The Wizard's Adventure Game [1]

Eric Bailey

October 14, 2017 [2]

[1] Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 5, pages 67–84. No Starch Press, 2010. ISBN 9781593273491. URL http://landoflisp.com

[2] Last updated October 15, 2017

In this game, you are a wizard's apprentice.
You'll explore the wizard's house.

Contents

1 ⟨* 1⟩≡

```
(in-package :cl-user)
(defpackage lol.wizard5
  (:use :cl)
  (:export :look
           :walk
           :pickup
           :inventory))
(in-package :lol.wizard5)
```

This definition is continued in chunks 2, 9, 16, 22, 25, 30, 34, 41, 45, and 49.
Root chunk (not used in this document).
Defines:
`lol.wizard5`, used in chunks 56–58.
Uses `inventory` 49, `look` 34, `pickup` 45, and `walk` 41.

Setting the Scene

This world consists of only three locations:

- the living room
- a beautiful garden
- the attic

6 ⟨*define the global variables* 6⟩≡
```
(defparameter *nodes*
  '((living-room (⟨living-room description 3⟩))
    (garden      (⟨garden description 4⟩))
    (attic       (⟨attic description 5⟩))))
```
This definition is continued in chunks 15, 23, and 32.
This code is used in chunk 2.
Defines:
nodes, used in chunks 8, 34, and 57.

Describing the Location

To find the description, ⟨*look up a location* 7⟩ and take the `cadr`. Preferring the *functional programming* style, pass `nodes` as an argument, instead of referencing `*nodes*` directly.

9 ⟨* 1⟩+≡
```
(defun describe-location (location nodes)
  (cadr ⟨look up a location 7⟩))
```
Defines:
describe-location, used in chunks 8, 34, and 57.

2 ⟨* 1⟩+≡
```
⟨define the global variables 6⟩
```

3 ⟨*living-room description* 3⟩≡
```
you are in the living room.
a wizard is snoring loudly on the couch.
```
This code is used in chunks 6, 8, 33, 57, and 58.

4 ⟨*garden description* 4⟩≡
```
you are in a beautiful garden.
there is a well in front of you.
```
This code is used in chunks 6 and 58.

5 ⟨*attic description* 5⟩≡
```
you are in the attic.
there is a giant welding torch in the corner.
```
This code is used in chunk 6.

7 ⟨*look up a location* 7⟩≡
```
(assoc location nodes)
```
This code is used in chunk 9.

8 ⟨*Example Session* 8⟩≡
```
> (describe-location 'living-room *nodes*)
(⟨living-room description 3⟩)
```
This definition is continued in chunks 14, 21, 26, 31, 33, 46, and 48.
Root chunk (not used in this document).
Uses *nodes* 6 and describe-location 9.

Describing the Paths

From the `living-room`, you can move to the `garden` by going `west` through the `door`, or to the `attic` by going `upstairs` via the `ladder`.

From the `garden`, you can move to the `living-room` by going `east` through the `door`.

From the `attic`, you can move to the `living-room` by going `downstairs` via the `ladder`.

15 ⟨*define the global variables* 6⟩+≡
```
(defparameter *edges*
  '((living-room ⟨living-room paths 10⟩)
    (garden      ⟨garden path 11⟩)
    (attic       ⟨attic path 12⟩)))
```
This code is used in chunk 2.
Defines:
edges, used in chunks 21, 34, 35, and 57.

To describe a path, take the means (`caddr`) and direction (`cadr`) and return a descriptive list.

16 ⟨* 1⟩+≡
```
(defun describe-path (edge)
  `(there is a ,(caddr edge) going ,(cadr edge) from here.))
```
Defines:
describe-path, used in chunks 14 and 57.

Describing Multiple Paths at Once

To describe multiple paths:

1. ⟨*Find the relevant edges.* 17⟩
2. ⟨*Convert the edges to descriptions.* 18⟩
3. ⟨*Join the descriptions.* 19⟩

22 ⟨* 1⟩+≡
```
(defun describe-paths (location edges)
  (⟨Join the descriptions. 19⟩ (⟨Convert the edges to descriptions. 18⟩ ⟨Find the relevant edges. 17⟩)))
```
Defines:
describe-paths, used in chunks 21, 34, and 57.

