O&P 2014

VENTTI IN ANSI COMMON LISP

Markus Viljanen

TABLE OF CONTENTS

Contents

Game Idea

GAME RULES

The player plays his hand against the House's hand. The sum of the cards in a hand determine victory, and the party with higher sum less or equal to 21 wins. Turns alternate, and at each turn a party may draw one card from the deck or opt out. A party who has opted out cannot opt back in. A party who has a sum over 21 has overdrafted and thus lost. A party who draws so that the sum adds to 21 has won, with no tie-in turn. Unless aforementioned events occur, game continues until both parties have opted out. In the event of an equal sum, the House wins.

CALCULATION OF THE SUM:

- Cards above 10 (with pictures) have a value of 10.
- Ace has, at the party's leisure, a value of either 1 or 10.
- Other cards have their normal values.

Ace Remainder Arithmetic

Due to the rules, a hand with aces has several possible values. We define such hand to sum to the value which has the largest value less than or equal to 21. Or if this is not possible, the smallest number over 21. This uniquely defines the optimal choice of ace values for each hand. A party does not thus have to specify the valuation, but it is up to them to interpret the possible alternate base sums for a draw.

The possible sums values for k aces are:

$$\{10i + 1(k - i) : i \in \{0, \dots, k\}\}$$

There are thus k unique sums for k aces. For different k and i, the sums are for example, with coloring with respect to achieving 21:

$i \backslash k$	1	2	3	4
4				40
3			30	31
2		20	21	22
1	10	11	12	13
0	1	2	3	4

However in a hand we can have other cards as well. Define the other cards sum as x:

$$x = \text{sum}(\{i : i \in Hand \wedge i \neq 1\})$$

And let c be the remainder:

$$c = \text{sum}(\{i : i \in Hand\}) - x$$

Then we want i^* such that:

$$i^* = \max(\{i : i \in \{0, \dots, k\} \wedge 10i + 1(k - i) \leq c\})$$

Solving the inequality:

$$\begin{aligned} 10i + 1(k - i) &\leq c \\ 9i + k &\leq c \\ i &\leq \frac{c - k}{9} \end{aligned}$$

Thus:

ACE REMAINDER ARITHMETIC

$$i^* = \min(\left\lfloor \frac{c-k}{9} \right\rfloor, k)$$

For c, from the definition of an optimal hand, we have:

$$x + k \leq 21 \Rightarrow c = 21 - x$$
$$x + k > 21 \Rightarrow c = k$$

Letting $s = \text{sum}(\{i : i \in Hand\})$ and solving for x:

$$s = x + k$$
$$x = s - k$$

We obtain new expressions for c:

$$x + k \leq 21 \Rightarrow c = 21 - (s - k) = k + (21 - s)$$
$$x + k > 21 \Rightarrow c = k$$

And thus:

$$c = k + \max(\{21 - s\}, 0\})$$

Which is much nicer since it replaces programming language *if*s with a mathematical function.
Programming this in Lisp:

```
(defun sum (hand)
      (let ((x (reduce #'+ (set-difference (normalize (nums hand)) '(1))) )
             (k (count 1 (nums hand)) ))
            (+ x (ace-remainder k (+ k (max (- 21 (+ x k)) 0)) ) )
      )
)

(defun normalize (hand)
      (mapcar (lambda (n) (if (> n 10) 10 n)) hand)
)

(defun nums (hand)
      (mapcar #'car hand)
)

(defun ace-remainder (k c)
      (let ((i (min (floor (/ (- c k ) 9)) k) ))
            (+ (* 10 i) (- k i) )
      )
)
```

House Rules

HOUSE RULES

A statistically optimal strategy, it is claimed but not proven here, is to draw when the sum is less than or equal to 16 and opt out otherwise. Of course, we have to take into account the fact that if the player has a larger hand, we would lose opting out since the players next move would be to opt out, so we have to draw regardless of the sum. If this is not the case, so that we have the larger hand, and player has opted out, there is no use risking the victory regardless of the sum, and we should opt out.

