User Manual

Graphical User Interface for the Psychoacoustic Listening Laboratory (PALILA)

For versions v1.2.x

Josephine Pockelé

PALILA GUI

Delft University of Technology

TU Delft

PALILA

– This page is intentionally left blank. –

Contents

1

Introduction

The *Graphical User Interface for the PsychoAcoustic Listening Laboratory*[1] *(PALILA GUI)* is a free, open-source, Python software developed by Josephine Pockelé to conduct listening experiments at the Delft University of Technology Faculty of Aerospace Engineering.

The interface is designed to allow for self-paced participation, without involvement of the responsible researchers during the listening part of the experiment. The GUI has a simple look with muted, neutral colours to avoid emotional interference with the results. Buttons are designed for use with touchscreen devices, to create better accessibility for participants.

The highlight features of the PALILA GUI are:

- A built-in questionnaire system to collect population statistics,
- Different types of questions, to enable a variety of listening tests,
- Ability to compare two sound samples,
- Simple result format, in the common csv format,
- Experiment configuration in an easy-to-understand language.

This manual is structured as follows:

Structure of this manual goes here :)

[1] More information about the facility: `https://www.tudelft.nl/lr/palila`

2

A Quick Start Guide to the PALILA GUI

A quick warning before using this software: this GUI was designed for the Laptop in PALILA, which has a $16{:}10$ aspect ratio screen with a $1920{:}1200$ pixels resolution. Other aspect ratios and screen resolutions will result in visual defects!! Future versions of the interface will provide greater screen compatibility.

This software was developed for *Microsoft Windows 10 and 11*. While it does not use any Windows-specific Python packages, it has not been tested on other operating systems. Usage on *Apple Mac*, *Linux*, and other Unix-based systems is not guaranteed to work. In case it does work without issue, feel free to inform Josephine Pockelé (j.s.pockele@tudelft.nl).

2.1. Setting up the GUI

First things first, let's get everything set up so you can run this GUI:

1. Download the latest version of the PALILA GUI software from *Zenodo*[1],
2. Unpack the *.zip* file into the folder where you want to use the GUI,
3. Install *Python 3.12*[2]. **IMPORTANT**: during installation, select "*Add Python 3.12 to PATH*".

After these steps, run the *SETUP.bat* script. This will create the virtual environment required to run the GUI. In case the script closes without prompting to "*Press enter to exit...*", use the following steps to set up the virtual environment:

1. Open a terminal window in the GUI directory.
2. Create a Python virtual environment *venv*[3].
3. Update the virtual environment:
   .\venv\Scripts\python.exe -m pip install –upgrade pip setuptools
4. Install the required Python packages into the virtual environment:
   .\venv\Scripts\python.exe -m pip install -r .\support_scripts\requirements.txt

Now, you should be good to go! Run *PALILA.bat*, and the GUI should appear in full-screen mode on your primary display. You can exit the interface with *Alt+F4*.

[1] Download available: `https://doi.org/10.5281/zenodo.15100497`
[2] Download available: `https://www.python.org/downloads/release/python-31210/`
[3] For further instructions, see: `https://docs.python.org/3.12/library/venv.html`

2.2. PALILA listening experiment structure

This section is a short explanation of how an experiment in this software will be structured. Figure 2.1 provides a visual representation. This structure cannot be altered without modifying the source code of the interface. The overall structure has built-in flexibilities, meaning a lot of variations are possible.

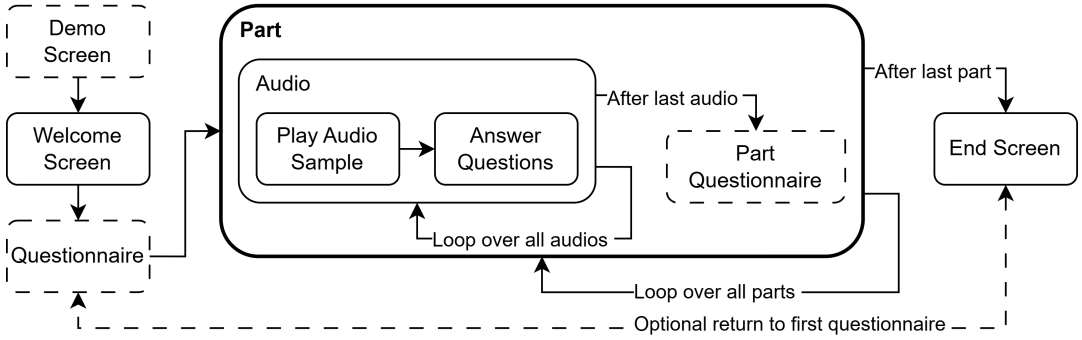


Figure 2.1: Overall structure of an experiment in the PALILA GUI.

The experiment normally starts with a *welcome screen*, but a *demonstration screen* can be added beforehand. This demonstration can be used by the researcher to show participants what to expect from the interface. The *welcome screen* shows a message, and allows for setting a custom participant ID, by the researcher. Before the listening part, a *questionnaire* can be conducted. The software contains a default set of questions for establishing population statistics and hearing health. This *questionnaire* can be fully disabled.

The listening portion of the experiment is divided into so-called *parts*. This grouping of *audios* can be used to test different types of sound with a clear distinction for the participants, and in the result files. Within a part, a list of *audios* is defined, which the software will automatically loop over. For each *audio*, participants will have to listen and answer one or two *question(s)*. At the end of each *part*, another questionnaire can be added. This can be used to ask further questions specific to that part, or questions that should be asked at the end of the experiment.

The experiment ends with the *end screen*. On this screen, participants can return to the first questionnaire to check their answers.

2.3. Creating your first experiment

With the GUI software installed, we can proceed to creating your first new expiriment. This is a very simple process: run the *NEW_EXPERIMENT.bat* script and enter a name when prompted. This will create one file and one folder with the name you entered. The folder will contain one audio file: *tone500Hz.wav*. Do not remove this file, since it is required by some interface elements.

Figure 2.2 shows the structure of the experiment created by the script. This is the simplest configuration of the GUI, and forms the baseline for all other explanations in this manual.

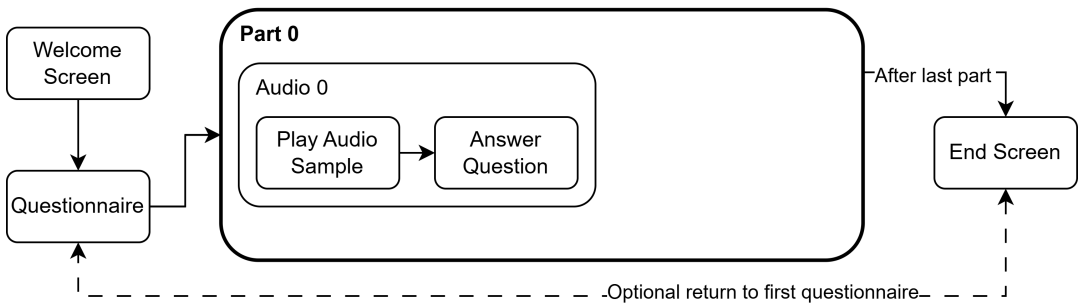


Figure 2.2: Structure of the new experiment.

