Abertay University

Levenik Code Documentation

Leverhulme Trust Project: Literacy Acquisition in Situations of Dialect Exposure

Nikolay Panayotov
Glenn P. Williams
Vera Kempe

March 2019

1 TABLE OF CONTENTS

2 About the Project Code and This Documentation

The Project, codenamed *Levenik*, is a series of web-based psychology experiments investigating the influence of dialect exposure on literacy acquisition, employing an artificial language paradigm. The project was carried out by Dr Glenn P. Williams and Prof Vera Kempe with Nikolay Panayotov building the necessary technical solutions. The present documentation outlines the design and implementation of the experiments from a technical standpoint. Familiarity with the project at large is assumed.

All project files referred to here are built with HTML5 and related web technologies, such as JavaScript for client-side code and PHP for server-side code.

2.1 Included Project Files

2.1.1 allp-demo:

This is a greatly abridged demo version of the experimental procedure that does not store any data on a server with the exception of temporary session variables. It was simply used for illustrative purposes.

2.1.2 word-permutations:

This tiny web-tool was used as part of the process of generating the artificial words in the language used in the experiments. It allows for the generation of all possible strings from the predefined set of consonants and vowels, based on a specified syllable structure and subject to predefined consonant clusters. The tool is self-contained and not the focus of this documentation – it is provided as is, for information purposes in relation to the generation of words in the language.

2.1.3 levenik-exp1:

The first Levenik experiment (note: numbers do not reflect the exact order of construction, but labelling conventions set by the researchers) only includes a reading task.

2.1.4 levenik-exp2ab:

The second Levenik experiment also includes a writing task.

2.1.5 levenik-exp3:

The third Levenik experiment extends 2ab by introducing more exposure and training screens.

2.1.6 levenik-exp4:

The forth Levenik experiment again changes the exact length of the experiment, but also restructures the order and type of presentation. It includes facilities to change some experimental variables mid-experiment and includes multiple speakers in one session.

2.1.7 levenik-db-structure.sql:

This SQL file contains the formal definition of the database used in Levenik-exp4, which is predominantly the same for all of the others and compatible with them.

This documentation focuses on Levenik-exp4, as the most recent and most complete version of the codebase. Most of what is written is applicable to all the others with minor exceptions.

3 Experiment Design

3.1 Sample Experiment Design

To understand the architecture of the codebase, it helps to have a clear example of the type of experiments it has set out to implement in the first place. Figure 1 demonstrates a simplified example of an experiment design flow.