In the game logic we check for the player having opted out, and if this is the case, we continue the move cycle recursively until the cycle opts out. The player is proactive about calling his move functions, so we do not need to explicitly relinquish the turn, but he is not proactive about calling the House move functions so we need to do this if he has no moves left which would call them:

```
(defun house-move ()
      (if (< (sum *house*) (sum *hand*))
            (house-deal)
            (if (and (<= (sum *house*) 16) (not *player-out*))
                  (house-deal)
                  (house-out)
            )
      )
)

(defun house-out ()
      (setf *house-out* T)
      ;(print-state)
      (if (check-state)
            (print-over)
            (if *player-out* (house-move))
      )
)

(defun house-deal ()
      (push (pop *deck*) *house*)
      (print-state)
      (if (check-state)
            (print-over)
            (if *player-out* (house-move))
      )
)
```

User Interface

PLAYER FUNCTIONS

The player move functions are like House move functions, calling the House to move afterwards if it has not opted out. The game is played by calling either of *(deal)* or *(out*); at player's choice:

```
(defun deal ()
      (push (pop *deck*) *hand*)
      (print-state)
      (if (check-state)
            (print-over)
            (if (not *house-out*) (house-move))
      )
)

(defun out ()
      (setf *player-out* T)
      ;(print-state)
      (if (check-state)
            (print-over)
            (if (not *house-out*) (house-move))
      )
)
```

PRINT HELPERS

(print-state) prints the current state, the House's hand and sum, and the player's hand and sum, respectively.

(print-over) prints the winner when called.

(print-help) prints available commands: *(new-game) (deal) (out) (quit)*.

OTHER HELPERS

(check-state) checks if the game is over according to the rules, returning *T* or *nil*,

(make-deck) makes a new deck of cards, which is a list of 52 *cons cells*, each containing a value and a suit.

(shuffle list) is Knuth's shuffling algorithm for shuffling the deck, copied from the web.

Playing the game

The Lisp implementation the game was tested in is clisp. To run the game with it:

```
clisp .\ventti.lisp
```

This launches a very simple read-eval-loop.

Alternatively, after disabling the automatic execution of the main-loop, the game can also be played from the Clisp REPL:

```
(progn (print-help) (main-loop)) -> ;(progn (print-help) (main-loop))
```

And then type in the command line:

```
clisp
[1]>(load "ventti.lisp")
[2]>(new-game)
...
```

which may be useful for debugging.

Example:

```
PS C:\Users\m\Desktop> clisp .\ventti.lisp
Commands: (new-game) (deal) (out) (quit)
(new-game)

(GAME STATE)
(HOUSE HAS (10 . 2) (9 . 2))
(= . 19)
(PLAYER HAS (1 . 2) (3 . 4))
(= . 13)
(deal)

(GAME STATE)
(HOUSE HAS (10 . 2) (9 . 2))
(= . 19)
(PLAYER HAS (3 . 1) (1 . 2) (3 . 4) )
(= . 16)
(deal)

(GAME STATE)
(HOUSE HAS (10 . 2) (9 . 2))
```

PLAYING THE GAME

```
(= . 19)
(PLAYER HAS (1 . 1) (3 . 1) (1 . 2) (3 . 4) )
(= . 17)
(deal)

(GAME STATE)
(HOUSE HAS (10 . 2) (9 . 2))
(= . 19)
(PLAYER HAS (10 . 4) (1 . 1) (3 . 1) (1 . 2) (3 . 4) )
(= . 18)
(deal)

(GAME STATE)
(HOUSE HAS (10 . 2) (9 . 2))
(= . 19)
(PLAYER HAS (7 . 3) (10 . 4) (1 . 1) (3 . 1) (1 . 2) (3 . 4) )
(= . 25)

(HOUSE WINS !)
```

We lost. But on this day of sorrow, we can be soothed by the fact that the ace arithmetic, which was arguably the most complicated part of the program, works well.

Final thoughts

LISPY AND UNLISPY PROGRAMMING

There were some **lispy** elements in the implementation:

- Most things are symbols or numbers in a list.
- Many operations applied a function over these lists.
- Some operations assume the underlying structure of the cons cells.
- Recursive function calling as a game flow.

There were some **unlispy** elements in the implementation:

- Ugly global variables.
- Loops where list operations would have done.
- Procedual programming style conditionals.

FUTURE DIRECTIONS

When I learn more lisp, the following improvements come to mind:

- Exploiting the symmetry of House/player move methods, replacing them with a generic one.
- More secure and prettier, possibly utf-8 suits, read-eval-print-loop.
- More players?

REFERENCES

A fun book: The Land of Lisp - Conrad Barski, M.D