This file with *.palila* extension will be called the “PALILA file” throughout this manual. You can open the PALILA file with your favourite text editor (e.g.: Windows Notepad, VIM, Notepad++, etc). It will contain the following configuration code:

```
1  ## This is your experiment input file.
2  ## It is set up with a basic structure for a PALILA GUI experiment.
3
4  pid mode = auto
5  randomise = no
6  demo = no
7
8  [questionnaire]
9      default = yes
10
11 [part 0]
12     randomise = yes
13
14     [[questions]]
15         [[[question annoyance]]]
16             type = Annoyance
17
18     [[audio 0]]
19         filename = 'tone500Hz.wav'
```

Let’s explain what this file has configured, referring to Figure 2.1 and Figure 2.2. First, we cover the lines defining the three left blocks of Figure 2.1:

- Line 4: the GUI will set the participants ID to a timestamp. The *welcome screen* will state: “Your participant ID is set automatically.”
- Line 5: the interface will keep the order of the *parts* in the PALILA file.
- Line 6: disables the optional *demo screen* of the GUI.
- Lines 8-9: the software adds a questionnaire at the start, containing the default set of questions.

The parameters *randomise* and *demo* are so-called ‘optional’ parameters. When they are not defined in the PALILA file, they will be set to ‘no’. A full list of these parameters and their default values are found in section 2.5 The default questionnaire questions are listed in section 5.2.

Then the listening part of the experiment is defined with the following lines:

- Line 11: defines the name of the *part*. Each part can be given a descriptive name, which will be used for outputting the answers.
- Line 12: the interface will shuffle the different *audios* within *part* ‘0’. *Audios* in other *parts* are not affected.
- Line 14-16: this block defines the *questions* used for all *audios* in *part* ‘0’. In this case, the GUI will display an 11-point ICBEN annoyance rating scale to the participants.
- Line 18-19: defines the one *audio* in *part* ‘0’, which is the default 500 Hz tone in the folder with the name of the experiment.

For details about the different experiment configurations, please refer to the other chapters of the manual. All input parameters are described in detail, and with visual examples.

2.4. PALILA file syntax

With the setup out of the way, let's look at the actual syntax of the PALILA file. The PALILA GUI uses the *ConfigObj* package to read PALILA files, hence the overall syntax is the same as described in their documentation [4]. This section will shortly summarise the most relevant items for the PALILA GUI.

Firstly, you will notice the top two lines of the setup code start with hashtags (#). These can be used to put comments in your PALILA file. Comments can also start at the end of a line of configuration code, again using a hashtag. See the below example:

```
# This line is a comment
parameter = value  # this is also a comment
```

This example also shows the basic syntax to set a value for any of the parameters. Note that parameter names can contain spaces. The values can take many forms, also called *parameter types*. The types and how to define them are described below:

2.4.1. String values

A String value can contain any form of text. There are three ways to define a string value:

```
parameter = A single line string
parameter = 'A single line string'
parameter = '''A single line string'''
```

For multi-line strings, triple quotes should always be used. In general, tab characters will be ignored to allow for indentation. Multiple lines in the configuration file will always result in multiple lines in the GUI.

```
welcome = '''A string which spans multiple lines
    should be in triple string quotes.'''
```

2.4.2. Numerical values

The PALILA GUI uses two types of numerical values. Currently, the code will read these as string values and convert them to numerical. The two types are *integers* and *floats*. When defining numbers, it is very important to use the correct representation.

Integer parameters will only accept pure integer numbers, such as '2', '40', '5000'. Any other form will result in a conversion error in Python.

Floating point numbers can be defined in three ways:

1. pure integer (e.g. '2', '40', '5000', ...),
2. pure float (e.g. '1.2', '4.56', ...),
3. scientific notation (e.g. '1.2e3', '4.56e7', ...)

2.4.3. Boolean values

There are multiple ways to define a boolean value [5]. The four pairings below are guaranteed to work (both capitalised and lower case):

True	False
1	0
Yes	No
On	Off

[4] See: `https://configobj.readthedocs.io/`

[5] See the "as_bool" function: `https://configobj.readthedocs.io/en/latest/configobj.html#section-methods`

2.4.4. List of string values
For some parameters, multiple string values have to be entered. This can be done using a list of string values. To avoid possible confusion, the single quote string syntax is recommended. Other notations of strings (no quotes, or triple quotes) may work, but this is not guaranteed. Multi-line strings are known to not work correctly!

2.4.5. Code blocks
Next to the different parameter types, PALILA files have a hierarchy in the form of blocks, and sub-blocks. In the current state of the GUI, there will be three levels of blocks. Blocks on the first level are defined with single square brackets [ ]. Each level down the hierarchy requires an extra set of brackets. In the examples throughout this manual, indentation is used in combination with these different block levels. These indentations provide good visual aids while writing a configuration file, but they have no meaning and will be ignored. PLEASE NOTE: always use TAB as indentations (some IDEs will put 4 spaces instead, but this will result in problems. The use of these blocks will be further explained in chapter 3 with an example configuration. See the example below:

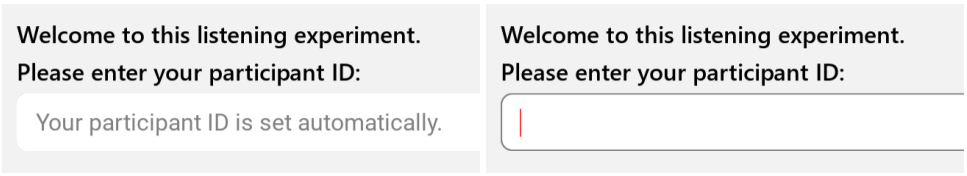
```
[block level 1]
    [[block level 2]]
        [[[block level 3]]]
```

2.5. Base parameters
This section will cover the paremeters at the top of a PALILA file, which control some basic aspects of the interface and the experiment structure. These parameters should ALWAYS be defined in the lines above any blocks, because otherwise they will not be read correctly.

2.5.1. pid mode (parameter)
***Required parameter: sets mode of identifying participant results. Options:* auto*, input*.**

The Participant IDentification mode defines how the individual participant results are identified in the output filenames. Currently, two modes are accepted:

auto: Assigns the timestamp containing date and time of the participant starting the experiment.
input: An ID will be manually entered in the *welcome screen*.



(a) *auto* **(b)** *input*

Figure 2.3: Difference between the PID modes, in the *welcome screen*

When set to *auto*, the result file name will be the timestamp of the date and time of clicking *continue* in the *welcome screen*. Example: the participant clicks continue on 23 April 2025 at 09:41, which results in the filename: *250423-0941.csv*.

In *input* mode, the result file will carry the ID entered in the *welcome screen*. Example: the participant enters ‘palila123’, which results in the filename: *palila123.csv*.

2.5.2. welcome (parameter)
Optional parameter: defines the welcome message. String value.

The *welcome* parameter can be used to define a custom message on the *welcome screen*. Any message can be entered, as long as it can be interpreted as a string by Python. The “*Please enter your*

participant ID: " line will always be added. When no message is defined, it reverts to its defaults (also shown in Figure 2.3):

"*Welcome to this listening experiment.*"

Section 2.4 explains the syntax of string values, which applies for this parameter.

2.5.3. goodbye (parameter)

Optional parameter: a custom message at the end of the experiment. String value.