10 ⟨*living-room paths* 10⟩≡
```
(garden west door)
(attic upstairs ladder)
```
This code is used in chunk 15.

11 ⟨*garden path* 11⟩≡
```
(living-room east door)
```
This code is used in chunk 15.

12 ⟨*attic path* 12⟩≡
```
(living-room downstairs ladder)
```
This code is used in chunk 15.

13 ⟨*garden door* 13⟩≡
```
THERE IS A DOOR GOING WEST FROM HERE.
```
This code is used in chunks 14, 21, 50, and 57.

14 ⟨*Example Session* 8⟩+≡
```
> (describe-path '(garden west door))
⟨garden door 13⟩
```
Uses describe-path 16.

17 ⟨*Find the relevant edges.* 17⟩≡
```
(cdr (assoc location edges))
```
This code is used in chunk 22.

18 ⟨*Convert the edges to descriptions.* 18⟩≡
```
mapcar #'describe-path
```
This code is used in chunk 22.

19 ⟨*Join the descriptions.* 19⟩≡
```
apply #'append
```
This code is used in chunks 22 and 30.

20 ⟨*attic ladder* 20⟩≡
```
THERE IS A LADDER GOING UPSTAIRS FROM HERE.
```
This code is used in chunks 21 and 50.

21 ⟨*Example Session* 8⟩+≡
```
> (describe-paths 'living-room *edges*)
(⟨garden door 13⟩
 ⟨attic ladder 20⟩)
```
Uses *edges* 15 and describe-paths 22.

Describing Objects at a Specific Location

23 ⟨*define the global variables* 6⟩+≡
```
(defparameter *objects* '(whiskey bucket frog chain))

(defparameter *object-locations*
  '((whiskey living-room)
    (bucket living-room)
    (chain garden)
    (frog garden)))
```
This code is used in chunk 2.
Defines:
object-locations, used in chunks 26, 31, 34, 43, 44, 47, 57, and 58.
objects, used in chunks 26, 31, 34, 44, 47, 57, and 58.

25 ⟨* 1⟩+≡
```
(defun objects-at (loc objs obj-locs)
  (labels (⟨at-loc-p 24⟩)
    (remove-if-not #'at-loc-p objs)))
```
Defines:
objects-at, used in chunks 26, 28, 44, 47, 57, and 58.

26 ⟨*Example Session* 8⟩+≡
```
> (objects-at 'living-room *objects* *object-locations*)
(WHISKEY BUCKET)
```
Uses *object-locations* 23, *objects* 23, and objects-at 25.

24 ⟨*at-loc-p* 24⟩≡
```
(at-loc-p (obj)
  (eq (cadr (assoc obj obj-locs)) loc))
```
This code is used in chunk 25.

Describing Visible Objects

To describe the objects visible at a given location:

1. ⟨*Find the objects at the current location.* 28⟩
2. ⟨*Convert the objects to descriptions.* 29⟩
3. ⟨*Join the descriptions.* 19⟩

27 ⟨*describe-obj* 27⟩≡
```
(describe-obj (obj)
  `(you see a ,obj on the floor.))
```
This code is used in chunk 30.

28 ⟨*Find the objects at the current location.* 28⟩≡
```
(objects-at loc objs obj-loc)
```
This code is used in chunk 30.
Uses objects-at 25.

29 ⟨*Convert the objects to descriptions.* 29⟩≡
```
mapcar #'describe-obj
```
This code is used in chunk 30.

30 ⟨* 1⟩+≡
```
(defun describe-objects (loc objs obj-loc)
  (labels (⟨describe-obj 27⟩)
    (⟨Join the descriptions. 19⟩
          (⟨Convert the objects to descriptions. 29⟩
                ⟨Find the objects at the current location. 28⟩)))))
```
Defines:
describe-objects, used in chunks 31, 34, 57, and 58.

31 ⟨*Example Session* 8⟩+≡
```
> (describe-objects 'living-room *objects* *object-locations*)
(⟨living-room object descriptions 51⟩)
```
Uses *object-locations* 23, *objects* 23, and describe-objects 30.

