

*The Wizard's Adventure Game*¹

Eric Bailey

October 14, 2017²

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>Full Listing</i>	7
<i>Chunks</i>	9
<i>Index</i>	9

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

```
5 <define the global variables 5>≡
  (defparameter *nodes*
    '((living-room (you are in the living room.
                    <living-room description 2>))
      (garden      (you are in a beautiful garden.
                    <garden description 3>))
      (attic       (you are in the attic.
                    <attic description 4>))))
```

This definition is continued in chunks 13, 20, and 29.

This code is used in chunk 1.

Defines:

nodes, used in chunks 7 and 31.

Describing the Location

To find the description, <look up a location 6> 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 6>))
```

Defines:

describe-location, used in chunks 7 and 31.

```
1 <* 1>≡
  <define the global variables 5>
  This definition is continued in chunks 8,
  14, 19, 22, 27, 31, 38, 42, and 46.
  Root chunk (not used in this document).

2 <living-room description 2>≡
  a wizard is snoring loudly on the couch.
  This code is used in chunk 5.

3 <garden description 3>≡
  there is a well in front of you.
  This code is used in chunk 5.

4 <attic description 4>≡
  there is a giant welding torch in the corner.
  This code is used in chunk 5.

6 <look up a location 6>≡
  (assoc location nodes)
  This code is used in chunk 8.

7 <session 7>≡
  > (describe-location 'living-room *nodes*)
  (YOU ARE IN THE LIVING-ROOM.
   A WIZARD IS SNORING LOUDLY ON THE COUCH.)
  This definition is continued in
  chunks 12, 18, 23, 28, 30, 43, and 45.
  Root chunk (not used in this document).
  Uses *nodes* 5 and describe-location 8.
```

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**.

```
9 <living-room paths 9>≡
  (garden west door)
  (attic upstairs ladder)
This code is used in chunk 13.
```

```
10 <garden path 10>≡
  (living-room east door)
This code is used in chunk 13.
```

```
11 <attic path 11>≡
  (living-room downstairs ladder)
This code is used in chunk 13.
```

```
12 <session 7>+≡
  > (describe-path '(garden west door))
  (THERE IS A DOOR GOING WEST FROM HERE.)
Uses describe-path 14.
```

```
13 <define the global variables 5>+≡
  (defparameter *edges*
    '((living-room <living-room paths 9>)
      (garden <garden path 10>)
      (attic <attic path 11>)))
```

This code is used in chunk 1.

Defines:

edges, used in chunks 18, 31, and 32.

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

```
14 <* 1>+≡
  (defun describe-path (edge)
    '(there is a ,(caddr edge) going ,(cadr edge) from here.))
```

Defines:

describe-path, used in chunk 12.

```
15 <Find the relevant edges. 15>≡
  (cdr (assoc location edges))
This code is used in chunk 19.
```

```
16 <Convert the edges to descriptions. 16>≡
  mapcar #'describe-path
This code is used in chunk 19.
```

```
17 <Join the descriptions. 17>≡
  apply #'append
This code is used in chunks 19 and 27.
```

```
18 <session 7>+≡
  > (describe-paths 'living-room *edges*)
  (THERE IS A DOOR GOING WEST FROM HERE.
   THERE IS A LADDER GOING UPSTAIRS FROM HERE.)
Uses *edges* 13 and describe-paths 19.
```

Describing Multiple Paths at Once

To describe multiple paths:

1. *<Find the relevant edges. 15>*

2. *<Convert the edges to descriptions. 16>*

3. *<Join the descriptions. 17>*

