FLXLab 2.2

An experiment generator for the free world

Tutorial

Contents

1 The Basic Demo Experiments

This section of the user guide introduces the basic principles of designing an experiment in FLXLab, using the set of demo experiments provided with the program as examples.

FLXLab experiments are controlled by a *script*. A script is a text file containing commands that set up the experiment. Once FLXLab is installed, you should be able to execute each of the demo scripts contained in the `demos` directory. The sections below review the script files for each of these experiments, and explain how the script files work. They are organized starting with the simplest demo and progressing to increasingly more complicated experiments.

1.1 Reaction Time I

The Reaction Time I demo experiment functions as follows. Each trial begins with a one-second pause, after which a small rectangle appears in the middle of the screen. It remains on the screen until a key on the keyboard is pressed, at which point it disappears. This cycle repeats five times. The text of the script file is reproduced in Table 1. Note that the numbers on the left are for reference only, and are not part of the script.

Table 1: Script for Reaction Time I

```
1   DelayEvent pause 1000
2   RectangleObject rectangle
3   DisplayEvent stimulus
4   AddObject rectangle
5   WaitEvent wait_for_key "until key any"
6   TrialEvent trial
7   AddEvent pause
8   AddEvent stimulus
9   AddEvent wait_for_key
10  BlockEvent reaction_time "repeat 5"
11  AddEvent trial
12  Start reaction_time
```

1.1.1 Commands

Each line of the script contains one FLXLab command. The name of the command is always the first word on the line (e.g., `DelayEvent`, `DisplayEvent`, etc.). The remaining words on the line are *arguments* to the command. Arguments are additional information that affects how the command functions.

Commands may have anywhere from zero to four arguments; in some cases one or more arguments may be optional. If an argument contains more than one word, it must be surrounded by quotes (either single or double) to tell FLXLab to treat it as a single unit.

1.1.2 Basic Events

The building blocks of an experiment in FLXLab are *events*. "Simple" events are those that control stimulus presentation and data collection. In this experiment, there are three such events. One of them controls the pause at the beginning of each trial. This event is created with the `DelayEvent` command, which takes two arguments (line 1). The first argument specifies a name that will be used to refer to this event (here, `pause`). The second specifies the duration of the delay in milliseconds (here, 1000, or one second).

A second event controls the presentation of the rectangle. The first thing we need to do is to create a *display object* which will be shown on the screen. This is accomplished on line 2 with the `RectangleObject` command, which creates a rectangle named `rectangle`. The actual event is created with the `DisplayEvent` command, which just takes one argument, the name of the event. By itself, this event would display nothing on the screen. To tell FLXLab to display the rectangle, we need to associate it with the `stimulus` event. This is accomplished with the `AddObject` command on line 4.

The third simple event is created with the `WaitEvent` command on line 5. The first argument is the name of the event; the second specifies what we're waiting for, in this case for the user to press a key on the keyboard.

At the next level up, events can be combined into a *trial*. This kind of event can be created with the `TrialEvent` command. The argument to this command specifies the name of the event. Lines 7, 8 and 9 then use the `AddEvent` command to add the events `pause`, `stimulus`, and `wait_for_key` to `trial`.

One more level up, trials can be combined to form *blocks*. This kind of event can be created with the `BlockEvent` command (line 10). Again, the first argument to this command is the name of the event. The second argument (`‘‘repeat 5’’`) indicates that the block will cycle through five times. We then add `trial` to our block event `reaction_time`, again using the `AddEvent` command.

Note that there is an even higher level of event called an *experiment*; we will encounter this in some of the more complicated demo experiments described below.

Finally, we need to tell FLXLab to start the event `reaction_time`; this is done with the `Start` command on line 12.

This script contains many of the major elements of an experiment, but it is not a very good experiment. The timing of the rectangle is predictable, and the script does not actually record the reaction times. These and other issues are addressed in the next demo experiment.

1.2 Reaction Time II

The second version of the reaction time experiment involves a somewhat more sophisticated procedure. Each trial begins with a one-second pause, as before. This is followed by a fixation cross in the middle of the screen. After a delay of variable duration (either 800, 1000, 1200, 1400 or 1600 ms), a small filled rectangle appears in one of eight positions around the screen. Again, the task is to press a key on the keyboard as soon as the rectangle appears. After the experiment, there will be a file called `data.txt` in the `reaction_time_II` directory that contains the reaction times for each trial, measured from the onset of the rectangle. The text of the script file is reproduced in Table 2.

