Assignment: Task 3

Abstract: This task dealt with dealing cards to hand 1, hand2, a prize suite, and a discard suite. The prize suite is then shuffled and after the shuffle a prize card is picked and shown.

Interaction: Task 3 Demos

```
[203]> ( demo3 )

>>> Running Task 3 Demo.

--- creating the deck.

>>> display the deck...

--- *deck* =

((ACE . CLUB) (2 . CLUB) (3 . CLUB) (4 . CLUB) (5 . CLUB) (6 . CLUB)
(7 . CLUB) (8 . CLUB) (9 . CLUB) (10 . CLUB) (JACK . CLUB)

 (QUEEN . CLUB) (KING . CLUB) (ACE . DIAMOND) (2 . DIAMOND) (3 .
DIAMOND) (4 . DIAMOND) (5 . DIAMOND) (6 . DIAMOND) (7 . DIAMOND)

 (8 . DIAMOND) (9 . DIAMOND) (10 . DIAMOND) (JACK . DIAMOND) (QUEEN .
DIAMOND) (KING . DIAMOND) (ACE . SPADE) (2 . SPADE) (3 . SPADE)

 (4 . SPADE) (5 . SPADE) (6 . SPADE) (7 . SPADE) (8 . SPADE) (9 .
SPADE) (10 . SPADE) (JACK . SPADE) (QUEEN . SPADE) (KING . SPADE)

 (ACE . HEART) (2 . HEART) (3 . HEART) (4 . HEART) (5 . HEART) (6 .
HEART) (7 . HEART) (8 . HEART) (9 . HEART) (10 . HEART) (JACK . HEART)

 (QUEEN . HEART) (KING . HEART))

>>> display hand1...

--- *hand1* = NIL

>>> display hand2...

--- *hand2* = NIL

>>> display prize suite...

--- *prize-suite* = NIL

>>> display discard suite...

--- *discard-suite* = NIL
```

```
>>> dealing cards to hands...

>>> display hand1...

--- *hand1* =

((ACE . SPADE) (2 . SPADE) (3 . SPADE) (4 . SPADE) (5 . SPADE) (6 .
SPADE) (7 . SPADE) (8 . SPADE) (9 . SPADE) (10 . SPADE) (JACK . SPADE)

 (QUEEN . SPADE) (KING . SPADE))

>>> display hand2...

--- *hand2* =

((ACE . HEART) (2 . HEART) (3 . HEART) (4 . HEART) (5 . HEART) (6 .
HEART) (7 . HEART) (8 . HEART) (9 . HEART) (10 . HEART) (JACK . HEART)

 (QUEEN . HEART) (KING . HEART))

>>> display prize suite...

--- *prize-suite* =

((ACE . CLUB) (2 . CLUB) (3 . CLUB) (4 . CLUB) (5 . CLUB) (6 . CLUB)
(7 . CLUB) (8 . CLUB) (9 . CLUB) (10 . CLUB) (JACK . CLUB)

 (QUEEN . CLUB) (KING . CLUB))

>>> display discard suite...

--- *discard-suite* =

((ACE . DIAMOND) (2 . DIAMOND) (3 . DIAMOND) (4 . DIAMOND) (5 .
DIAMOND) (6 . DIAMOND) (7 . DIAMOND) (8 . DIAMOND) (9 . DIAMOND)

 (10 . DIAMOND) (JACK . DIAMOND) (QUEEN . DIAMOND) (KING . DIAMOND))

>>> Done creating and dealing.




>>> Removing unused suite.

>>> display discard suite...

--- *discard-suite* =

((ACE . DIAMOND) (2 . DIAMOND) (3 . DIAMOND) (4 . DIAMOND) (5 .
DIAMOND) (6 . DIAMOND) (7 . DIAMOND) (8 . DIAMOND) (9 . DIAMOND)

 (10 . DIAMOND) (JACK . DIAMOND) (QUEEN . DIAMOND) (KING . DIAMOND))

>>> display discard suite...
```

```
--- *discard-suite* = NIL

>>> Done removing unused suite.




>>> Shuffling prize suite...

>>> display prize suite...

--- *prize-suite* =

((ACE . CLUB) (2 . CLUB) (3 . CLUB) (4 . CLUB) (5 . CLUB) (6 . CLUB)
(7 . CLUB) (8 . CLUB) (9 . CLUB) (10 . CLUB) (JACK . CLUB)

 (QUEEN . CLUB) (KING . CLUB))

>>> display prize suite...

--- *prize-suite* =

((KING . CLUB) (10 . CLUB) (QUEEN . CLUB) (5 . CLUB) (ACE . CLUB)
(JACK . CLUB) (6 . CLUB) (7 . CLUB) (2 . CLUB) (9 . CLUB) (4 . CLUB)

 (8 . CLUB) (3 . CLUB))

>>> Done shuffling.




>>> Display the prize card ...

>>> display prize suite...

--- *prize-suite* =

((KING . CLUB) (10 . CLUB) (QUEEN . CLUB) (5 . CLUB) (ACE . CLUB)
(JACK . CLUB) (6 . CLUB) (7 . CLUB) (2 . CLUB) (9 . CLUB) (4 . CLUB)

 (8 . CLUB) (3 . CLUB))

--- prize-card = NIL

>>> Fetching the prize card...

>>> display prize suite...

--- *prize-suite* =

((10 . CLUB) (QUEEN . CLUB) (5 . CLUB) (ACE . CLUB) (JACK . CLUB) (6 .
CLUB) (7 . CLUB) (2 . CLUB) (9 . CLUB) (4 . CLUB) (8 . CLUB)
```

```
 (3 . CLUB))
--- prize-card = (KING . CLUB)



>>> Finished Task 3 Demo.
NIL
[204]>
```

