Written by Raquel Ibáñez Alcalá, and Sofia Cooper
Created Sep 11, 2023, last updated Jul 2, 2024 (Rev. 1.2)

Human Model for Analyzing Neuroeconomic Situations

H.U.M.A.N.S. Application

User Guide

Revision 1.1

Table of Contents

H.U.M.A.N.S.

We have developed the Human Model for Analysing Neuroeconomic Situations (HUMANS) app. The purpose of this app is to measure decision-making (DM) in human subjects by presenting them with a questionnaire specifically designed to elicit DM behaviours in the subject while also collecting biometric data as the subject goes through the different trials.

Data collected

Subjects are not directly identified by name in the data produced by this application. Rather, a randomised ID number between 10000 and 99999 is generated and assigned to the subject. We then collect demographic information including but not limited to sex, gender identity, age range, race and ethnicity, and education. We also collect interests such as hobbies and what kind of media they consume. We use this data to classify subjects into clusters to attempt to identify any meaningful trends.

During trials, we collect biometric data, including gaze (where the subject is looking at on the screen, pupil dilation (estimated, in millimetres), and heart rate (measured in beats per minute), to potentially measure attentiveness and stress response. All trials are time stamped from when the question is presented (trial start), to when the subject submits their answer (trial_end).

All data is packed into a PostgreSQL table and uploaded as soon as the subject submits their answer. Additionally, demographic information (identified by ID number) is stored in a local text file.

How it works

Each subject is presented with a selected number of scenarios, in which a scenario (story) context is given. The subject must then put themself in that particular situation and think about how they would react.

Next, a list of potentially rewarding options to solve the problem at hand is presented to the subject. The subject then chooses which options would be most preferable by ranking them on a continuous 0 - 100 scale. This is then repeated for a list of potentially costly or unpleasant options to solve the problem and the subject must then rank them from least unpleasant to most unpleasant on the same continuous 0 - 100 scale. We call these rewarding and costly options "preferences".

Finally, the subject is presented with a predetermined number of questions that pertain to the preferences that were previously chosen in the actual trials. The subject must then answer the question with yes, no, or maybe on a scale. In the case where two different options are presented in the trial, the subject must indicate which option they lean toward the most (this is not the same as ranking preferences). As subjects answer these questions, we collect biometric data (see "Data collected" section).

App operation

The app is written using mostly Python, and runs within a Flask webserver. All webpage templates are written in HTTP and styled with CSS. All data is stored in a local or remote PostgreSQL database (depending on how the app is set up), as well as locally for redundancy in case of database failures.

What is included our repository

App elements

This includes files and scripts that critically pertain to the operation of the application. This includes Python scripts, HTML sheets, and CSS styling. This also critically includes the settings INI file.

Helper scripts

These are scripts that isolate certain functionalities that are built into the app, but are not referenced from the app's code. Such scripts can be changed and run independently from the app without any impact. The following is a full list of these scripts:

- create_map.py
- distribute_stories.py
- grab_ids.py
- import_demodata.py
- randomise_relationship_levels.py
- write_to.py
- breakdown_stories.py

Resource links

1. HUMANS App Github Repository*
2. ANT+ Heart Rate (HR) monitor (example)
3. ANT+ USB Antenna (example)
4. Eye tracker (Tobii Pro Spark)
5. Tobii Eye Tracker Manager (software)
6. Tobii Pro SDK (software)
7. Tobii Pro Spark Runtime Driver (software)**
8. Tobii Pro Spark User Guide (documentation)**
9. Pulse Monitor (software)

* This guide pertains to only the code contained in the main branch of our Github repository. The code contained in any other branch is not documented in this guide. Please ensure you are downloading code from the main branch.

** Which runtime driver and documentation will depend on what eye tracking system you are using. In this guide, we assume the Tobii Pro Spark eye tracker, and all links in this guide were made with that in mind. If your eye tracking system differs from ours, consult the manufacturer's instructions for setup and installation.

Setup

Installing the app

Our Human Decision-Making App (HDMA) is contained in our github repository. After zipping all files and downloading, extract the files onto any location on the hard drive.

Tweaking app settings

Navigate to the directory where you extracted the decision-making app files to. Navigate to the 'bin' folder and open 'settings.ini' with your text editor of choice (Notepad, Notepad++, Atom, etc…).

We first enter the PostgreSQL database credentials. This will look different depending on how the database was set up. However, refer to the following guidelines (or to the "App Settings" section) to help you know what each parameter is under the 'postgresql' section, or seek help from your database administrator:

- `host : The host computer's IPv4 address or domain name. If the HMDA is running on the host computer, enter the word localhost.`
- `database : The name of your PostgreSQL database.`
- `port : The host computer's port-forwarded port for servicing database queries (try 5432 if you don't know)`
- `user : The PostgreSQL username. Keep in mind that the account that the HMDA uses must have database read-write privileges.`
- `password : The password tied to the account identified by the 'user' parameter.`

Next, we need to change the settings that will affect how the HMDA behaves. This will look different depending on your own study's needs. However, refer to the following guidelines to help you know what each parameter is under the 'app_settings' section:

- `data_table : The database table to insert data into`
- `auto_create_table : If the table specified above does not exist in the database, the app may be able to create it using the participant's data. If auto_create_table is set to 1, the app will attempt to create the target data table once it has all the required data from the user. If set to 0, the app will simply`

crash when attempting to upload data if the table does not exist in the database.

- enable_consecutive_users : Enables back-to-back use of the app without the need to close and re-run startup.bat (which restarts the flask server. If set to 1, the app will reset all global app and user parameters to default when loading the landing portal (setup_session.html, routed as '/'). If this option is not enabled (set to 0), the flask server must be closed and re-run. Any changes to the source code or templates are not affected by this option.
- data_upload : Ensures that data is automatically uploaded to the database after each session.
- unique_ids_from : Controls where IDs are read from when trying to generate a unique ID for each user. Possible values are "database" and "local". "database" will look into the database and find all the unique IDs stored there. "local" will only look to the "/data/" directory to find all the taken IDs. The app will then generate a new random ID that does not exist in the retrieved list of existing IDs.
- next_story_from : Controls where the next story index is referenced from. Possible values are "database" and "local". "database" will look into the database to find the next story the user will see by counting all the unique entries of 'tasktypedone'. "local" will look into the user's local demographic info record in '/data/' and read the 'next_story_index' entry.
- timestamp_timezone : The time zone that timestamps are collected in. Recommend 'UTC', if this parameter is not set, the app will default to 'UTC'.
- minimum_topics : The minimum amount of topics that the subject must select. Replaces 'min_stories_to_choose' from version 30.
- questions_per_story : The number of questions selected to show the user PER STORY.
- ignore_legacy_story_data : How the app handles story data ('pref_stories' and 'story_order') from legacy app versions. If this is set to 0, the app will NOT ignore legacy story data from returning users and continue the remainder of the users' sessions with only 'approach_avoid' task types, keeping their preferred

stories and previously calculated story order. If set to 1, the app will discard this information and ask the user to restart their sessions to add the new task types to their story data.

- randomise_relation_levels : FOR SOCIAL TASK ONLY: Whether to randomly choose a random relationship keyword and replace it into the text of the social task stories. If set to 1, a random relationship keyword will be picked from the list under the 'relation_levels' list parameter and replaced into every snippet of text throughout the entire story.
- relation_levels : FOR SOCIAL TASK ONLY: The words to look for and replace in story text. The app will randomly sample a word and replace it throughout the entire story text.
- relation_level_stories : FOR SOCIAL TASK ONLY: Stories where replacing the relationship level can be done
- validate_stories : Validates that the stories described in the 'Human DM Topics' relationship table are actually contained in '../stories/task_types/'. Stories that don't exist in that directory will be deleted from the pool of stories that can be selected for each subject. Setting this parameter to 1 can cause errors if the validated story pool ends up being smaller than the sample size for each task type (see below). If this is the case, check that there are enough stories for each task type in '../stories/task_types/'. Conversely, not validating the stories increases the odds of trying to access a story that does not exist, causing a different error.
- The next 7 parameters describe how many stories are to be selected (sample size) per task type at each user's first session:
    - approach_avoid
    - benefit_benefit
    - cost_cost
    - moral
    - multi_choice
    - probability
    - social

These settings may be tweaked further later, but keep in mind that they must be tweaked **before** running the HMDA.

