

The Wizard's Adventure Game ¹

Eric Bailey

October 14, 2017 ²

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

Contents

<i>Setting the Scene</i>	2
<i>Describing the Location</i>	2
<i>Describing the Paths</i>	3
<i>Describing Multiple Paths at Once</i>	3
<i>Describing Objects at a Specific Location</i>	4
<i>Describing Visible Objects</i>	4
<i>Describing It All</i>	5
<i>Walking Around in Our World</i>	5
<i>Picking Up Objects</i>	6
<i>Checking Our Inventory</i>	6
<i>Tests</i>	6
<i>Running the Tests</i>	8
<i>Full Listing</i>	9
<i>Chunks</i>	11
<i>Index</i>	11

```
1  <* 1>≡
    (in-package #:lol)
    (defpackage #:lol.wizard5
      (:use #:cl #:lisp-unit)
      (:export #:look
               #:walk
               #:pickup))
    (in-package #:lol.wizard5)
```

This definition is continued in chunks 2, 8, 13, 17, 20, 24, 26, 33, 37, and 39–47.
Root chunk (not used in this document).
Uses look 26, pickup 37, and walk 33.

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 14, 2017

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 12, 18, and 25.

This code is used in chunk 2.

Defines:

nodes, used in chunks 26 and 40.

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.

```
8 <* 1>+≡
  (defun describe-location (location nodes)
    (cadr <look up a location 7>))
```

Defines:

describe-location, used in chunks 26 and 40.

```
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, 40,
and 45.
```

```
4 <garden description 4>≡
  you are in a beautiful garden.
  there is a well in front of you.
This code is used in chunk 6.
```

```
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 8.
```

```
> (describe-location 'living-room *nodes*)
(YOU ARE IN THE LIVING-ROOM.
 A WIZARD IS SNORING LOUDLY ON THE COUCH.)
```

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**.

```
<living-room paths 9>≡
  (garden west door)
  (attic upstairs ladder)
This code is used in chunk 12.
```

```
<garden path 10>≡
  (living-room east door)
This code is used in chunk 12.
```

```
<attic path 11>≡
  (living-room downstairs ladder)
This code is used in chunk 12.
```

```
> (describe-path '(garden west door))
(THERE IS A DOOR GOING WEST FROM HERE.)
```

```
12 <define the global variables 6>+≡
    (defparameter *edges*
      '((living-room <living-room paths 9>)
        (garden      <garden path 10>)
        (attic       <attic path 11>))))
```

This code is used in chunk 2.

Defines:

edges, used in chunks 26, 27, and 42.

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

```
13 <* 1>+≡
    (defun describe-path (edge)
      '(there is a ,(caddr edge) going ,(cadr edge) from here.))
```

Defines:

describe-path, used in chunk 41.

```
14 <Find the relevant edges. 14>≡
    (cdr (assoc location edges))
This code is used in chunk 17.
```

```
15 <Convert the edges to descriptions. 15>≡
    mapcar #'describe-path
This code is used in chunk 17.
```

```
16 <Join the descriptions. 16>≡
    apply #'append
This code is used in chunks 17 and 24.
```

```
> (describe-paths 'living-room *edges*)
(THERE IS A DOOR GOING WEST FROM HERE.
 THERE IS A LADDER GOING UPSTAIRS FROM HERE.)
```

```
17 <* 1>+≡
    (defun describe-paths (location edges)
      ((Join the descriptions. 16) ((Convert the edges to descriptions. 15) <Find the relevant edges. 14>)))
```

Defines:

describe-paths, used in chunks 26 and 42.

Describing Multiple Paths at Once

To describe multiple paths:

1. *<Find the relevant edges. 14>*
2. *<Convert the edges to descriptions. 15>*
3. *<Join the descriptions. 16>*

```
17 <* 1>+≡
    (defun describe-paths (location edges)
      ((Join the descriptions. 16) ((Convert the edges to descriptions. 15) <Find the relevant edges. 14>)))
```

Defines:

describe-paths, used in chunks 26 and 42.

Describing Objects at a Specific Location

```
18 <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, 35, 36, 38, 43, and 44.
 objects, used in chunks 26, 36, 38, 43, and 44.

```
20 <* 1>+≡
    (defun objects-at (loc objs obj-locs)
      (labels ((at-loc-p 19))
        (remove-if-not #'at-loc-p objs)))
```

Defines:

objects-at, used in chunks 22, 36, 38, and 43.

