Task 5: Heuristic GOPS Machine

Abstract: Code was split up into multiple files as to not make the AI_Project file unwieldy. I moved tasks 2 through 4 into a different file (first_iter.l) that is not included here. The later tasks will be included in the file sec_iter.l

Game.l file contains code to run the game mostly re-writing the code from previous tasks to use CLOS.

Hm_rules.l file contains the heuristics. Currently there are two rules the first being the random machine and the second is a machine that plays the same card as the prize card.

Player.l contains the code for the current model of the players. Currently only machine players are modeled. Later I will add a human player model so a person can play against a machine.

Demos.l file contains the demo code.

Sec_iter.l contains more code for task 5. Mainly to the function to define a heuristic machine vs a heuristic machine. (Matching-card machine vs Random-machine).

Demo:

```
[3]> ( demo--task5 )

>>> Demo for task 5 <<<

Name of machine 1? m1

Name of machine 2? m2


>>>--------- Round:  1 ---------<<<

--- Prize Card = (ACE . HEART)

--- M1's Card = (ACE . DIAMOND)

--- M2's Card = (9 . SPADE)

--- M2  won ---

--- M1 Score is: 0.0

--- M2 Score is: 1.0


>>>--------- Round:  2 ---------<<<

--- Prize Card = (10 . HEART)
```

--- M1's Card = (10 . DIAMOND)

--- M2's Card = (6 . SPADE)

--- M1 won ---

--- M1 Score is: 10.0

--- M2 Score is: 1.0

>>>--------- Round: 3 ---------<<<

--- Prize Card = (9 . HEART)

--- M1's Card = (9 . DIAMOND)

--- M2's Card = (2 . SPADE)

--- M1 won ---

--- M1 Score is: 19.0

--- M2 Score is: 1.0

>>>--------- Round: 4 ---------<<<

--- Prize Card = (3 . HEART)

--- M1's Card = (3 . DIAMOND)

--- M2's Card = (KING . SPADE)

--- M2 won ---

--- M1 Score is: 19.0

--- M2 Score is: 4.0

>>>--------- Round: 5 ---------<<<

--- Prize Card = (5 . HEART)

--- M1's Card = (5 . DIAMOND)

--- M2's Card = (ACE . SPADE)

--- M1 won ---

--- M1 Score is: 24.0

--- M2 Score is: 4.0

>>>--------- Round: 6 ---------<<<

--- Prize Card = (2 . HEART)

--- M1's Card = (2 . DIAMOND)

--- M2's Card = (8 . SPADE)

--- M2 won ---

--- M1 Score is: 24.0

--- M2 Score is: 6.0

>>>--------- Round: 7 ---------<<<

--- Prize Card = (KING . HEART)

--- M1's Card = (KING . DIAMOND)

--- M2's Card = (QUEEN . SPADE)

--- M1 won ---

--- M1 Score is: 37.0

--- M2 Score is: 6.0

>>>--------- Round: 8 ---------<<<

--- Prize Card = (4 . HEART)

--- M1's Card = (4 . DIAMOND)

--- M2's Card = (7 . SPADE)

--- M2 won ---

--- M1 Score is: 37.0

--- M2 Score is: 10.0

>>>--------- Round: 9 ---------<<<

--- Prize Card = (7 . HEART)

--- M1's Card = (7 . DIAMOND)

--- M2's Card = (10 . SPADE)

--- M2 won ---

--- M1 Score is: 37.0

--- M2 Score is: 17.0

>>>--------- Round: 10 ---------<<<

--- Prize Card = (8 . HEART)

--- M1's Card = (8 . DIAMOND)

--- M2's Card = (4 . SPADE)

--- M1 won ---

--- M1 Score is: 45.0

--- M2 Score is: 17.0

>>>--------- Round: 11 ---------<<<

--- Prize Card = (QUEEN . HEART)

--- M1's Card = (QUEEN . DIAMOND)

--- M2's Card = (5 . SPADE)