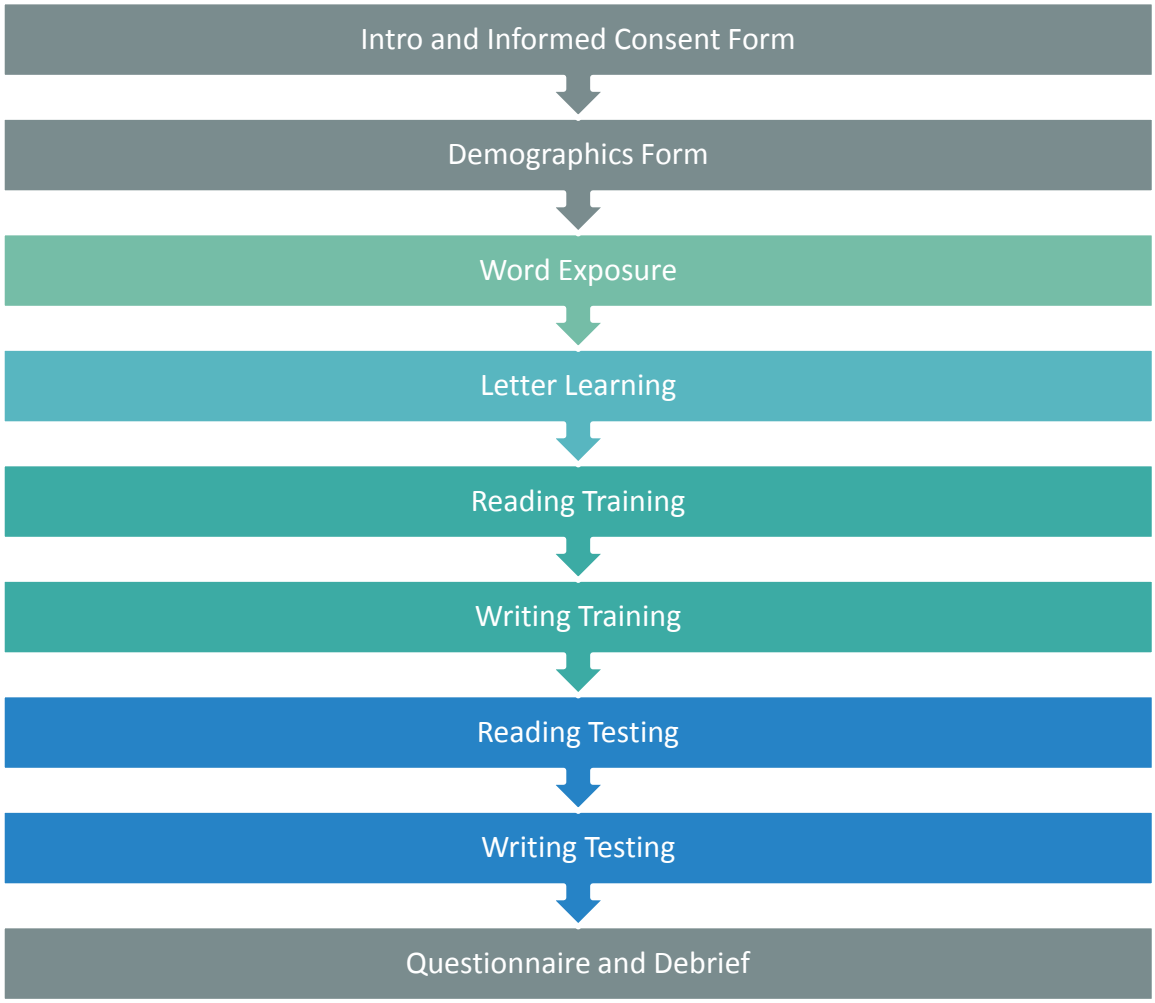

Figure 1. Example Experiment Structure Flow.

3.1.1 Intro Screens

These are standard information screens about the study, including an informed consent form and a basic demographics questionnaire like age and gender.

3.1.2 Word Exposure Screens

In word exposure, participants hear all the words of the artificial language and might also see pictures of their meanings. However, they do not see the written form of the word in the artificial script (Figure 2). There is an active and a passive type of exposure – where participants are asked to repeat the word or not respectively.

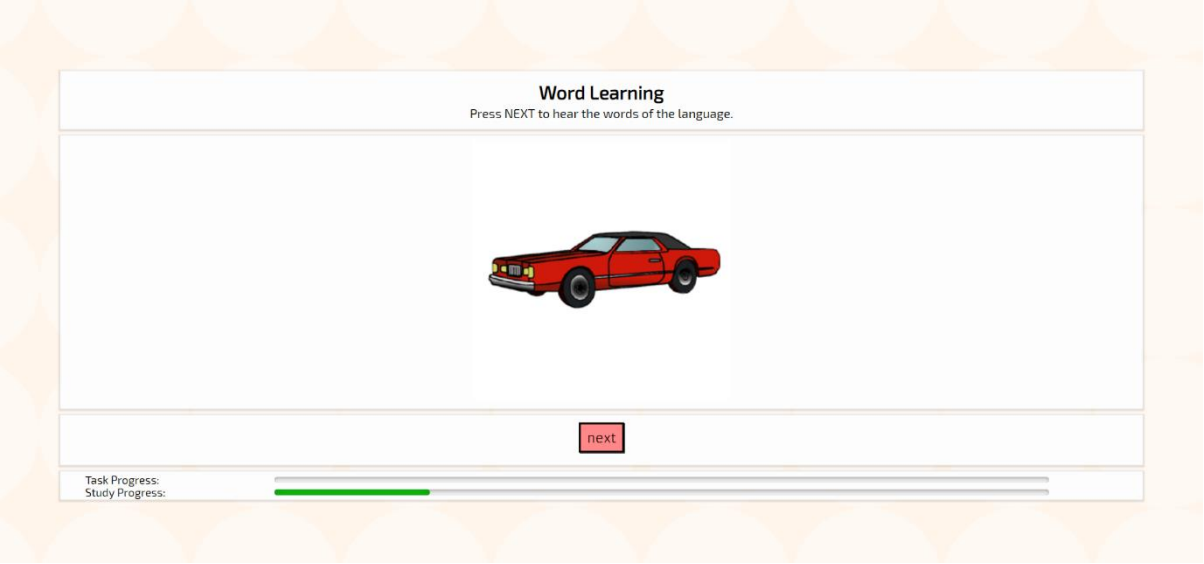

Figure 2. Example exposure screen.