Describing It All

34 ⟨* 1⟩+≡
```
(defun look ()
  (append (describe-location *location* *nodes*)
          (describe-paths *location* *edges*)
          (describe-objects *location* *objects* *object-locations*)))
```

Defines:
look, used in chunks 1, 33, 38, and 58.
Uses *edges* 15, *location* 32, *nodes* 6, *object-locations* 23, *objects* 23, describe-location 9, describe-objects 30, and describe-paths 22.

Walking Around in Our World

Given a direction, ⟨*locate the path marked with the appropriate direction* 37⟩ and ⟨*try to go in that direction* 38⟩. Since the direction will be there, ⟨*match against the* cadr *of each path* 36⟩.

37 ⟨*locate the path marked with the appropriate direction* 37⟩≡
```
(find direction
      ⟨look up the available walkings paths 35⟩
      ⟨match against the cadr of each path 36⟩)
```
This code is used in chunk 41.

If such a path is found, ⟨*adjust the player's position* 39⟩, otherwise ⟨*admonish the player* 40⟩.

38 ⟨*try to go in that direction* 38⟩≡
```
(if next
    (progn ⟨adjust the player's position 39⟩
           (look))
    ⟨admonish the player 40⟩)
```
This code is used in chunk 41.
Uses look 34.

41 ⟨* 1⟩+≡
```
(defun walk (direction)
  (let ((next ⟨locate the path marked with the appropriate direction 37⟩))
    ⟨try to go in that direction 38⟩))
```

Defines:
walk, used in chunks 1 and 58.

N.B. The look function is **not** functional, since it reads global variables.

32 ⟨*define the global variables* 6⟩+≡
```
(defparameter *location* 'living-room)
```

This code is used in chunk 2.
Defines:
location, used in chunks 34, 35, 39, and 44.

33 ⟨*Example Session* 8⟩+≡
```
> (look)
(⟨living-room description 3⟩
 ⟨living-room path descriptions 50⟩
 ⟨living-room object descriptions 51⟩)
```
Uses look 34.

35 ⟨*look up the available walkings paths* 35⟩≡
```
(cdr (assoc *location* *edges*))
```
This code is used in chunk 37.
Uses *edges* 15 and *location* 32.

36 ⟨*match against the* cadr *of each path* 36⟩≡
```
:key #'cadr
```
This code is used in chunk 37.

39 ⟨*adjust the player's position* 39⟩≡
```
(setf *location* (car next))
```
This code is used in chunk 38.
Uses *location* 32.

40 ⟨*admonish the player* 40⟩≡
```
'(you cannot go that way.)
```
This code is used in chunks 38 and 58.

Picking Up Objects

If ⟨*the object is on the floor* 42⟩, ⟨*pick it up* 43⟩.

44 ⟨*get the list of objects here* 44⟩≡
```
(objects-at *location* *objects* *object-locations*)
```
This code is used in chunk 42.
Uses *location* 32, *object-locations* 23, *objects* 23, and objects-at 25.

45 ⟨* 1⟩+≡
```
(defun pickup (object)
  (if ⟨the object is on the floor 42⟩
      (progn ⟨pick it up 43⟩)
      '(you cannot get that.)))
```
Defines:
pickup, used in chunks 1, 46, and 58.

46 ⟨*Example Session* 8⟩+≡
```
> (pickup 'whiskey)
(YOU ARE NOW CARRYING THE WHISKEY)
```
Uses pickup 45.

Checking Our Inventory

49 ⟨* 1⟩+≡
```
(defun inventory ()
  (cons 'items- ⟨retrieve the list of carried objects 47⟩))
```
Defines:
inventory, used in chunks 1, 48, and 58.

Tests

56 ⟨*test/wizard5.lisp* 56⟩≡
```
(in-package :lol.wizard5)

(prove:plan 2)

⟨Test the private functions in lol.wizard5 57⟩

⟨Test the exported functions in lol.wizard5. 58⟩

(prove:finalize)
```
Root chunk (not used in this document).
Uses lol.wizard5 1.