--- M1 won ---

--- M1 Score is: 57.0

--- M2 Score is: 17.0

>>>--------- Round: 12 ---------<<<

--- Prize Card = (JACK . HEART)

--- M1's Card = (JACK . DIAMOND)

--- M2's Card = (3 . SPADE)

--- M1 won ---

--- M1 Score is: 68.0

--- M2 Score is: 17.0

>>>--------- Round: 13 ---------<<<

--- Prize Card = (6 . HEART)

--- M1's Card = (6 . DIAMOND)

--- M2's Card = (JACK . SPADE)

--- M2 won ---

--- M1 Score is: 68.0

--- M2 Score is: 23.0

--- M1 won the game! ---

Name of machine 1? Rosie

Name of machine 2? Robert

>>>--------- Round: 1 ---------<<<

--- Prize Card = (4 . DIAMOND)

--- ROSIE's Card = (4 . HEART)

--- ROBERT's Card = (9 . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 0.0

--- ROBERT Score is: 4.0

>>>--------- Round: 2 ---------<<<

--- Prize Card = (7 . DIAMOND)

--- ROSIE's Card = (7 . HEART)

--- ROBERT's Card = (10 . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 0.0

--- ROBERT Score is: 11.0

>>>--------- Round: 3 ---------<<<

--- Prize Card = (6 . DIAMOND)

--- ROSIE's Card = (6 . HEART)

--- ROBERT's Card = (8 . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 0.0

--- ROBERT Score is: 17.0

>>>--------- Round: 4 ---------<<<

--- Prize Card = (3 . DIAMOND)

--- ROSIE's Card = (3 . HEART)

--- ROBERT's Card = (4 . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 0.0

--- ROBERT Score is: 20.0

>>>--------- Round: 5 ---------<<<

--- Prize Card = (JACK . DIAMOND)

--- ROSIE's Card = (JACK . HEART)

--- ROBERT's Card = (7 . SPADE)

--- ROSIE won ---

--- ROSIE Score is: 11.0

--- ROBERT Score is: 20.0

>>>--------- Round: 6 ---------<<<

--- Prize Card = (2 . DIAMOND)

--- ROSIE's Card = (2 . HEART)

--- ROBERT's Card = (5 . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 11.0

--- ROBERT Score is: 22.0

>>>--------- Round: 7 ---------<<<

--- Prize Card = (8 . DIAMOND)

--- ROSIE's Card = (8 . HEART)

--- ROBERT's Card = (2 . SPADE)

--- ROSIE won ---

--- ROSIE Score is: 19.0

--- ROBERT Score is: 22.0

>>>--------- Round: 8 ---------<<<

--- Prize Card = (QUEEN . DIAMOND)

--- ROSIE's Card = (QUEEN . HEART)

--- ROBERT's Card = (3 . SPADE)

--- ROSIE won ---

--- ROSIE Score is: 31.0

--- ROBERT Score is: 22.0

>>>--------- Round: 9 ---------<<<

--- Prize Card = (ACE . DIAMOND)

--- ROSIE's Card = (ACE . HEART)

--- ROBERT's Card = (QUEEN . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 31.0

--- ROBERT Score is: 23.0

>>>--------- Round: 10 ---------<<<

--- Prize Card = (10 . DIAMOND)

--- ROSIE's Card = (10 . HEART)

--- ROBERT's Card = (6 . SPADE)

--- ROSIE won ---

--- ROSIE Score is: 41.0

--- ROBERT Score is: 23.0

>>>--------- Round: 11 ---------<<<

--- Prize Card = (KING . DIAMOND)

--- ROSIE's Card = (KING . HEART)

--- ROBERT's Card = (KING . SPADE)

--- The round ended in a draw ---

--- ROSIE Score is: 47.5

--- ROBERT Score is: 29.5

>>>--------- Round: 12 ---------<<<

--- Prize Card = (5 . DIAMOND)

--- ROSIE's Card = (5 . HEART)

--- ROBERT's Card = (ACE . SPADE)