3.1.3 Letter Learning

This stage presents all the letters of the artificial alphabet and their typical phonetic values one by one to the participant. Participants simply click through to hear the sounds of the language and their corresponding letters (Figure 3).

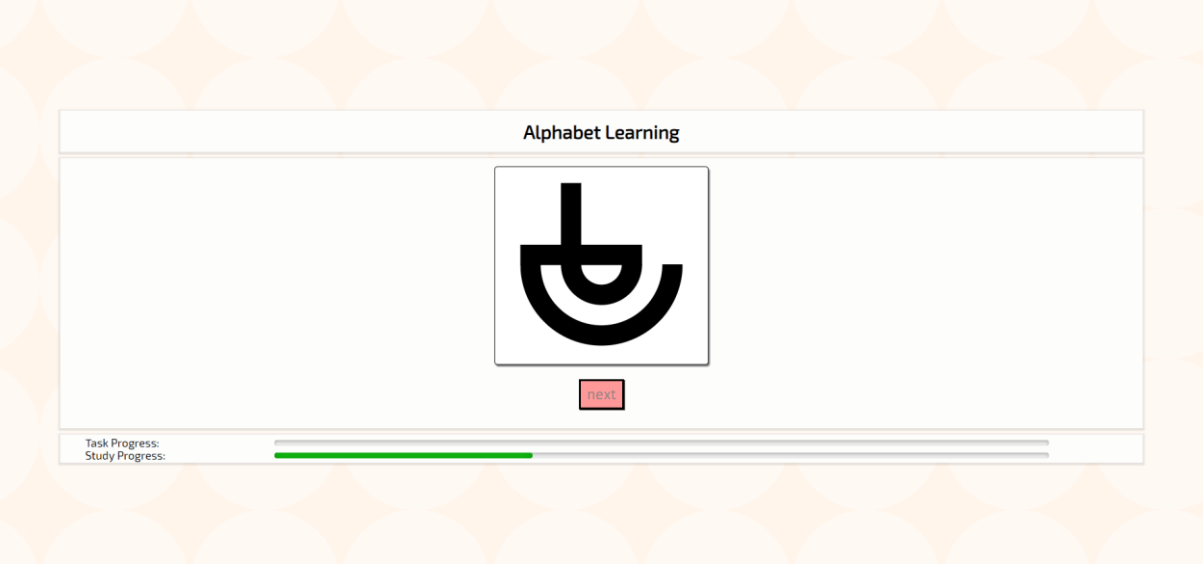

Figure 3. Example letter learning screen.

3.1.4 Reading Training

In the reading task, participants are shown the word in written form in the artificial script (which may also be accompanied by a picture of the meaning). Participants are asked to click on a button and record themselves reading the word out loud into their microphone (Figure 4). Their sound file is recorded on the study server. After recording, the participant is given feedback by hearing the correct pronunciation of the word before moving to the next word.

Figure 4. Example reading task screen.

3.1.5 Writing Training

In the writing task, participants hear the pronunciation of a word (which may also be accompanied by a picture of the meaning). Participants are asked to spell out the word in the artificial script by pressing on a virtual keyboard on the screen (Figure 5). Their input is recorded and before moving on to the next word, are also given feedback showing the correct spelling alongside their own.