At the end of the experiment, after clicking the *Finish Experiment* button, participants will see the *goodbye* message. The default message is:

"*Thank you for your participation in this experiment!*"

Since the input is a string value, it works similarly to the welcome message. Please refer to the examples above to see the behaviour of this parameter. The *goodbye* message shows up in the middle of the screen, with centred alignment.

2.5.4. randomise (parameter)

Optional parameter: activates shuffling of the experiment parts. Boolean value (default = 'no').

Normally, the experiment parts will appear in the order of the configuration file. The two code snippets below will ALWAYS result in the following order: ... → *part beta* → *part alpha* → ...

```
[part beta]
    ...
[part alpha]
    ...
```

```
randomise = no

[part beta]
    ...
[part alpha]
    ...
```

When *randomise* is set to 'yes', some participants will receive the order ... → *part beta* → *part alpha* → ...
Other participants will get the order ... → *part alpha* → *part beta* → ...
NOTE: this affects all *part* blocks in the PALILA file. This parameter does not affect the order of samples within each *part* (see section 4.1).

```
randomise = yes

[part beta]
    ...

[part alpha]
    ...
```

2.5.5. demo (parameter or block)

Optional parameter: activates the default demonstration screen. Boolean value.
Optional block: defines a custom demonstration screen. Level 1 block.

By default, the demonstration screen (see Figure 2.1) is deactivated (*demo* = *no*). The demonstration screen can be configured in two ways. To show the default demonstration, with one audio and the ICBEN 11-point annoyance scale question, only a single line of code is required:

```
demo = yes
```

IMPORTANT: the default demonstration screen requires the presence of *tone500Hz.wav* in your experiment folder. This file can be recovered from the *GUI/assets* folder, if required.

The default demonstration screen will show the elements in Figure 2.4:

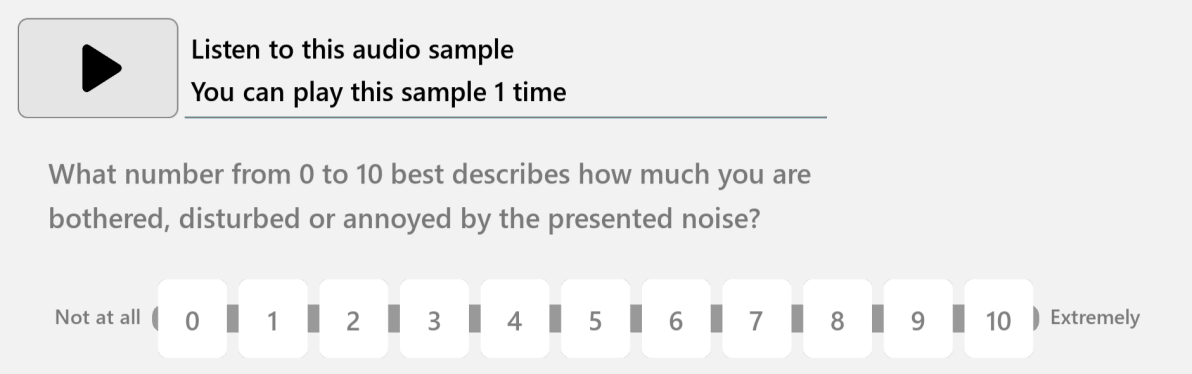


Figure 2.4: Audio and question elements of the default demonstration screen.

The second method is defining a *demo* block. This block has the same structure as an *audio* block, which is described in chapter 4. Note that the *demo* block is a block on level 1, while an *audio* block is normally on level 2. The second difference is in defining the audio file name, which becomes optional. The *repeat* keyword will not serve any function. The default audio file is the 500 Hz tone created by the *NEW_EXPERIMENT.bat* script.

Functionally, the extent of a *demo* block will be limited to the following code snippet:

```
[demo]
    filename_2 = ...
    max replays = ...

    [[question 1]]
        ...
    [[question 2]]
        ...
```

3

Example Configuration File

To show how the PALILA file's block structure works, this chapter follows an example of building a complex experiment. The overall structure of the final experiment is shown in Figure 3.1. The final experiment PALILA file is given in Appendix A, and can be found in the GUI repository as "example/example.palila".

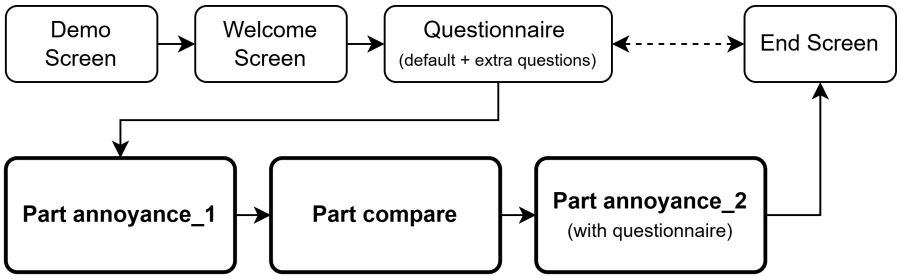


Figure 3.1: Final structure of the example experiment.

We start with the PALILA file from the *NEW_EXPERIMENT.bat* script, shown in section 2.3. First, we will set up the base parameters to our liking. We want an automatic participant ID, the default *welcome* and *goodbye* messages, and no randomisation of the parts, as well as the default demonstration screen. The layout of the default demonstration screen is previously shown in Figure 2.4.

```
4 pid mode = auto
5 randomise = no
6 demo = yes
```

3.1. Questionnaire

While we want the default questionnaire, the experiment also calls for a few extra questions. The default questionnaire has been manually configured with three screens, which require manual assignment of a questionnaire screen for the extra questions. A diagram of the desired layout of the questionnaire for this experiment is shown in Figure 3.2:

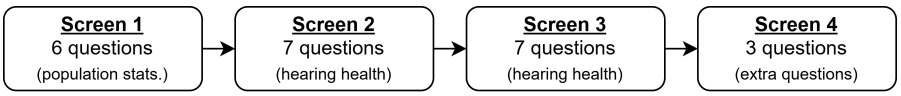


Figure 3.2: Diagram of the experiment questionnaire structure.

Let's add our extra questions to the questionnaire block. The code is shown below. Most importantly, these three question blocks are started with double brackets, since they are on the second block level. The *manual screen* is set to 4 so they appear after the default question set. The remaining parameters are further explained in chapter 5.

```
[questionnaire]
  default = yes

  [[question heard_before]]
    text = 'Have you ever heard wind turbine noise before?'
    type = MultipleChoice
    choices = 'Yes', 'No', 'Not sure'
    manual screen = 4

  [[question live_near]]
    text = 'Can you see a wind turbine from your home?'
    type = MultipleChoice
    choices = 'Yes', 'No'
    manual screen = 4

  [[question live_distance]]
    text = '''If yes, what is the approximate distance from
        your home to this/these wind turbines?'''
    type = MultipleChoice
    choices = '< 500 m', '500 - 1000 m', '1 - 2 km', '> 2 km'
    manual screen = 4

    unlocked by = 'live_near'
    unlock condition = 'Yes'
```

This code results in the questionnaire according to Figure 3.2, and the questions on screen 4 will look like Figure 3.3:

Figure 3.3: The extra screen of questions in the questionnaire of the experiment.