--- ROSIE won ---

--- ROSIE Score is: 52.5

--- ROBERT Score is: 29.5

>>>--------- Round: 13 ---------<<<

--- Prize Card = (9 . DIAMOND)

--- ROSIE's Card = (9 . HEART)

--- ROBERT's Card = (JACK . SPADE)

--- ROBERT won ---

--- ROSIE Score is: 52.5

--- ROBERT Score is: 38.5

--- ROSIE won the game! ---

Name of machine 1? M101

Name of machine 2? M202

>>>--------- Round: 1 ---------<<<

--- Prize Card = (2 . DIAMOND)

--- M101's Card = (2 . HEART)

--- M202's Card = (4 . CLUB)

--- M202 won ---

--- M101 Score is: 0.0

--- M202 Score is: 2.0

>>>--------- Round: 2 ---------<<<

--- Prize Card = (9 . DIAMOND)

--- M101's Card = (9 . HEART)

--- M202's Card = (3 . CLUB)

--- M101 won ---

--- M101 Score is: 9.0

--- M202 Score is: 2.0

>>>--------- Round: 3 ---------<<<

--- Prize Card = (10 . DIAMOND)

--- M101's Card = (10 . HEART)

--- M202's Card = (8 . CLUB)

--- M101 won ---

--- M101 Score is: 19.0

--- M202 Score is: 2.0

>>>--------- Round: 4 ---------<<<

--- Prize Card = (ACE . DIAMOND)

--- M101's Card = (ACE . HEART)

--- M202's Card = (10 . CLUB)

--- M202 won ---

--- M101 Score is: 19.0

--- M202 Score is: 3.0

>>>--------- Round: 5 ---------<<<

--- Prize Card = (JACK . DIAMOND)

--- M101's Card = (JACK . HEART)

--- M202's Card = (QUEEN . CLUB)

--- M202 won ---

--- M101 Score is: 19.0

--- M202 Score is: 14.0

>>>--------- Round: 6 ---------<<<

--- Prize Card = (QUEEN . DIAMOND)

--- M101's Card = (QUEEN . HEART)

--- M202's Card = (2 . CLUB)

--- M101 won ---

--- M101 Score is: 31.0

--- M202 Score is: 14.0

>>>--------- Round: 7 ---------<<<

--- Prize Card = (7 . DIAMOND)

--- M101's Card = (7 . HEART)

--- M202's Card = (7 . CLUB)

--- The round ended in a draw ---

--- M101 Score is: 34.5

--- M202 Score is: 17.5

>>>--------- Round: 8 ---------<<<

--- Prize Card = (3 . DIAMOND)

--- M101's Card = (3 . HEART)

--- M202's Card = (9 . CLUB)

--- M202 won ---

--- M101 Score is: 34.5

--- M202 Score is: 20.5

>>>--------- Round: 9 ---------<<<

--- Prize Card = (KING . DIAMOND)

--- M101's Card = (KING . HEART)

--- M202's Card = (KING . CLUB)

--- The round ended in a draw ---

--- M101 Score is: 41.0

--- M202 Score is: 27.0

>>>--------- Round: 10 ---------<<<

--- Prize Card = (4 . DIAMOND)

--- M101's Card = (4 . HEART)

--- M202's Card = (ACE . CLUB)

--- M101 won ---

--- M101 Score is: 45.0

--- M202 Score is: 27.0

>>>--------- Round: 11 ---------<<<

--- Prize Card = (8 . DIAMOND)

--- M101's Card = (8 . HEART)

--- M202's Card = (5 . CLUB)

--- M101 won ---

--- M101 Score is: 53.0

--- M202 Score is: 27.0

>>>--------- Round: 12 ---------<<<

--- Prize Card = (6 . DIAMOND)

--- M101's Card = (6 . HEART)

--- M202's Card = (JACK . CLUB)

--- M202 won ---

--- M101 Score is: 53.0

--- M202 Score is: 33.0

>>>--------- Round: 13 ---------<<<

--- Prize Card = (5 . DIAMOND)