```
19 <* 1>+≡
  (defun describe-paths (location edges)
    ((Join the descriptions. 17) ((Convert the edges to descriptions. 16) <Find the relevant edges. 15>)))
```

Defines:

describe-paths, used in chunks 18 and 31.

Describing Objects at a Specific Location

20 *<define the global variables 5>+≡*
 (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 1.

Defines:

object-locations, used in chunks 23, 28, 31, 40, 41, and 44.
objects, used in chunks 23, 28, 31, 41, and 44.

22 *<* 1>+≡*
 (defun *objects-at* (loc objs obj-locs)
 (labels (*<at-loc-p 21>*)
 (remove-if-not #'at-loc-p objs)))

Defines:

objects-at, used in chunks 23, 25, 41, and 44.

23 *<session 7>+≡*
 > (*objects-at* 'living-room **objects** **object-locations**)
 (WHISKEY BUCKET)

Uses **object-locations** 20, **objects** 20, and *objects-at* 22.

Describing Visible Objects

To describe the objects visible at a given location:

1. *<Find the objects at the current location. 25>*
2. *<Convert the objects to descriptions. 26>*
3. *<Join the descriptions. 17>*

27 *<* 1>+≡*
 (defun *describe-objects* (loc objs obj-loc)
 (labels (*<describe-obj 24>*)
 (*<Join the descriptions. 17>*
 (*<Convert the objects to descriptions. 26>*
 (*<Find the objects at the current location. 25>*))))

Defines:

describe-objects, used in chunks 28 and 31.

21 *<at-loc-p 21>≡*
 (at-loc-p (obj)
 (eq (cadr (assoc obj obj-locs)) loc))

This code is used in chunk 22.

24 *<describe-obj 24>≡*
 (describe-obj (obj)
 '(you see a ,obj on the floor.))

This code is used in chunk 27.

25 *<Find the objects at the current location. 25>≡*
 (*objects-at* loc objs obj-loc)

This code is used in chunk 27.

Uses *objects-at* 22.

26 *<Convert the objects to descriptions. 26>≡*
 mapcar #'describe-obj

This code is used in chunk 27.

```

28 <session 7>+≡
    > (describe-objects 'living-room *objects* *object-locations*)
    (YOU SEE A WHISKEY ON THE FLOOR.
     YOU SEE A BUCKET ON THE FLOOR)

```

Uses **object-locations** 20, **objects** 20, and *describe-objects* 27.

Describing It All

```

31 <* 1>+≡
    (defun look ()
      (append (describe-location *location* *nodes*)
               (describe-paths *location* *edges*)
               (describe-objects *location* *objects* *object-locations*)))

```

Defines:

look, used in chunks 30 and 35.

Uses **edges** 13, **location** 29, **nodes** 5, **object-locations** 20, **objects** 20, *describe-location* 8, *describe-objects* 27, and *describe-paths* 19.

Walking Around in Our World

Given a *direction*, *<locate the path marked with the appropriate direction 34>* and *<try to go in that direction 35>*. Since the *direction* will be there, *<match against the cadr of each path 33>*.

```

34 <locate the path marked with the appropriate direction 34>≡
    (find direction
      (look up the available walkings paths 32)
      (match against the cadr of each path 33))

```

This code is used in chunk 38.

If such a path is found, *<adjust the player's position 36>*, otherwise *<admonish the player 37>*.

```

35 <try to go in that direction 35>≡
    (if next
      (progn (adjust the player's position 36)
              (look))
      (admonish the player 37))

```

This code is used in chunk 38.

Uses *look* 31.

```

38 <* 1>+≡
    (defun walk (direction)
      (let ((next (locate the path marked with the appropriate direction 34)))
        (try to go in that direction 35)))

```

Defines:

walk, never used.

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

```

29 <define the global variables 5>+≡
    (defparameter *location* 'living-room)

```

This code is used in chunk 1.

Defines:

location, used in chunks 31, 32, 36, and 41.

```

30 <session 7>+≡
    > (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.)

```

Uses *look* 31.

```

32 <look up the available walkings paths 32>≡
    (cdr (assoc *location* *edges*))

```

This code is used in chunk 34.

Uses **edges** 13 and **location** 29.

```

33 <match against the cadr of each path 33>≡
    :key #'cadr

```

This code is used in chunk 34.

```

36 <adjust the player's position 36>≡
    (setf *location* (car next))

```

This code is used in chunk 35.

Uses **location** 29.

```

37 <admonish the player 37>≡
    '(you cannot go that way.)

```

This code is used in chunk 35.

Picking Up Objects

If *<the object is on the floor 39>*, *<pick it up 40>*.

```
41 <get the list of objects here 41>≡
    (objects-at *location* *objects* *object-locations*)
This code is used in chunk 39.
Uses *location* 29, *object-locations* 20, *objects* 20, and objects-at 22.