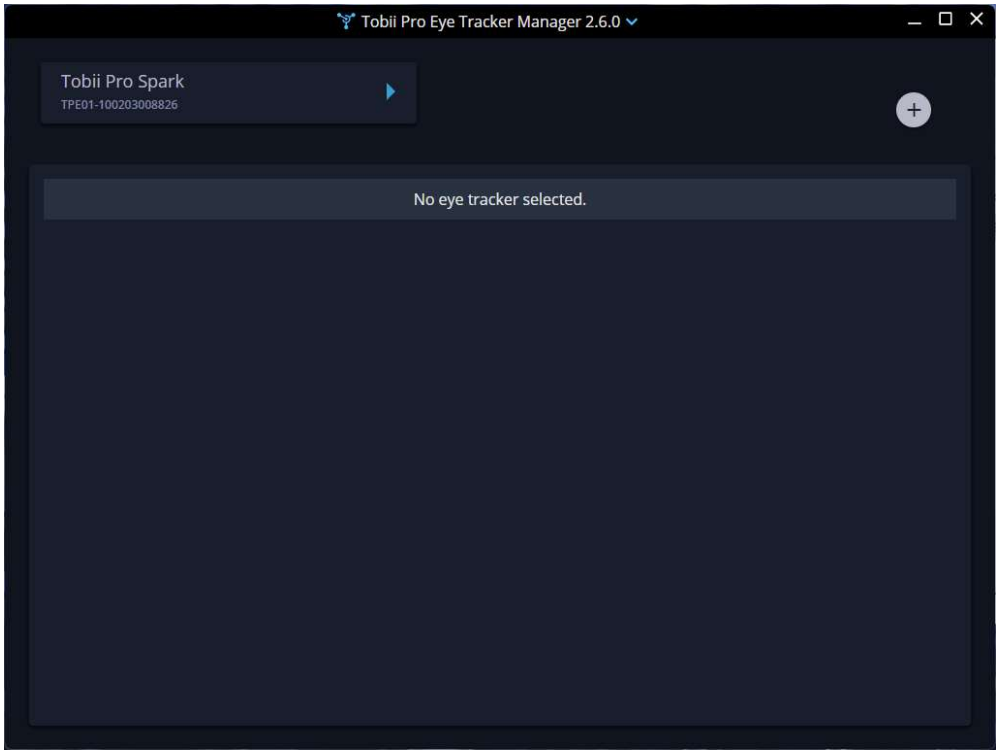
Once you have made the changes needed, save the file and exit the text editor.

Installing the eye tracker

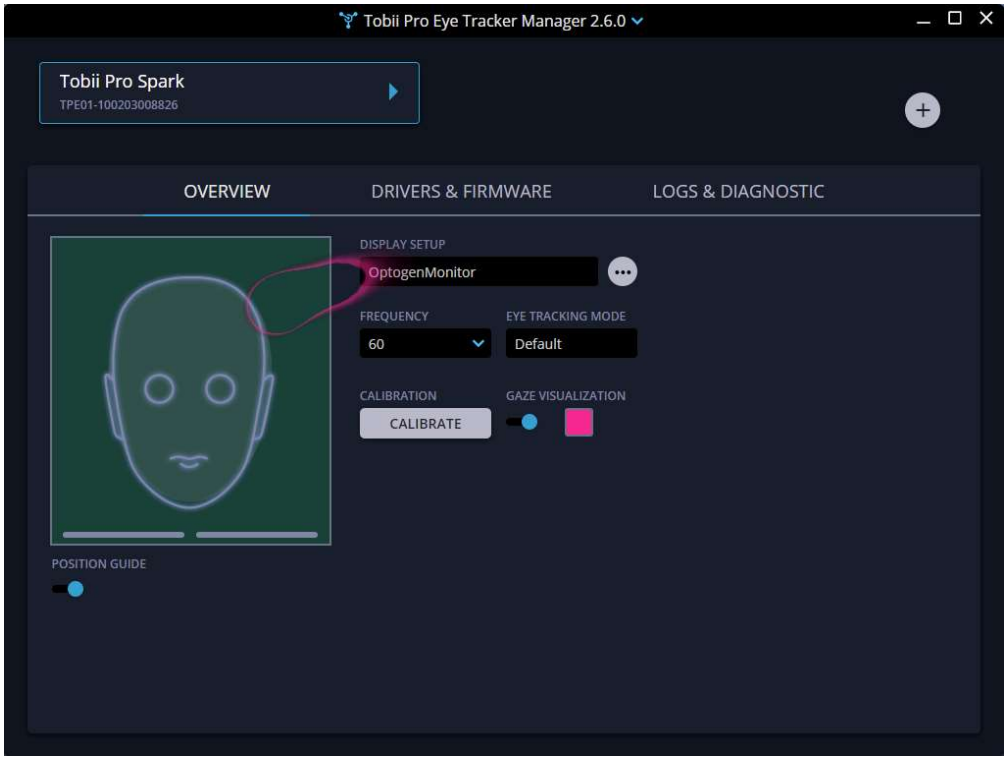
Hardware and software setup

The eye tracker setup requires one hardware device, and three different software. Refer to the "Resource links" section of this guide. Download and install items 5, 6, and 7 first, then mount your tracking device following the manufacturer's guidelines (item 8). Connect the device to any available USB port, preferably one directly on the computer tower and not through a USB hub, as this could cause issues with data throughput.

Test the eye tracker by opening the eye tracker manager (item 5). Your eye tracking device should show up in the list of available devices at the top.



Click your device and the user interface (UI) will expand. Toggle both 'gaze visualisation' and 'position guide', then look around and see if the gaze visualiser (in pink in the example below) moves to where you are looking. If it doesn't, click the 'Calibrate' button and follow the calibration steps that show on screen.



Tweaking app settings

Navigate to the directory where you extracted the decision-making app files to. Navigate to the ‘bin’ folder and open ‘settings.ini’ with your text editor of choice (Notepad, Notepad++, Atom, etc…). Make the following changes under the ‘eye_tracker’ section:

- `manager_install_path=C:\Users\{YOUR_WINDOWS_USERNAME}\AppData\Local\Programs\TobiiProEyeTrackerManager\TobiiProEyeTrackerManager.exe (this may differ depending on where you installed the Eye Tracker Manager program)`
- `subscriptions=[‘gaze’,’position’] (‘openness’ is also valid, however it may not be supported for your device)`
- `eyetracker_index=0 (if you only have one eye tracker connected, otherwise specify your device’s index)`
- `use_eyetracker=1`

Save the file and exit the text editor.

```
[eye_tracker]
; Where in the local system the Tobii Eye Tracker Manager is installed
manager_install_path=C:\Users\{USER}\AppData\Local\Programs\TobiiProEyeTrackerManager\TobiiProEyeTrackerManager.exe
; Which eye tracker data streams to subsribe to, i.e. what eye tracker data to collect. Write these values separated by a comma and enclosed
; in square brackets, for example ['gaze','openness','position']. Only choose out of the items provided in the example.
subscriptions=['gaze','position']
; The index number of the eye tracker to connect to. Likely will not change unless more than one tracker is connected.
eyetracker_index=0
; Whether to use the eye tracker. 1 means yes, 0 means no.
use_eyetracker=1
```

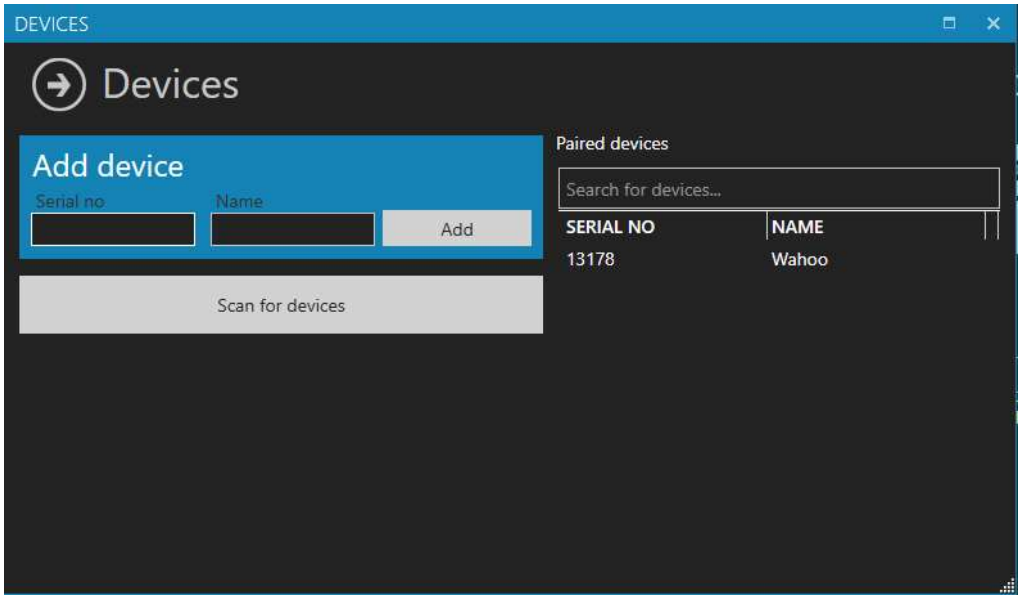
Installing a heart rate monitor

Throughout this section, refer to the "Resource links" section of this guide. If you are following our methods to collect heart rate data, you'll need to install the ANTUSB driver for Windows. You may follow this guide to do so.

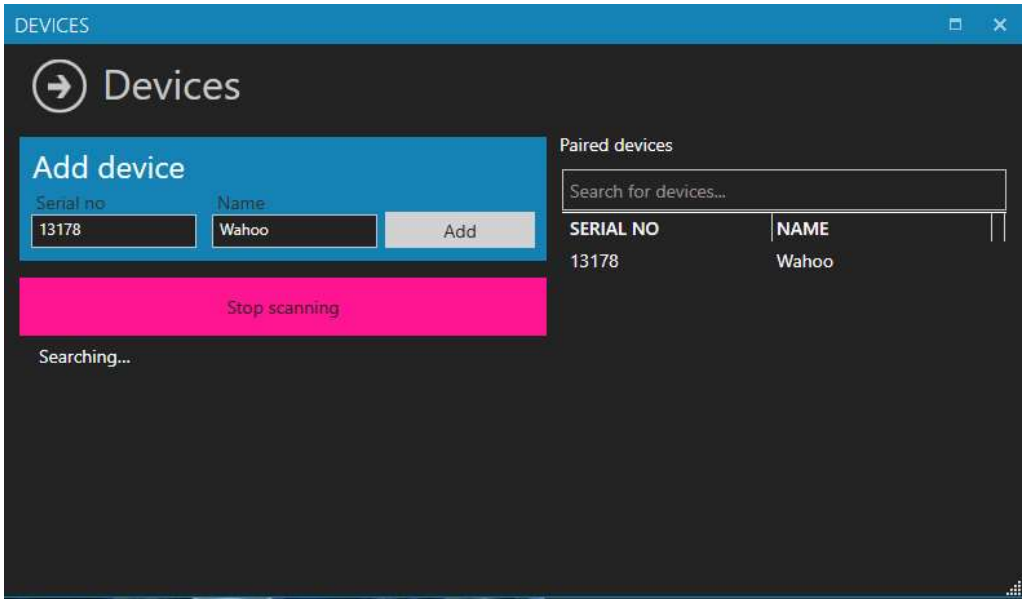
If using an external program like Pulse Monitor

Make sure your program of choice is compatible with ANT+ devices. After installing your program, be sure to set up your device in whichever way the program needs. Here, we provide an example using Pulse Monitor (item 9).