Table 2: Script for Reaction Time II

```
1   # set up the stimulus list
2   StimulusList stimulus_list stimulus_list.txt
3   LabelListColumn 1 stimulus_onset_time
4   LabelListColumn 2 stimulus_position
5
6   # define the positions at which the rectangle can appear
7   DefinePosition 1,1 25% 25%
8   DefinePosition 1,2 50% 25%
9   DefinePosition 1,3 75% 25%
10  DefinePosition 2,1 25% 50%
11  DefinePosition 2,3 75% 50%
12  DefinePosition 3,1 25% 75%
13  DefinePosition 3,2 50% 75%
14  DefinePosition 3,3 75% 75%
15
16  # set up the delay between trials
17  DelayEvent intertrial_delay 1000
18
19  # create an event to display a fixation cross
20  TextEvent fixation_cross +
21
22  # the delay between the fixation cross and the
23  # rectangle; the duration of this delay is variable
24  # and is taken from the stimulus file
25  DelayEvent stimulus_onset_delay $stimulus_onset_time
26
27  # create an event to display the rectangle; the
28  # position of the rectangle is variable and is
29  # taken from the stimulus file
30  RectangleObject rectangle
31  Filled
32  DisplayEvent stimulus
```

```
33  ResetDataTime
34  AddObject rectangle $stimulus_position
35
36  WaitEvent wait_for_key "until key any"
37
38  # create an event to record the reaction time
39  DataEvent record_rt
40  DataColumn $stimulus_onset_time
41  DataColumn $stimulus_position
42  DataColumn $time
43
44  # add all the events together to make a trial
45  TrialEvent trial
46  AddEvent intertrial_delay
47  AddEvent fixation_cross
48  AddEvent stimulus_onset_delay
49  AddEvent stimulus
50  AddEvent wait_for_key
51  AddEvent record_rt
52
53  # continue to do trials until we get to the end
54  # of the stimulus list
55  BlockEvent reaction_time "until list end"
56  AddEvent trial
57
58  # start the experiment
59  Start reaction_time
```

1.2.1 Aborting an experiment

This experiment is considerably longer than Reaction Time I, and you may be wondering if there is a way to stop the experiment in the middle. The answer is yes: All you have to do is type Control-Q. Note that this won't work if you have a dialog box open on the screen (as occurs in some of the later demo experiments); you'll have to close it first.

1.2.2 Blank lines and comments

One of the major differences between this script and the reaction time I script is that this script contains numerous blank lines and lines beginning with a `#`. Lines beginning with `#` are called *comments*, and allow you to make notes, explain the purpose of particular commands, or anything else you'd like. Both blank lines and comments are ignored by FLXLab.

1.2.3 Using a stimulus list

In this experiment, both the timing of the stimulus and its position vary from trial to trial. This information is stored in a separate text file, in this case a file called `stimulus_list.txt`. This file contains one line for each trial of the experiment. There are two columns in the file. The first specifies the time from the onset of the fixation cross to the onset of the rectangle. The second specifies the position of the rectangle.

Lines 2-4 in the script configure FLXLab to use this stimulus file. The `StimulusList` command indicates that we will be using a stimulus list called `stimulus_list` read from the file `stimulus_list.txt`. The two `LabelListColumn` commands define labels we can use to refer to values in the first and second columns of the file.

1.2.4 Positions

By default, display objects are presented at the center of the screen. However, it is possible to present them at some other location by the use of *positions*. There are nine pre-defined positions: `topleft`, `top`, `topright`, `left`, `center`, `right`, `bottomleft`, `bottom`, and `bottomright`. Additional positions can be defined using the `DefinePosition` command. The first argument of this command is the name to be given to this position (e.g., the position defined on line 7 is given the name `1,1`). The next two arguments indicate the horizontal and vertical coordinates of the position. These may be specified either as a percentage of the screen width, as done here, or as absolute pixel values. The use of percentages is generally preferred, as it ensures that stimuli appear in the same relative locations even if you change the monitor you are using.

To specify that a particular display object should appear at a particular position, we use a second, optional argument to the `AddObject` command, as is done on line 34. Note that `$stimulus_position` is a label for the second column in the stimulus file (see section 1.2.7 for an explanation of the `$`). If you look in the stimulus file, you will see that this column contains entries like `2,3`, `3,1`, `3,2`, etc. Thus, to determine the position of the rectangle on any given trial, FLXLab looks at the appropriate entry from the stimulus file.

1.2.5 Displaying text

The fixation cross in this experiment is implemented as a plus sign. This requires a new kind of event, created using the `TextEvent` command (see line 20). As with the `DisplayEvent` command, the first argument specifies the name for the event (here, `fixation_cross`. There is also a second argument that specifies the text to be displayed. Remember that if the text is more than one word, you must surround it with quotes. Here this text is provided explicitly in the script file; it is also possible to place the text in the stimulus file so that it changes from

trial to trial. This is done in the same way as for positions or times (discussed next).