--- M101's Card = (5 . HEART)

--- M202's Card = (6 . CLUB)

--- M202 won ---

--- M101 Score is: 53.0

--- M202 Score is: 38.0

--- M101 won the game! ---

>>> Finished Demo for task 5 <<<

NIL

Code AI_Project.l File:

```
;--------------------------------------------
; Name: AI Project: Heuristic Machine
;                          for the
;                  Game of Pure Strategy
; File: AI_Project.l
; Description: A program that will feature Heuristic Machines to play
;              GOPS.
;--------------------------------------------




;--------------------------------------------
;
;  Load code from files
;  helpers - helper functions
;  demo - demos for the tasks
;
;


( defun start ( &optional num )
  ( format t ">>> Loading files.~%" )
```

```
  ( load "library/helpers.l" )
  ( load "library/demos.l" )
  ( load "library/player.l" )
  ( load "library/hm_rules.l" )
  ( format t ">>> Finished loading.~%" )
  ( format t ">>> type ( rund ) to run all demos.~%" )
  ( format t ">>> type ( demo--task# ) to demo a specific task.~%" )
  ( cond
    ( ( equal num 1 )
      ( format t ">>> Loading first four tasks~%" )
      ( load "library/first_iter.l")
     )
    ( t
      ( format t ">>> Loading CLOS objects~%" )
      ( load "library/game.l" )
      ( load "library/sec_iter.l" )
     )
   )
  nil
  )
```

Code sec_iter.l File:

```
;-----------------------------------------------
;
; Task 5:
;
; set-up two machines to play
;
;-----------------------------------------------
```

```
( defun hm-hm-game ( &aux player1 player2 p1name p2name game )
  ( princ "Name of machine 1? " ) ( setf p1name ( read ) )
  ( princ "Name of machine 2? " ) ( setf p2name ( read ) )
  ( setf player1 ( make-instance 'h-machine-player :name p1name ) )
  ( setf player2 ( make-instance 'h-machine-player :name p2name ) )
  ( set-machine-rules player1 'hm-rule01-stg01 )
  ( setf game
      ( make-instance 'game
                  :player1 player1
                  :player2 player2
                  )
      )
  ( init-cards game )
  game
  )
```

Code demos.l File:

```
;-------------------------------------------
;
;
;
;  Task 5 Demo
;
;
```

```
( defun demo--task5 ()
  ( format t ">>> Demo for task 5 <<< ~%" )
  ( setf g ( hm-hm-game ) )
  ( play g )
  ( setf g ( hm-hm-game ) )
  ( play g )
  ( setf g ( hm-hm-game ) )
  ( play g )
  ( format t ">>> Finished Demo for task 5 <<< ~%" )
  nil
  )
```

Code hm_rules.l File:

```
;-----------------------------------------------
;
; Task 5:
;
; Heuristic rules for the machine
;
;-----------------------------------------------


; format for arguments is take value of the card ( number ) and the
machine
; player's hand
; default rule is random.



; value will be unused in random as it is not needed.
( defun random-rules ( value hand ( p player ) &aux my-card )
  ( setf my-card ( a-random-card hand p ) )
```

```
  ;( display-hand p )
  my-card
  )

;supporting random rules
( defun a-random-card ( hand ( p player ) &aux card number clcr )
  ( setf clcr ( get-cards-left ) )
  ( setf number ( + ( get-random-number clcr ) 1 ) )
  ;( format t "--- Number  = ~A~%" number )
  ( setf card ( select ( - number 1 ) hand ) )
  ( display-card card p )
  card
  )

; Used by the players to pick a card from the cards left.
; After the round is over the hand shrinks. This tells us
; How many cards are left to pick from.
; Formatted so if one is a number from the random player
; the card picked will be the first element from the list
; instead of the second.
( defun get-cards-left ( &aux round cards-left )
  ( setf round *current-round* )
  ( setf cards-left ( - 14 *current-round* ) )
  ;( format t "--- Cards-left  = ~A~%" cards-left )
  cards-left
  )



; simple heuristic rule to pick the card
```

```
; Pick the card that has the same value as the prize card


( defun hm-rule01-stg01 ( value hand ( p player ) &aux my-card number
)
  ( setf my-card ( select 0 hand ) )
;  ( format t "--- Value: ~A~%" value )
;  ( format t "--- Hand: ~A~%" hand )
;  ( format t "--- My-card: ~A~%" my-card )
  ( dotimes ( i ( get-cards-left ) )
    ( setf my-card ( select i hand ) )
    ( cond
      ( ( equal ( value-of my-card ) value )
      ( display-card my-card p )
      ( return-from hm-rule01-stg01 my-card )
     )
      )
    )
;  my-card
  )
```