```
> (objects-at 'living-room *objects* *object-locations*)
(WHISKEY BUCKET)
```

Describing Visible Objects

To describe the objects visible at a given location:

1. <Find the objects at the current location. 22>
2. <Convert the objects to descriptions. 23>
3. <Join the descriptions. 16>

```
24 <* 1>+≡
    (defun describe-objects (loc objs obj-loc)
      (labels ((describe-obj 21))
        (<Join the descriptions. 16>
          (<Convert the objects to descriptions. 23>
            (<Find the objects at the current location. 22>))))
```

Defines:

describe-objects, used in chunks 26 and 44.

```
> (describe-objects 'living-room *objects* *object-locations*)
(YOU SEE A WHISKEY ON THE FLOOR.
 YOU SEE A BUCKET ON THE FLOOR.)
```

```
19 <at-loc-p 19>≡
    (at-loc-p (obj)
      (eq (cadr (assoc obj obj-locs)) loc))
This code is used in chunk 20.
```

```
21 <describe-obj 21>≡
    (describe-obj (obj)
      '(you see a ,obj on the floor.))
This code is used in chunk 24.
```

```
22 <Find the objects at the current location. 22>≡
    (objects-at loc objs obj-loc)
This code is used in chunk 24.
Uses objects-at 20.
```

```
23 <Convert the objects to descriptions. 23>≡
    mapcar #'describe-obj
This code is used in chunk 24.
```

Describing It All

```

26  ⟨* 1⟩+≡
    (defun look ()
      (append (describe-location *location* *nodes*)
               (describe-paths *location* *edges*)
               (describe-objects *location* *objects* *object-locations*)))

```

Defines:

look, used in chunks 1, 30, and 45.

Uses *edges* 12, *location* 25, *nodes* 6, *object-locations* 18, *objects* 18, describe-location 8, describe-objects 24, and describe-paths 17.

Walking Around in Our World

Given a *direction*, ⟨locate the path marked with the appropriate direction 29⟩ and ⟨try to go in that direction 30⟩. Since the *direction* will be there, ⟨match against the cadr of each path 28⟩.

```

29  ⟨locate the path marked with the appropriate direction 29⟩≡
    (find direction
      ⟨look up the available walkings paths 27⟩
      ⟨match against the cadr of each path 28⟩)

```

This code is used in chunk 33.

If such a path is found, ⟨adjust the player's position 31⟩, otherwise ⟨admonish the player 32⟩.

```

30  ⟨try to go in that direction 30⟩≡
    (if next
      (progn ⟨adjust the player's position 31⟩
              (look))
      ⟨admonish the player 32⟩)

```

This code is used in chunk 33.

Uses look 26.

```

33  ⟨* 1⟩+≡
    (defun walk (direction)
      (let ((next (locate the path marked with the appropriate direction 29)))
        ⟨try to go in that direction 30⟩))

```

Defines:

walk, used in chunk 1.

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

```

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

```

This code is used in chunk 2.

Defines:

location, used in chunks 26, 27, 31, and 36.

```

> (look)
(YOU ARE IN THE LIVING ROOM.
 A WIZARD IS SNORING LOUDLY ON THE COUCH.
 THERE IS A DOOR GOING WEST FROM HERE.
 THERE IS A LADDER GOING UPSTAIRS FROM HERE.
 YOU SEE A WHISKEY ON THE FLOOR.
 YOU SEE A BUCKET ON THE FLOOR.)

```

```

27  ⟨look up the available walkings paths 27⟩≡
    (cdr (assoc *location* *edges*))

```

This code is used in chunk 29.

Uses *edges* 12 and *location* 25.

```

28  ⟨match against the cadr of each path 28⟩≡
    :key #'cadr

```

This code is used in chunk 29.

```

31  ⟨adjust the player's position 31⟩≡
    (setf *location* (car next))

```

This code is used in chunk 30.

Uses *location* 25.

```

32  ⟨admonish the player 32⟩≡
    '(you cannot go that way.)

```

This code is used in chunk 30.

Picking Up Objects

If *<the object is on the floor 34>*, *<pick it up 35>*.

```
36 <get the list of objects here 36>≡
    (objects-at *location* *objects* *object-locations*)
This code is used in chunk 34.
Uses *location* 25, *object-locations* 18, *objects* 18, and objects-at 20.