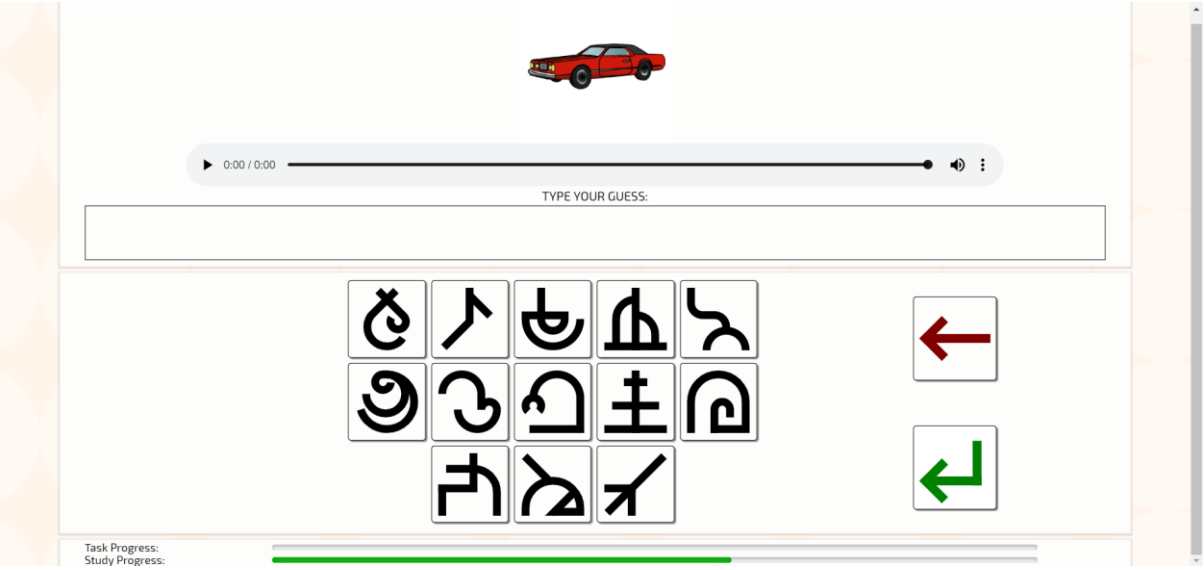

Figure 5. Example writing task screen.

3.1.6 Testing Screens

Testing screens (both reading and writing) are identical to the training screens, but exclude the feedback mechanisms. Furthermore, the test screens include a few more additional words, not seen or heard by the participant before.

3.1.7 Debrief Screens

At the end participants are asked about their experience with the study and are given some information on the purpose of the study.

3.2 Possible Conditional Variables

Several aspects of the experiment can be manipulated.

3.2.1 Task Type Order

The order of reading and writing tasks (either reading first or writing first) can be counterbalanced.

3.2.2 Speaker Gender

The speaker can be one of two voices, a male or a female.

3.2.3 Picture Associations for Words

When words are presented they can be accompanied by pictures of their meaning or not.

3.2.4 Dialect Variants

The variant of the language heard in the exposure screens can be either the same as the testing language or a different variety.

3.2.5 Orthographic Transparency

There are two possible orthographies to use. One is completely transparent and has a one-to-one correspondence between sounds and letters, while the other is more opaque and follows special predefined rules and exceptions.

3.2.6 Social Cue

The different language variants throughout the study can be signalled by additional instructions, a different voice for a different variant, and a picture representing the location of the speaker. Those are referred to as social cues and can be either present or absent.

3.2.7 Dialect Training

Training blocks reflect the variant used in testing, but in a different condition the last two training blocks can be in the other variant. This is referred to as dialect training.

3.2.8 Dialect Location

The pictures used (either a mountain village or a coastal village) for locations as social cues (see above) can be counterbalanced between the two language varieties used.

