Homework, Week 7, part C

lightprogramming.org

October 21, 2016

This continues with the Hangman game. We know how to get the answer from the first player and how to obscure the answer for the second player (at least if the second player doesn't peek — we'll solve this problem in a few weeks.) We can now write the next bit, which is how to process the guesses.

To begin with, we note that the game consists of a series of guesses, and that the same code can be used for every guess. This strongly suggests some kind of loop. However, we need to check with each occurance of a loop to see if the game has ended, that is, whether the second player has guessed the word or has run out of guesses. The second part is easy: declare a variable for the number of guesses and increment it with each wrong guess. When the bad guesses exceed the number of attempts, the player loses. How would you check to see if the player wins? I took a very simple approach. If no letters are replaced by the obscuring function, the player wins. In other words, if there are no * in *obscure* the game is over.

Here is my game loop function. It takes three arguments, all lists: the list of the correct answer, the list of good guesses, and the list of bad guesses. It first gets the obscured answer, then gives clues to the player, and then checks the game status. If the length of the bad guesses list exceeds the number of attempts, the player loses. If the obscured text does not contain an *, the player wins. Otherwise. the function calls a second function to process the player's guess.

```
1  (defun game-loop (answer good-guesses bad-guesses)
2    (setf obscure (get-obscure good-guesses answer))
3    (format t " Your clue is ~a~%Your bad guesses are ~a~%"
4              obscure bad-guesses)
5    (cond
6      ((= (length bad-guesses) tries)
7          (format t "Sorry, you lose. The answer is ~a~%"
8              (coerce answer 'string)))
9      ((not (member #\* obscure))
10         (format t "You win. The answer is ~a.~%"
11             (coerce answer 'string)))
12     (t (process-guess answer good-guesses bad-guesses))))
```

The function to process the guess is very simple. It takes three arguments, all lists: the correct answer, the list of good guesses, and the list of bad guesses. It prompts the player to enter a guess. reads the character he enters, and converts it to upper case. If the guess is contained in the answer, it calls `game-loop`, consing the guess to the good-guesses list. If not, it calls `game-loop`, consing the guess to the bad-guesses list. (The `remove-if-not` function is necessary to remove any characters that are not alphabetical characters, such as the newline character.)

```
1  (defun process-guess (answer good-guesses bad-guesses)
2    (format t "~%Enter a letter to guess: ~%")
3    (setf guess (char-upcase (read-char)))
4    (cond
5      ((member guess answer)
6        (game-loop answer (cons guess good-guesses) bad-guesses))
```

```
7       (t
8         (game-loop answer good-guesses (remove-if-not #'alpha-char-p (cons
              guess bad-guesses))))))
```