The Wizard's Adventure Game REPL [1]

Eric Bailey

October 14, 2017 [2]

Contents

Setting Up a Custom REPL

1b ⟨* 1a⟩+≡
```
(defun game-repl ()
  (let ((cmd (game-read)))
    (unless (eq (car cmd) 'quit)
      (game-print (game-eval cmd))
      (game-repl))))

(export (find-symbol "GAME-REPL"))
```

Defines:
  `game-repl`, never used.
Uses `game-eval` 2i, `game-print` 3b, and `game-read` 2d.

[1] Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 6, pages 85–101. No Starch Press, 2010. ISBN 9781593273491. URL http://landoflisp.com

[2] Last updated October 19, 2017

1a ⟨* 1a⟩≡
```
(in-package :cl-user)
(in-package :lol.wizard5)
```

⟨*define the allowed commands.* 2e⟩

This definition is continued in chunks 1–3.
Root chunk (not used in this document).
Defines:
  `lol.wizard6`, never used.

Writing a Custom read Function

`game-read` needs to:

2a 1. ⟨*Read a command.* 2a⟩≡
```
(read-from-string (concatenate 'string "(" (read-line) ")"))
```
This code is used in chunk 2d.

2. Take the `cdr` and ⟨*quote it.* 2b⟩

2b ⟨*quote it.* 2b⟩≡
```
(quote-it (x) (list 'quote x))
```
This code is used in chunk 2d.

2c 3. ⟨`cons` *the* `car` *to the result.* 2c⟩≡
```
(cons (car cmd) (mapcar #'quote-it (cdr cmd)))
```
This code is used in chunk 2d.

2d ⟨* 1a⟩+≡
```
(defun game-read ()
  (let ((cmd ⟨Read a command. 2a⟩))
    (flet (⟨quote it. 2b⟩)
      ⟨cons the car to the result. 2c⟩)))
```

Defines:
`game-read`, used in chunk 1b.

Writing a game-eval Function

First, we need to:

2e ⟨*define the allowed commands.* 2e⟩≡
```
(defparameter *allowed-commands* '(look walk pickup inventory))
```
This code is used in chunk 1a.
Defines:
`*allowed-commands*`, used in chunk 2f.

Then, when evaluating user input, if `an entered command is allowed`, ⟨*evaluate it.* 2g⟩ Otherwise ⟨*admonish the user.* 2h⟩

2i ⟨* 1a⟩+≡
```
(defun game-eval (sexp)
  (if ⟨an entered command is allowed 2f⟩
      ⟨evaluate it. 2g⟩
      ⟨admonish the user. 2h⟩))
```

Defines:
`game-eval`, used in chunk 1b.

2f ⟨*an entered command is allowed* 2f⟩≡
```
(member (car sexp) *allowed-commands*)
```
This code is used in chunk 2i.
Uses `*allowed-commands*` 2e.

2g ⟨*evaluate it.* 2g⟩≡
```
(eval sexp)
```
This code is used in chunk 2i.

2h ⟨*admonish the user.* 2h⟩≡
```
'(i do not know that command.)
```
This code is used in chunk 2i.

Writing a game-print Function

3a ⟨* 1a⟩+≡

```
(defun tweak-text (lst caps lit)
  (when lst
    (let ((item (car lst))
          (rest (cdr lst)))
      (cond ((eql item #\space) (cons item (tweak-text rest caps lit)))
            ((member item '(#\! #\? #\.)) (cons item (tweak-text rest t lit)))
            ((eql item #\") (tweak-text rest caps (not lit)))
            (lit (cons item (tweak-text rest nil lit)))
            (caps (cons (char-upcase item) (tweak-text rest nil lit)))
            (t (cons (char-downcase item) (tweak-text rest nil nil)))))))
```

Defines:
  tweak-text, used in chunk 3b.

3b ⟨* 1a⟩+≡

```
(defun game-print (lst)
  (princ (coerce (tweak-text (coerce (string-trim "() "
                                                  (prin1-to-string lst))
                                     'list)
                             t
                             nil)
                 'string))
  (fresh-line))
```

Defines:
  game-print, used in chunk 1b.
Uses tweak-text 3a.

Full Listing

```lisp
(defparameter *allowed-commands* '(look walk pickup inventory))


(defun game-repl ()
  (let ((cmd (game-read)))
    (unless (eq (car cmd) 'quit)
      (game-print (game-eval cmd))
      (game-repl))))

(export (find-symbol "GAME-REPL"))


(defun game-read ()
  (let ((cmd (read-from-string (concatenate 'string "(" (read-line) ")"))))
    (flet ((quote-it (x) (list 'quote x)))
      (cons (car cmd) (mapcar #'quote-it (cdr cmd))))))


(defun game-eval (sexp)
  (if (member (car sexp) *allowed-commands*)
      (eval sexp)
      '(i do not know that command.)))


(defun tweak-text (lst caps lit)
  (when lst
    (let ((item (car lst))
          (rest (cdr lst)))
      (cond ((eql item #\space) (cons item (tweak-text rest caps lit)))
            ((member item '(#\! #\? #\.)) (cons item (tweak-text rest t lit)))
            ((eql item #\") (tweak-text rest caps (not lit)))
            (lit (cons item (tweak-text rest nil lit)))
            (caps (cons (char-upcase item) (tweak-text rest nil lit)))
            (t (cons (char-downcase item) (tweak-text rest nil nil)))))))


(defun game-print (lst)
  (princ (coerce (tweak-text (coerce (string-trim "() "
                                                  (prin1-to-string lst))
                                     'list)
                             t
                             nil)
                 'string))
  (fresh-line))
```

Chunks

Index

References

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 6, pages 85–101. No Starch Press, 2010. ISBN 9781593273491. URL http://landoflisp.com.

Glossary

`car`

1. a. the first component of a `cons`; the other is the `cdr`.
   b. the head of a list, or `nil` if the list is the *empty list*.
2. the *object* that is held in the `car`. "The function `car` returns the `car` of a `cons`."

2, 5

`cdr`

1. a. the second component of a `cons`; the other is the `car`.
   b. the tail of a list, or `nil` if the list is the *empty list*.
2. the *object* that is held in the `cdr`. "The function `cdr` returns the `cdr` of a `cons`."

2, 5

`cons`

1. a compound data *object* made up of a `car` and a `cdr`.
2. to create such an *object*.
3. to create any *object* or to allocate storage.

2, 5

empty list the list containing no elements. 5

`nil` represents both boolean `false` and the *empty list*. Alternatively notated as `()` to emphasize its use as an *empty list*. 5

object any Lisp datum. 5