The Guess-My-Number Game[1]

Eric Bailey

January 14, 2017 [2]

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

1a ⟨* 1a⟩≡
```
(in-package :cl-user)
(defpackage lol.guess
  (:use :cl :prove)
  (:export :bigger
           :smaller
           :start-over))
(in-package :lol.guess)
```

⟨*reset the global state* 1b⟩

This definition is continued in chunks 1 and 2.
Root chunk (not used in this document).
Defines:
lol.guess, used in chunk 3.
Uses bigger 2f, smaller 2c, and start-over 2g.

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* 1b⟩ as such whenever we want to restart the game,

1b ⟨*reset the global state* 1b⟩≡
```
(defparameter *small* 1)
(defparameter *big* 100)
```
This code is used in chunks 1a and 2g.
Defines:
big, used in chunks 1c and 2b.
small, used in chunks 1c and 2e.

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* 1c⟩. 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
```

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

Now, when we want to ⟨*have the computer guess a number* 1d⟩, we simply call `guess-my-number` as follows.

1d ⟨*have the computer guess a number* 1d⟩≡
```
(guess-my-number)
```
This code is used in chunk 2.
Uses guess-my-number 1e.

1e ⟨* 1a⟩+≡
```
(defun guess-my-number ()
  ⟨halve the sum of the limits and shorten the result 1c⟩)
```

Defines:
guess-my-number, used in chunk 1d.

[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 18, 2017

src/guess.lisp:

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

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* 2b⟩ then ⟨*have the computer guess a number* 1d⟩. 2a

2b ⟨*set* `*big*` *to one less than the last guess* 2b⟩≡
```
(setf *big* ⟨subtract one from the most recent guess 2a⟩)
```
This code is used in chunk 2c.
Uses `*big*` 1b.

2c ⟨* 1a⟩+≡
```
(defun smaller ()
  ⟨set *big* to one less than the last guess 2b⟩
  ⟨have the computer guess a number 1d⟩)
```

Defines:
`smaller`, used in chunks 1a and 3.

To appropriately adjust `*big*`, ⟨*subtract one from the most recent guess* 2a⟩.

⟨*subtract one from the most recent guess* 2a⟩≡
```
(1- ⟨have the computer guess a number 1d⟩)
```
This code is used in chunk 2b.

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* 2e⟩ then ⟨*have the computer guess a number* 1d⟩. 2d

2e ⟨*set* `*small*` *to one greater than the last guess* 2e⟩≡
```
(setq *small* ⟨add one to the most recent guess 2d⟩)
```
This code is used in chunk 2f.
Uses `*small*` 1b.

2f ⟨* 1a⟩+≡
```
(defun bigger ()
  ⟨set *small* to one greater than the last guess 2e⟩
  ⟨have the computer guess a number 1d⟩)
```

Defines:
`bigger`, used in chunks 1a and 3.

To appropriately adjust `*small*`, ⟨*add one to the most recent guess* 2d⟩.

⟨*add one to the most recent guess* 2d⟩≡
```
(1+ ⟨have the computer guess a number 1d⟩)
```
This code is used in chunk 2e.

Defining the Start-Over Function

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

2g ⟨* 1a⟩+≡
```
(defun start-over ()
  ⟨reset the global state 1b⟩
  ⟨have the computer guess a number 1d⟩)
```
Defines:
`start-over`, used in chunks 1a and 3.

Full Listing

```
(in-package :cl-user)
(defpackage lol.guess
  (:use :cl :prove)
  (:export :bigger
           :smaller
           :start-over))
(in-package :lol.guess)


(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))
```

Tests

3 ⟨*test/guess.lisp* 3⟩≡

```
(in-package :lol.guess)


(plan 1)


(subtest "A Plausible Session"
  (is (start-over) 50 "(start-over) ; => 50")
  (is (smaller)    25 "(smaller)    ; => 25")
  (is (bigger)     37 "(bigger)     ; => 37")
  (is (bigger)     43 "(bigger)     ; => 43")
  (is (smaller)    40 "(smaller)    ; => 40")
  (is (bigger)     41 "(bigger)     ; => 41")
  (is (bigger)     42 "(bigger)     ; => 42"))


(finalize)
```

Root chunk (not used in this document).
Uses `bigger` 2f, `lol.guess` 1a, `smaller` 2c, and `start-over` 2g.

Chunks

Index

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.