Open Pulse Monitor and click the 'Devices' button.



Connect an ANT+ USB antenna (item 3) and power up your heart rate monitor (item 2). Enter the heart rate monitor's serial number and name in the corresponding dialogue boxes, then click the 'Scan for devices' button and wait for your device to appear. Once your device appears, click the 'Add' button. Close this dialogue box when you're done.



Next navigate to the directory where you extracted the decision-making app files to. Navigate to the ‘bin’ folder and open ‘settings.ini’ with your text editor of choice (Notepad, Notepad++, Atom, etc…). Change the ‘use_external_app’ parameter under the ‘hr_tracker’ section to 1 and copy your heart rate monitoring app’s install directory path to the ‘external_app_install_path’ parameter (see example below for suggested format). Finally, set ‘use_hrtracker’ to 1.

```
[hr_tracker]
; Whether to use the external application to collect heart rate data. If set to 1, the app will attempt to run the program linked by the
; 'external_app_install_path' parameter as a subprocess to collect heart rate data. If set to 0, a thread will be run in parallel with the
; app that will directly collect hr data.
use_external_app=1
; Where in the local system Pulse Monitor is installed
external_app_install_path=D:\Program Files\PulseMonitor
; The heart rate monitor's device index/ID. This will likely not change unless there are more than one trackers connected.
hrtracker_index=0
; Whether to use the heart rate tracker. 1 means yes, 0 means no.
use_hrtracker=0
; Whether to emulate a heart rate device for the hr monitor thread. This is mainly useful for development and data captured from the emulator
; will not be sent to the database. Use this if you're testing the app and don't have a heart rate monitor device connected to the computer.
emulate_device=1
; Whether to run the heart rate monitor thread as a daemon. If set to 1, hr monitor thread will run alongside app as a daemon.
; This ensures that the thread will exit along with the app, however, this could cause problems if the thread is not manually stopped.
run_thread_as_daemon=1
; Toggles verbose output of the hr monitor thread. This has little impact on functionality, but is useful when debugging the heart rate monitor.
; Setting this to 1 will allow the hr monitor thread to output to the same console window the app is running on.
verbose=1
; Whether to test the heart rate monitor when the app starts
test_on_startup=1
```

Example is ‘settings.ini’ open in Notepad++ with Dark Mode enabled.

This will signal the HDMA to open Pulse Monitor as a subprocess to collect heart rate data. Keep in mind that since this program runs asynchronously from the HDMA, the data produced by this program will not be directly uploaded to your database, rather, it must be exported, saved, proprocessed, and then uploaded.

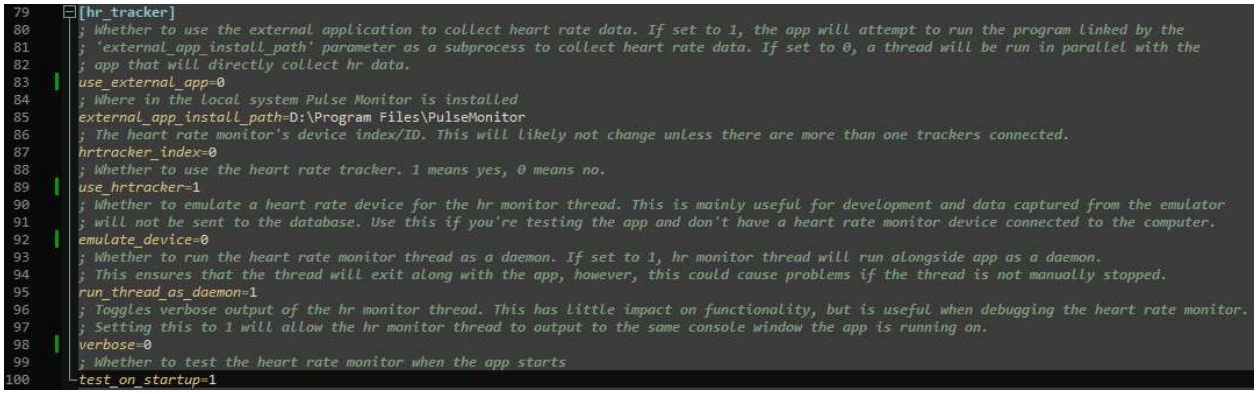
If using ‘built-in’ heart rate monitoring

Note: This feature is currently experimental and may not function as expected.

Navigate to the directory where you extracted the decision-making app files to. Navigate to the ‘bin’ folder and open ‘settings.ini’ with your text editor of choice (Notepad, Notepad++, Atom, etc…). Make the following changes under the ‘hr_tracker’ section:

- `use_external_app=0`
- `use_hrtracker=1`
- `hrtracker_index=0 (if you only have one heart rate monitor connected, otherwise specify your device’s index)`
- `emulate_device=0`
- `run_thread_as_daemon=1 (optional)`
- `verbose=0 (optional, can be set to 1 if you want the heart rate monitor output displayed on the command line)`
- `test_on_startup=1 (optional)`

Save the file and exit the text editor. These settings will ensure that the heart rate monitor data is directly collected by the HDMA and stored in the database. No additional steps are required.



App usage

Materials required

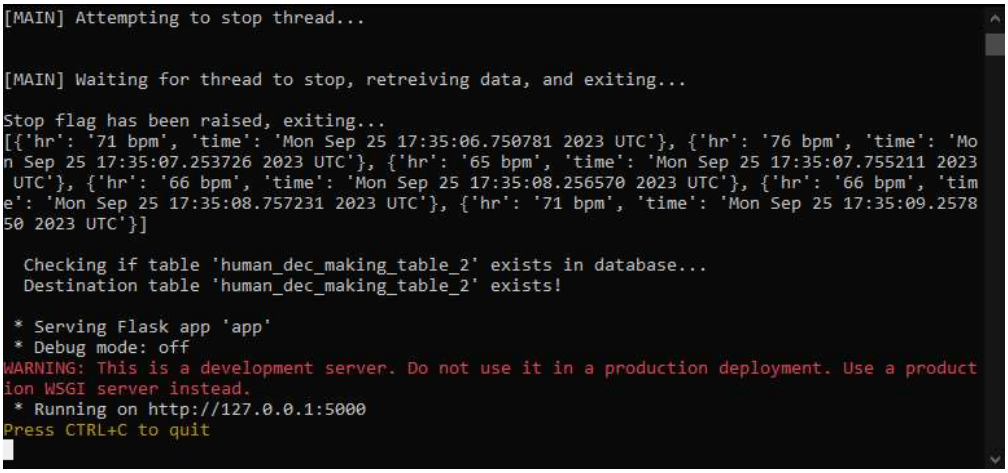
- HUMANS App (see GitHub repository under "Resource Links")
- ANT+ Heart Rate (HR) monitor (example)
- ANT+ USB Antenna (example)
- Alcohol wipes
- Keyboard and mouse
- *If subject requires:* Eye correction (for example eyeglasses, reading glasses, blue light filtering glasses, or contact lenses, if applicable)
- Eye tracker (Tobii Pro Spark)
- Tobii Eye Tracker Manager (software)
- Tobii Pro SDK (software)
- Tobii Pro Spark Runtime Driver (software)
- Pulse Monitor* (software)

* The use of Pulse Monitor is a temporary solution to collecting heart rate data. This software will be optional for later versions of the app. If you are running version 32.1 and later, no setup is needed for the heart rate monitor if the 'use_external_app' option is set to 0.

Running HUMANS

Double click on the batch script named **'startup.bat'** in the DM app folder.

This will open a command line window, which will remain open as long as the app is running**. Do not close this window!!** This window is the program that powers the web app, and will display information essential for troubleshooting**.**



Subject Preparation

Before beginning a session, prepare the subject for the session by doing the following:

1. Allow the subject to choose whether they would like to wear the heart rate monitor. If they accept to wear the tracker, allow them to put on the tracker wherever they are comfortable doing so (for example, the restroom). If they do not accept to wear the tracker, do not force them, simply take note of this so that the absence of HR data is noted and move on.
2. Ask the participant for any relevant information you may need in private, make sure to store this information in a secure location. This may include:
    a. Consent forms
    b. Name to be associated with ID numbers for bookkeeping purposes
    c. Eyesight conditions that may require corrective lenses (see ‘Eye Tracking Study Considerations’)
    d. Use of pace-makers or health conditions that could complicate collection of heart rate data.
    e. Whether they choose to not wear a heart rate monitor.
3. If this is the participant’s first session, give them a run-down of what they can expect from the recording session. Mention the eye tracker, what it does, and what it does not collect (the subject will notice the tracker turn on and off throughout the session, this could be intimidating to some participants!).
    a. The eye tracker is not programmed to capture any images during the trial. Only gaze (where the subject is looking on the screen), pupil dilation, and user position in front of the tracker are collected and recorded.
    b. Make sure that they know that they may choose not to disclose certain demographic information that the app asks for, and that their data will only ever be identified by a randomly-generated ID number and not by their name.
4. **Ensure that the participant is wearing eye correction if they require it!** Eye correction such as contacts and eyeglasses must be worn during the eye tracker’s calibration and all the way into the end of the session.