Code player.l File:

```
;-----------------------------------------------
;
; Task 5:
; Infrastructure
; Modelling a player
;
;-----------------------------------------------
```

```
( defclass player ()
  (
   ( name :accessor player-name :initarg :name )
   ( hand :accessor player-hand :initarg :hand :initform '() )
   ( winnings :accessor player-winnings :initarg :winnings :initform
'() )
   ( score :accessor player-score :initarg :score :initform 0.0 )
   )
  )


( defclass h-machine-player ( player )
  (
   ( rules :accessor h-machine-player-rules :initarg :rules :initform
'random-rules )
   ;( hand :accessor h-machine-player-hand :initarg :hand :initform
'() )
   )
  )


( defmethod display ( ( p player ) )
  ( princ "< Player Name = " )
  ( prin1 ( player-name p ) )
  ( princ " >" )
  ( terpri )
  )


( defmethod display-card ( ( p player ) card )
  ( format t "--- ~A's card: ~A ~A's score: ~A ~%" ( player-name p )
card ( player-name p ) ( player-score p ) )
  )
```

```
( defmethod display-hand ( ( hm h-machine-player ) )
  ( princ "< My hand is: " )
  ( prin1 ( player-hand hm ) )
  ( princ " >" )
  ( terpri )
  )

( defmethod set-machine-rules ( ( hm h-machine-player ) hm-rules )
  ( setf ( h-machine-player-rules hm ) hm-rules )
  )

( defmethod set-hand ( ( p player ) p-hand )
  ( setf ( player-hand p ) p-hand )
  )

( defmethod set-winnings ( ( p player ) value )
  ( setf ( player-winnings p ) ( cons value ( player-winnings p ) ) )
  )

( defmethod set-score ( ( p player ) &optional draw-total )
  ( setf ( player-score p ) ( + ( sum ( player-winnings p ) ) draw-
total ) )
  )

( defmethod get-score ( ( p player ) )
  ( player-score p )
  )

;-------------------------------------------
;
```

```
;
; Playing a card
;
;
;-----------------------------------------------



( defmethod play-card ( ( p h-machine-player ) p-card &aux value card
)
  ( setf value ( value-of p-card ) )
  ( setf card ( funcall ( h-machine-player-rules p ) value ( player-
hand p ) p ) )
  card
  )
```