With the *unlocked by* and *unlock condition* in the third question, participants only have to answer how far they live from wind turbines, if they answered 'Yes' to the previous question. This mechanism of conditional questions is further explained in chapter 5.

3.2. Part 1 - Annoyance (1)

The first part of the experiment's main body is called 'annoyance_1' (see Figure 3.1). Within this part, we want the questions to be in a random order for each participant. We therefore start the *part* block with:

```
[part annoyance_1]
  randomise = yes
```

To give more context to the participants, we will add a *intro* block with a custom text, and set a hold *time* of 10 seconds. Note that *intro* is again in double brackets, to indicate it is inside the *part* block.

```
37  [[intro]]
38    text = '''Thank you for filling out our questionnaire!
39
40      In the first part of the listening experiment, you will listen
41      to noise recordings of a wind turbine.
42
43      While listening, imagine that this is the sound situation
44      outdoors (e.g. your garden or a park).'''
45
46    time = 10
```

The resulting introduction screen will look like Figure 3.4. The blue timer bar on top of the *continue* button will last 10 seconds, and the *continue* button will unlock once the timer bar is filled. This introduction screen is further explained in subsection 4.1.4.

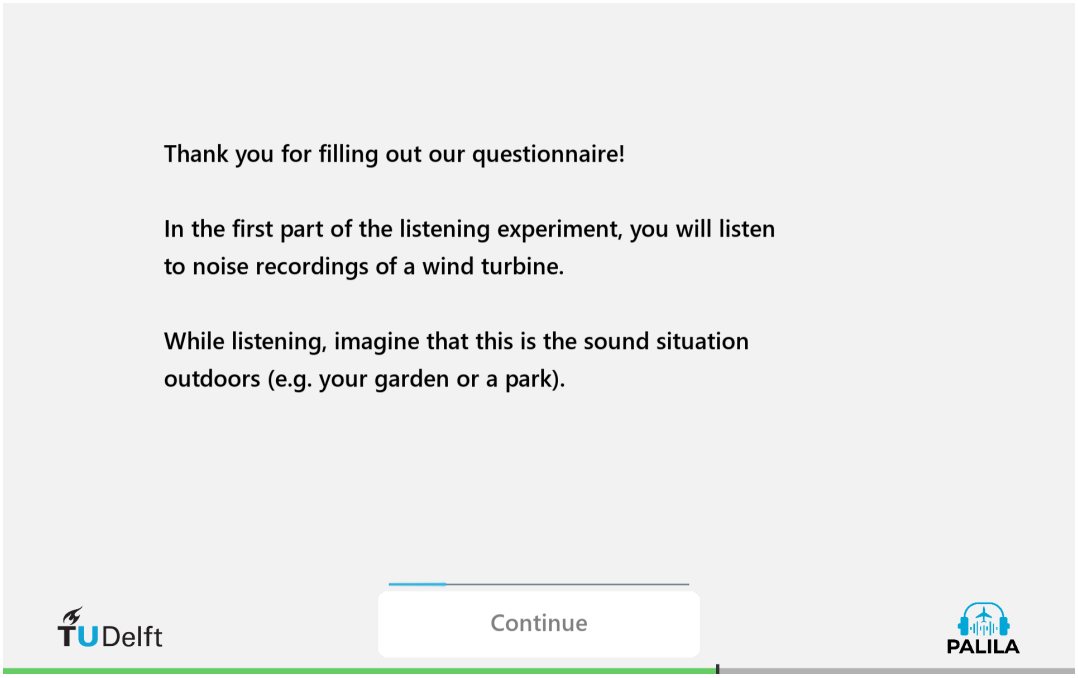


Figure 3.4: Introduction screen of the 'annoyance_1' part of the example experiment.

Because this part requires a single question for all audio samples, we define this in a separate *questions* block, so we don't have to duplicate the question every time. First, note the triple brackets around the *question* definition, to signify it is a block inside the *questions* block.

```
48  [[questions]]
49    [[[question annoyance]]]
50      type = Annoyance
```

The *Annoyance* type question results simply in the ICBEN 11-point scale for measuring annoyance:

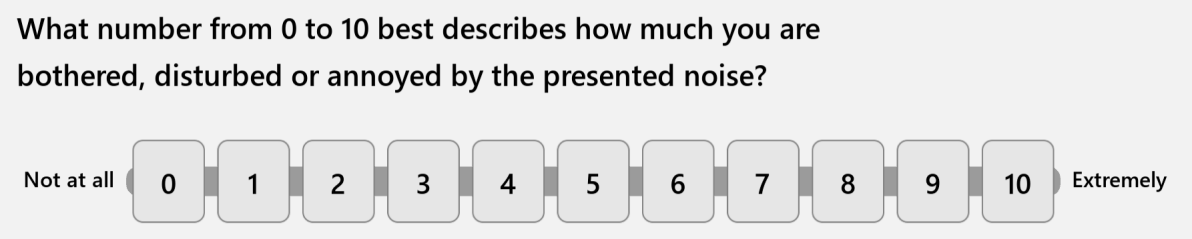


Figure 3.5: The ICBEN 11-point scale for measuring noise annoyance resulting from the *Annoyance* question type.

The 'annoyance' identification in line 49 will be used in the identifier of the answers in the output file. A more thorough explanation of defining the *questions* and their related types is provided in subsection 4.1.3 and section 4.3.

The following block, with identifier 'sample_1', defines one *audio* sub-block. This tells the GUI which sound samples to play, and some related settings. More information can be found in subsection 4.1.2. You have to define one of these blocks for each sample (or pair of samples) you want to include in your experiment.

For this part, we want each sample-question pairing to appear twice, which is achieved with the *repeat* parameter. For this example, we will use the 500 Hz tone audio file every time. You can replace the *filename* with any audio file inside the experiment folder. Two more identical *audio* blocks are defined with identifiers 'sample_2' and 'sample_3'.

```
52   [[audio sample_1]]
53     filename = 'tone500Hz.wav'
54     repeat = 2
```

3.3. Testing your configuration

At this point, we have reached the minimum requirement for the GUI to work. To test the configuration more easily, use the *override* parameter. This will unlock most continue buttons so you can scroll through your experiment quickly:

```
override = yes
```

Part intros and *breaks* cannot be skipped with this method, but setting the hold *time* to 0 seconds will instantly unlock the *continue* buttons:

```
time = 0
```

Before running your experiment with participants, **DO NOT FORGET** to reset these lines, so your participants cannot skip parts of your experiment.