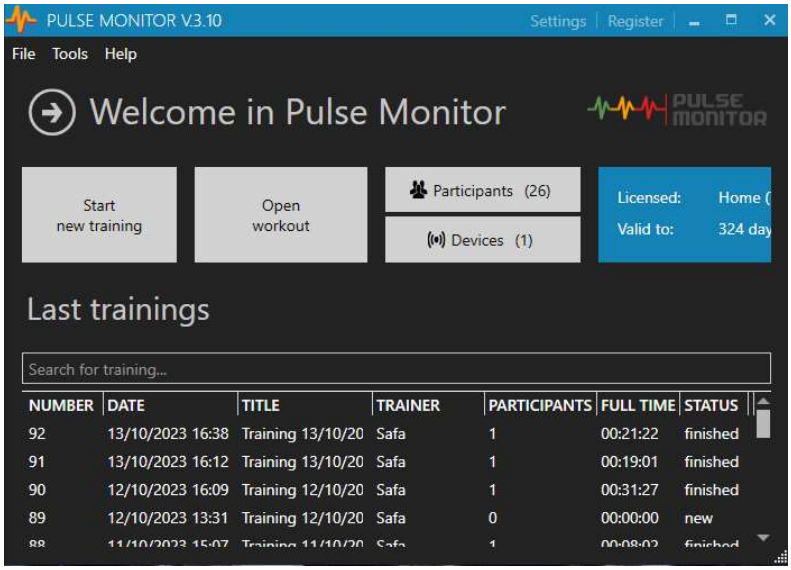
5. Allow the participant to get comfortable in their seat. Any adjustments to chair height or positioning can be done during eye tracker calibration.
6. Allow the participant to answer the questions in the first screen of the app. Ask them to let you know before continuing to the next screen.

Eye Tracker Calibration

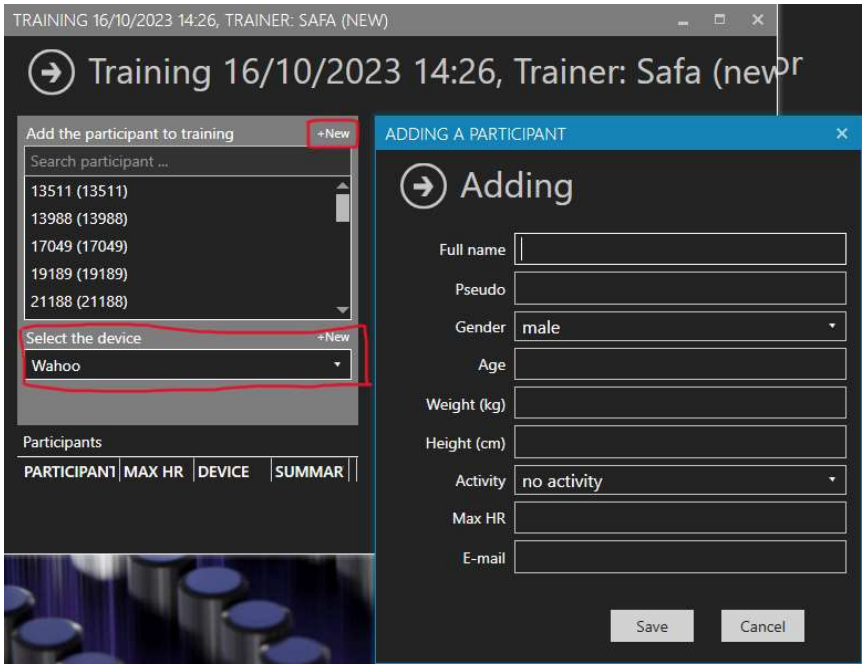
Every single subject should go through the calibration process at least once before starting the trials. Remember that eye corrections such as eye glasses or contact lenses must be worn during this process for calibration to be accurate!

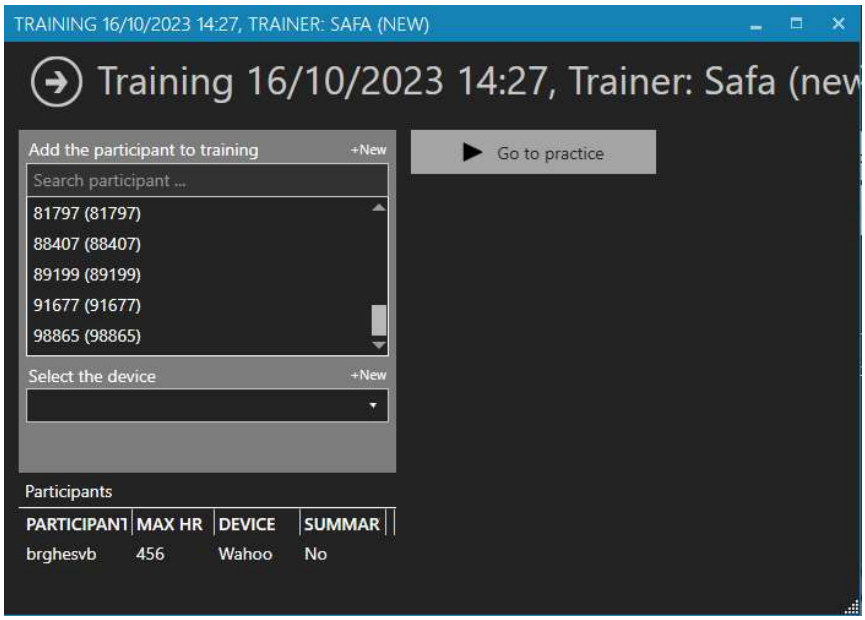
7. Upon continuing to the second screen, one or two* windows will be opened. Bring only the **Tobii Eye Tracker Manager** window to focus. Click on the button that says 'Pro Spark' on the top of this window.
    a. **If Tobii Eye Tracker Manager does not open, stop and check that the eye tracker is connected to the computer and that the 'use_eyetracker' option is set to 1 in '\bin\settings.ini'.**
    b. The app will ask you (through the command window) if you wish to continue without the eye tracker. Type 'Y' and press Enter on your keyboard to continue with the eye tracker disabled; or type 'N' to exit the app and troubleshoot.
    c. The use of Pulse Monitor is a temporary solution to collecting heart rate data. This software will not be needed for later versions of the app.
8. Ask the participant to get comfortable in their seat and click the 'Calibrate' button. Instructions will show up on-screen and calibration will begin.
9. After calibration is done, **close** ('X' out of) the **Tobii Eye Tracker Manager** window and the app will continue.
10. Allow the participant to click through the next screen ('Are you a new participant?').
11. If the subject is wearing the HR monitor and/or 'use_eyetracker' in '\bin\settings.ini' is set to 1, continue to the Heart Rate Monitor Setup section, otherwise skip ahead.

Heart Rate Monitor Setup (optional)



12. If the subject is a new participant, register them into the **Pulse Monitor** software by clicking the ‘Participants’ button. Click on ‘Add Participant’ and allow the subject to input their information; ask them to enter their generated ID number (from the web browser) under ‘name’ and ‘pseudo’. Click ‘Save’ when done.
    a. None of this information is used for our study, we only input this information because otherwise the Pulse Monitor software will not allow us to collect HR data.
13. Click on ‘Start New Training’. Add the participant’s ID number to the list of participants in the ‘workout’ and choose ‘Wahoo’ from the ‘Device’ dropdown menu. Finally, click ‘Add’ and ‘Go to practice’.





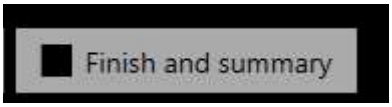
14. Start the ‘workout’, click the small grey button on the bottom right of the screen, and set this window aside.

