AI Assignment: Missionaries and Cannibals Problem Solver

Code

```
( defun mc ()
        ( establish-world )
        ( init-move-list )
        ( make-moves )
)

(defun establish-world ()
        (setf *left-bank* '(M M M C C C B))
(setf *right-bank* '())
)

(defun init-move-list ()
        (setf *move-list* '())
)

( defun make-moves ()
        ( display-world )
        ( cond
                ( ( goalp )
                        ( write-line "Good work!" )
                nil
                )
                ( ( feast-state-p )
                        ( write-line "Yummy yummy yummy, I got Good in my tummy!!")
```

```
                nil
                )
                ( t
                        ( let ( m )
                        ( format t ">>> " ) ( setf m ( read ) )
                                ( if ( applicable-p m )
                                        ( let () ( perform-move m ) ( make-moves ) )
                                ( let () ( write-line "Move inapplicable" ) nil )
                                )
                        )
                )
        )
)


(defun display-world ()
(write-string "*left-bank* ")
(write *left-bank*)
(write-line "")
(write-string "*right-bank* ")
(write *right-bank*)
(write-line "")
)


(defun goalp ()
( cond
        ( (eq (length *right-bank*) 7) T)
)
)
(
```

```
d
e
f
u
n
f
e
a
s
t-
s
t
a
t
e
-
p
()
(
c
o
n
d
        ( ( and ( > ( count 'M *left-bank* ) 0 ) ( > ( count 'C *left-bank* ) (
count 'M *left-bank* ) ) ) T )
        ( ( and ( > ( count 'M *right-bank* ) 0 ) ( > ( count 'C *right-bank* ) (
count 'M *right-bank* ) ) ) T )
)
)
```

```
( defun applicable-p ( move )
( cond
        ( ( or ( < ( length move ) 0 ) ( > ( length move ) 3 ) ) nil )
        ( ( not ( member 'b move ) ) nil )
        ( ( > ( count 'b move ) 1) nil )
        ( t
                ( cond
                        ( ( equal ( first move ) ( second move ) )
                                ( > ( count ( first move ) ( current-bank ) ) 1 ) )
                        ( ( equal ( first move ) ( third move ) )
                                ( > ( count ( first move ) ( current-bank ) ) 1 ) )
                        ( ( equal ( second move ) ( third move ) )
                                ( > ( count ( second move ) ( current-bank ) ) 1 ) )
                        ( ( and ( member ( car move ) ( current-bank ) ) ( member ( cadrmove ) ( current-bank) )
                        )
T )
                        ( t nil )
                )
        )
)
)
(
d
e
f
u
n
p
e
rf
o
```

r

m

-

m

o

v

e

(

m

o

v

e

)

```
        ( setf *move-list* ( snoc move *move-list* ) )
        ( if ( equal ( current-bank ) *left-bank* )
        ( move-lr move )
        ( move-rl move )
        )
)

( defun current-bank ()
( cond
        ( ( member 'b *left-bank* ) *left-bank* )
        ( t *right-bank* )
)
)

(defun snoc (o l)
        (cond
                ((null l)
                        (list o)
```

```
        )
        (t
                (cons (car l)(snoc o (cdr l)))
        )
        )
)


( defun move-lr ( ml )
        ( if ( null ml ) ( return-from move-lr ) )
        ( move-lr-1 ( first ml ) )
        ( move-lr ( rest ml ) )
)

(defun move-lr-1 (ml)
        (setf *left-bank* (remove ml *left-bank* :count 1))
        (setf *right-bank* (cons ml *right-bank*))
)

( defun move-rl ( ml )
        ( if ( null ml ) ( return-from move-rl ) )
        ( move-rl-1 ( first ml ) )
        ( move-rl ( rest ml ) )
)

(defun move-rl-1 (ml)
        (setf *right-bank* (remove ml *right-bank* :count 1))
        (setf *left-bank* (cons ml *left-bank*))
)

(defun display-solution ()
```

```
(display *move-list*)
)

( defun display ( move-list )
( cond
        ( ( equal move-list '() ) )
        ( t
                ( format t "~A~%" ( car move-list ) )
                ( display ( cdr move-list ) )
        )
)
)
```

Demo

```
[1]> (load "mc.l")
;; Loading file mc.l ...
;; Loaded file mc.l
#P"/home/asigdel/public_html/CSC416WorkSite/mc.l".
[2]> (mc)
*left-bank* (MM MCCCB)
*right-bank* NIL
>>> (ccb)
*left-bank* (M MMC)
*right-bank* (BCC)
>>> (cb)
*left-bank* (BCMMMC)
*right-bank* (C)
>>> (bcc)
*left-bank* (MMM)
```

right-bank (CC BC)

>>> (b c)

left-bank (CBMMM)

right-bank (CC)

>>> (mm b)

left-bank (CM)

right-bank (BM MCC)

>>> (m bc)

left-bank (CBM CM)

right-bank (MC)

>>> (mmb)

left-bank (CC)

right-bank (B MMMC)

>>> (bc)

left-bank (CBCC)

right-bank (MMM)

>>> (cc b)

left-bank (C)

right-bank CB C CMMM)

>>> (b c)

left-bank (CBC)

right-bank (CM MM)

>>> (b c c)

left-bank NIL *right-

bank* (CC BCMMM) Good

work!

NIL

[3]> (mc)

left-bank (MM MCCCB)

right-bank NIL

>>> (mmb)

left-bank (MCCC)

right-bank (B MM)

Yummy yummy yummy, I got Good in my tummy!! NIL [4]> (display-solution) CM MB)

T