3.4. Part 2 - Comparison

As the name of the second experiment block implies, it is an example of including two sound samples in a single *audio* screen. The *part* block is started with the following:

```
65 [part compare]
66   randomise = yes
67
68   [[intro]]
69     ...
```

A 10-second break is added to the middle of this listening block, using a *breaks* sub-block inside the *part* block. The *interval* parameter is set to '2' to split this part in the middle. A negative number is used

so no break screen appears at the end of this *part*. This is further explained in subsection 4.1.5.

```
78  [[breaks]]
79    text = '''This is a short break from listening.
80      Please stay as long as you need.
81      When you continue, imagine that this is the sound situation
82      outdoors (e.g. your garden or a park).'''
83
84    interval = -2
85    time = 10
```

During these comparisons, two questions will be asked to participants. A *questions* sub-block is used once more, since these questions are used for all comparisons. The first question uses the *Multiple-Choice* type, to ask participants which sample they find more annoying. If one sample is more annoying, the *IntegerScale* question will ask by how much, from 1 to 10. The *unlocked by* and *unlock condition* are used, so participants do not have to answer the second question if they find both samples equally annoying.

```
87  [[questions]]
88    [[[question which]]]
89      text = '''Which sample did you experience as more annoying?'''
90      type = MultipleChoice
91
92      choices = 'Left', 'Equally Annoying', 'Right'
93
94    [[[question rating]]]
95      text = '''On this scale from 1 to 10, how much MORE bothered,
     disturbed or annoyed were you by the most annoying sample, compared to
     the less annoying sample?'''
96      type = IntegerScale
97
98      min = 1
99      max = 10
100     left note = 'Almost equal'
101     right note = 'A lot more'
102
103     unlocked by = which
104     unlock condition = Left;Right
```

Adding a second sound sample to a *audio* block is done with the *filename_2* parameter. As this example shows, it is possible to add the same sample twice to the same screen.

In this part of the experiment, the *max replays* parameter is set to '2', so participants can listen to each of the samples two times. The *repeat* parameter is again set to '2', so each comparison will be presented twice.

An identical *audio* block named 'comparison_2' is also added to this experiment (see Appendix A).

```
107 [[audio comparison_1]]
108   filename = 'tone500Hz.wav'
109   filename_2 = 'tone500Hz.wav'
110
111   repeat = 2
112   max replays = 2
```

Ultimately, the *audio* and *questions* sub-blocks will result in four times the following screen:

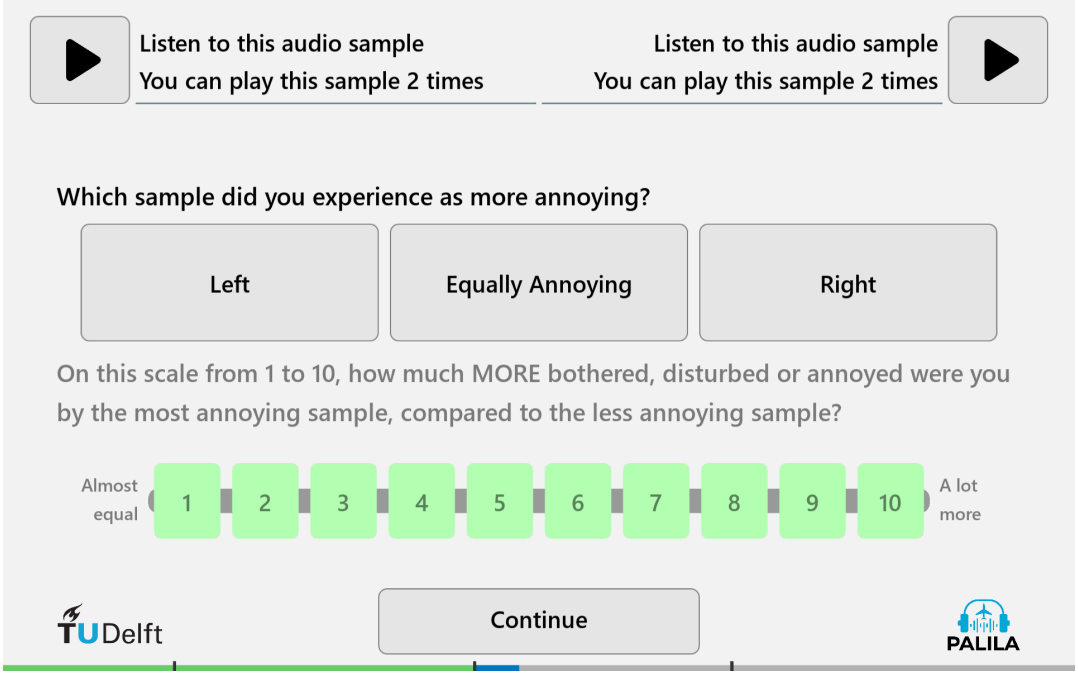


Figure 3.6: The audio screen in the comparison part of the example GUI.

3.5. Part 3 - Annoyance (2)

Lastly, a third *part* with simple annoyance questions is added to the experiment. Since this is the last *part* of the experiment, a small questionnaire is added at the end. For the purpose of this example, we make a copy of the 'annoyance_1' part and rename it to 'annoyance_2':

```
[part annoyance_2]
  ...
```

At the end of this *part* block, a *questionnaire* sub-block is added with two multiple choice questions:

```
  [[questionnaire]]
    [[[question wt_opinion_change]]]
      type = MultipleChoice
      text = '''Has this experiment changed your attitude
        towards wind turbines?'''
      choices = 'Yes, positively', 'No', 'Yes, negatively'

    [[[question wtn_acceptable]]]
      type = MultipleChoice
      text = '''Would you find any of the presented sound situations
        acceptable in your daily life?'''
      choices = 'None', 'Some', 'Most', 'All'
```

This final questionnaire's questions will appear as Figure 3.7

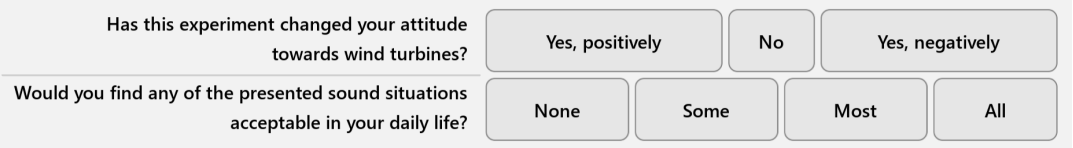


Figure 3.7: Questionnaire questions at the end of the third part of the example GUI configuration.

3.6. Output files

TODO: Add explanation of the resulting output file here.

4

Audio Configuration

Chapter intro

4.1. Part Block

Required block: defines a part of the experiment with similar audios. Level 1 block

The most important components of the experiment configuration are the *part* blocks. For one, an experiment configuration requires at least one, and secondly, they contain the information about the audio samples of your experiment. The *part* block contains some sub-blocks to define elements not related to the sound samples, and sub-blocks to define the individual sound samples or sound sample pairings.

Defining a *part* block consists of a line of configuration code with three pieces of information:

1. Single square brackets to indicate it is a level 1 block,
2. The word 'part' to indicate it is a *part* block,
3. And the name of the part.

Thus each part should start with the following line of code, where 'part_name' will be used in the identifier of a specific audio-question combination in the output file:

```
[part part_name]
```

Usually, a *part* block will be structured as follows:

```
[part part_name]
  randomise = <boolean>

  [[intro]] (optional)
    ...
  [[breaks]] (optional)
    ...
  [[questions]] (optional)
    ...

  [[questionnaire]] (optional)
    ...

  [[audio audio_name_1]]
    ...
  [[audio audio_name_2]]
    ...
  ...
```

The following sub-sections will explain each of these sub-blocks and parameters one-by-one.

4.1.1. Randomise (parameter)

Optional parameter: activates shuffling the audios within a part. Boolean value (default = 'no')
Similar to how *part* blocks can be shuffled in the overall experiment, the *audio* sub-blocks can be shuffled within each *part*. This parameter ensures each participant will be presented with a pseudo-random order of these audios that will be different from the other participants.

