Workflow

0. Preregistration
1. Extract variable forms from webcorpus frequency list
2. Generate nonce words based on syllable ngrams in variable forms
3. Run a wug test with variable forms to gauge how much they prefer A or B variant
4. Create nonce word lists for matching game based on wug test
5. Set up matching game with high / low ; typical / reversed co-players based on nonce word lists
6. Run matching game
7. Run matching game post-sleep integration task
8. Analyse results

In theory, you need to run all this once. In practice, you need to run 1-5 once *right*. Doing 3 once is relatively easy since all you do is run one big wug test online. Running 6 and 7 is immensely finicky. You run a couple conditions. You run them in iterations and broken up across testing periods (twice a year). So there's really no point in writing a container or a terraform config, here's a pile of code instead. I write down the details for myself so I remember when I come back to it.

0. Prereg

In 'preregs' and on aspredicted. I like aspredicted. You can cross-check github and gitlab commits and the date of the preregistrations to see that I didn't do anything before the preregs were submitted.

1. Extract variable forms from webcorpus frequency list

Webcorpus 2 is *here*. I compiled a frequency list which is *here*.

2. Generate nonce words based on syllable ngrams in variable forms

The scripts to identify the three patterns and to generate nonce words is *here*. It has its own readme.

3. Run a wug test with variable forms to gauge how much they prefer A or B variant

Nonce words were split into three lists that each have a 1:1:1 ratio of the three variable patterns. This was fed to psychopy which picked a list at random,

looped through it in random order, and pushed each line (=word) into a written forced-choice prompt.

The exp code is *here*. This was just run until we've run out of students that semester.

This is a simple forced-choice task coded in psychopy. Psychopy takes care of setting up the Pavlovia repository for you and moving data back and forth. It works most of the time.

You can ask Pavlovia to store data in a database format or store it in the gitlab repo. I do the latter so I can pull the data to local in the CLI.

4. Create nonce word lists for matching game based on wug test

The code is *here*. The lists are turned into js arrays for jspsych

5. Set up matching game with high / low ; typical / reversed co-players based on nonce word lists

The exp code is *here*.

It was coded in jspsych. There are two versioning vulnerabilities here. Jspsych uses plugins that implement simple trial types people might want to use in an experiment. You can get some things done using accessible parameters of the plugin. You can get other things done by editing the plugin. So I have two custom button clicker plugins stored locally that are not maintained anywhere and will eventually get out of sync with the rest of js/jspsych. Also, the last version of the jspsych pavlovia plugin simply didn't work for me so I had to try previous versions to find one that would work. That's also included locally and not maintained. The latest version may or may not work. Also also an experiment set up to work on pavlovia will not work locally.

There are three js files storing arrays for the three variable patterns. There are three lists for each pattern. Each list can be used as a high typical / low typical / high reversed / low reversed list. This gives us 3 x 3 x 4 = 36 lists. I need 7 people per list to get 21 people per condition (e.g. "levelling, low, typical" etc.).

I have an index file for the three patterns each in the indices folder in the exp dir. One for running on pavlovia using our local participant recruitment and the main matching task and, for levelling and vowel deletion, one for running on pavlovia through prolific for the integration task.

I pull data in using the CLI. The code to analyse the data is *here*.

The script `process_esp_data` is for the main task. The script `process_esp_data2` is for the integration task run on prolific. There are pre-registered exclusion

criteria for the main task. So you might have enough participants in one condition (7 per list x 3 x 4) but some of them don't qualify and their list has to be re-run. This is a lot of back and forth:

6. Run matching game

i. You start running e.g. the levelling index.html. It randomly picks one out of twelve arrays (3 lists x 4 conditions) to run with the participant.
ii. You keep track of how many lists ran how many times using the `process` script and once you hit seven you remove the number from the randomiser in index.html.
iii. Unless the list overflowed since you last checked. Then, you effectively lost a participant.
iv. This is impossible / very hard to do any other way, since jspsych can't really write anything other than the output into its own gitlab folder. So you have to do this by hand.
v. You don't want to store **all** data in one output folder because then the processor script will run for ages. You want to batch it.

7. Run matching game integration task

This has the added complexity that you have to get the same people back on prolific. I did this by paying them the bare minimum for the first part (this is the same as the main task) and then told them they'll get more for doing the second part which will be like five minutes. Almost all people came back, so that worked nicely. But it was a lot of fiddling with submitting the task and checking etc.

8. Analysis

Script *here*. Of primary interest is `esp_analysis`. `source_esp` loads all the data and the best model. `esp_analysis` selects best model using `lme4`. I played around with `rstan` and realised I have no idea what the priors should be, was, ironically, uncertain about the posteriors as a result, and generally found the whole experience slow and confusing. Since everything is pre-registered we can go with alpha = .05. Analysis outlined in big `Readme.Rmd`. Distilled for paper.