Code game.l File:

```
;-----------------------------------------------
;
; Task 5:
; Infrastructure
; Setup a game
;
;-----------------------------------------------



( defclass game ()
  (
   ( player1 :accessor game-player1 :initarg :player1 )
   ( player2 :accessor game-player2 :initarg :player2 )
   )
```

```
  )

;-----------------------------------------------
;
;
;
; Class Functions
;
;-----------------------------------------------

( defmethod play ( ( g game ) )
  ( loop
    ( get-prize-card )
    ( play-round g )
    ( if ( game-over-p g ) ( return-from play ( wind-up-game g ) ) )
    )
  )

( defmethod wind-up-game ( ( g game ) )
  ( display-results g )
  nil
  )

;-----------------------------------------------
;
;
;
; Round Functions
;
;-----------------------------------------------
```

```
; set globals, deal the cards to the hands and shuffle the prize suite
( defmethod init-cards ( ( g game ) )
  ( set-globals )
  ( deal-cards ( game-player1 g ) ( game-player2 g ) )
  ( shuffle-suite )
  nil
  )
( defmethod play-round ( ( g game ) )
  ( display-current-round )
  ( display-prize-card *prize-card* )


;  ( display-hand ( game-player1 g ) )
;  ( display-hand ( game-player2 g ) )
  ( play-cards ( game-player1 g ) ( game-player2 g ) )
;  ( display-hand ( game-player1 g ) )
;  ( display-hand ( game-player2 g ) )

  ( display-scores g ( sum-draw-winnings ) )
  ( setf *current-round* ( + *current-round* 1 ) )
  nil
  )

( defmethod game-over-p ( ( g game ) )
  ( equal ( length *prize-suite* ) 0 )
  )

( defmethod who-won ( card1 card2 prize-card ( p1 player ) ( p2 player
) &aux val1 val2 )
  ( setf val1 ( value-of card1 ) )
```

```
  ( setf val2 ( value-of card2 ) )
  ( cond
    ( ( > val1 val2 )
      ( display-player1-won p1 )
      ( add-prize-to-player p1 prize-card )
      )
    ( ( > val2 val1 )
      ( display-player2-won p2 )
      ( add-prize-to-player p2 prize-card )
      )
    ( t
      ( round-winner-draw )
      ( add-prize-draw prize-card )
      )
    )
  nil
  )



;-----------------------------------------------
;
;
; Dealing Functions
;
;-----------------------------------------------

;---------------------------
;
```

```
; Declare gloabals
;
;

( defmethod set-globals ()
  ( setf *prize-suite* '() )
  ( setf *prize-card* '() )
  ( setf *discard-suite* '() )
  ( setf *current-round* 1 )
  ( setf *deck* ( create-deck ) )
  ( setf *draw-winnings* '() )
  nil
  )

( defmethod create-deck ()
  ( mapcan #'create-cards '( club diamond spade heart ) )
  )

( defmethod deal-cards ( ( p1 player ) ( p2 player ) )
  ( setf dealing ( deal-order ) )
  ( setf start-deal ( start-card dealing ) )
  ( set-hand p1 ( set-of-cards start-deal 13 '() ) )
  ( setf start-deal ( start-card ( list ( select 1 dealing ) ) ) )
  ( set-hand p2 ( 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
```

```
  )

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

( defmethod shuffle ( suite &aux card s-suite )
  ( cond
    ( ( equal suite nil )
      '()
     )
    (t
     ( setf card ( pick suite ) )
     ( setf s-suite ( take-from card suite ) )
     ( cons card ( shuffle s-suite ) )
    )
   )
  )

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

;-----------------------------------------------
;
;
; Card Functions
;
```

```
;---------------------------------------------

( defmethod create-cards ( suite &aux ranks )
  ( setf ranks '( ace 2 3 4 5 6 7 8 9 10 jack queen king ) )
  ( setf suite-duplicates ( duplicate ( length ranks ) suite ) )
  ( mapcar #'cons ranks suite-duplicates )
  )

( defmethod play-cards ( ( p1 player ) ( p2 player ) &aux card1 card2
)
  ( setf card1 ( play-the-card p1 ) )
  ( setf card2 ( play-the-card p2 ) )
  ( who-won card1 card2 *prize-card* p1 p2 )
  ( remove-cards card1 card2 p1 p2 )
  nil
  )

( defmethod play-the-card ( ( p player ) &aux card )
  ( setf card ( play-card p *prize-card* ) )
  card
  )

( defmethod get-prize-card ()
  ( set-prize-card ( prize-card ) )
  nil
  )

( defmethod remove-cards ( card1 card2 ( p1 player ) ( p2 player ) )
  ( set-hand p1 ( take-from card1 ( player-hand p1 ) ) )
  ( set-hand p2 ( take-from card2 ( player-hand p2 ) ) )
  nil
```

```
  )

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

( defmethod get-card ( number )
  ( select number *deck* )
  )



( defmethod set-prize-card ( card )
  ( setf *prize-card* card )
  )

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

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