4 Database Design

4.1 Database Relationships

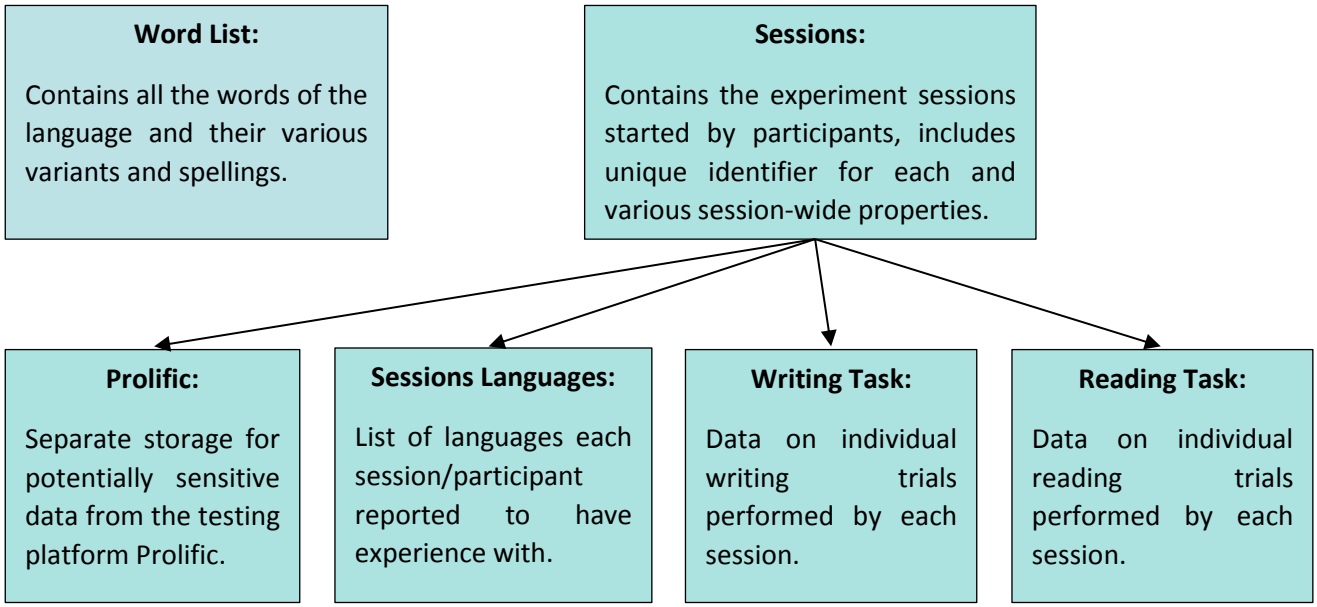

Figure 6. Basic relationship diagram and short descriptions of each table.

4.2 Table Outlines

4.2.1 `word_list`

`word_id`: unique numerical identifier for the word;

`word`: string representing the typical, variant 1, transparent spelling of the word;

`dialect_version`: string representing an alternative variant 2 transparent spelling of the word;

`opaque_spelling`: string representing an opaque spelling of variant 1;

`opaque_dialect_spelling`: string representing an opaque spelling of variant 2;

4.2.2 `sessions`

`session_number`: unique numerical identifier of the session – this links all the tables together (except word_list, which is independent);

`progress`: a string showing the exact screen (experiment state) at which a session currently is;

`completion_code`: a unique and randomly generated completion code presented to the participant at the end of the experiment;

`alphabet_key`: a string describing the associations between artificial letter and sound. Each artificial letter is known by a numerical identifier 1-14 while sounds are represented in Latin letters.

In the ‘alphabet_key’ the 14 locations of each character represents 1 to 14 artificial letters, while the value of the character (Latin letter) represents the sound for that artificial letter. Every session gets randomly assigned letter to sound correspondences, hence the alphabet_key stores this information concisely.

`language_condition`: the variant of the language used in exposure and dialect screens. ‘s’ (standard, variant matches testing); ‘d’ (dialect, variant is different from testing);

`order_condition`: describes the order of the type of task in training and in testing. ‘rw’ (first reading, then writing); ‘wr’ (first writing, then reading);

`picture_condition`: Boolean, describing whether or not words are simultaneously presented with pictures;

`speaker_condition`: the gender of the main speaker in the study (in some conditions the speaker can change mid-experiment). ‘m’ (male); ‘f’ (female);

`orthography_condition`: either an ‘o’ (opaque) or ‘t’ (transparent) version of the written words’ spelling is used.

`social_cue_condition`: Boolean, describing whether or not social cues (described in the experiment design section) are present.

`dialect_training_condition`: Boolean, describing whether or not dialect training (described in the experiment design section) is present.

`dialect_location_condition`: picture to be displayed as a social cue for the dialect. 1 (mountain village); 0 (coastal village picture);

`start_timestamp`: the time at which the session was started;

`end_timestamp`: the time at which the session reached the end of the experiment (if it has, otherwise blank);

`age`: the self-reported age of the participant;

`gender`: the self-reported gender of the participant (m: male; f: female; o: other);

`english`: the self-reported English proficiency of the participant (1: low proficiency; 5: high proficiency);

`fun`: the self-reported ‘enjoyment’ of the study by the participant (1: low enjoyment; 5: high enjoyment);

`noise`: the self-reported ‘hearing clarity’ of the participant, i.e. "I could hear the words in the study clearly." (1: strongly disagree; 5: strongly agree);

`browser`: the browser’s user agent string, i.e. the HTTP_USER_AGENT server variable as gathered by PHP, this typically shows the type of browser the participant was using;

4.2.3 `prolific`

`session_number`: links session to Prolific data;

`prolific_id`: Prolific ID provided by the Prolific platform;

`prolific_session`: Prolific Session provided by the Prolific platform;

`completion_code`: a unique and randomly generated completion code presented to the participant at the end of the experiment (repeated from sessions table);

4.2.4 `sessions_languages`

`input_id`: unique ID for individual entries;

`session_number`: links each stated language to the session that reported it;

`language`: the language as inputted by the participant;

`self_rating`: the fluency rating given by the participant (1 to 5, where 5 is native or bilingual proficiency);

4.2.5 `reading_task`

`trial_id`: unique identifier for the trial;

`timestamp`: when the trial was recorded;

`session_number`: the session that performed the trial;

