The Guess-My-Number Game[1]

Eric Bailey

January 14, 2017 [2]

[1] Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 2, pages 21–30. No Starch Press, 2010. ISBN 9781593273491. URL http://landoflisp.com

[2] Last updated October 15, 2017

In this game, you pick a number from 1 to 100, and the computer has to guess it.

Defining the Small and Big Variables

To give the computer a range of numbers in which to guess, we define the lower and upper limits, `*small*` and `*big*`, respectively. We'll need to ⟨*reset the global state* 1⟩ as such whenever we want to restart the game,

1 ⟨*reset the global state* 1⟩≡
```
(defparameter *small* 1)
(defparameter *big* 100)
```
This code is used in chunks 2 and 12.
Defines:
big, used in chunks 3 and 7.
small, used in chunks 3 and 10.

"Global variable names should start and end with asterisks (also known in this context as earmuffs)" [Brown and Rideau, 2017].

2 ⟨* 2⟩≡
⟨*reset the global state* 1⟩

This definition is continued in chunks 5, 8, 11, and 12.
Root chunk (not used in this document).

Defining the Guess-My-Number Function

With `*small*` and `*big*` defined, we can tell the computer how to guess a number (`guess-my-number`) within those limits.

The basic algorithm is to ⟨*halve the sum of the limits and shorten the result* 3⟩. To achieve that, we use Common Lisp's `ash` function to perform an *arithmetic right shift* by 1, i.e. $\lfloor \texttt{sum} \times 2^{-1} \rfloor$.

To define the `guess-my-number` function, we simply implement the algorithm described in pseudocode in Figure 1.

Figure 1: The guessing algorithm

```
sum ← small + big
right shift sum by 1
return sum
```

3 ⟨*halve the sum of the limits and shorten the result* 3⟩≡
```
(ash (+ *small* *big*) -1)
```
This code is used in chunk 5.
Uses *big* 1 and *small* 1.

Now, when we want to ⟨*have the computer guess a number* 4⟩, we simply call `guess-my-number` as follows.

4 ⟨*have the computer guess a number* 4⟩≡
```
(guess-my-number)
```
This code is used in chunks 6, 8, 9, 11, and 12.
Uses guess-my-number 5.

5 ⟨* 2⟩+≡
```
(defun guess-my-number ()
  ⟨halve the sum of the limits and shorten the result 3⟩)
```

Defines:
guess-my-number, used in chunk 4.

Defining the Smaller and Bigger Functions

To define the `smaller` function, we need to update the global state such that the next guess is *smaller* than the last, i.e. ⟨*set* `*big*` *to one less than the last guess* 7⟩ then ⟨*have the computer guess a number* 4⟩.

7 ⟨*set* `*big*` *to one less than the last guess* 7⟩≡
```
(setf *big* ⟨subtract one from the most recent guess 6⟩)
```
This code is used in chunk 8.
Uses `*big*` 1.

8 ⟨* 2⟩+≡
```
(defun smaller ()
  ⟨set *big* to one less than the last guess 7⟩
  ⟨have the computer guess a number 4⟩)
```

Defines:
`smaller`, used in chunk 13.

To appropriately adjust `*big*`, ⟨*subtract one from the most recent guess* 6⟩.

6 ⟨*subtract one from the most recent guess* 6⟩≡
```
(1- ⟨have the computer guess a number 4⟩)
```
This code is used in chunk 7.

To define the `bigger` function, we need to update the global state such that the next guess is *bigger* than the last, i.e. ⟨*set* `*small*` *to one greater than the last guess* 10⟩ then ⟨*have the computer guess a number* 4⟩.

10 ⟨*set* `*small*` *to one greater than the last guess* 10⟩≡
```
(setq *small* ⟨add one to the most recent guess 9⟩)
```
This code is used in chunk 11.
Uses `*small*` 1.

11 ⟨* 2⟩+≡
```
(defun bigger ()
  ⟨set *small* to one greater than the last guess 10⟩
  ⟨have the computer guess a number 4⟩)
```

Defines:
`bigger`, used in chunk 13.

To appropriately adjust `*small*`, ⟨*add one to the most recent guess* 9⟩.

9 ⟨*add one to the most recent guess* 9⟩≡
```
(1+ ⟨have the computer guess a number 4⟩)
```
This code is used in chunk 10.

Defining the Start-Over Function

At this point, to define the `start-over` function is trivial. We simply ⟨*reset the global state* 1⟩ then ⟨*have the computer guess a number* 4⟩.

12 ⟨* 2⟩+≡
```
(defun start-over ()
  ⟨reset the global state 1⟩
  ⟨have the computer guess a number 4⟩)
```
Defines:
`start-over`, used in chunk 13.

Full Listing

```
(defparameter *small* 1)
(defparameter *big* 100)


(defun guess-my-number ()
  (ash (+ *small* *big*) -1))


(defun smaller ()
  (setf *big* (1- (guess-my-number)))
  (guess-my-number))


(defun bigger ()
  (setq *small* (1+ (guess-my-number)))
  (guess-my-number))


(defun start-over ()
  (defparameter *small* 1)
  (defparameter *big* 100)
  (guess-my-number))
```

Example Session

After loading src/guess.lisp, you might have ⟨*a session* 13⟩ like this:

```
$ rlwrap sbcl --load src/guess.lisp
```

13 ⟨*a session* 13⟩≡

```
> (start-over)
50
> (smaller)
25
> (bigger)
37
> (bigger)
43
> (smaller)
40
> (bigger)
41
> (bigger)
42
```

Root chunk (not used in this document).
Uses bigger 11, smaller 8, and start-over 12.

Index

Chunks

References

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 2, pages 21–30. No Starch Press, 2010. ISBN 9781593273491. URL http://landoflisp.com.

Robert Brown and François-René Rideau. Google Common Lisp Style Guide: Global variables and constants. https://google.github.io/styleguide/lispguide.xml?showone=Global_variables_and_constants#Global_variables_and_constants, September 2017. Accessed: 2017-10-08.