Final setup

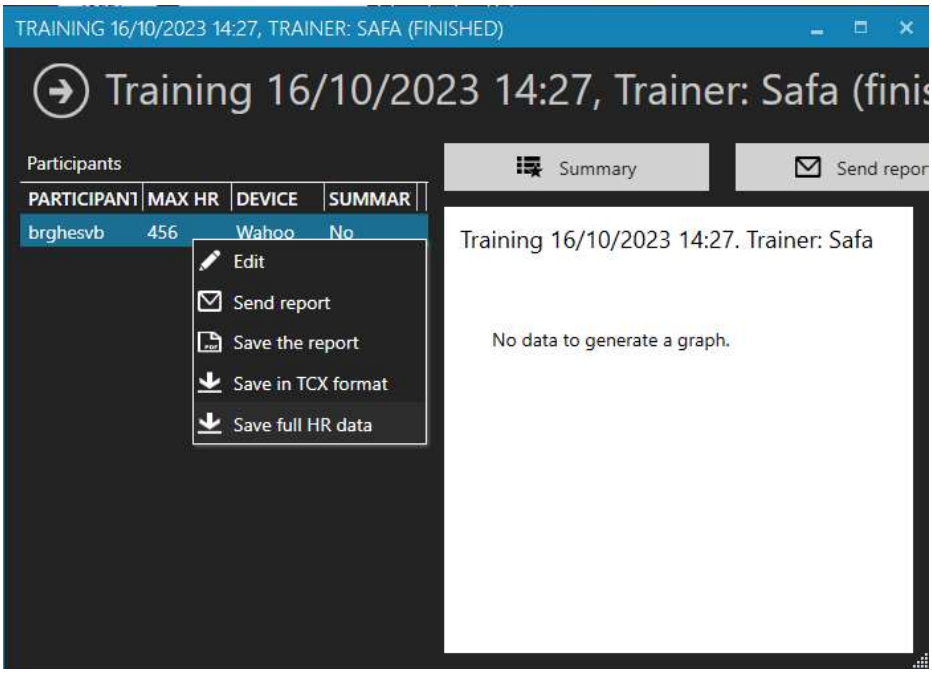
15. Take note of the participant’s ID number and ask them to do the same. Allow them to continue filling out the questionnaire.
16. The session will officially start as soon as the participant submits this questionnaire.

Ending a session

17. Once the subject sees the ‘You are done for today!’ screen, this will indicate the end of the session. Ask them to step away from the computer at this time.
18. Write down any notes about the session you wish to let the data analysts know. Such notes may include:
    a. Interruptions due to environmental factors
    b. Any comments that the participant may have had
    c. Catastrophic app failures
    d. Having to split the session into two
19. Click on the green button that reads ‘Finished!’.
20. Ask the subject to remove the heart rate monitor. If you are using external software (such as Pulse Monitor) to collect heart rate data, do the following:
    a. Stop the ‘workout’ on the Pulse Monitor software.



    b. Export the subject’s data by right clicking on their session summary and clicking ‘Save full HR data’.



    c. Save this file under the directory named after the participant’s ID number in ‘\data\’ with the following format:
    ‘{participant ID}_{mm}-{dd}-{yyyy}_{hh}-{mm}-{ss}.csv’
21. Wipe down the heart rate monitor with alcohol wipes for the next participant to wear.
22. If ‘enable_consecutive_users’ is set to 1 in ‘\bin\settings.ini’, the app will be ready to take the next subject. Otherwise, close the command window, run **‘startup.bat’**, and refresh the app in the browser window.

Troubleshooting

As mentioned before, a command window will open whenever you run the app. This command window will be your best friend during troubleshooting! The program that powers the app will display information such as…

1. The user's retrieved or generated ID number.
2. The way in which the app samples stories to add to the user's **story order**.
3. The user's story order.
4. The current **task type**, story number, and story index in the **story order** list.
5. The current story **topic**.

This information is displayed to help you identify where issues come from whenever they arise. The command window is also where the program will dump a description of the issue it ran into. These messages can be cryptic and might seem difficult to understand, however we'll cover some of the most common problems you may encounter, where they're likely to come from, and how to fix them.

Story data format

In this section, we refer to '**story data**' as any content that was written for each specific scenario that the app displays. This content is written by hand, then run through a script that breaks each scenario down into four different files; *context.txt, pref_cost.txt, pref_reward.txt, and questions.txt*.

Each of these files has to be in the correct format for the app to recognise what is in the file. Some of the most common problems are caused by the formatting of these files. The following is the expected format for each of the files listed above:

`context.txt`

`Each story context should be written in one paragraph, meaning no line breaks such as new paragraphs. Punctuation is allowed, but no unicode characters are allowed.`

`pref_cost.txt and pref_reward.txt`

1) `The content of these files must be in list form`
2) `Each item must be preceded by a number, an unpaired right parenthesis, and a space`
3) `Do not add an indent before the number`
4) `No new lines are allowed within one list item`

5) Punctuation is allowed but unicode characters are not
6) Do not add empty line in-between items, before, or after the list

questions.txt

```
Question items must be in list form? (R1, C1)
Each element in this list is un-numbered, un-bulleted, and starts in a new line, do not add spaces at the beginning? (R1, C2)
Questions must begin with a capitalised word and end in a question mark, period, or exclamation point. (R1, C3)
Each item must also end with a code which represents a reward and cost level before the new line, enclosed in left and right parentheses? (R1, C4)
The cost and reward level codes do not need to be separated, but should contain either a capital R or C, followed by a number! (R1, C5)
Parenthesis are allowed in the middle of a question (the app will simply display them as they are), but be careful to also include the code at the end? (R1, C6)
The code follows a certain pattern, but list items do not have to be ordered by code? (R2, C1)
Breaking sentences up is allowed. Each sentence in the questions will be stitched together and shown in a single line? (R2, C2)
```

Problems with story data

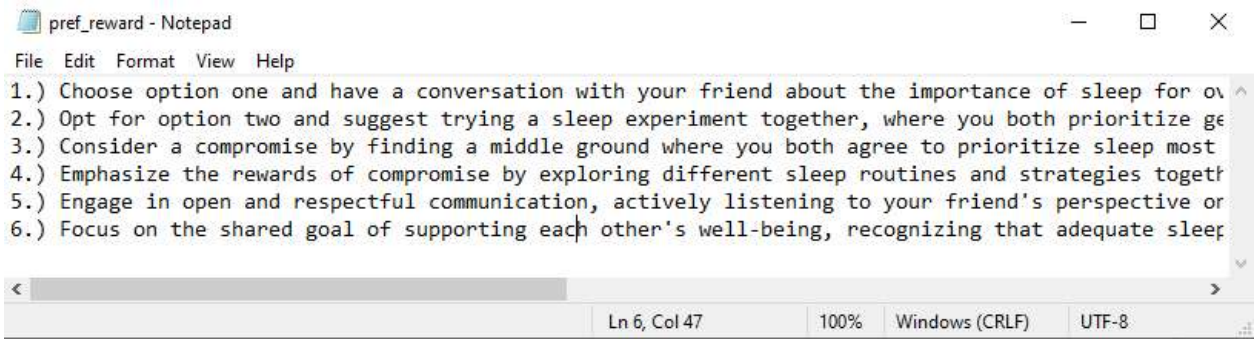
Problems with **story data** might be the most common problem you'll see. They will manifest themselves as one of many types of errors ranging from indexing errors to type casting errors. Here are few examples and how to fix them:

ValueError: invalid literal for int() with base 10: '1.'

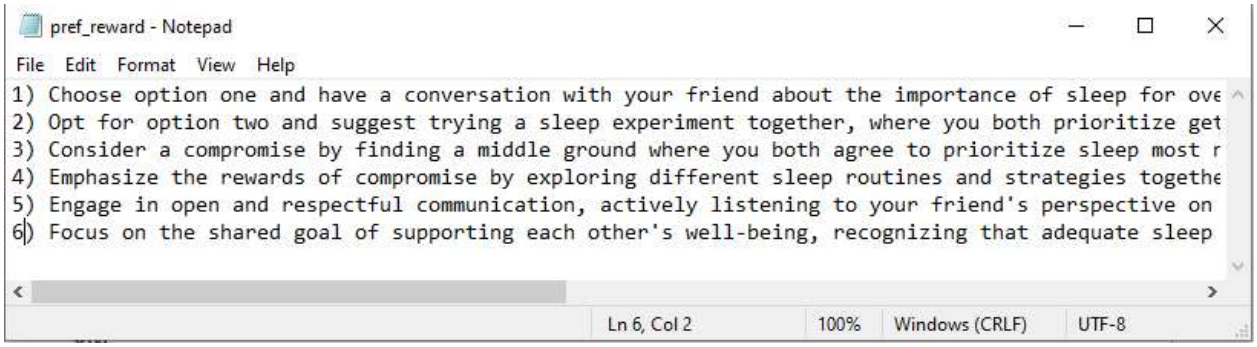
```
127.0.0.1 - - [12/Sep/2023 15:17:30] "GET /static/main.css HTTP/1.1" 304 -

Current story number: 2.
Story: /social/story_18.

127.0.0.1 - - [12/Sep/2023 15:18:10] "GET /prefs/reward HTTP/1.1" 200 -
127.0.0.1 - - [12/Sep/2023 15:18:10] "GET /static/main.css HTTP/1.1" 304 -

Current story number: 2.
Story: /social/story_18.

[2023-09-12 15:18:22,230] ERROR in app: Exception on /prefs/reward [GET]
Traceback (most recent call last):
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 2528, in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1825, in full_disp
atch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1823, in full_disp
atch_request
    rv = self.dispatch_request()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1799, in dispatch_
request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\Raquel\Desktop\Decision Making App\dec-making-app-v32\app.py", line 1102, in rank_prefs
    opt_num = int(line[0])
ValueError: invalid literal for int() with base 10: '1.'
127.0.0.1 - - [12/Sep/2023 15:18:22] "GET /prefs/reward HTTP/1.1" 500 -
```

This error tends to happen because 'pref_reward.txt' or 'pref_cost.txt' was formatted incorrectly. In this case, the 'value error' line gives us a clue on what might be wrong. The code is trying to convert the string '1.' into a number, but it can't because there is a period in the way.