37 <* 1>+≡
    (defun pickup (object)
      (if <the object is on the floor 34>
        (progn <pick it up 35>)
        '(you cannot get that)))
```

Defines:

pickup, used in chunks 1 and 46.

```
> (pickup 'whiskey)
(YOU ARE NOW CARRYING THE WHISKEY)
```

Checking Our Inventory

```
39 <* 1>+≡
    (defun inventory ()
      (cons 'items- <retrieve the list of carried objects 38>))
```

Defines:

inventory, used in chunk 47.

Tests

```
40 <* 1>+≡
    (define-test describe-living-room
      (assert-equal '(<living-room description 3>)
                    (describe-location 'living-room *nodes*)))
```

Uses *nodes* 6 and describe-location 8.

```
41 <* 1>+≡
    (define-test garden-path
      (assert-equal '(THERE IS A DOOR GOING WEST FROM HERE.)
                    (describe-path 'garden west door))))
```

Uses describe-path 13.

```
<the object is on the floor 34>≡
    (member object <get the list of objects here 36>))
This code is used in chunk 37.
```

```
35 <pick it up 35>≡
    (push (list object 'body) *object-locations*)
    '(you are now carrying the ,object)
This code is used in chunk 37.
Uses *object-locations* 18.
```

```
38 <retrieve the list of carried objects 38>≡
    (objects-at 'body *objects* *object-locations*)
This code is used in chunk 39.
Uses *object-locations* 18, *objects* 18,
and objects-at 20.

> (inventory)
(ITEMS- WHISKEY)
```

```

42  <* 1>+≡
    (define-test living-room-paths
      (assert-equal '(THERE IS A DOOR GOING WEST FROM HERE.
                      THERE IS A LADDER GOING UPSTAIRS FROM HERE.)
                    (describe-paths 'living-room *edges*)))

    Uses *edges* 12 and describe-paths 17.

43  <* 1>+≡
    (define-test living-room-objects
      (assert-equal '(WHISKEY BUCKET)
                    (objects-at 'living-room *objects* *object-locations*)))

    Uses *object-locations* 18, *objects* 18, and objects-at 20.

44  <* 1>+≡
    (define-test describe-living-room-objects
      (assert-equal '(YOU SEE A WHISKEY ON THE FLOOR.
                      YOU SEE A BUCKET ON THE FLOOR.)
                    (describe-objects 'living-room *objects* *object-locations*)))

    Uses *object-locations* 18, *objects* 18, and describe-objects 24.

45  <* 1>+≡
    (define-test look
      (assert-equal '(<living-room description 3>
                    THERE IS A DOOR GOING WEST FROM HERE.
                    THERE IS A LADDER GOING UPSTAIRS FROM HERE.
                    YOU SEE A WHISKEY ON THE FLOOR.
                    YOU SEE A BUCKET ON THE FLOOR.)
                    (look)))

    Uses look 26.

46  <* 1>+≡
    (define-test pickup-whiskey
      (assert-equal '(YOU ARE NOW CARRYING THE WHISKEY)
                    (pickup 'whiskey)))

    Uses pickup 37.

47  <* 1>+≡
    (define-test have-whiskey
      (assert-equal '(ITEMS- WHISKEY)
                    (inventory)))

    Uses inventory 39.

```

Running the Tests

Describe lisp-unit

The package under test is specified as the first argument on the command line and prefixed with `:lol.` and used to *set the value of `*package*`* 48) in *bin/runtests* 58).

52 *Run the lisp-unit tests.* 52)≡
 (let* ((results *Run all tests in the package.* 53))
 (failures *Collect the failures.* 54))
 (status *Set the exit status.* 50)))
 Print the failures. 55)
 Exit with an appropriate status code. 51))

This code is used in chunk 59.

53 *Run all tests in the package.* 53)≡
 (lisp-unit:run-tests :all *the specified package* 49))
 This code is used in chunk 52.

54 *Collect the failures.* 54)≡
 (lisp-unit:failed-tests results)
 This code is used in chunk 52.

55 *Print the failures.* 55)≡
 (lisp-unit:print-failures results)
 This code is used in chunk 52.

Describe nix-shell shebang

58 *bin/runtests* 58)≡
 #! /usr/bin/env nix-shell
 #! nix-shell -i sh -p sbcl
 This definition is continued in chunk 59.
 Root chunk (not used in this document).

59 *bin/runtests* 58)+≡
Run SBCL quietly 56) \
 Load init.lisp as the user initialization file. 57) \
 *set the value of `*package*`* 48) \
 -eval "*Run the lisp-unit tests.* 52)"

\$./bin/runtests wizard5