`session_trial_id`: the consecutive trial number for this session;

`section`: the experiment screen name at which the trial was recorded;

`section_trial_id`: the consecutive trial number for this particular experiment screen;

`word_id`: the unique identifier for the word related to this trial;

`novel_word_for_task`: Boolean showing whether this word was encountered in a similar task before;

`exposure_count`: a count of how many times the session has encountered the word by the time of this trial;

`picture_id`: identifies the picture shown to the session for this word if in the picture condition;

`word_length`: the string length of the expected answer target for the trial;

`target`: the actual string of the expected answer target for the trial;

`participant_input`: the string the participant actually produced during the trial;

`correct`: a Boolean showing if the participant's input was correct (i.e. matched the target);

`edit_distance`: a normalised Levenshtein edit distance between the target and input strings;

4.2.6 `writing_task`

`trial_id`: unique identifier for the trial;

`timestamp`: when the trial was recorded;

`session_number`: the session that performed the trial;

`session_trial_id`: the consecutive trial number for this session;

`section`: the experiment screen name at which the trial was recorded;

`section_trial_id`: the consecutive trial number for this particular experiment screen;

`word_id`: the unique identifier for the word related to this trial;

`novel_word_for_task`: Boolean showing whether this word was encountered in a similar task before;

`exposure_count`: a count of how many times the session has encountered the word by the time of this trial;

`picture_id`: identifies the picture shown to the session for this word if in the picture condition;

`word_length`: the string length of the expected answer target for the trial;

`target`: the actual string of the expected answer target for the trial;

`participant_input`: the string the participant actually produced during the trial;

`correct`: a Boolean showing if the participant's input was correct (i.e. matched the target);

`edit_distance`: a normalised Levenshtein edit distance between the target and input strings;

5 CODE ORGANISATION

5.1 DIRECTORY AND FILES STRUCTURE

5.1.1 Entry part: outside the 'study' folder

This folder is the entry point for participants. Here there are some introductory pages and an ethics form, which is completely client-side and does not record any data until participants accept the terms and move on to the main part. URL variables are passed on to the URL of the main part.

5.1.2 Main part: inside the 'study' folder

Participants move on to this part after agreeing to the ethics form. This results in the creation of a new session from *create_new_session_for_index.php*, which connects to the database through *database_connection_for_index.php* and creates a new entry in the sessions table. The URL variables are extracted with *get_url_data.php* and conditions are set and recorded. The main study control flow is organised in a state machine in *index.php*.

5.1.3 Resources

There are two resource folders, one for images and one for audio.

Images contains the pictures used for words, language locations, and the interface, as well as SVG files for the artificial script letters.

Audio contains the sounds used for correct and incorrect responses, the sounds for individual letters, and the audio recordings for all the words in their different variants (both in male and female voice).

5.1.4 Pages / HTML

These are all the pages used in the experiment in HTML format. They are presented to the user from the experiment main control flow in *index.php*.

5.1.5 CSS

There are four styling sheets for the entire experiment:

normalize: standardises the default styling across browsers and is used on every page.

global_style: is specific to this experiment, but also applies to all pages.

listening_and_writing_style: this is a special case style applied to the writing task – styling the on-screen keyboard, etc.

reading_and_speaking_style: this is a special case style applied to the reading tasks – styling recording controls, etc.

5.1.6 JavaScript

The folder *recorderjs_dist* contains the framework used for recording audio from the user's microphone. *jQuery* and *Modernizr* are also used.

progress_bars.js is used to keep the study progress bars on the page it is attached to up to date. It can also be used to check the current section's progress and perform different actions based on whether all trials have been exhausted or not.

location_picture.js queries the server for the dialect location picture, which is determined by *get_location_picture.php* based on the preset conditions and the current screen type. After finding out the picture's reference, the client-side code displays it if in the relevant social cue condition.

The rest of the JavaScript files contain specific code to be run by particular sections of the experiment. Details are discussed in the next section.

5.1.7 PHP

This is the server-side code of the program, which is entirely written in PHP. Details of the implementation and specific files are discussed in the next section.

6 Implementation Design and Architecture

6.1 Setting Up

1. Use *levenik-db-structure.sql* to create the database on your server.

2. Modify the function **connect()** in *database_connection_for_index.php* and *database_connection.php* to direct the **parse_ini_file** function to an appropriate *config.ini* file (ideally, outside your server's *public_html* files folder) containing your database's credentials. Plug those in the **mysqli** constructor.