To solve this, notice the story that the user encountered the error in; this is indicated in the command line window, where it reads 'Story:'. In the example above, the culprit is '/social/story_18'. For this example, we would need to navigate to '{path_to_DM_app}\stories\task_types\social\story_18\pref_reward.txt', where we encounter…



Sure enough the problem was caused by the periods after the numbers. Remove them and save…

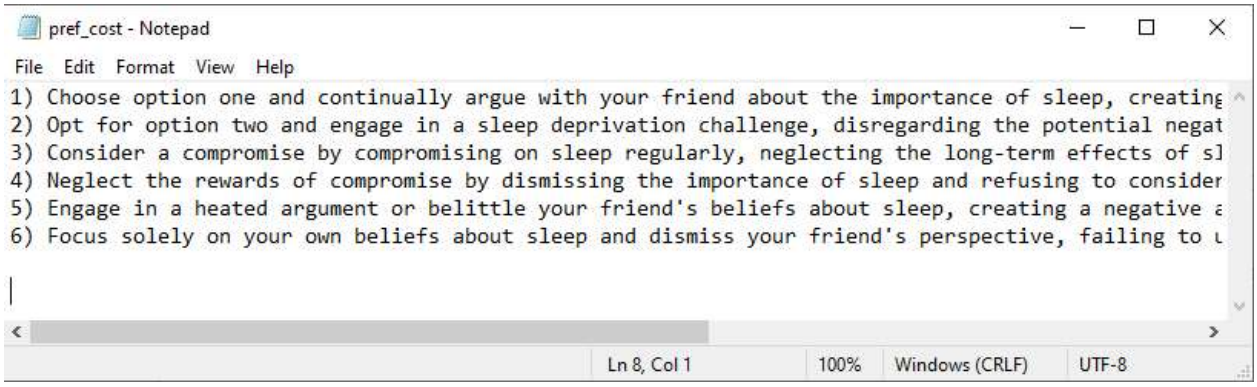


…then refresh the page the user left off on.

Empty preference options

Although this error may not cause the app to halt, it may be very confusing for the participant to encounter these glitches and can negatively affect the data.

Check the command line window to figure out what story this happened in. In our example, the culprit is ‘/social/story_18’. For this example, we would need to navigate to ‘{path_to_DM_app}\stories\task_types\social\story_18\pref_reward.txt’, where we encounter…



The problem is the two empty lines at the end of the list. To solve this problem, delete these two lines so that there are no empty lines at the end of the list, save the file, then refresh.

IndexError: list index out of range

```
Current story number: 2.
Story: /approach_avoid/story_15.


Current story: 2.
Story: /approach_avoid/story_15.
[2023-09-12 15:47:15,478] ERROR in app: Exception on /refresh [GET]
Traceback (most recent call last):
  File "C:\Users\FriedmanLab Optogen\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 2190,
 in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\FriedmanLab Optogen\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1486,
 in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\FriedmanLab Optogen\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1484,
 in full_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\FriedmanLab Optogen\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1469,
 in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\FriedmanLab Optogen\Desktop\Human Decision-Making App\dec-making-app-v32\app.py", line 1118, in context
_refresh
    relevant_questions = choose_questions()
  File "C:\Users\FriedmanLab Optogen\Desktop\Human Decision-Making App\dec-making-app-v32\app.py", line 501, in choose_q
uestions
    linelist[1] = re.findall('\(.*?\)', l)[-1]  # Find everything in parenthesis and take the last element.
IndexError: list index out of range
127.0.0.1 - - [12/Sep/2023 15:47:15] "GET /refresh HTTP/1.1" 500 -
```

This error can be caused by many things. It essentially means that the code tries to access an item that does not exist within a list of variables. This could mean that the list is incomplete or empty. In the example above, the line above the 'IndexError' gives us a clue of what might have happened.

When choosing the questions that will be shown to the user for each trial, the numbers in the parentheses are used to select only the most relevant questions for the user. The code looks for anything that is enclosed in parentheses on each line of the text and separates it from the rest. If the code is not enclosed in parenthesis, the program will not find it and thus cannot continue.

Make sure all parentheses are closed, save the file, and refresh the page.

Missing question during trials

This error does not cause an internal server error, however, it may be confusing for the user and can confound the data. The error is caused because the question was not found in the ‘questions.txt’ file.

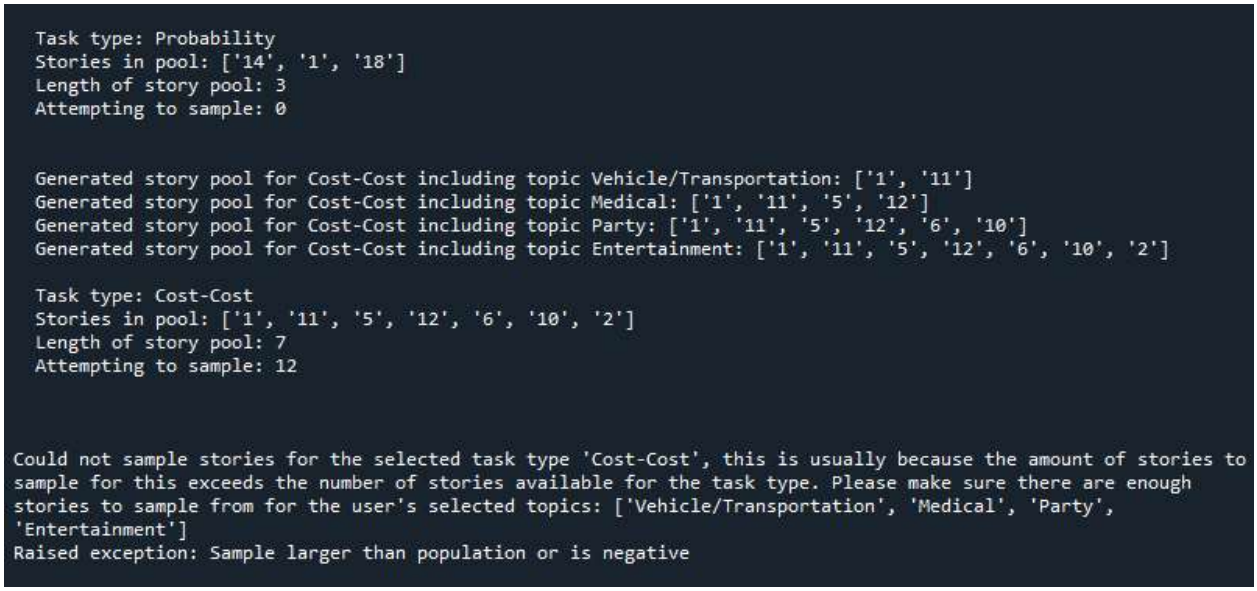
To find the question, the program loads the ‘questions.txt’ file and looks through each line of the file searching for sentences. It looks for a capitalised word and continues scanning the sentence until it finds some kind of punctuation (a period, a question mark, or an exclamation mark). If one or both of these conditions isn’t met, then the program will not be able to recognise a sentence as a question and won’t have anything to display to the user.

To solve this, make note of which story caused the issue, find this by looking at the command line and find the most recent line that starts with ‘Story:’. Navigate to ‘{path_to_DM_app}\stories\task_types\{target_story}\’, open ‘questions.txt’, and check that all sentences start with a capitalised word and end with punctuation. Save the file, **but don’t refresh the page as this may cause data loss.**

After saving, you may proceed in one of two ways; please refer to the section ‘Recovering from catastrophic failures’.

Problems with story sampling

`Raised exception: Sample larger than population or is negative`



This error can present itself when a particular task type does not have the amount of stories the ‘settings.ini’ file is asking the app to sample.

The app goes through each topic that the user selected, then for each task type, creates a story pool by adding the stories that fall into both those categories. Once the story pool is created, the app randomly samples a certain amount of stories for each task type. The amount of stories that get sampled is the number indicated by the approach_avoid, benefit_benefit, cost_cost, moral, multi_choice, probability, and social (lines 51 - 57) settings in 'settings.ini'.

To solve this issue, first identify which task type is causing the problem by looking at what was printed to the command window before the error occurred. The command window will break down the process of sampling stories. The last 'Task type:' line will tell you which task type caused the problem, and the different topics where stories were sampled from will be indicated in the preceding lines. If the length of the resulting story pool is lesser than the amount of stories that the app attempts to sample, this error will be raised.

For the example above, we can see that the task type that caused the issue was 'Cost-Cost'. If we look at the breakdown of how the story pool was generated, we can see that the generated pool ended up being smaller than the amount to be sampled (7 vs. 12).