1.2.6 Delays of variable duration

Line 25 of the script defines a delay event for the time between the onset of the fixation cross and the onset of the rectangle. To make this delay variable, we place the duration of the delay in the stimulus file, and use the label for this column (`$stimulus_onset_time`) in place of the literal time (compare the second argument to the `DelayEvent` commands on lines 17 and 25). See section 1.2.7 for an explanation of the `$`.

1.2.7 Variables and literal values

Note that the column label on line 25 is preceded by a `$`. This signals to FLXLab that we are providing the name of a variable that contains the appropriate value, rather than providing the value directly. Without this convention, strings like `1000` (line 17) or `+` (line 20) would be ambiguous: Should they be taken as the names of a variables, or as the actual values themselves? Of course, `1000` and `+` are rather odd names for a variable, but there are also variables with names like `time` (line 42), in which case distinguishing between a variable and a literal value is not so easy.

The `$` should be used with any variables created by the `LabelListColumn` command; in this script, these are `stimulus_onset_time` and `stimulus_position`. It must also be used with "built-in" variables such as `time` (see line 42).

1.2.8 Changing attributes of shapes

In the Reaction Time I experiment, the stimulus rectangle was a small, black outline. It is possible to change various attributes of shape objects, including their size, their color, the weight of the line used to draw them, and whether or not they are filled. This last possibility is utilized on line 31, where the `Filled` command specifies to use a filled rectangle rather than the default outline. Other attributes can be changed with the `Size`, `Color`, `BoxColor`, and `LineWidth` commands; more details are provided in the description of later scripts and in the reference section. Note that in addition to rectangles, you can create ellipses with the `EllipseObject` command.

1.2.9 Defining a reference point for measuring reaction times

In recording reaction times in our experiment, we want those times measured from the onset of the rectangle stimulus. By default, FLXLab measures reaction times from the beginning of the trial. You can indicate that you want reaction

times measured relative to a specific event using the `ResetDataTime` command, as is done on line 33.

1.2.10 Recording data to a file

For our experiment to be useful, we have to be able to record the reaction times to a data file for analysis. One you have run the experiment, you will find a file called `data.txt` in the `reaction_time_II` directory. This file contains three columns. The first two are the same as the stimulus file. The last is the reaction time for each trial.

This information is recorded to the file by means of the `DataEvent` command (line 39). This command defines an event that will write one line to the data file. The contents of that line are specified using one or more `DataColumn` commands. Lines 40 and 41 indicate that the presentation time and position of the rectangle stimulus should be written to the file; recall that `stimulus_onset_time` and `stimulus_position` were used to label these values in the stimulus file. Line 42 uses the special value `time`, which always refers to the current elapsed time (relative to the beginning of the trial or an event identified with the `ResetDataTime` command).

1.2.11 Determining the number of trials from the stimulus file

When a stimulus file is used, it is convenient to have the experiment simply run until all the lines in the stimulus file have been used. This is achieved in the current experiment by using `‘‘until list end’’` as the second argument to the `BlockEvent` command on line 55. This simply indicates that FLXLab should keep cycling through the block until the end of the stimulus list is reached.

1.3 Lexical Decision

This experiment introduces a new procedure: Lexical decision. On each trial, a string of letters if presented on the screen, and the participant has to decide whether or not it is an actual word of English, and press one of two keys accordingly. In addition, two more improvements are made. At the beginning of the experiment, a dialog box is presented for the experimenter to enter in the subject ID, which is used to create distinct data files for each subject. Second, instructions are presented on the screen prior to the beginning of the actual lexical decision trials. The text of the script file is reproduced in Table 3.

Table 3: Script for Lexical Decision

```
1   # Get the subject id from the user
2   EditDialog subject_id "Enter subject id:"
3   # Use the subject id to generate the name of the data file
```

```
4   JoinStrings data_file "data_files $path_separator $subject_id .txt"
5   # Set the name of the data file
6   UseDataFile $data_file
7
8   # Create labels for the three columns in the stimulus file
9   StimulusList stimulus_list stimulus_list.txt
10  LabelListColumn 1 item_number
11  LabelListColumn 2 stimulus_item
12  LabelListColumn 3 item_type
13
14  # Use 36 point URWNimbus throughout the experiment
15  FontFace URWNimbus
16  FontSize 36
17
18  # This creates an event to display the instructions
19  LoadTextFromFile instructions_text instructions.txt
20  TextBoxEvent instructions $instructions_text
21
22  WaitEvent wait_for_key "until key any"
23
24  TextEvent warning_signal ****
25
26  DelayEvent pause 500
27
28  ClearScreenEvent clear_screen
29
30  TextEvent stimulus $stimulus_item
31  # Reaction times will be relative to the onset of the stimulus
32  ResetDataTime
33
34  DataEvent record_response
35  DataColumn $item_number
36  DataColumn $stimulus_item
37  DataColumn $item_type
38  DataColumn $key
39  DataColumn $time
40
41  TrialEvent trial
42  AddEvent warning_signal
43  AddEvent pause
44  AddEvent clear_screen
45  AddEvent pause
46  AddEvent stimulus
47  AddEvent wait_for_key
48  AddEvent record_response
49
```

```
50  BlockEvent mainblock "until list end"
51  AddEvent trial
52
53  ExperimentEvent lexical_decision
54  AddEvent clear_screen
55  AddEvent instructions
56  AddEvent wait_for_key
57  AddEvent mainblock
58
59  Start lexical_decision
```