Both *randomise* parameters do not affect the identifiers in the output file, so researchers will always know a response comes from the same audio-question pairing, no matter which order they were presented to participants.

4.1.2. Audio (sub-block)

Required sub-block: defines one audio sample or pairing of two samples. Level 2 block

Similar to the *part* blocks, this block is defined with three pieces of information:

1. Double square brackets to indicate it is a level 2 block,
2. The word 'audio' to indicate it is an *audio* sub-block,
3. And the name of the audio.

This again translates to the following line of code, where 'audio_name' will be used in the identifier of this specific audio-question combination in the output file:

```
[[audio audio_name]]
```

The *audio* sub-block has four parameters. It is possible to define questions per *audio* block, using (a) *question* sub-block(s). This is explained further in section 4.2.

filename (parameter)

Required parameter: defines the relative path of the sound sample. String value

This parameter defines the path relative to your experiment folder, to a sound sample. For example, the PALILA file is named 'experiment.palila', the GUI will try to find a sound sample with filename 'sound_sample.wav' at *path_to_gui* *experiment* *sound_sample.wav*

IMPORTANT: the *filename* should always include the file extension (i.e. '.wav', '.mp3', etc.)!

filename_2 (parameter)

Optional parameter: defines the relative path of a second sound sample. String value

Similar to *filename*, this parameter will add a second sound sample on the same screen. This can be used, for example, for comparisons. The second audio will have to be listened to by participants. The two sound samples are blocked from playing simultaneously. The sound sample bar on top of the *audio* screen will look like:

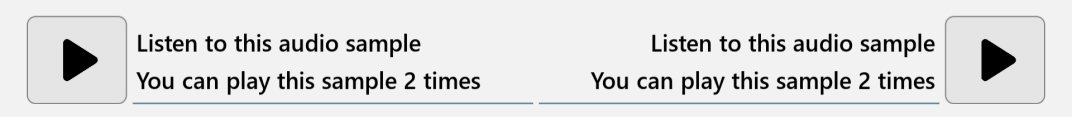


Figure 4.1

repeat (parameter)

Optional parameter: number of times participants get this audio-question pairing. Integer value

With this parameter, the *audio* screen from one *audio* sub-block is repeated multiple times, without the need for duplicated definitions. In the output file, these repetitions will result in separate outputs, distinguished by an additional *repetition identifier*.

max replays (parameter)
Optional parameter: number of allowed replays of the sound sample(s). Integer value (default = 1)

This parameter defines the number of times a sound sample in one *audio* screen can be played by participants. In the example configuration in Figure 3.6 and Figure 4.1, *max replays* is set to '2', meaning participants can listen to each sample twice. In case of two sound samples, using *filename_2*, this parameter applies to both samples.

4.1.3. Questions (sub-block)
Optional sub-block: defines common questions for all audios in a part. Level 2 block

For most experiments, one group of sound samples will require the same question(s). The *questions* sub-block allows you to define the common questions for all *audios* within a *part*. Simply define one or two *question* sub-blocks within this sub-block, according to the same syntax as described in section 4.2.

The identifier will follow the convention outlined in section 4.1, subsection 4.1.2, and section 4.2.

IMPORTANT: this sub-block will overwrite any questions defined within an *audio* sub-block!

4.1.4. Intro (sub-block)
Optional sub-block: defines a custom introduction to the part. Level 2 block

This sub-block can be used to customise the introduction screen of a *part*. A custom text can be set, as well as customised timing. This sub-block requires two parameters.

text (parameter)
Required parameter: defines the custom text of the introduction. String value

time (parameter)
Required parameter: defines the hold time of this introduction. Float value

The *time* parameter defines in seconds how long the participants will be held in the introduction before they can continue.

4.1.5. Breaks (sub-block)
Optional sub-block: defines mandatory breaks for participants. Level 2 block

To alleviate participant fatigue, you can put mandatory breaks into the experiment. This sub-block defines all breaks within a single *part*.

text (parameter)
Optional parameter: defines the custom text of the introduction. String value

time (parameter)
Required parameter: defines the hold time of this introduction. Float value

The *time* parameter defines in seconds how long the participants will be held in a break before they can continue.

interval (parameter)
Required parameter: defines the interval between breaks. Integer value

There are three ways to define the interval:

1. Zero: a break will be placed at the end of the part, after the questionnaire

4.1.6. Questionnaire (sub-block)
Optional sub-block: defines a questionnaire at the end of the part. Level 2 block

Use of this sub-block enables and extra questionnaire at the end of an experiment part. This can be used to ask supporting questions about the *audios*, or as a closing questionnaire. This sub-block is

defined with the exact syntax as the main questionnaire at the start of the experiment. See chapter 5 for more.

IMPORTANT: the *default* parameter does not work for part questionnaires. It can only be used in the initial questionnaire.

NOTE: *questions* in the *part* questionnaires are defined as LEVEL 3 BLOCKS, instead of level 2!

4.2. Question Sub-Block

Optional sub-block: defines a single question. Level 3 block

A *Question* sub-block, similarly to *part* blocks and *audio* sub-blocks, is defined with three pieces of information:

1. Triple square brackets to indicate it is level 3 block,
2. The word 'question' to indicate it is a *question* sub-block,
3. And the name of the question.

Combining these results in the following line of code, where 'question_name' will be used in the identifier of this question within each audio-question pairing:

```
[[[question question_name]]]
```

4.2.1. text (parameter)

Required parameter: defines the question to be asked to participants. String value

This parameter defines the wording of the question. A clear example of its use can be found in section 3.4. The design of the GUI will automatically fit this *text* in the available space above the answering system. This space is two lines high, and approximately 80 characters wide. The recommended limit for this parameter is 160 characters.

4.2.2. type (parameter)

Required parameter: defines the type of this question. String value

For each question, the type has to be defined. The choices for question types on an *audio* screen are listed in section 4.3. The value entered in this parameter should match exactly with the names listed below, including capitalisation. Depending on the set type, more parameters will be required, which will be listed below.

4.2.3. unlocked by (parameter)

Optional parameter: identifier of a question which unlocks this question. String value

Together with the *unlock condition*, this parameter allows for a question to only require an answer if a different question is answered in a certain way. This parameter defines the identifier of the other question which unlocks this one. This identifier is 'question_name' in the example at the top of this section.

The unlock system will only work within the scope of one *audio* screen. Therefore, this system will not work across different *audio* blocks. An example of this unlocking system is shown in section 3.4.

4.2.4. unlock condition (parameter)

Optional parameter: answers which will unlock this question. String value

This parameter is required when *unlocked by* is defined. This parameter will list all answers of the other question that unlock this question. In case of a single value, this looks like the example on the left. The right example shows how this works for multiple unlock values.

```
[[[question other_question_name]]]
  ...

[[[question question_name]]]
  ...

  unlocked by = 'other_question_name'
  unlock condition = 'value_1'
```

```
[[[question other_question_name]]]
  ...

[[[question question_name]]]
  ...

  unlocked by = 'other_question_name'
  unlock condition = value_1;value_2
```

For one *unlock condition* value, any form of the *string* value syntax can be used (see subsection 2.4.1. Multiple values requires a very specific formatting. No quotation marks should be used, and the values should be split by semicolons (;). This semicolon cannot be followed by a space!