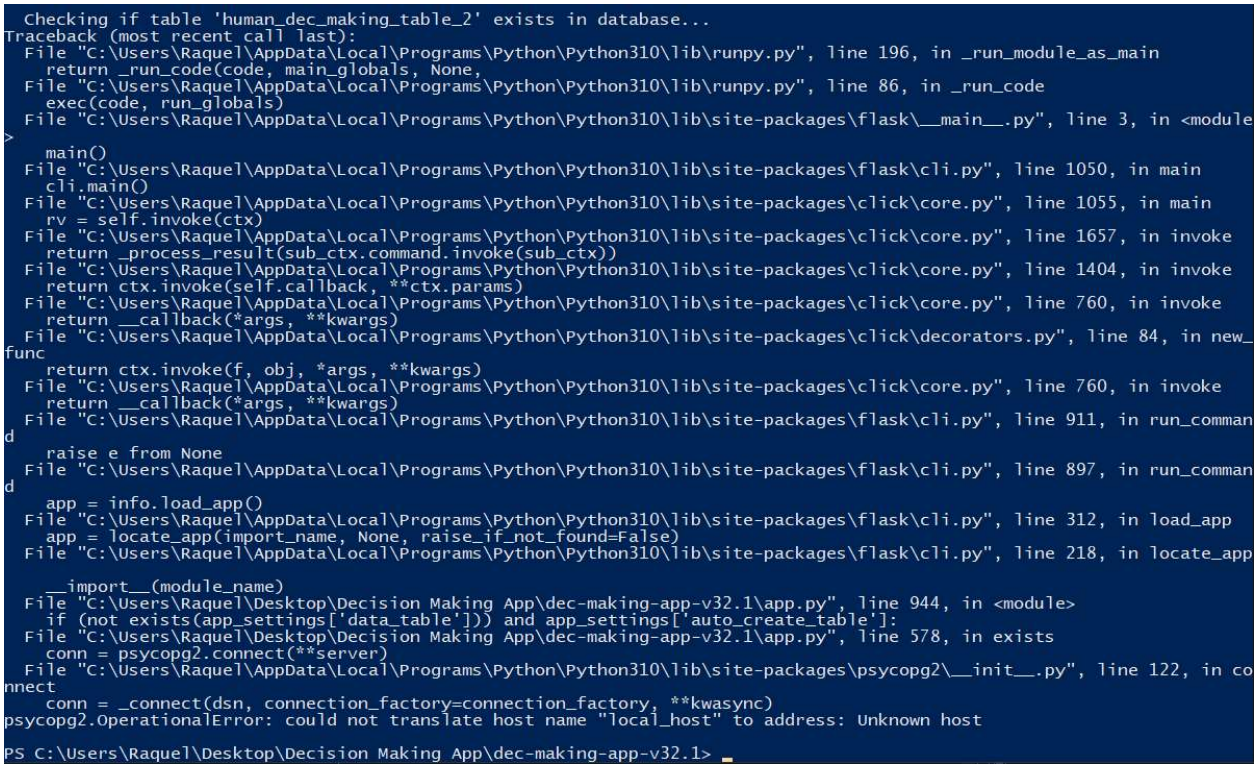
Solving this problem can be tricky, until more content will be added to the app, here is a temporary solution…

1. Stop the app (close the command window). The user's data will likely need to be discarded.
2. The amount of stories that get sampled is the number indicated by the approach_avoid, benefit_benefit, cost_cost, moral, multi_choice, probability, and social (lines 51 - 57) settings in 'settings.ini'. Decrease the appropriate values to however large the story pool ended up being.
3. Restart the app.
4. Repeat the user setup process.

Problems with database

Problems with the database can arise when the host, database, port, user, or password parameters are not set correctly in 'settings.bin'. Here are a few problems that you may encounter:

`psycopg2.OperationalError: could not translate host name "local_host" to address: Unknown host`



```
  Checking if table 'human_dec_making_table_2' exists in database...
Traceback (most recent call last):
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\__main__.py", line 3, in <module
>
    main()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\cli.py", line 1050, in main
    cli.main()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\click\core.py", line 1055, in main
    rv = self.invoke(ctx)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\click\core.py", line 1657, in invoke
    return _process_result(sub_ctx.command.invoke(sub_ctx))
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\click\core.py", line 1404, in invoke
    return ctx.invoke(self.callback, **ctx.params)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\click\core.py", line 760, in invoke
    return __callback(*args, **kwargs)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\click\decorators.py", line 84, in new_
func
    return ctx.invoke(f, obj, *args, **kwargs)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\click\core.py", line 760, in invoke
    return __callback(*args, **kwargs)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\cli.py", line 911, in run_comman
d
    raise e from None
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\cli.py", line 897, in run_comman
d
    app = info.load_app()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\cli.py", line 312, in load_app
    app = locate_app(import_name, None, raise_if_not_found=False)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\cli.py", line 218, in locate_app

    __import__(module_name)
  File "C:\Users\Raquel\Desktop\Decision Making App\dec-making-app-v32.1\app.py", line 944, in <module>
    if (not exists(app_settings['data_table'])) and app_settings['auto_create_table']:
  File "C:\Users\Raquel\Desktop\Decision Making App\dec-making-app-v32.1\app.py", line 578, in exists
    conn = psycopg2.connect(**server)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\psycopg2\__init__.py", line 122, in co
nnect
    conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
psycopg2.OperationalError: could not translate host name "local_host" to address: Unknown host

PS C:\Users\Raquel\Desktop\Decision Making App\dec-making-app-v32.1>
```

This error arises when the ‘host’ parameter is set to something that cannot be recognised by Python. In this case, ‘local_host’ was entered when the correct term would be ‘localhost’.

Problems with missing peripherals

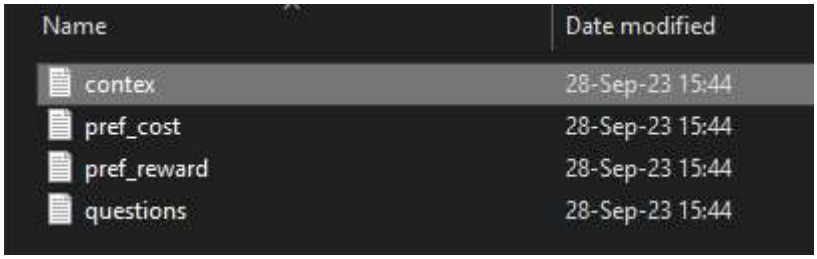
Problems with missing or misnamed files/directories

Missing file problems tend to arise when the code tries to access some external file to read or write, but does not find anything from the file path that was specified. Thus, the file structure of the app is extremely important; all files and folders must have the correct names, otherwise these errors may arise.

FileNotFoundError: [Errno 2] No such file or directory

```
Starting story /moral/story_24
Current topic: Trip (5)

127.0.0.1 - - [19/Oct/2023 15:57:54] "GET /story_num_overall HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2023 15:57:54] "GET /static/main.css HTTP/1.1" 304 -

Current story number: 11.
Story: /moral/story_24.

[2023-10-19 15:58:09,710] ERROR in app: Exception on /context [GET]
Traceback (most recent call last):
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 2528, in w
sgi_app
    response = self.full_dispatch_request()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1825, in f
ull_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1823, in f
ull_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1799, in d
ispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\Raquel\Desktop\Decision Making App\dec-making-app-v32.2\app.py", line 1261, in context
    txt = open(path).read().replace("'", "'")
FileNotFoundError: [Errno 2] No such file or directory: 'stories/task_types/moral/story_24/context.txt'
127.0.0.1 - - [19/Oct/2023 15:58:09] "GET /context HTTP/1.1" 500 -
```

This error has to do with a particular file not being accessible because it either does not exist, or the name of the file is not what is expected. In the example above, the code tried to find a context file using the path ‘stories/task_types/moral/story_24/context.txt’. If we navigate to this directory, we will find that the context file is misnamed.



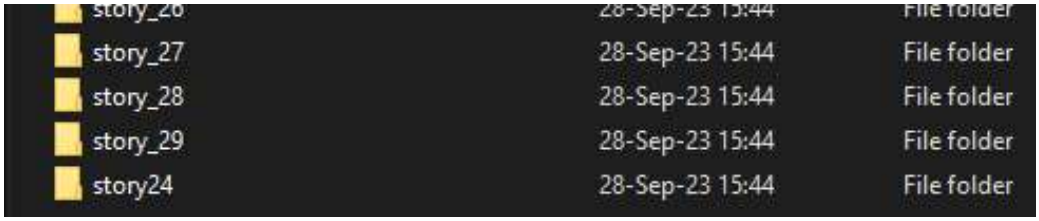
To fix this, simply rename the file to what the code expects. Most of the time, one can recover from this error by refreshing the page, but in some cases, a full restart may be necessary.

KeyError: ‘topic’

```
Calling eye tracker manager to initiate calibration!

127.0.0.1 - - [19/Oct/2023 15:55:34] "GET /welcome HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2023 15:55:34] "GET /static/main.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Oct/2023 15:55:36] "GET /not_new HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2023 15:55:36] "GET /static/main.css HTTP/1.1" 304 -

Retrieved user ID: 32083.

Retrieved starting story index: 10.

127.0.0.1 - - [19/Oct/2023 15:55:38] "POST /not_new HTTP/1.1" 302 -

Starting story /moral/story_24
[2023-10-19 15:55:38,349] ERROR in app: Exception on /story_num_overall [GET]
Traceback (most recent call last):
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 2528, in w
sgi_app
    response = self.full_dispatch_request()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1825, in f
ull_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1823, in f
ull_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\Raquel\AppData\Local\Programs\Python\Python310\lib\site-packages\flask\app.py", line 1799, in d
ispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\Raquel\Desktop\Decision Making App\dec-making-app-v32.2\app.py", line 1244, in story_num_refres
h
    blurb = f"{ story_info['topic'] }"
KeyError: 'topic'
127.0.0.1 - - [19/Oct/2023 15:55:38] "GET /story_num_overall HTTP/1.1" 500 -
```

This error is much more obscure and may not have anything to do with file or directory names. However, if you find that this error occurs at the start of a story, when the story context is supposed to appear, the cause of the error could be a misnamed folder/directory.

To identify the problem, refer to the command line printout that identifies the story that was last attempted to be loaded. In the example above, we see ‘/moral/story_24’. If we navigate to the directory that contains all the ‘moral’ tasks, we find the following:



In this case, the 24th story directory is misnamed. Story directories must be named ‘story_{story number}’ in order for the code to recognise them as such. To fix this issue, we rename this directory to ‘story_24’.