1.3.1 Getting input from the user

It is often convenient to collect certain information at the beginning of the experiment, such as what the subject ID should be or which list to use. This can be achieved with the `EditDialog` command, which displays a dialog box where the user can enter in information. The first argument to this command is a name which you can use to refer to whatever the user entered in. The second argument is a prompt which will be displayed in the dialog box. See line 2 of the script for an example of how to use this command to obtain a subject ID.

1.3.2 Combining strings of text

Now that we have the subject ID, we would like to use it to generate the name for a file to store that subject's data. This can be done with the `JoinStrings` command. The first argument to this command is a name which can used to refer to the combined strings. The second argument is a series of strings (surrounded by quotes) to combine. To create the name of the data file, we take the name of the data directory (`data_files`), add the special variable `path_separator` (see section 1.3.3), the subject ID, and the suffix `.txt` to indicate that it is a text file (see line 4 of the script).

1.3.3 Generating paths to files

The special term `path_separator` represents whatever character separates directory names in a path on your system - on Unix/Linux it is `/`, in Windows it is \. By using `path_separator` rather than the actual character used on your system, you can ensure that your scripts will work correctly on any operating system that FLXLab runs on. Taking the use of the `JoinStrings` command on line 4 as an example, if the subject ID was `subject01`, and FLXLab is being run on Windows, this would yield `data_files\subject01.txt`. We could then refer to this combined string with the name `data_file`.

1.3.4 Specifying the data file

By default, FLXLab will record data to the file `data.txt`. Each time you run the experiment, this file will be replaced. To specify a different data file, you can use the `UseDataFile` command. This command takes one argument, the name of the file (see line 6). This name will be the value of `data_file`, e.g., in Windows `data_files\subject01.txt`.

1.3.5 More advanced commands for text: Reading text from files, text boxes, and text events

In the Reaction Time II demo, we learned how to display text on the screen. This script introduces several new commands for working with text. The `LoadTextFromFile` command takes two arguments: A name to be used to refer to the text, and a file to read the text from. It is used on line 19 of the script to load the text of the instructions. The name `instructions_text` can then be used any place a text argument is expected (e.g., with a `TextEvent` command or the `TextBoxEvent` command, discussed below).

If a large amount of text is in the file, it may not all fit on one line of the screen. With a text event, the text will simply be cut off at the edge of the screen. If you want the text to wrap around instead, you can use the `TextBoxEvent` command (see line 20).

1.3.6 Changing the font

Like shape objects, text objects have a color attribute which may be changed with the `Color` command. However, they also have several attributes of their own. Font face and size may be specified with the `FontFace` and `FontSize` commands (see lines 15 and 16). For the font face, in general you would just use the name of the font as it appears in the font menu of a word processor, with the spaces removed. Thus, Franklin Gothic Medium is `FranklinGothicMedium`. To specify bold or italic, append this to the name of the font, e.g., `BookAntiquaBold`, `FranklinGothicMediumItalic`, `GautamiBoldItalic`. Note that `Bold` should come before `Italic` in this case.

FLXLab is distributed with three font faces: URW Bookman, URW Nimbus (similar to Courier), and URW Palladio (similar to Palatino). The default font is URW Palladio. However, FLXLab should be able to utilize any PostScript or TrueType font on your system. To access these fonts, you can use the `AddFontDirectory` command. In Windows, placing the following line in the configuration file for the text module should allow you to access many (though not all) of the fonts on the system:

```
AddFontDirectory \windows\Fonts
```

If you downloaded and installed the Windows version of the program, this has probably already been done for you. In Linux, things are a bit trickier, because there generally isn't a single directory where most fonts are stored. You may have to look around a bit to see what is available and where it is located.

Once you have chosen a font face, the font size is specified with the `FontSize` command, e.g.,

```
FontSize 48
```

Because FLXLab uses PostScript and TrueType fonts, it is generally possible to have any font size you like, though some will look better than others. You can also display any character you like, including Unicode characters. In order for FLXLab to recognize Unicode characters in text files (e.g., script files or stimulus lists), the files should be saved in the UTF-8 format.

Note that setting the font before any text objects are defined, as is done here, changes the default font, which will then be used for all text objects unless otherwise specified.

