# *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 14, 2017

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

## *Contents*

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.

## 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.

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.

## 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.

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.

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

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

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

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

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

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.

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

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.

## 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⟩

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 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.

19 ⟨*at-loc-p* 19⟩≡

```
(at-loc-p (obj)
  (eq (cadr (assoc obj obj-locs)) loc))
```

This code is used in chunk 20.

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⟩

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.

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.)
```

## 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.

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.)
```

## 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.

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.

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.

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.

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.

## Picking Up Objects

If ⟨*the object is on the floor* 34⟩, ⟨*pick it up* 35⟩.

34 ⟨*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.

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

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)
```

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.

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

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
```

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⟩.

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.

7 ⟨*Load* `init.lisp` *as the user initialization file.* 57⟩≡
```
  -userinit init.lisp
```
This code is used in chunk 59.

## *Full Listing*

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


(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 ()
  (cons 'items- (objects-at 'body *objects* *object-locations*)))
```

```

(define-test describe-living-room
  (assert-equal '(you are in the living room.
                  a wizard is snoring loudly on the couch.)
                (describe-location 'living-room *nodes*)))


(define-test garden-path
  (assert-equal '(THERE IS A DOOR GOING WEST FROM HERE.)
                (describe-path '(garden west door))))


(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*)))
(define-test living-room-objects
  (assert-equal '(WHISKEY BUCKET)
                (objects-at 'living-room *objects* *object-locations*)))


(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*)))


(define-test look
  (assert-equal '(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.)
                 (look)))


(define-test pickup-whiskey
  (assert-equal '(YOU ARE NOW CARRYING THE WHISKEY)
                (pickup 'whiskey)))


(define-test have-whiskey
  (assert-equal '(ITEMS- WHISKEY)
                (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`.