3. Outside your server's *public_html* files folder create a '*private*' folder. Inside the *private* folder, create a '*recordings*' folder: this is where the participant's audio recording will go.

4. Back in the levenik code, modify the session variable ***$_SESSION['folder']*** at the bottom of *create_new_session_for_index.php* to point to the newly created recordings folder.

5. Upload the updated levenik experiment folder to your server. Make sure your modified paths in the database connection and session creation files match your server's file directory.

6. Check that everything is working.

7. All done!

6.2 Initialising a Session

The study officially commences when the user requests the *index.php* file in the study folder and the program begins creating records and storing data. The first thing done by *index.php* is to check if a session has been created by checking an 'exists' property; if not, any potential URL variables are grabbed by *get_url_data.php* and a new session is initated through the *create_new_session_for_index.php*.

The *create_new_session_for_index.php* does several important tasks:

- Selects experimental conditions, based either on what *get_url_data.php* gathers or (if URL variables are not provided) on the session's order;
- Randomises the presentation of trials during the experiment so as to assure the correct balance of dialect and non-dialect words and storing that in ***$_SESSION['training_set']***.
- Using *generate_word_pictures.php* randomises which object picture corresponds to which word in a ***picture_key*** containing the names of the pictures in order of the word identifiers they represent.
- Using *generate_alphabet.php* randomises which artificial letter corresponds to which sound in an ***alphabet_key*** containing the sound names of each letter in order of the letter identifiers they represent.
- Creates a new session record in the sessions table and the associated Prolific table;

- Initialises and resets all experiment-wide variables to be discussed in other sections of this documentation.
- Sets the initial state of the program as 'START', which will be used by the study state machine. This is stored in ***$_SESSION['state']***.

6.3 Setting Experimental Conditions

Experimental conditions, as described previously, are set in the *create_new_session_for_index.php* either automatically based on the session number following a simple alternation pattern or explicitly by a custom URL of the study. The file *get_url_data.php* extracts specific conditional settings and stores them in the session, which is subsequently recorded in the session record in the sessions table. Other experimental conditions can easily be added by including them in the same way as the others.

6.4 The Study State Machine

The main role of *index.php* is to manage the experiment's state machine – this is implemented with a simple switch statement. The first 'START' state initiates the order and total number of states (these are used in the progress bar shown to participants throughout the study).

6.4.1 Handling States

Each state (or section) in the experiment has a certain number of trials (items) that the participants must pass before moving to the next section, called ***'item_total'***. When a trial is passed, a counter called ***'item_order'*** is incremented by 1. When ***item_order*** exceeds ***item_total***, *index.php* is refreshed by the client page and the state machine progresses to the next state. In order for the client to find out that the ***item_order*** has reached ***item_total***, it uses the function **updateProgressBars(continueAction, finishedAction)** from *progress_bars.js*. This function first queries the server for the exact numbers of ***item_order*** and ***item_total*** to visually update the on-screen progress bars. Then it decides to either perform **continueAction** or **finishedAction**, which are passed to it in the form of function variables. Usually the **finishedAction** is a window refresh that allows for the *index.php* state machine to be run again and updated so that the participant can progress to the next screen. On the other hand, **continueAction** tends to simply prepare the client to query the server for the next item/trial in the experiment, until all items are passed.

The function **stateTransition($current_state, $next_state)** in *index.php* determines the HTML page to be loaded for the current state and the name of the next state to go to once the total number of trials (items) in the current state have been passed. When a state is updated, this is also recorded in the session record in the sessions table, under the ***'progress'*** field. This field is used to keep track of the state machine.

At the end of the state machine, the study completion code is also created and recorded, which is then displayed to the participant in the final HTML page.

6.4.2 Initiating Sections

When switching to a new state, the state machine runs an initialisation code from *initialise_section.php*. This primarily does the randomisations of the item order in the various sections and serves the correct subset of items from the 'training_set' to the 'section_set' (or in the case of testing sections, the complete set).

6.5 Study Sections

6.5.1 Information Screens and Questionnaires

Information screens and questionnaire forms are simply treated like regular sections that have only 1 total trials/items which is passed when the 'next' or 'submit' button is pressed: this is accomplished through the *next_item.php* file. As usual, this triggers a page refresh and running the state machine again to transition to the appropriate next screen.