1.3.7 Clearing the screen

FLXLab automatically clears the screen at the beginning of each trial. However, if you want to clear the screen at some other point, you can do this with the `ClearScreenEvent` command (see line 28). This command simply creates a special kind of event that clears the screen.

1.3.8 Recording which key was pressed

Since the participant can press one of two keys in this experiment, we would like to know which key they pressed. We can do this with a special value similar to `time` (as in Reaction Time II and line 39 of the current experiment), called `key` (see line 38). This holds the key that last triggered a key condition (as in ``when key any''.

1.3.9 Experiment events

Because we want to present instructions prior to the main experiment, we need a level of events above that of a block. This is achieved with the `ExperimentEvent` command (see line 53). Events may be added to an experiment in the same way they are added to a block. Thus, our experiment begins by clearing the screen, after which the instructions are displayed. FLXLab will then wait until a key is pressed, after which it begins the actual block of trials.

1.4 Masked Priming

This experiment is a variant on the lexical decision demo. On each trial, the participant sees a mask for 500 ms, followed by a prime word for 35 ms, followed by the target. The target stays on the screen until a response has been made.

Table 4: Script for Masked Priming

```
# Get the subject id from the user
EditDialog subject_id "Enter subject id:"
# Use the subject id to generate the name of the data file
JoinStrings data_file "data_files $path_separator $subject_id .txt"
# Set the name of the data file
UseDataFile $data_file

# Create labels for the three columns in the stimulus file
StimulusList stimulus_list stimulus_list.txt
LabelListColumn 1 item_number
LabelListColumn 2 prime_item
LabelListColumn 3 target_item
LabelListColumn 4 item_condition
LabelListColumn 5 item_type

FontFace URWNimbus
FontSize 36

UseMicroseconds

ClearScreenEvent clear_screen

# This creates an event to display the instructions
LoadTextFromFile instructions_text instructions.txt
TextBoxEvent instructions $instructions_text

WaitEvent wait_for_key "until key any"

TextEvent mask #####
WaitForRefresh
ResetEventTime

TextObject prime_text $prime_item
DisplayEvent prime
Color white
AddObject prime_text

TextObject target_text $target_item
```

```
39  DisplayEvent target
40  Color white
41  AddObject target_text
42  # Reaction times will be relative to the onset of the target
43  ResetDataTime
44
45  DataEvent record_response
46  DataColumn $item_number
47  DataColumn $target_item
48  DataColumn $item_condition
49  DataColumn $item_type
50  DataColumn $key
51  DataColumn $time
52
53  TrialEvent trial "until event record_response"
54  AddEvent mask
55  AddEvent prime "when time 500000"
56  AddEvent target "when time 533333"
57  AddEvent record_response "when key any"
58
59  BlockEvent mainblock "until list end"
60  AddEvent trial
61
62  ExperimentEvent lexical_decision
63  AddEvent clear_screen
64  AddEvent instructions
65  AddEvent wait_for_key
66  AddEvent mainblock
67
68  Start lexical_decision
69
```

1.4.1 Synchronizing events with the display refresh

For paradigms like masked priming, where it is import to precisely control the amount of time the prime is visible, it is useful to be able to synchronize stimulus presentation with refresh of the display. Typical computer displays are refreshed somewhere between 60 and 100 times a second. Using the 60 Hz rate as an example, this means a refresh occurs every 16.67 ms. Regardless of when you tell FLXLab to present a stimulus and when you tell it to replace it with something else, these actions will only take place when the next refresh occurs. Thus, if your refresh rate is 60 Hz, your prime will actually be visible for a multiple of this number, i.e., 16.67 ms, 33.34 ms, etc.

If you tell FLXLab to display the prime for a multiple of the refresh rate (or very close to it), then it will be visible for that amount of time, whether or

not you sychronize with the start of the refresh cycle. However, there is some possibility that "tearing" will occur, i.e., that the top half and the bottom half of the prime will be out of sync with each other. You can avoid this by using the `WaitForRefresh` command to tell a display event to wait for the beginning of the refresh cycle before drawing to the screen (see line 30). Note that at this time, this command only works consistently in the Windows version of FLXLab.

Once you've synchronized with the display refresh once, you can usually keep your stimuli in sync with the refresh simply by ensuring the pauses between them are multiples of the refresh rate. In this experiment, we synchronize with the display once (for the mask), and then just precisely control the time between presentation of the mask and presentation of the prime and target. Because the refresh cycle often is not an even number of milliseconds, it's usually a good idea to specify times in microseconds (millionths of a second) in such experiment; see lines 55 and 56. To make FLXLab interpret times as microseconds, use the `UseMicroseconds` command. We also want to measure our times relative to the onset of the mask, because the amount of time we have to wait for the beginning of the refresh cycle can vary from trial to trial; to do this, we use the `ResetEventTime` command with the DisplayEvent that shows the mask (line 31). This basically means that when the `mask` event executes, it resets the timer that measures how long the trial has been running for, and which is used on lines 55 and 56 to determine when the prime and target should be displayed.