Interaction: Task 3 Code

```
;---------------------------------------------
;
; Declare gloabals
;
; hand 1, hand 2, deck
;

( defun set-globals ()
  ( setf *hand1* '() )
  ( setf *hand2* '() )
  ( setf *prize-suite* '() )
  ( setf *discard-suite* '() )
  ( setf *deck* ( create-deck ) )
  nil
  )

;---------------------------------------------
;
; functions for task 3
```

```
;
;

( defun deal-cards ()
  ( setf dealing ( deal-order ) )
  ( setf start-deal ( start-card dealing ) )
  ( setf *hand1* ( set-of-cards start-deal 13 '() ) )
  ( setf start-deal ( start-card ( list ( select 1 dealing ) ) ) )
  ( setf *hand2* ( set-of-cards start-deal 13 '() ) )
  ( setf start-deal ( start-card ( list ( select 2 dealing ) ) ) )
  ( setf *prize-suite* ( set-of-cards start-deal 13 '() ) )
  ( setf start-deal ( start-card ( list ( select 3 dealing ) ) ) )
  ( setf *discard-suite* ( set-of-cards start-deal 13 '() ) )
  ( setf *deck* '() )
  nil
  )

( defun set-of-cards ( number length location )
  ( cond
    ( ( equal length 0 )
      location
      )
    ( t
      ( set-of-cards ( + number 1 ) ( - length 1 ) ( snoc ( get-card
number ) location ) )
      )
    )
  )

( defun get-card ( number )
  ( select number *deck* )
```

```
  )

( defun remove-suite ()
  ( setf *discard-suite* '() )
  )

( defun prize-card ( &aux card )
  ( setf card ( top-card ) )
  ( remove-top-card-from-prize-suite )
  card
  )

( defun remove-top-card-from-prize-suite ()
  ( setf *prize-suite* ( take-from ( top-card ) *prize-suite* ) )
  )

( defun top-card ()
  ( select 0 *prize-suite* )
  )

( defun shuffle-suite ()
  ( setf *prize-suite* ( shuffle *prize-suite* ) )
  nil
  )

( defun shuffle ( suite )
  ( cond
    ( ( equal suite nil )
      '()
      )
```

```
    (t
     ( setf card ( pick suite ) )
     ( setf s-suite ( take-from card suite ) )
     ( cons card ( shuffle s-suite ) )
     )
    )
  )
```

Interaction: Task 3 Helper functions

```
;-----------------------------------------------
;
; Helper functions for task 3
;
;
;

( defun start-card ( list &aux number )
  ( setf number ( * ( car list ) 13 ) )
  number
  )


( defun deal-order ( &aux list )
  ( setf list '() )
  ( setf list ( create-list list 4 ) )
  list
  )
```

```
( defun create-list ( list number &aux num )
  ( setf num ( add-to-list list ) )
  ( cond
    ( ( equal ( length list ) number )
      list
      )
    ( ( p-in-list num list )
      ( create-list list number )
      )
    ( t
      ( create-list ( cons num list ) number )
      )
    )
  )

( defun add-to-list ( list &aux number )
  ( setf number ( get-random-number 4 ) )
  ( cond
    ( ( p-in-list number list )
      ( setf number ( get-random-number 4 ) )
      )
    ( ( not ( p-in-list number list ) )
      number
      )
    ( t
      ( add-to-list )
      )
    )
  )
```

```
( defun p-in-list ( number list )
  ( cond
    ( ( equal list nil )
      nil
      )
    ( ( equal number ( car list ) )
      t
      )
    ( t
      ( p-in-list number ( cdr list ) )
      )
    )
  )

( defun get-random-number ( number )
  ( random number )
  )

( defun select ( num l )
  ( cond
    ( ( = num 0 )
      ( car l )
      )
    ( ( > num 0 )
      ( select ( - num 1 ) ( cdr l ) )
      )
    )
  )

( defun pick ( l )
```

```
  ( select ( random ( length l ) ) l )
  )

( defun take-from ( ob lis )
  ( cond
    ( ( equal ( car lis ) nil )
      '()
      )
    ( ( equal ( car lis ) ob )
      ( take-from ob ( cdr lis ) )
      )
    ( t
      ( cons ( car lis ) ( take-from ob ( cdr lis ) ) )
      )
    )
  )
```