

# COMP1811 – Scheme Project Report

Name	Radoslav Gadzhovski	SID	001200583
Partner's name	Riwaj Siral	SID	001195216

## 1. SOFTWARE DESIGN

Our team created a card game named “Seven and Half Computer-Aided Solitaire” its based on the well-known game “Black Jack”. Before starting the game, we removed cards with numerals 8,9,10 as well as “Jockers” from the deck. The goal is to reach a mark of 7,5 or as closer as much, but you don’t have to exceed 7,5 as you will lose the game instantly. All cards with “faces” (Jack, Queen, King) counts as 0,5 points and the numeral cards counts points as its own value (3 of Hearts = 3 points).

As shown below for the implementation of the game strategy, we used mostly recursion.(Figure 1.1/2)

```
67 ;; 3 .- The deck.
68
69 (define (deck? values)           ;; Provided a list of values, returns #t if it is a valid deck.
70   (cond
71     [(empty? values) #t]
72     [else
73      (and
74       (card? (first values))
75       (deck? (rest values)))]))
76
77
```

Figure 1.1 Recursion (The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.)

```
77
78 (define (valueOf round)           ;; Returns the total value of the cards in the round.
79   (cond
80     [(empty? round) 0]
81     [else
82      (+ (value (first round)) (valueOf (rest round)))]))
83
84
```

Figure 1.2

## 2. CODE LISTING

## 2.1 FUNCTION: CARD?

```
(define (card? value)      ;; Returns #t if the the card has both "numeral/face" and "suite" .
  (and
    (pair? value)
    (list? (member (car value) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
    (list? (member (cdr value) '(\H \S \C \D)))
  )
)
```

## 2.2 FUNCTION: SUITE

```
(define (suite card) ;; Returns the second part of the card which is its"suite".
  (cdr card))
```

## 2.3 FUNCTION: NUMERAL

```
(define (numeral card) ;; Returns the first part of the card which is its "rank/numeral".
  (car card))
```

## 2.4 FUNCTION: FACE?

```
(define (face? card)
  (not (integer? (numeral card)))) ;; Check if the has "face" (Jack, Queen or King) returns #t.
```

## 2.5 FUNCTION: VALUE

```
(define (value card)      ;; Returns the "value" of a given card.
  (cond
    [(face? card) 0.5]
    [else (numeral card)]))
```

## 2.6 FUNCTION: CARD->STRING

```
(define (card->string value)      ;; Given a valid card, it returns a human readable string for it.
  (cond
    [(face? value) (string (numeral value) (suite value))])
```

```
[else
 (string-append
  (number->string (numeral value))
  (string (suite value))))]
```

## 2.7 FUNCTION: DECK?

```
(define (deck? values)      ;; Provided a list of values, returns #t if it is a valid deck.
  (cond
    [(empty? values) #t]
    [else
     (and
      (card? (first values))
      (deck? (rest values)))]))
```

## 2.8 FUNCTION: VALUEOF

```
(define (valueOf round)      ;; Returns the total value of the cards in the round.
  (cond
    [(empty? round) 0]
    [else
     (+ (value (first round)) (valueOf (rest round)))])
```

## 2.9 FUNCTION: DO-SUITE

```
(define (do-suite value) ;; Entire list of all 40 valid cards( 7 numerical and 3 faces)*4).
  (map
    (lambda (num) (cons num value))
    '( 1 2 3 4 5 6 7 #\J #\Q #\K )))
```

## 2.10 FUNCTION: DECK

```
(define deck (append (do-suite #\H) (do-suite #\S) (do-suite #\C) (do-suite #\D))) ;;
Generating deck
```

## 2.11 FUNCTION: DECK->STRINGS

```
(define (deck->strings deck) ;; Passed a valid deck, it returns a human representation of it.
  (cond
    [(empty? deck) empty]
```

```
[else
  (cons
    (card->string (first deck))
    (deck->strings (rest deck))))))
```

## 2.12 FUNCTION: PROBABILITY

(define (probability comp number deck) ;; Passed (<, >, =), a number, and a list of cards, works out the number of cards in list having values less,

```
(cond
  ;; greater or equal than the provided number respectively.
  [(empty? deck) 0]
  [(comp (value (first deck)) number)
   (+ 1 (probability comp number (rest deck)))]
  [else (probability comp number (rest deck))])
```


## 2.13 FUNCTION: CHEAT

(define cheat #f) ;; Hides the next card from the deck

## 3. RESULTS – OUTPUT OBTAINED

*Provide screenshots that demonstrate the results generated by running your code. That is show the output obtained in the REPL when calling your functions. Alternatively, you may simply cut and paste from the REPL.*

### 3.1 TASK-1 CARD? TEST



```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (card? "7") (card? 7)
#f
#f
> (card? "7 of Spades")
#f
> (card? (cons 4 #\S))
#t
> (card? (cons 3 #\Q))
#f
>
```

### 3.2 TASK-2 SUITE TEST, NUMERAL TEST

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (suite (cons 2 #\S)) (suite (cons #\J #\H))
#\S
#\H
> (numeral (cons 2 #\S)) (numeral (cons #\K #\S))
2
#\K
>
```

Pretty Big ▼ 92 420.36 MB

### 3.3 TASK-3 FACE? TEST

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (face? (cons 2 #\S)) (face? (cons #\J #\S))
#f
#t
>
```

Pretty Big ▼ 62 421.95 MB

### 3.4 TASK-4 VALUE TEST

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (value (cons 4 #\S)) (value (cons #\K #\C))
4
0.5
>
```

Pretty Big ▼ 62 427.50 MB

### 3.5 TASK-5 CARD->STRING TEST