In case a question remains locked (i.e. no answer required), the output value will be ”na”.

4.3. Question Types

4.3.1. Text

This question type only shows the question text. Because there is no answering system, the space for text is increased to five lines, increasing the maximum recomended number of characters to $5 \times 80 = 400$.

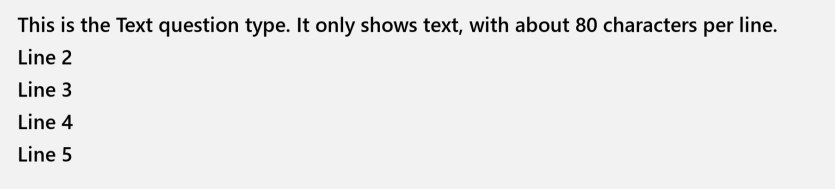


Figure 4.2: Example of the Text question type.

4.3.2. MultipleChoice

This question type provides a button-based multiple-choice answering system. This question type can force users to select a single answer, or it can allow for multiple answers. The recommended limit of choices is 5, but the actual maximum depends on the length of the choices. Below is an example of what the question looks like before answering, and after selecting two answers:

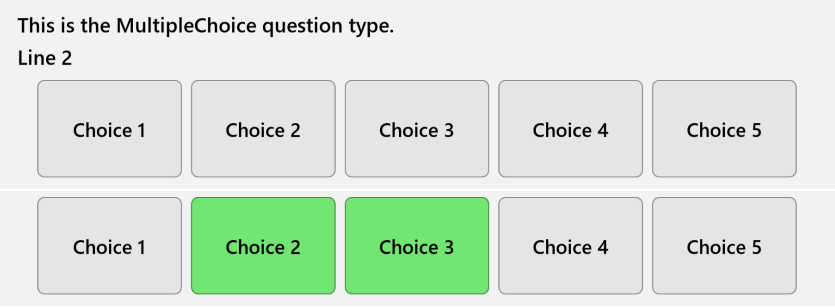


Figure 4.3

This question type requires one additional parameter inside the question sub-block:

choices (parameter)
Required parameter: list of the possible answers. List of string values

This parameter defines the options to show on the buttons. These can be any string value, but are limited to one line, due to the syntax (see subsection 2.4.4).

For the above example, the following line was used to define *choices*:

```
choices = 'choice 1', 'choice 2', 'choice 3', 'choice 4', 'choice 5'
```

multi (parameter)
Optional parameter: allows for multiple answers. Boolean value (default = 'no')

When set to 'yes', this parameter allows participants to select multiple answers on the multiplce choice question. In the results file, the answers will be separated by a semicolon (;). The example above shows multiple selected answers with this parameter turned on.

4.3.3. Spinner

A spinner is essentially a *MultipleChoice* question, but with a dropdown menu, instead of buttons. This question type can only be used for multiple-choice, single answer questions. The example below shows how this question works for the participants. Once they have selected an answer, it cannot be removed like in the button variant.

choices (parameter)
Required parameter: list of the possible answers. List of string values

See the *MultipleChoice* question type (subsection 4.3.2).

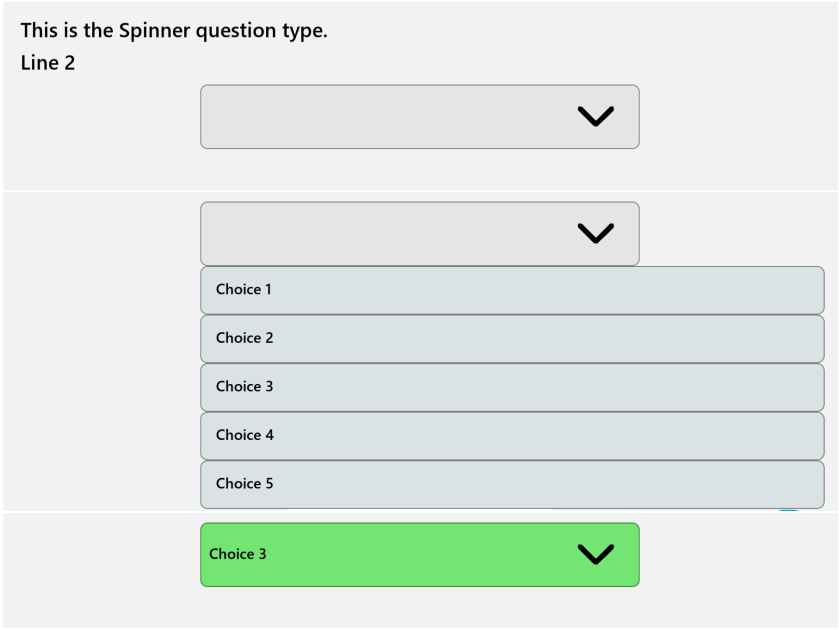


Figure 4.4

4.3.4. IntegerScale

This question type gives participants a scale with integer values to answer. It is highly customisable. The GUI will automatically scale the answer buttons depending on the number of them. The step between numbers is always 1, thus the number of buttons is always $max - min + 1$. Text can be defined on the left and right side of the scale, to give participants an indication of the meaning of the extreme values. Since this question type is derived from *MultipleChoice*, the variable *multi* can also be used.

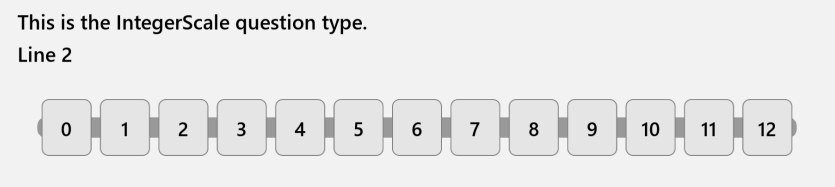


Figure 4.5

min, max (parameters)
Required parameters: define the range of the scale. Integer values

These parameters define the minimum and maximum values on the scale, respectively. The recommended maximum range ($max - min$) is 10 with text on either end of the scale, and 12 for a scale without.

left note, right note (parameters)
Optional parameters: define text on either side of the scale. String values

On both ends of the scale, a *note* can be placed. An example of this is shown below. One of these notes can be empty, but the scale will always be adjusted to fit both notes. These notes can fit approximately 10 characters on one line, with a recommended maximum of two lines (i.e. 20 characters total).

```
left note = 'Left (0)'
right note = 'Right (10)'
```

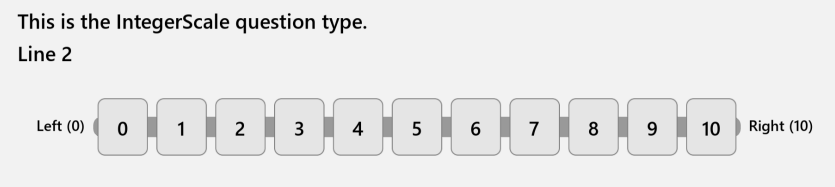


Figure 4.6

4.3.5. Annoyance

A predefined version of the IntegerScale with the ICBEN standardised annoyance scale. The example below shows this configuration. This question type only requires the *type* parameter. The *text* parameter is strictly optional for this question type and will simply overwrite the default text.