```
  )

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

; gets the value of the card. Card is processed from ( rank.suite ).
; Returns a value for the rank
( defmethod value-of ( card &aux part number )
  ( setf part ( car card ) )
  ( cond
    ( ( equal part 'ace )
      ( setf number 1 )
     )
    ( ( equal part 2 )
      ( setf number 2 )
     )
    ( ( equal part 3 )
      ( setf number 3 )
     )
    ( ( equal part 4 )
      ( setf number 4 )
     )
    ( ( equal part 5 )
      ( setf number 5 )
     )
    ( ( equal part 6 )
      ( setf number 6 )
     )
    ( ( equal part 7 )
```

```
      ( setf number 7 )
      )
    ( ( equal part 8 )
      ( setf number 8 )
      )
    ( ( equal part 9 )
      ( setf number 9 )
      )
    ( ( equal part 10 )
      ( setf number 10 )
      )
    ( ( equal part 'jack )
      ( setf number 11 )
      )
    ( ( equal part 'queen )
      ( setf number 12 )
      )
    ( ( equal part 'king )
      ( setf number 13 )
      )
    )
  number
  )

;-----------------------------------------------
;
;
;
; Player Functions
;
```

```
;---------------------------------------------

( defmethod add-prize-to-player ( ( p player ) card )
  ( set-winnings p ( value-of card ) )
  )


;---------------------------------------------
;
;
;
; Draw Functions
;
;---------------------------------------------

; Add the value of the prize card to the draw score to be summed
; and divided by 2 for each player.
; The score is a list that is summed after each round.
( defmethod add-prize-draw ( card )
  ( setf *draw-winnings* ( cons ( value-of card ) *draw-winnings* ) )
  nil
  )

( defmethod sum-draw-winnings ( &aux score )
  ( setf score ( float ( / ( sum *draw-winnings* ) 2 ) ) )
  score
  )

;---------------------------------------------
;
```

```
;
;
; Display Functions
;
;-----------------------------------------------

; formating for nice display to the user.
( defmethod display-player1-won ( ( p player ) )
  ( format t "--- ~A won --- ~%" ( player-name p ) )
  nil
  )

; formating for nice display to the user.
( defmethod display-player2-won ( ( p player ) )
  ( format t "--- ~A  won --- ~%" ( player-name p ) )
  nil
  )

; formating for nice display to the user.
( defmethod round-winner-draw ()
  ( format t "--- The round ended in a draw --- ~%" )
  nil
  )

( defmethod display-card ( card ( p player ) )
  ( format t "--- ~A's Card = ~A~%" ( player-name p ) card )
  )

; calculate the score of each player and display the scores.
; Score is total rounds won ( get value of the prize card add to total
)
```

```
; take the rounds ending in a draw then sum the draws and divide by 2.
; Giving each player half the points.
( defmethod display-scores ( ( g game ) &optional draw )
  ( set-score ( game-player1 g ) draw )
  ( set-score ( game-player2 g ) draw )
  ( format t "--- ~A Score is: ~A~%" ( player-name ( game-player1 g ) ) ( get-score ( game-player1 g ) ) )
  ( format t "--- ~A Score is: ~A~%" ( player-name ( game-player2 g ) ) ( get-score ( game-player2 g ) ) )
  nil
  )


; checks to see which player won the game. Total higher score wins.
( defun display-results ( ( g game ) )
  ( cond
    ( ( > ( get-score ( game-player1 g ) ) ( get-score ( game-player2 g ) ) )
      ( format t "--- ~A won the game! --- ~%" ( player-name ( game-player1 g ) ) )
      )
    ( ( > ( get-score ( game-player2 g ) ) ( get-score ( game-player1 g ) ) )
      ( format t "--- ~A won the game! --- ~%" ( player-name ( game-player2 g ) ) )
      )
    ( t
      ( format t "--- The game ended in a draw. --- ~%" )
      )
    )
  nil
  )
```