42 <* 1>+≡
    (defun pickup (object)
      (if <the object is on the floor 39>
        (progn <pick it up 40>)
        '(you cannot get that)))
```

Defines:
pickup, used in chunk 43.

```
43 <session 7>+≡
    > (pickup 'whiskey)
    (YOU ARE NOW CARRYING THE WHISKEY)
Uses pickup 42.
```

Checking Our Inventory

```
46 <* 1>+≡
    (defun inventory ()
      (cons 'items- <retrieve the list of carried objects 44>))
Defines:
inventory, used in chunk 45.
```

```
39 <the object is on the floor 39>≡
    (member object <get the list of objects here 41>))
This code is used in chunk 42.

40 <pick it up 40>≡
    (push (list object 'body) *object-locations*)
    '(you are now carrying the ,object)
This code is used in chunk 42.
Uses *object-locations* 20.
```

```
44 <retrieve the list of carried objects 44>≡
    (objects-at 'body *objects* *object-locations*)
This code is used in chunk 46.
Uses *object-locations* 20, *objects* 20,
and objects-at 22.

45 <session 7>+≡
    > (inventory)
    (ITEMS- WHISKEY)
Uses inventory 46.
```

Full Listing

```

(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*)))

```


Chunks

⟨* 1⟩ [1](#), [8](#), [14](#), [19](#), [22](#), [27](#), [31](#), [38](#), [42](#), [46](#)
 ⟨adjust the player's position 36⟩ [35](#), [36](#)
 ⟨admonish the player 37⟩ [35](#), [37](#)
 ⟨at-loc-p 21⟩ [21](#), [22](#)
 ⟨attic description 4⟩ [4](#), [5](#)
 ⟨attic path 11⟩ [11](#), [13](#)
 ⟨Convert the edges to descriptions. 16⟩ [16](#), [19](#)
 ⟨Convert the objects to descriptions. 26⟩ [26](#), [27](#)
 ⟨define the global variables 5⟩ [1](#), [5](#), [13](#), [20](#), [29](#)
 ⟨describe-obj 24⟩ [24](#), [27](#)
 ⟨Find the objects at the current location. 25⟩ [25](#), [27](#)
 ⟨Find the relevant edges. 15⟩ [15](#), [19](#)
 ⟨garden description 3⟩ [3](#), [5](#)
 ⟨garden path 10⟩ [10](#), [13](#)
 ⟨get the list of objects here 41⟩ [39](#), [41](#)
 ⟨Join the descriptions. 17⟩ [17](#), [19](#), [27](#)
 ⟨living-room description 2⟩ [2](#), [5](#)
 ⟨living-room paths 9⟩ [9](#), [13](#)
 ⟨locate the path marked with the appropriate direction 34⟩ [34](#), [38](#)
 ⟨look up a location 6⟩ [6](#), [8](#)
 ⟨look up the available walkings paths 32⟩ [32](#), [34](#)
 ⟨match against the cadr of each path 33⟩ [33](#), [34](#)
 ⟨pick it up 40⟩ [40](#), [42](#)
 ⟨retrieve the list of carried objects 44⟩ [44](#), [46](#)
 ⟨session 7⟩ [7](#), [12](#), [18](#), [23](#), [28](#), [30](#), [43](#), [45](#)
 ⟨the object is on the floor 39⟩ [39](#), [42](#)
 ⟨try to go in that direction 35⟩ [35](#), [38](#)

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: [13](#), [18](#), [31](#), [32](#)
 location: [29](#), [31](#), [32](#), [36](#), [41](#)
 nodes: [5](#), [7](#), [31](#)
 object-locations: [20](#), [23](#), [28](#), [31](#), [40](#), [41](#), [44](#)
 objects: [20](#), [23](#), [28](#), [31](#), [41](#), [44](#)
 describe-location: [7](#), [8](#), [31](#)
 describe-objects: [27](#), [28](#), [31](#)
 describe-path: [12](#), [14](#)
 describe-paths: [18](#), [19](#), [31](#)
 inventory: [45](#), [46](#)
 look: [30](#), [31](#), [35](#)
 objects-at: [22](#), [23](#), [25](#), [41](#), [44](#)
 pickup: [42](#), [43](#)
 walk: [38](#)