The Wizard's Adventure Game REPL [1]

Eric Bailey

October 14, 2017 [2]

Contents

Setting Up a Custom REPL 2

Writing a Custom read Function 2

Writing a game-eval Function 3

Writing a game-print Function 3

Tests 4

Full Listing 5

Chunks 6

Index 6

[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

1 ⟨* 1⟩≡

```
(in-package :cl-user)
(in-package :lol.wizard5)
```

⟨*define the allowed commands.* 3a⟩

This definition is continued in chunks 2–4.

Root chunk (not used in this document).

Defines:

`lol.wizard6`, never used.

Setting Up a Custom REPL

2a ⟨* 1⟩+≡
```
(defun game-repl ()
  (let ((cmd (game-read)))
    (unless (eq (car cmd) 'quit)
      (game-print (game-eval cmd))
      (game-repl))))
```

Defines:
  `game-repl`, never used.
Uses `game-eval` 3e, `game-print` 4a, and `game-read` 2e.

Writing a Custom read Function

`game-read` needs to:

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

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

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

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

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

Defines:
  `game-read`, used in chunks 2a and 4c.

Writing a game-eval Function

First, we need to:

3a ⟨*define the allowed commands.* 3a⟩≡
```
(defparameter *allowed-commands* '(look walk pickup inventory))
```
This code is used in chunk 1.
Defines:
allowed-commands, used in chunk 3b.

Then, when evaluating user input, if `an entered command is allowed`, ⟨*evaluate it.* 3c⟩ Otherwise ⟨*admonish the user.* 3d⟩

3e ⟨* 1⟩+≡
```
(defun game-eval (sexp)
  (if ⟨an entered command is allowed 3b⟩
      ⟨evaluate it. 3c⟩
      ⟨admonish the user. 3d⟩))
```
Defines:
game-eval, used in chunk 2a.

3b ⟨*an entered command is allowed* 3b⟩≡
```
(member (car sexp) *allowed-commands*)
```
This code is used in chunk 3e.
Uses *allowed-commands* 3a.

3c ⟨*evaluate it.* 3c⟩≡
```
(eval sexp)
```
This code is used in chunk 3e.

3d ⟨*admonish the user.* 3d⟩≡
```
'(i do not know that command.)
```
This code is used in chunk 3e.

Writing a game-print Function

3f ⟨* 1⟩+≡
```
(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 4a.

4a ⟨* 1⟩+≡

```
(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 2a.
Uses tweak-text 3f.

4b ⟨* 1⟩+≡

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

Tests

4c ⟨*test/wizard5.lisp* 4c⟩≡

```
(in-package :lol.wizard5)


(prove:plan 1)


;; > (game-read)
;; walk east
;; (WALK 'EAST)


(prove:finalize)
```

Root chunk (not used in this document).
Uses game-read 2e.

Full Listing

Chunks

⟨* 1⟩ 1, 2a, 2e, 3e, 3f, 4a, 4b
⟨`cons` *the* `car` *to the result.* 2d⟩ 2d, 2e
⟨*admonish the user.* 3d⟩ 3d, 3e
⟨*an entered command is allowed* 3b⟩ 3b, 3e
⟨*define the allowed commands.* 3a⟩ 1, 3a
⟨*evaluate it.* 3c⟩ 3c, 3e
⟨*quote it.* 2c⟩ 2c, 2e
⟨*Read a command.* 2b⟩ 2b, 2e
⟨*test/wizard5.lisp* 4c⟩ 4c

Index

`*allowed-commands*`: 3a, 3b
`game-eval`: 2a, 3e
`game-print`: 2a, 4a
`game-read`: 2a, 2e, 4c
`game-repl`: 2a
`lol.wizard6`: 1
`tweak-text`: 3f, 4a

[3]

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, 6, 7

`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, 7

`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, 6, 7

empty list the list containing no elements. 7

`nil` represents both boolean `false` and the *empty list*. Alternatively notated as `()` to emphasize its use as an *empty list*. 7

object any Lisp datum. 7

[3] Kent M. Pitman. CLHS: Glossary. http://www.lispworks.com/documentation/HyperSpec/Body/26_a.htm, April 2005. Accessed: 2017-10-17

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.

Kent M. Pitman. CLHS: Glossary. http://www.lispworks.com/documentation/HyperSpec/Body/26_a.htm, April 2005. Accessed: 2017-10-17.