One final comment: Inserting a DelayEvent between stimuli is not a good way to ensure a precise interval between them. This is because drawing a display to the screen takes time - usually several milliseconds. This incidental delay is added onto the intentional delay created by your DelayEvent. So if you put a DelayEvent of 50 ms between two stimuli, the actual interval between them might be more like 54 ms. Specifying times relative to a fixed point in the trial, as done here, is a better approach.

1.5 Temporal Order Judgment

This script implements an auditory temporal order judgment task. On each trial, the participant hears two very brief tones that are separated by a variable interval (in the case, anywhere from 0 to 50 ms in 5 ms increments). The tones differ slightly in pitch. The task is to decide whether the low tone preceded the high tone or vice versa, and press one of two keys accordingly. The text of the script file is reproduced in Table 5.

Table 5: Script for Temporal Order Judgment

```
1   # Get the subject ID and specify the data file
2   EditDialog subject_id "Please enter the subject id:"
3   JoinStrings data_file "data_files $path_separator $subject_id .txt"
4   UseDataFile $data_file
5
```

```
6   # Define a block to display the instructions
7
8   ClearScreenEvent clear_screen
9
10  LoadTextFromFile instructions_text instructions.txt
11  TextBoxEvent instructions $instructions_text
12
13  WaitEvent wait_for_key "until key any"
14
15  BlockEvent do_instructions
16  AddEvent clear_screen
17  AddEvent instructions
18  AddEvent wait_for_key
19
20  # Define a trial
21
22  StimulusList stimulus_list stimulus_list.txt
23  LabelListColumn 1 order
24  LabelListColumn 2 interval
25
26  DelayEvent intertrial_interval 1000
27
28  JoinStrings sound_file "sound_files $path_separator $order $interval .wav"
29  PlaySoundEvent tones $sound_file
30
31  DelayEvent pause 500
32
33  TextEvent prompt "lo-hi hi-lo"
34
35  DataEvent record_response
36  DataColumn $order
37  DataColumn $interval
38  DataColumn $key
39
40  TrialEvent trial
41  AddEvent intertrial_interval
42  AddEvent tones
43  AddEvent pause
44  AddEvent prompt
45  AddEvent wait_for_key
46  AddEvent record_response
47
48  # Define a block of trials
49
50  BlockEvent mainblock "until list end"
51  AddEvent trial
```

```
52
53  # Define a block for displaying a thank you message
54
55  TextEvent thank_you "Thank you for your help!  (press any key to exit)"
56
57  BlockEvent do_thank_you
58  AddEvent clear_screen
59  AddEvent thank_you
60  AddEvent wait_for_key
61
62  # Combine the three blocks into an experiment
63
64  ExperimentEvent temporal_order_judgment
65  AddEvent do_instructions
66  AddEvent mainblock
67  AddEvent do_thank_you
68
69  Start temporal_order_judgment
```

1.5.1 A more complex experiment structure

Although the term *block* is conventially used to refer to a block of trials, in FLXLab a block event is simply a way to combine a number of events into one unit. The current script creates three block events which are combined into one experiment event. The first block displays the instructions, the second block runs the trials, and the third block displays a thank you message.

1.5.2 Playing sounds

FLXLab can play sounds that are stored in a WAV file (currently there is no support for other file formats). This is achieved with the `PlaySoundEvent` command (see line 29). This command takes two arguments: The first is the name for the event, the second is the file to load the sound from. In this experiment, there are 22 different sound files, corresponding to the 22 conditions of the experiment (11 different intervals between the tones x 2 tone orders). These sound files are contained in the `sound_files` directory. The name of the appropriate file is constructed out of the entries in the stimulus file by use of the `JoinStrings` command, as seen on line 28.

Note that it would also be possible to have only two sound files, one low tone and one high tone, and use a delay event to create the interval between them. However, there are often slight delays associated with beginning to play a sound. One reason for this is that sound cards generally don't start playing a sound immediately, but wait until they've received a certain amount of sound data (this helps avoid so-called *underruns*). There are also sometimes delays associated

with activating the sound hardware. Thus, the method used in the current script may provide more precise control over the interval between the tones.

1.6 Picture Naming

This script implements a basic picture naming task. On each trial, the participant sees a warning signal, followed by a picture. The task is to say what the picture is. As soon as the participant starts speaking, a voice key is triggered, removing the picture from the screen. The spoken response is recorded to an audio file, and the latency to start speaking is recorded to the data file. The participant is given 2 seconds to finish their response, and then the next trial commences. Note that you will need to connect a microphone to your computer and ensure that recording is functioning properly in order for this experiment to work. The text of the script file is reproduced in Table 6.