```
Unit Test Summary
8 assertions total
8 passed
0 failed
0 execution errors
0 missing tests
```

48 *set the value of `*package*`* 48)≡
 -eval "(in-package *the specified package* 49))"
 This code is used in chunk 59.

49 *the specified package* 49)≡
 :lol.\$1
 This code is used in chunks 48 and 53.

50 *Set the exit status.* 50)≡
 (if (null failures) 0 1)
 This code is used in chunk 52.

51 *Exit with an appropriate status code.* 51)≡
 (sb-posix:exit status)
 This code is used in chunk 52.

56 *Run SBCL quietly* 56)≡
 sbcl -noinform -non-interactive
 This code is used in chunk 59.

Load init.lisp as the user initialization file. 57)≡
 -userinit init.lisp
 This code is used in chunk 59.

Full Listing

```

1  (in-package #:lol)
2  (defpackage #:lol.wizard5
3    (:use #:cl #:lisp-unit)
4    (:export #:look
5             #:walk
6             #:pickup))
7  (in-package #:lol.wizard5)
8
9
10 (defparameter *nodes*
11   '((living-room (you are in the living room.
12                  a wizard is snoring loudly on the couch.))
13     (garden      (you are in a beautiful garden.
14                  there is a well in front of you.))
15     (attic       (you are in the attic.
16                  there is a giant welding torch in the corner.))))
17
18 (defparameter *edges*
19   '((living-room (garden west door)
20                  (attic upstairs ladder))
21     (garden      (living-room east door))
22     (attic       (living-room downstairs ladder))))
23
24 (defparameter *objects* '(whiskey bucket frog chain))
25
26 (defparameter *object-locations*
27   '((whiskey living-room)
28     (bucket living-room)
29     (chain garden)
30     (frog garden)))
31
32 (defparameter *location* 'living-room)
33
34
35 (defun describe-location (location nodes)
36   (cadr (assoc location nodes)))
37
38
39 (defun describe-path (edge)
40   `(there is a ,(caddr edge) going ,(cadr edge) from here.))
41
42
43 (defun describe-paths (location edges)
44   (apply #'append (mapcar #'describe-path (cdr (assoc location edges)))))
45
46
47 (defun objects-at (loc objs obj-locs)
48   (labels ((at-loc-p (obj)
49             (eq (cadr (assoc obj obj-locs)) loc)))
50     (remove-if-not #'at-loc-p objs)))
51
52
53 (defun describe-objects (loc objs obj-loc)
54   (labels ((describe-obj (obj)
55             `(you see a ,obj on the floor.)))
56     (apply #'append
57            (mapcar #'describe-obj
58                    (objects-at loc objs obj-loc)))))
59
60
61 (defun look ()
62   (append (describe-location *location* *nodes*)
63           (describe-paths *location* *edges*)
64           (describe-objects *location* *objects* *object-locations*)))
65
66
67 (defun walk (direction)
68   (let ((next (find direction
69                     (cdr (assoc *location* *edges*))
70                     :key #'cadr)))
71     (if next
72         (progn (setf *location* (car next))
73               (look))
74         '(you cannot go that way.)))
75
76
77 (defun pickup (object)
78   (if (member object (objects-at *location* *objects* *object-locations*))
79       (progn (push (list object 'body) *object-locations*)
80             `(you are now carrying the ,object))
81       '(you cannot get that)))
82
83
84 (defun inventory ()
85   (cons 'items- (objects-at 'body *objects* *object-locations*)))

```