```

Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (card->string (cons 3 #\S)) (card->string (cons #\J #\H))
"3S"
"JH"
> |

```

Pretty Big ▾

62

427.40 MB



## 3.6 TASK-6 DECK? TEST

```

Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (deck? (list (cons 10 #\S) (cons 4 #\D)))
#f
> (deck? (list (cons 4 #\S) (cons 2 #\Q)))
#f
> (deck? (list (cons 4 #\S) (cons 2 #\D)))
#t
>

```

Pretty Big ▾

92

420.79 MB



## 3.7 TASK-7 VALUE OF TEST

```

Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (valueOf (list
  (cons 4 #\S)
  (cons #\K #\C)))
4.5
> (valueOf (list
  (cons 2 #\H)
  (cons #\Q #\D)))
2.5
> (valueOf empty)
0
>

```

Pretty Big ▾

11:17

417.12 MB



## 3.8 TASK-8 DO-SUITE TEST

```

Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (do-suite #\C)
(1 . #\C)
(2 . #\C)
(3 . #\C)
(4 . #\C)
(5 . #\C)
(6 . #\C)
(7 . #\C)
(#\J . #\C)
(#\Q . #\C)
(#\K . #\C)
> deck
(1 . #\H)
(2 . #\H)
(3 . #\H)
(4 . #\H)
(5 . #\H)
(6 . #\H)
(7 . #\H)
(#\J . #\H)
(#\Q . #\H)
(#\K . #\H)
(1 . #\S)
(2 . #\S)
(3 . #\S)
(4 . #\S)
(5 . #\S)
(6 . #\S)
(7 . #\S)
(#\J . #\S)
(#\Q . #\S)
(#\K . #\S)
(1 . #\C)
(2 . #\C)
(3 . #\C)
..

```

### 3.9 TASK-9 DECK->STRING TEST

```

Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (deck->strings deck)
("1H"
 "2H"
 "3H"
 "4H"
 "5H"
 "6H"
 "7H"
 "JH"
 "QH"
 "KH"
 "1S"
 "2S"
 "3S"
 "4S"
 "5S"
 "6S"
 "7S"
 "JS"
 "QS"
 "KS"
 "1C"
 "2C"
 "3C"
 "4C"
 "5C"
 "6C"
 "7C"
 "JC"
 "QC"
 "KC"
 "1D"
 "2D"
 "3D"
 "4D"
 "5D"
 "6D"
 "7D"
 "JD"
 "QD"
 "KD")

```

### 3.10 TASK-10 PROBABILITY TEST

```

Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (probability > 7 (list (cons 7 #\H) (cons 2 #\D) (cons #\J #\H)))
0
> (probability = 7 (list (cons 7 #\H) (cons 2 #\D) (cons #\J #\H)))
1
> (probability < 3 (list (cons 7 #\H) (cons 2 #\D) (cons #\J #\H)))
2
> |

```

## 4. TESTING

### Loosing game ‘LOST’

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (play (shuffle deck) '())
P(>7.5):0/40
P(<7.5):40/40
P(=7.5):0/40
HAND: ()
VALUE:0
DECK:*****
accept
P(>7.5):19/39
P(<7.5):20/39
P(=7.5):0/39
HAND: (5D)
VALUE:5
DECK:*****
accept
P(>7.5):30/38
P(<7.5):0/38
P(=7.5):0/38
HAND: (6D 5D)
VALUE:11
DECK:*****
LOST!
>
```

Card accepted on the first deal (5D – value 5)

Card accepted on the second deal (6D – value 6)

Result:  $5 + 6 = 11$        $11 > 7.5$  “LOST”

### Wining game “WIN”

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (play (shuffle deck) '())
P(>7.5):0/40
P(<7.5):40/40
P(=7.5):0/40
HAND: ()
VALUE:0
DECK:*****
accept
P(>7.5):0/39
P(<7.5):35/39
P(=7.5):4/39
HAND: (KC)
VALUE:0.5
DECK:*****
accept
P(>7.5):30/38
P(<7.5):0/38
P(=7.5):0/38
HAND: (7H KC)
VALUE:7.5
DECK:*****
WIN
> |
```

Card accepted on the first deal (KC – value 0.5)

Card accepted on the second deal (7H – value 7)

Result:  $0.5 + 7 = 7.5$        $7.5 = 7.5$  “WIN”



# Cheating

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (define cheat #t)
> (play (shuffle deck) '())
P(>7.5):0/40
P(<7.5):40/40
P(=7.5):0/40
HAND: ()
VALUE:0
DECK: (7C KS JS KH)...
accept
P(>7.5):27/39
P(<7.5):0/39
P(=7.5):12/39
HAND: (7C)
VALUE:7
DECK: (KS JS KH 7H)...
accept
P(>7.5):38/38
P(<7.5):0/38
P(=7.5):0/38
HAND: (KS 7C)
VALUE:7.5
DECK: (JS KH 7H QH)...
WIN
>
```

Using (define cheat #t) we can see which cards are coming next from the deck, this way we can easily win the game.

## 5. EVALUATION

---

During the process of building this application we learned a lot about functional programming and recursion in general, and how we can create such a nice game in just a few lines of code. Recursive thinking is one of the most important parts in programming. It is helpful for breaking down big problems into smaller ones. Most of the times it can be even simpler to read than the iterative one. From now I will try to use it in every project as it could be implemented in most of the programming languages. Seven and Half Computer-Aided Solitaire is simple game and if I have free time, I will try to create a GUI for it and also, I will change the strategy of the game so it will use a full deck of cards and the “winning” value will be bigger. Furthermore, I will include starting points to the player and after each winning game the points will be increased.