Table 6: Script for Picture Naming

```
1   EditDialog subject_id "Please enter the subject id:"
2   JoinStrings data_file "data_files $path_separator $subject_id .txt"
3   UseDataFile $data_file
4
5   StimulusList stimulus_list stimulus_list.txt
6   LabelListColumn 1 picture_name
7
8   ClearScreenEvent clear_screen
9
10  LoadTextFromFile instructions_text instructions.txt
11  TextBoxEvent instructions $instructions_text
12
13  WaitEvent wait_for_key "until key any"
14
15  # Use a large font for the warning signal to make the cross more visible
16  TextEvent warning_signal +
17  FontSize 72
18
19  DelayEvent pause 500
20
21  # Set up the file that spoken responses will be recorded to
22  JoinStrings sound_file "sound_files $path_separator $subject_id - $picture_name .wav"
23  # An event to record the spoken responses
24  RecordSoundEvent toggle_recording $sound_file
25
26  # Create a string with the file name that images will be loaded from
27  JoinStrings picture_file "images $path_separator $picture_name .bmp"
28  # An event to present the images
29  ImageEvent stimulus $picture_file
```

```
30  ResetDataTime
31
32  WaitEvent wait_for_voice "until voice_key"
33
34  # The data file records the name of the picture and the latency for the
35  # voice key to trigger
36  DataEvent rt_data
37  DataColumn $picture_name
38  DataColumn $time
39
40  DelayEvent record_response 2000
41
42  TrialEvent trial
43  AddEvent warning_signal
44  AddEvent pause
45  AddEvent clear_screen
46  AddEvent pause
47  # Turn audio recording on here
48  AddEvent toggle_recording
49  AddEvent stimulus
50  AddEvent wait_for_voice
51  AddEvent rt_data
52  AddEvent clear_screen
53  AddEvent record_response
54  # Turn audio recording off here
55  AddEvent toggle_recording
56
57  BlockEvent mainblock "until list end"
58  AddEvent trial
59
60  ExperimentEvent picture_naming
61  AddEvent clear_screen
62  AddEvent instructions
63  AddEvent wait_for_key
64  AddEvent mainblock
65
66  Start picture_naming
```

1.6.1 Presenting images from picture files

FLXLab can display images stored in several formats, including BMP (Windows bitmaps), JPEG, LBM, PCX, and TGA. This is achieved with the `ImageEvent` command (see line 29), which is roughly analogous to the `PlaySoundEvent` command. This command takes two arguments. The first is the name for the event, the second is the file to load the image from. The name of the file is

constructed on line 27 using the `JoinStrings` command.

1.6.2 Recording sound

FLXLab can record sound from a microphone or other external source, and write the sound data to a WAV file. Sound recording is controlled by the `RecordSoundEvent` command (see line 24). This command takes two arguments. The first is the name for the event, the second is a file to write the sound data to. Sound recording events are a bit unusual in that they work like a switch. The first time the event occurs, it turns sound recording on (line 48). The second time it occurs, it turns recording off again, and writes the sound data to the file (line 55). In this script, the sequence is as follows: Turn recording on, display the stimulus, wait for the voice key to trigger, write the speech initiation latency to the data file, clear the screen, wait two seconds, then turn recording off.

1.6.3 Using the voice key

FLXLab provides a software voice key. To use the voice key, you must first start recording sound. If you don't actually want to record the sound to a file, you can use a variant of the `RecordSoundEvent` command called `VoiceKeyEvent`, which does not require a file name. The two types of events work exactly the same except a voice key event discards the sound data at the end of the trial.

You can check whether the voice key has triggered in a similar way as you can check whether a key has been pressed, using phrases like `"until voice_key"` and `"when voice_key"`. Note that the underscore is necessary here, as `voice_key` is one unit. In the current script, we define an event `wait_for_voice` that simply pauses until the voice key has been triggered (line 32).

The voice key is auto-calibrating, meaning that it adjusts its sensitivity on each trial based on the level of background noise. Thus, the voice key should work correctly regardless of the sound input level or the background noise level, provided there is a sufficient signal-to-noise ratio.

2 Additional Demo Experiments

This section briefly describes additional demo experiments included with FLXLab to illustrate other uses of the program.

2.1 Reaction Time III

The third variant on the reaction time demo illustrates how to randomly generate stimulus items, rather than specifying them via a stimulus list. As before, each trial begins with a one-second pause, followed by a fixation cross in the

middle of the screen. However, rather than specifying the delay before the appearance of the stimulus in the stimulus file, a random integer between 800 and 1600 is used. Similarly, rather than specifying the location of the stimulus in the stimulus file, random values are used to determined the x and y coordinates. In addition, both the color and the dimensions of the rectangle also randomly vary from trial to trial. As before, the task is to press a key on the keyboard as soon as the rectangle appears.