42 ⟨*the object is on the floor* 42⟩≡
```
(member object ⟨get the list of objects here 44⟩)
```
This code is used in chunk 45.

43 ⟨*pick it up* 43⟩≡
```
(push (list object 'body) *object-locations*)
'(you are now carrying the ,object)
```
This code is used in chunk 45.
Uses *object-locations* 23.

47 ⟨*retrieve the list of carried objects* 47⟩≡
```
(objects-at 'body *objects* *object-locations*)
```
This code is used in chunk 49.
Uses *object-locations* 23, *objects* 23, and objects-at 25.

48 ⟨*Example Session* 8⟩+≡
```
> (inventory)
⟨All you have is whiskey. 55⟩
```
Uses inventory 49.

50 ⟨*living-room path descriptions* 50⟩≡
```
⟨garden door 13⟩
⟨attic ladder 20⟩
```
This code is used in chunks 33, 57, and 58.

51 ⟨*living-room object descriptions* 51⟩≡
```
YOU SEE A WHISKEY ON THE FLOOR.
YOU SEE A BUCKET ON THE FLOOR.
```
This code is used in chunks 31, 33, 57, and 58.

52 ⟨*garden path description* 52⟩≡
```
THERE IS A DOOR GOING EAST FROM HERE.
```
This code is used in chunk 58.

53 ⟨*garden object descriptions* 53⟩≡
```
YOU SEE A FROG ON THE FLOOR.
YOU SEE A CHAIN ON THE FLOOR.
```
This code is used in chunk 58.

54 ⟨*You've got whiskey!* 54⟩≡
```
'(YOU ARE NOW CARRYING THE WHISKEY)
```
This code is used in chunk 58.

55 ⟨*All you have is whiskey.* 55⟩≡
```
'(ITEMS- WHISKEY)
```
This code is used in chunks 48 and 58.

lol.wizard5 (Private Parts)

57 ⟨*Test the private functions in* lol.wizard5 57⟩≡

```
(prove:subtest "lol.wizard5 (Private Parts)"
  (prove:is (describe-location 'living-room *nodes*)
            '(⟨living-room description 3⟩))
  (prove:is (describe-path '(garden west door))
            '(⟨garden door 13⟩))
  (prove:is (describe-paths 'living-room *edges*)
            '(⟨living-room path descriptions 50⟩))
  (prove:is (describe-objects 'living-room *objects* *object-locations*)
            '(⟨living-room object descriptions 51⟩))
  (prove:is (objects-at 'living-room *objects* *object-locations*)
            '(WHISKEY BUCKET)))
```

This code is used in chunk 56.
Uses *edges* 15, *nodes* 6, *object-locations* 23, *objects* 23, describe-location 9, describe-objects 30, describe-path 16, describe-paths 22, lol.wizard5 1, and objects-at 25.

lol.wizard5 (Public API)

58 ⟨*Test the exported functions in* lol.wizard5. 58⟩≡

```
(prove:subtest "lol.wizard5 (Public API)"
  (prove:is (look)
            '(⟨living-room description 3⟩
              ⟨living-room path descriptions 50⟩
              ⟨living-room object descriptions 51⟩))
  (prove:subtest "Pick up the whiskey"
    (prove:is (pickup 'whiskey)
              ⟨You've got whiskey! 54⟩)
    (prove:is (objects-at 'living-room *objects* *object-locations*)
              '(BUCKET))
    (prove:is (describe-objects 'living-room *objects* *object-locations*)
              '(YOU SEE A BUCKET ON THE FLOOR.)))
  (prove:is (pickup 'the-pace)
            '(you cannot get that.))
  (prove:is (walk 'west)
            '(⟨garden description 4⟩
              ⟨garden path description 52⟩
              ⟨garden object descriptions 53⟩))
  (prove:is (walk 'south)
            ⟨admonish the player 40⟩)
  (prove:is (inventory)
            ⟨All you have is whiskey. 55⟩))
```

This code is used in chunk 56.
Uses *object-locations* 23, *objects* 23, describe-objects 30, inventory 49, lol.wizard5 1, look 34, objects-at 25, pickup 45, and walk 41.

