

Running E-Z Reader 10 Simulations

The program for running simulations using the E-Z Reader 10 model was written in Java, version 1.5. The executable (.jar) version of the program is available at: www.pitt.edu/~reichle/ezreader.html. The source code (i.e., .java classes) is also available from me (Erik Reichle, at: reichle@pitt.edu) upon request. These instructions are specifically for E-Z Reader 10. Because the program for E-Z Reader 10 is very similar to the one for E-Z Reader 9, these instructions will only describe those details that differentiate the former model from the latter. For a general (but very detailed) description of running E-Z Reader models, including how to set up the input files and interpreting the simulation results, consult the instructions for E-Z Reader 9 (*EZR9Instructions.pdf*).

You will need three files to run E-Z Reader 10 simulations: (1) the program file containing the actual model (*E-Z_Reader_10.jar*), (2) a file containing the sentences/text that will be used in the simulation (e.g., *SRC98Corpus.txt*), and (3) a file used to identify specific target words of interest (e.g., *SRC98Targets.txt*). The format of the sentence corpus file is exactly as described in the instructions for E-Z Reader 9. The file that is used to identify the target words consists of a single column of integers, with each integer indicating the ordinal position of one target word per sentence. For example, in *SRC98Targets.txt*, the file begins with “5, 5, 3, 9...” arranged in a vertical column. These numbers identify the target words used by Schilling, Rayner, and Chumbley (1998) in their eye-movement experiment. These target words are contained in the corpus file *SRC98Corpus.txt*. If you examine the first four sentences in that file, the target words are “apartment,” “president,” “students,” and “baseball,” respectively. Please note that—as per Java conventions—both the sentences and words are numbered starting with zero, so that word #5 in the first sentence is actually the sixth word in the sentence.

The program will not run without a target word file (i.e., the field labeled “Target Word File Name:” cannot be left empty). By using this file to identify specific target words, it is possible to adjust the values of the free parameters (i.e., I , p_F , and p_N) that control post-lexical (integration) processing of those words independently of the parameter values that control post-lexical processing of the other words in the sentence corpus. To do this, note that, in the GUI panel labeled “Free Parameters” there is a panel labeled “Post-Lexical.” This panel contains six fields that can be used to set the values of the model’s post-lexical processing parameters. These parameters are as follows:

1. I = the integration time (in ms) for all words in the sentence corpus *except* the target words.
2. p_F = the probability of integration interruption/failure for all words in the sentence corpus *except* the target words.
3. p_N = the probability of directing an inter-word regression back to the problematic word (i.e., the word that resulted in integration failure/interruption) for all words in the sentence corpus *except* the target words. Note that, with

probability $1.0 - p_N$, the inter-word regression will be directed back towards the word preceding the problematic word.

4. $I(n)$ = the integration time (in ms) for the target words (indexed by n).

5. $p_F(n)$ = the probability of integration interruption/failure for the target words.

6. $p_N(n)$ = the probability of directing an inter-word regression back to the problematic target word. Again, with probability $1.0 - p_N$, the inter-word regression will be directed back towards the word preceding the problematic target word.

Notice that, by default, the parameters are set to values that treat the target words exactly the same as non-target words. Thus, if you are not interested in running simulations to examine how variables affecting post-lexical integration in turn affect eye movements on specific target words, simply construct a “dummy” file contain a single column of numbers (e.g., 0’s or 1’s, with one number per sentence) and set the values of $I(n)$, $p_F(n)$, and $p_N(n)$ to the values of I , p_F , and p_N , respectively. This will allow the program to run and the model will treat all of the words in the sentence corpus in the same manner.

One other point is worth noting regarding the model and integration failure: The values of $I(n)$ and $p_F(n)$ are set equal to I and p_F , respectively, after integration failure on target word n . This was done because completely precluding integration failure from happening a second time (as was done in the simulations reported in Reichle et al., 2009) is probably less plausible than assuming that it is less likely to occur during the second pass through the text (i.e., that integration failure occurs with the same “default” probability as with other words in the sentence). And irrespective of whether the target word is regressed to or not [i.e., depending on the value of $p_N(n)$ and/or saccadic error], the values of $I(n)$ and $p_F(n)$ will always revert to I and p_F , respectively (i.e., the probability of integration failure is reduced for the “problematic” word, not the word that is actually fixated following the regression). As indicated in Reichle et al. (2009, p. 7), these assumptions regarding what happens during the second pass through the text are included to make the modeling possible in the absence of having a detailed computational model of what actually happens with post-lexical language processing, how it fails, and how such failures are repaired. That being said, the model is not to be taken as a serious description of the cognitive processes that are operative during the re-reading of sentences; the model’s theoretical scope is instead much more modest, being a description of how problems with post-lexical processing might interrupt on-going lexical processing to produce pauses and/or short (1-3 word) inter-word regressions. A detailed computational model of post-lexical integration (and of all of the E-Z Reader processes, for that matter) is a “story for another day.” After all, as all modelers know, the process of developing a model is an incremental process. ☺

Finally, all of the other details related to running E-Z Reader 10 are exactly the same as for E-Z Reader 9.

Don't hesitate to contact me if you have any questions or run into any problems. Good luck!

Best regards,
Erik