Wer einen Stich macht spielt die nächste Karte aus. Farbe muss bedient werden, das bedeutet, daß der Spieler der als zweiter an der Reihe ist eine Karte der selben Farbe wie die des ersten Spielers spielen muss. Hat er keine Karte der gleichen Farbe, so verliert er den Stich, außer er spielt einen Trumpf aus. Gewinner ist der Spieler mit den meisten Punkten.

In diesem Übungsblatt geht es darum, ein vorgegebenes Programm für das Spiel Herzblatt zu vervollständigen, indem Du abstrakte Datentypen implementierst. Der Code zu diesem Übungsblatt ist unter der Adresse

<http://www-pu.informatik.uni-tuebingen.de/pfg-2001/software/herzblatt.scm>

zu finden.

1. [10 Punkte] Führe einen Datentyp *Card*, mit Konstruktor *make-card*, welcher den Namen der Karte und ihre Farbe als Parameter akzeptiert, Selektoren *card-color* und *card-name* ein um Karten zu repräsentieren. Definiere außerdem eine Prozedur *card<*, um Karten zu vergleichen; diese Prozedur soll annehmen, daß die Karte, die ihr als erstes Argument übergeben wurde, auch die Karte ist, die zuerst ausgespielt wurde. Gib außerdem eine Prozedur *display-card* an, die eine Karte in lesbarer Form ausgibt und eine Prozedur *card-value*, welche den Wert einer Karte zurückgibt. Eine Prozedur *name->value* ist im Code bereits vorhanden.
2. [10 Punkte] Der folgende (unvollständige) Code implementiert den Computerspieler. *computer-first-move* wird aufgerufen, wenn der Computer die erste Karte spielen soll, *computer-reply*, wenn der Computer als zweiter an der Reihe ist.

```
(define computer-reply
  (lambda (deck other-card)
    (let ((my-card (choose-card deck other-card)))
      (display "My move: ")
      (display-card my-card)
      my-card)))

(define choose-card
  (lambda (deck other-card)
    (let ((same-colors (deck-cards-with-color deck (card-color other-card))))
      (if (null? same-colors)
          (let ((trump-cards (deck-cards-with-color deck 'herz)))
            (if (null? trump-cards)
                (deck-any-card deck)
                (lowest-card trump-cards)))
          (higher-or-lowest-card same-colors other-card)))))

(define lowest-card
```

```

(lambda (card-list)
  (lowest-card-helper (cdr card-list) (car card-list)))

(define lowest-card-helper
  (lambda (card-list maybe-lowest)
    (if (null? card-list)
        maybe-lowest
        (if (card< (car card-list) maybe-lowest)
            (lowest-card-helper (cdr card-list) (car card-list))
            (lowest-card-helper (cdr card-list) maybe-lowest)))))

(define computer-first-move
  (lambda (deck)
    (let ((card (deck-any-card deck)))
      (display "I play ")
      (display-card card)
      card)))

(define higher-or-lowest-card
  (lambda (card-list the-card)
    (let ((a-higher-card (higher-card card-list the-card)))
      (if a-higher-card
          a-higher-card
          (lowest-card card-list)))))

(define higher-card
  (lambda (card-list other-card)
    (if (null? card-list)
        #f
        (if (card< other-card (car card-list))
            (car card-list)
            (higher-card (cdr card-list) other-card)))))

```

- Beschreibe die Strategie des Computers.
 - Implementiere den Datentyp *Deck*. Überlege dir dazu eine geeignete Repräsentation für ein Deck und gib den Konstruktor `make-deck`, der eine Liste aus Karten als Argument akzeptieren soll, den Ersetzer `deck-remove-card`, sowie die Prozeduren `deck-empty?`, `deck-cards-with-color`, `deck-any-card` an, die der Code des Computerspielers benötigt.
3. [10 Punkte] Der Rest des Codes läßt Dich mit dem Computer Herzblatt spielen. Implementiere den abstrakte Datentyp *Game-State*, indem du den Konstruktor `make-game-state`, welcher die Decks der beiden Spieler als Parameter nimmt, die Selektoren `game-state-computer-deck` und `game-state-human-deck` angibst und die Ersetzer `game-state-replace-decks`, `game-state-increment-human` und `game-state-increment-computer`. Außerdem werden noch die Prozeduren `game-over?`, `display-deck` und `winner` benötigt.