6.5.2 Word Learning

There are three versions of the *learn_words* server-side script in *levenik-exp4*:

- *learn_words_old.js* refers to the exposure sections in experiments (1-3) which used 'passive' exposure, where participants simply have to click through and hear all the words. Each item is queried with *php/get_random_word.php* (and 'exposure' passed as a section identifier string) and progress is moved simply with *php/next_item.php*.
- *learn_words.js* and *learn_words_test.js* are nearly identical to *reading.js* with the exception that the spelling of the words is not displayed. The regular *learn_words.js* also allows participants to hear the word first, before asking them to repeat it into their microphone. On the other hand, *learn_words_test.js* simply asks participants to speak the words from memory and are not given feedback.

More details of these reading-type sections (requiring the use of the microphone for recording) are found under the "Reading" section below.

6.5.3 Letter Learning

The letter learning section is handled by *learn_letters.js* on the client-side and queries *learn_letters.php* on the server-side for each letter's image reference. The client-side code simply operates a timer through **setTimeout** that reveals the letter's image and sound only for a brief amount of time each, before querying for the next one after the 'next' button is pressed. The interface between the JavaScript and PHP code also passes a message variable at the middle and end of the trial sequence to instruct the user.

6.5.4 Reading

The reading-type screens use *reading.js* with Recorderjs to capture audio from the user's microphone, export that into a WAV which is then passed to the server in the form of a blob though AJAX. There are two setup functions to be ran before recording can take place, namely

initializePage() and **startUserMedia(…)**. If these do not detect a microphone or cannot initialise the media recording facilities, the user is informed with a message on the screen.

On each trial, the *get_random_word.php* is queried by the reading client-side code to deliver the next word (trial) in the form of an array of numbers (corresponding to artificial letter identifiers), as well as its audio recording and associated picture. These are used in the presentation of the trial or feedback after it. The array of letter identifiers is used to select each letter individually and present it on the screen as a concatenation of images.

At the end of the trial, after participant recording is complete, the *save_file_to_server.php* is passed a blob of the recording. It then proceeds to create that as a file on the server's recordings folder and creating a record in the reading_task database. Finally, it increments the trial counts and returns the study state name (to be used to determine if in testing and hence not give feedback to the participant).

Additionally for the user interface, two HTML5 canvas elements are used to animate the microphone volume levels and 2.5 seconds timer for the participant.

6.5.5 Writing

Writing-type screens are controlled by *writing.js*. It first generates the on-screen keyboard with the help of *generate_keyboard.php* on the server, which randomises the location of the buttons. The buttons themselves are images that have onclick events which trigger the various interaction functions. There are several functions used to input individual letters or strings to the on-screen display as a concatenation of images (where each image is 1 letter).

New trials are gathered from *get_random_file.php*, which returns a reference to the audio file of the word and its associated picture. The participant input is captured in an array of numbers (representing letter identifiers) and is passed to *post_word_input.php*. It translates the array into their sound values (using the ***alphabet_key***) and finds the edit distance between those and the target for that trial. Finally, it creates a record of the trial in the database, increments trial counts and returns the name of the current state (if a testing state, feedback is not given) and the target string (for feedback purposes).

6.5.6 Testing

Testing screens use the exact same code as the training screens, namely *reading.js* and *writing.js*. However, *initialise_section.php* generates the full set of words for them to use, and the client-side code excludes feedback when it receives that the current section is a test.

7 RESOURCES

7.1 USED AUDIO:

Two sound effects in this experiment were used under the Creative Commons license: "*correct*" by "*ertfelda*" at https://www.freesound.org/people/ertfelda/sounds/243701/ and "*Game Sound Wrong.wav*" by "*Bertrof*" at https://www.freesound.org/people/Bertrof/sounds/131657/

7.2 USED IMAGES:

Images for the experiment were taken from: *Revisiting Snodgrass and Vanderwart's Object Pictorial Set: The Role of Surface Detail in Basic-Level Object Recognition*. Bruno Rossion and Gilles Pourtois. *Perception Vol 33*, Issue 2, pp. 217 – 236. First Published February 1, 2004. https://doi.org/10.1068/p5117

7.3 USED CODE:

Audio recording is a basic implementation of the Recorderjs, saving recordings to the server.

Recorderjs: https://github.com/mattdiamond/Recorderjs

Saving audio files to server: http://stackoverflow.com/questions/19015555/pass-blob-through-ajax-to-generate-a-file

Normalize.css: https://github.com/necolas/normalize.css

jQuery 3.1.1: https://jquery.com/

JavaScript Modernizr: https://modernizr.com/

Using PHP with MySQL - the right way (Lionite): https://www.binpress.com/tutorial/using-php-with-mysql-the-right-way/17