```

87
88 (define-test describe-living-room
89   (assert-equal '(you are in the living room.
90                  a wizard is snoring loudly on the couch.)
91                 (describe-location 'living-room *nodes*)))
92
93
94 (define-test garden-path
95   (assert-equal '(THERE IS A DOOR GOING WEST FROM HERE.)
96                  (describe-path '(garden west door))))
97
98
99 (define-test living-room-paths
100   (assert-equal '(THERE IS A DOOR GOING WEST FROM HERE.
101                  THERE IS A LADDER GOING UPSTAIRS FROM HERE.)
102                  (describe-paths 'living-room *edges*)))
103
104 (define-test living-room-objects
105   (assert-equal '(WHISKEY BUCKET
106                  (objects-at 'living-room *objects* *object-locations*)))
107
108 (define-test describe-living-room-objects
109   (assert-equal '(YOU SEE A WHISKEY ON THE FLOOR.
110                  YOU SEE A BUCKET ON THE FLOOR.)
111                  (describe-objects 'living-room *objects* *object-locations*)))
112
113
114 (define-test look
115   (assert-equal '(you are in the living room.
116                  a wizard is snoring loudly on the couch.
117                  THERE IS A DOOR GOING WEST FROM HERE.
118                  THERE IS A LADDER GOING UPSTAIRS FROM HERE.
119                  YOU SEE A WHISKEY ON THE FLOOR.
120                  YOU SEE A BUCKET ON THE FLOOR.)
121                  (look)))
122
123
124 (define-test pickup-whiskey
125   (assert-equal '(YOU ARE NOW CARRYING THE WHISKEY)
126                  (pickup 'whiskey)))
127
128
129 (define-test have-whiskey
130   (assert-equal '(ITEMS- WHISKEY)
131                  (inventory)))

```

Chunks

(* 1) [1](#), [2](#), [8](#), [13](#), [17](#), [20](#), [24](#), [26](#), [33](#), [37](#), [39](#), [40](#), [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#)
 <adjust the player's position 31> [30](#), [31](#)
 <admonish the player 32> [30](#), [32](#)
 <at-loc-p 19> [19](#), [20](#)
 <attic description 5> [5](#), [6](#)
 <attic path 11> [11](#), [12](#)
 <bin/runtests 58> [58](#), [59](#)
 <Collect the failures. 54> [52](#), [54](#)
 <Convert the edges to descriptions. 15> [15](#), [17](#)
 <Convert the objects to descriptions. 23> [23](#), [24](#)
 <define the global variables 6> [2](#), [6](#), [12](#), [18](#), [25](#)
 <describe-obj 21> [21](#), [24](#)
 <Exit with an appropriate status code. 51> [51](#), [52](#)
 <Find the objects at the current location. 22> [22](#), [24](#)
 <Find the relevant edges. 14> [14](#), [17](#)
 <garden description 4> [4](#), [6](#)
 <garden path 10> [10](#), [12](#)
 <get the list of objects here 36> [34](#), [36](#)
 <Join the descriptions. 16> [16](#), [17](#), [24](#)
 <living-room description 3> [3](#), [6](#), [40](#), [45](#)
 <living-room paths 9> [9](#), [12](#)
 <Load init.lisp as the user initialization file. 57> [57](#), [59](#)
 <locate the path marked with the appropriate direction 29> [29](#), [33](#)
 <look up a location 7> [7](#), [8](#)
 <look up the available walkings paths 27> [27](#), [29](#)
 <match against the cadr of each path 28> [28](#), [29](#)
 <pick it up 35> [35](#), [37](#)
 <Print the failures. 55> [52](#), [55](#)
 <retrieve the list of carried objects 38> [38](#), [39](#)
 <Run all tests in the package. 53> [52](#), [53](#)
 <Run SBCL quietly 56> [56](#), [59](#)
 <Run the lisp-unit tests. 52> [52](#), [59](#)
 <Set the exit status. 50> [50](#), [52](#)
 <set the value of *package* 48> [48](#), [59](#)
 <the object is on the floor 34> [34](#), [37](#)
 <the specified package 49> [48](#), [49](#), [53](#)
 <try to go in that direction 30> [30](#), [33](#)

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>.

Index

edges: [12](#), [26](#), [27](#), [42](#)
 location: [25](#), [26](#), [27](#), [31](#), [36](#)
 nodes: [6](#), [26](#), [40](#)
 object-locations: [18](#), [26](#), [35](#), [36](#), [38](#), [43](#), [44](#)
 objects: [18](#), [26](#), [36](#), [38](#), [43](#), [44](#)
 describe-location: [8](#), [26](#), [40](#)
 describe-objects: [24](#), [26](#), [44](#)
 describe-path: [13](#), [41](#)
 describe-paths: [17](#), [26](#), [42](#)
 inventory: [39](#), [47](#)
 look: [1](#), [26](#), [30](#), [45](#)
 objects-at: [20](#), [22](#), [36](#), [38](#), [43](#)
 pickup: [1](#), [37](#), [46](#)
 walk: [1](#), [33](#)