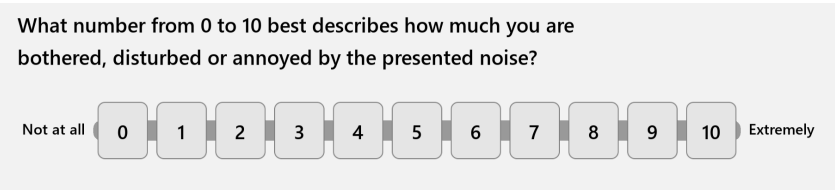


Figure 4.7

4.3.6. Slider

This is a variant of the IntegerScale question type, but with a slider instead of buttons. The main advantage is that this type allows for a more granular numerical answer. Below is an example of how this question may look for participants.

IMPORTANT: this question type will be revised in a future version! This revision will allow for more customisation than currently possible.

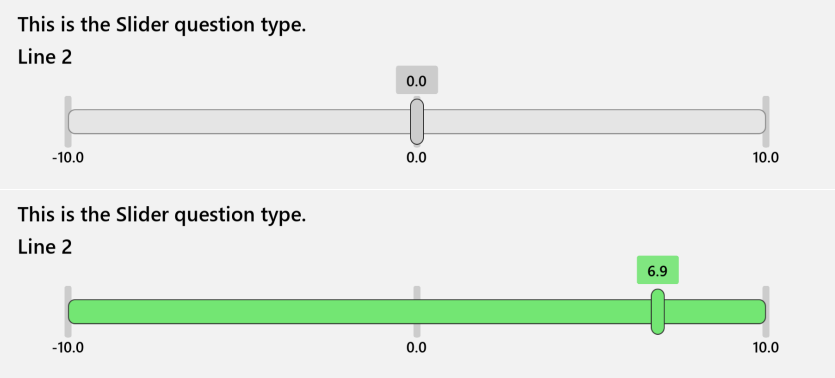


Figure 4.8

min, max (parameters)
Required parameters: define the range of the scale. Integer values

See the *IntegerScale* question type (subsection 4.3.4)

step (parameter)
Required parameter: defines the delta between possible answers. Float value

With the slider it is possible to allow for smaller steps than 1 between possible numbers. E.g. if the step is set to 0.1, it is possible to answer 6.9, 4.2, etc. With a step of 0.5, only 6.5 and 4.0 would be possible.

left note, right note (parameters)
Optional parameters: define text on either side of the scale. String values

See the *IntegerScale* question type (subsection 4.3.4)

5

Questionnaire Configuration

Introduction to chapter. Both main questionnaire and part questionnaires. Mention that a seperate end questionnaire is also in the development pipeline.

5.1. Questionnaire Block

Optional block: defines the main questionnaire at the start of an experiment. Level 1 block

5.1.1. default (parameter)

Optional parameter: turns on the default questions. Boolean value (default = 'no')

5.1.2. manual split (parameter)

Optional parameter: turn on manual question distribution. Boolean value (default = 'no')

5.2. Default Questionnaire

5.3. Question Types

5.3.1. FreeNumber

5.3.2. FreeText

5.3.3. MultipleChoice

5.3.4. Spinner

A

Example PALILA file from chapter 3

```
## This is your experiment input file. It is set up with a basic structure
    for a PALILA GUI experiment.
override = yes

pid mode = auto
randomise = no
demo = no

[questionnaire]
  default = yes

  [[question heard_before]]
    text = 'Have you ever heard wind turbine noise before?'
    type = MultipleChoice
    choices = 'Yes', 'No', 'Not sure'
    manual screen = 4

  [[question live_near]]
    text = 'Can you see a wind turbine from your home?'
    type = MultipleChoice
    choices = 'Yes', 'No'
    manual screen = 4

  [[question live_distance]]
    text = '''If yes, what is the approximate distance from
      your home to this/these wind turbines?'''
    type = MultipleChoice
    choices = '< 500 m', '500 - 1000 m', '1 - 2 km', '> 2 km'
    manual screen = 4

    unlocked by = 'live_near'
    unlock condition = 'Yes'


[part annoyance_1]
  randomise = yes

  [[intro]]
    text = '''Thank you for filling out our questionnaire!

      In the first part of the listening experiment, you will listen
      to noise recordings of a wind turbine.

      While listening, imagine that this is the sound situation
```

```
      outdoors (e.g. your garden or a park).'''

    time = 0.5e2

  [[questions]]
    [[[question annoyance]]]
      type = Annoyance

  [[audio sample_1]]
    filename = 'tone500Hz.wav'
    repeat = 2

  [[audio sample_2]]
    filename = 'tone500Hz.wav'
    repeat = 2

  [[audio sample_3]]
    filename = 'tone500Hz.wav'
    repeat = 2


[part compare]
  randomise = yes

  [[intro]]
    text = '''In the second part of the listening experiment, you will listen
      to noise recordings of a wind turbine. Each screen will contain two
      sound samples, which you will compare.

      While listening, imagine that this is the sound situation
      outdoors (e.g. your garden or a park).'''

    time = 0

  [[breaks]]
    text = '''This is a short break from listening.
      Please stay as long as you need.
      When you continue, imagine that this is the sound situation
      outdoors (e.g. your garden or a park).'''

    interval = -2
    time = 0

  [[questions]]
    [[[question which]]]
      text = '''Which sample did you experience as more annoying?'''
      type = MultipleChoice

      choices = 'Left', 'Equally Annoying', 'Right'

    [[[question rating]]]
      text = '''On this scale from 1 to 10, how much MORE bothered,
    disturbed or annoyed were you by the most annoying sample, compared to
    the less annoying sample?'''
      type = IntegerScale
```

```
97
98        min = 1
99        max = 10
100       left note = 'Almost equal'
101       right note = 'A lot more'
102
103       unlocked by = which
104       unlock condition = Left;Right
105
106
107     [[audio comparison_1]]
108       filename = 'tone500Hz.wav'
109       filename_2 = 'tone500Hz.wav'
110
111       repeat = 2
112       max replays = 2
113
114     [[audio comparison_2]]
115       filename = 'tone500Hz.wav'
116       filename_2 = 'tone500Hz.wav'
117
118       repeat = 2
119       max replays = 2
120
121
122   [part annoyance_2]
123     randomise = yes
124
125     [[intro]]
126       text = '''
127         In this final part of the listening experiment, you will listen
128         to noise recordings of a wind turbine, once more.
129
130         While listening, imagine that this is the sound situation
131         outdoors (e.g. your garden or a park).'''
132
133       time = 0
134
135     [[questions]]
136       [[[question annoyance]]]
137         type = Annoyance
138
139     [[audio sample_1]]
140       filename = 'tone500Hz.wav'
141       repeat = 2
142
143     [[audio sample_2]]
144       filename = 'tone500Hz.wav'
145       repeat = 2
146
147     [[audio sample_3]]
148       filename = 'tone500Hz.wav'
149       repeat = 2
150
151     [[questionnaire]]
152       [[[question wt_opinion_change]]]
```

```
153       type = MultipleChoice
154       text = '''Has this experiment changed your attitude
155         towards wind turbines?'''
156       choices = 'Yes, positively', 'No', 'Yes, negatively'
157
158     [[[question wtn_acceptable]]]
159       type = MultipleChoice
160       text = '''Would you find any of the presented sound situations
161         acceptable in your daily life?'''
162       choices = 'None', 'Some', 'Most', 'All'
```