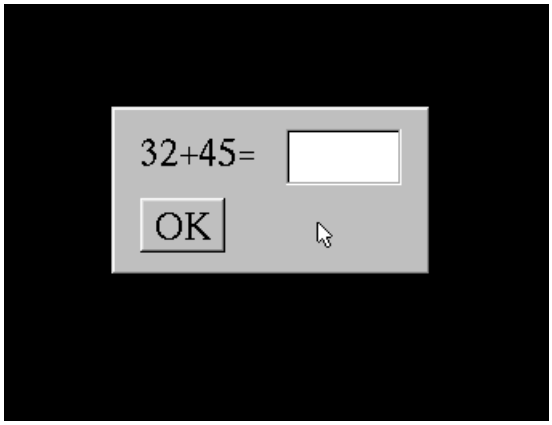
2.2 An Arithmetic Experiment

In FLXLab, dialog boxes would typically be used to collect necessary information at the beginning of an experiment, such as a subject ID, the condition, the list to use, etc. However, dialogs can also be used to collect responses during the experiment itself. This is particularly useful for collecting responses consisting of more than a single key press. Such responses could be words, sentences, or - as in this experiment - numbers.

On each trial of the demo, a simple arithmetic problem is presented on the screen, either addition, subtraction, multiplication, or division. For example, on the first trial of the demo, the participant sees `32+45=`. The task is to type in the correct answer to this problem, then click OK (or press the enter key). Then another problem is shown. The data file records the problem number, the problem, the correct answer, and the answer the participant provides.

A screen shot of the experiment is shown in Figure 1.

Figure 1: Screen shot from the Arithmetic Demo



2.3 Randomized List Demo

This script illustrates how to build a stimulus list out of several smaller lists, as well as how to randomize the items. The experiment consists of 15 trials: 5 practice trials, 5 filler trials, and 5 experimental trials. The practice trials are presented first, followed by the filler and experimental trials, randomly intermixed. The stimuli to be used for the experimental trials vary depending on the condition, which can be A, B, C or D. Prior to the start of the experiment, a dialog box is presented into which the condition is entered.

On each trial, some text is presented in the middle of the screen, and the script waits for a key press before advancing to the next trial. The text indicates the type and number of the item, e.g., `practice3`, `fill2`, `exp4C`, where in the last case the `C` indicates that the item comes from condition C. If you run the script repeatedly, you can see that the order of the filler and experimental items is different each time.

2.4 Animation Demo

This script illustrates how you can make objects move on the screen during a trial. The demo begins with a black square in the top left corner of the screen. The square moves slowly down and to the right. You can adjust the direction and speed of movement with the keys 'l', 'r', 'u' and 'd' (for left, right, up, and down, respectively). Press 'q' to exit the experiment.

Note that the position of the square is only updated with each screen refresh. If you have a low refresh rate (e.g., 60 Hz), the quality of the animation will be poor when the square is moving quickly.

2.5 Lexical Decision II Demo

This experiment is a more elaborate version of the Lexical Decision Demo. The structure of the experiment is exactly the same as in that experiment, except the subjects are presented with the message "Sorry, wrong answer" whenever they make a mistake, and the experiment terminates after 5 mistakes have been made.

2.6 Stroop Effect Demo

This experiment illustrates the classic Stroop effect. On each trial, the subject sees a word presented on the screen. The set of words is 'blue,' 'green,' 'red,' 'orange,' 'yellow,' and 'purple,' and the text color of the word is also drawn from the same set. On half the trials, the word and the color of the text are the same, e.g., the word 'red' with with the color of the text being red. On the other half of the trials, the word and the color of the text are different, e.g.,

the word 'red' with the color of the text being green. The task is to say the color of the text out loud as quickly as possible, while ignoring what the word is. The voice key is used to determine the latency to start speaking, so you'll need a microphone for this one. The typical finding is that the average latency is longer for different trials than for same trials.

2.7 Simon Effect Demo

This experiment illustrates the Simon effect, which was first reported by J. R. Simon in the late 1960's. On each trial, participants see either a red square or a blue square appear on the screen. They are instructed to press the '/' key (on the right side of the keyboard) if they see a red square, and the 'Z' key (on the left side of the keyboard) if they see a blue square. The squares can appear on either the left or the right edge of the screen. Although the location of the squares is irrelevant to the task, subjects typically respond faster when the locations of the stimulus and the response are congruent, i.e., a red square on the right side of the screen, and a blue square on the left side of the screen.

This demo also illustrates one way to use multiple stimulus lists. At the beginning of the script, the user is asked to enter a subject number, but also a list number, which is used to select the appropriate list from a folder named `lists`. List number can range from 1 to 5. In this particular case, all the lists are identical except for the order of trials.