Running the Tests

Describe `prove`

61 ⟨*Run the system tests.* 61⟩≡
```
(prove:run-test-system :lol-test)
```
This code is used in chunk 62.

62 ⟨*Run the system tests and exit.* 62⟩≡
```
(uiop:quit (if ⟨Run the system tests. 61⟩ 0 1))
```
This code is used in chunk 68.

Describe nix-shell shebang

63 ⟨*script header* 63⟩≡
```
#! /usr/bin/env nix-shell
#! nix-shell -i sh -p sbcl
```

This definition is continued in chunks 64 and 65.
This code is used in chunk 68.

68 ⟨*bin/runtests* 68⟩≡
```
⟨script header 63⟩
    -eval "⟨Load the test package. 67⟩" \
    -eval "⟨Run the system tests and exit. 62⟩"


⟨script footer 66⟩
```
Root chunk (not used in this document).

```
$ ./bin/runtests
✓ 2 tests completed (0ms)

Summary:
  All 1 file passed.
```

59 ⟨*Set the exit status.* 59⟩≡
```
(if (null failures) 0 1)
```
Root chunk (not used in this document).

60 ⟨*Exit with an appropriate status code.* 60⟩≡
```
(sb-posix:exit status)
```
Root chunk (not used in this document).

Run `sbcl` quietly:

64 ⟨*script header* 63⟩+≡
```
sbcl -noinform -non-interactive \
```
This code is used in chunk 68.

Load ⟨*init.lisp* (never defined)⟩ as the user initialization file:

65 ⟨*script header* 63⟩+≡
```
    -userinit init.lisp \
```
This code is used in chunk 68.

66 ⟨*script footer* 66⟩≡
```
# Local Variables:
# mode: sh
# End:
```
This code is used in chunk 68.

67 ⟨*Load the test package.* 67⟩≡
```
(asdf:load-system :lol-test)
```
This code is used in chunk 68.

Full Listing

```lisp
(defparameter *nodes*
  '((living-room (you are in the living room.
                  a wizard is snoring loudly on the couch.))
    (garden      (you are in a beautiful garden.
                  there is a well in front of you.))
    (attic       (you are in the attic.
                  there is a giant welding torch in the corner.))))

(defparameter *edges*
  '((living-room (garden west door)
                 (attic upstairs ladder))
    (garden      (living-room east door))
    (attic       (living-room downstairs ladder))))

(defparameter *objects* '(whiskey bucket frog chain))

(defparameter *object-locations*
  '((whiskey living-room)
    (bucket living-room)
    (chain garden)
    (frog garden)))

(defparameter *location* 'living-room)


(defun describe-location (location nodes)
  (cadr (assoc location nodes)))


(defun describe-path (edge)
  `(there is a ,(caddr edge) going ,(cadr edge) from here.))


(defun describe-paths (location edges)
  (apply #'append (mapcar #'describe-path (cdr (assoc location edges)))))


(defun objects-at (loc objs obj-locs)
  (labels ((at-loc-p (obj)
             (eq (cadr (assoc obj obj-locs)) loc)))
    (remove-if-not #'at-loc-p objs)))


(defun describe-objects (loc objs obj-loc)
  (labels ((describe-obj (obj)
             `(you see a ,obj on the floor.)))
    (apply #'append
           (mapcar #'describe-obj
                   (objects-at loc objs obj-loc)))))


(defun look ()
  (append (describe-location *location* *nodes*)
          (describe-paths *location* *edges*)
          (describe-objects *location* *objects* *object-locations*)))


(defun walk (direction)
  (let ((next (find direction
                    (cdr (assoc *location* *edges*))
                    :key #'cadr)))
    (if next
        (progn (setf *location* (car next))
               (look))
        '(you cannot go that way.))))


(defun pickup (object)
  (if (member object (objects-at *location* *objects* *object-locations*))
      (progn (push (list object 'body) *object-locations*)
             `(you are now carrying the ,object))
      '(you cannot get that.)))


(defun inventory ()
```

Chunks

Index

References

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 5, pages 67–84. No Starch Press, 2010. ISBN 9781593273491. URL http://landoflisp.com.