Unfortunately, this error occurred at a point where refreshing the page will not recover the code from the failure. An app restart is needed in this case. Refer to the ‘Recovering from catastrophic failures’ section for steps to recover from this.

Recovering from catastrophic failures

Catastrophic failures occur when the code raises an exception that cannot be recovered from by refreshing the page. This may be because some critical data was pre-loaded into memory and cannot be re-created using new information, or because external hardware prevents the code from re-running. A catastrophic failure may also occur when the code soft-fails (does not stop running), but shows incorrect information to the user, which can affect data.

What measures to take will depend on your unique situation and whether data may be potentially lost or 'corrupted'. If there is potential data loss during a trial, follow either **solution 1** or **solution 2**. If there is no expected data loss (for example, the error did not occur using a trial question), proceed to **solution 2.**

Solution 1
You may want to simply ask the subject to skip a question altogether and continue through the rest of the trials, however, the subject may encounter the same error again if there was more than one question that had the same issue in the 'questions.txt' file. If you choose to proceed with the trials, make note of which questions had a problem and write them down in the session notes at the end of the session.

The app will record the subject's data and immediately upload it to the database whenever the subject clicks the 'Submit' button during a trial, so there will be some data that can be preserved and analysed.

Solution 2
You might decide that because there were errors in the session, the data will be bad and unusable in its entirety, or you might not be able to recover from the error by simply refreshing the page. If this is the case, you may interrupt the session and ask the subject to re-do the problematic story after you've troubleshooted it. To do this, follow the following steps:

1. Bring the command line window up and enter the keyboard combination 'Ctrl + C'. If you're prompted to confirm, enter 'Y' and press the Enter key.
2. Make sure you make any necessary corrections to the 'questions.txt' file and navigate to '\bin\settings.ini'. Open this file with your text editor of choice (f.e. Notepad, or Notepad++). Make the following changes:
    a. (line 34) next_story_from=local
3. Save the settings file and minimise it. Restart the app as you did before. Ask the subject to select however many stories they had left in the previous session and run them through the same setup as before. When asked 'Are you a new participant?', ask the subject to click 'No' and enter their ID number.
4. The app should continue on the story where it left off before the error happened.
5. Once the subject is done with their session, make sure to add a note of what happened to the session notes.

6. Stop the app through the command line as in step 1, undo the changes you made to settings, and restart the app.

Following these steps should minimise data loss.

Adding or replacing stories in participant's story order

Sometimes it may be necessary to add or replace stories to an existing participant's story order. This may be because some new task types were included into the study, in which case a new story order including the new task types needs to be added. It could also be that a particular story needs to be removed or replaced from a participant's story order. In both of these cases, one can directly manipulate the stories that the subject will see.

Adding stories

To generate a new story order, refer to the `distribute_stories.py` script. This script runs directly from the DM app's root directory and references the 'settings.ini' file contained in the 'bin' directory.

Distribute_stories.py is the same algorithm that the dm app uses to create a user's story order upon beginning their first-ever session. It is a command line script that will take a list of at least X topics, where X is defined as the minimum amount of stories that users are required to select from the dm app; this is defined by the 'minimum_topics' parameter in 'bin\settings.ini'. If no parameters are entered, the script will use a hard-coded list of topics as an execution example. The script will fail if less than X topics are passed. You may pass more than X topics at a time, as long as they exist in the 'Human DM Topics.xlsx' file.

Assuming no errors are found in the code execution, this script will spit out a freshly-made story order following the same instructions the 'settings.ini' file gives. This means that you must modify and save
the appropriate parameters in 'settings.ini', this is a list of parameters that will affect the outcome of this script:

1. approach_avoid,
2. benefit_benefit,
3. cost_cost,
4. moral,
5. multi_choice,
6. probability,
7. social,

All of which indicate how many stories to pull from each task type, and...

8. minimum_topics

To specify which topics to pick stories from, add them as command line arguments by writing them one-by-one after 'python distribute_stories.py ', each inside single quotes and separated by a space (see example below).

USAGE EXAMPLE

> in 'settings.ini': (approach_avoid=12, benefit_benefit=0, cost_cost=0, moral=0, multi_choice=0, probability=0
social=12, and minimum_topics=4)

Write in command line:

```
python distribute_stories.py 'Education-Post-Education Life' 'Food'
'Vehicle/Transportation' 'Entertainment'
```

Output:

```
['/social/story_20', '/approach_avoid/story_16', '/social/story_23',
'/approach_avoid/story_14', '/social/story_24',
'/social/story_7', '/social/story_8', '/approach_avoid/story_9',
'/social/story_11', '/approach_avoid/story_19',
'/social/story_12', '/approach_avoid/story_24',
'/approach_avoid/story_15', '/approach_avoid/story_20',
'/social/story_6',
'/approach_avoid/story_2', '/approach_avoid/story_10',
'/social/story_22', '/social/story_3', '/approach_avoid/story_6',
'/approach_avoid/story_21', '/social/story_10', '/social/story_21',
'/approach_avoid/story_5']
```

Keep in mind that the topics must be written exactly as they are in the 'Human DM Topics.xlsx' file!

If you are generating an additional story order for a particular user, access their demographic data by navigating to '\data\{user_id}\demographic_info.txt', and reference the 'pref_stories' line.

Once the new story order is generated, copy it from the command line window and **append** it to the **end** of the user's 'story_order' line. Do not delete the existing story order!!

When pasting, make sure to delete the original closing square bracket (']'), add a comma and a space, and paste the new story order without the opening square bracket ('['). Also make sure you have not accidentally added an empty line to the end of the file.

Finally, save the demographics info file and close it. The app will continue from the last story it left off on. This is indicated by the user's 'next_story_index' (if 'next_story_from=local'), or by the number of different stories gone through registered in the database (if '`next_story_from=database`').

Replacing stories

To replace a story in the user's story order, navigate to the 'data' directory from the HMDA root directory, find the user's ID number, and open their 'demografic_info.txt' file. Under the 'story_order' line, replace whichever story you mean to replace, make sure the new story does exist in the 'stories/task_types' folder. Include the task type name and the story number, for example, '/social/story_5'.

If you need the user to go back a few stories to do the new story, modify the 'next_story_index' line by subtracting however many stories you need so that the user sees the story. Remember that 'next_story_index' is the index value of the next story in the story order list, where the first element in the list is element number 0.

Make sure you also set the 'next_story_from' parameter to 'local' so that the app references this number in the demographic data and does not look into the database to retrieve the next story index.

Removing stories

To remove a story from the user's story order, navigate to the 'data' directory from the HMDA root directory, find the user's ID number, and open their 'demografic_info.txt' file. Under the 'story_order' line, remove whichever story you need to remove.

You may need to modify the 'next_story_index' line if the removal impacts which story the user will see next. For example…
Say you need to remove '/social/story_7' from the following story order.

```
['/probability/story_3', '/multi_choice/story_11', '/moral/story_23',
'/multi_choice/story_1', '/social/story_7', '/cost_cost/story_2',
'/benefit_benefit/story_10', '/moral/story_15',
'/approach_avoid/story_7', '/probability/story_14', '/social/story_4',
'/social/story_17']
```

If we count the elements in the list starting from 0, we can figure out the index number (shown in superscript) for each story in the list…

```
['/probability/story_3'0, '/multi_choice/story_11'1, '/moral/story_23'2,
'/multi_choice/story_1'3, '/social/story_7'4, '/cost_cost/story_2'5,
'/benefit_benefit/story_10'6, '/moral/story_15'7,
'/approach_avoid/story_7'8, '/probability/story_14'9,
'/social/story_4'10, '/social/story_17'11]
```

Now, imagine that `next_story_index: 6`, which indicates that the user's next story will be '/benefit_benefit/story_10', the seventh element in the list.

If you simply remove ‘/social/story_7’ from the list, index 6 would point to '/moral/story_15' instead, which would mean that the user would never see ‘/benefit_benefit/story_10’.

```
['/probability/story_3'[0], '/multi_choice/story_11'[1], '/moral/story_23'[2],
'/multi_choice/story_1'[3], '/cost_cost/story_2'[4],
'/benefit_benefit/story_10'[5], '/moral/story_15'[6],
'/approach_avoid/story_7'[7], '/probability/story_14'[8],
'/social/story_4'[9], '/social/story_17'[10]]
```

In this case, in order for the user to see '/benefit_benefit/story_10', we would need to change `next_story_index` to 5.

Make sure you also set the 'next_story_from’ parameter to ‘local’ so that the app references `next_story_index` in the demographic data and does not look into the database to retrieve the next story index.

Eye Tracking Study Considerations

A full version of these guidelines can be found on the official Tobii Connect website. The information contained in this section was paraphrased or taken directly from the documentation provided there. See the sources section for details.

Participant Screening

Because corrective lenses (glasses or contacts) might cause issues with the eye tracker during our trials, it might be good to screen for the following people:

- Wearers of corrective lenses (glasses) for cataracts, as well as bifocals
- Individuals who have had eye surgeries: corneal, cataracts, intraocular implants
- Individuals with eye movement or alignment abnormalities: lazy eye, nystagmus, others
- If participants can wear contact lenses instead of glasses, please advise them to do so

Eye Tracker limitations

The eye tracker will not usually have a problem with corrective lenses, however issues may arise if:

- there are internal reflections being caused by lighting in the room,
- the glasses themselves keep moving on the participant,
- the lens correction is very strong (+/- 6 or more),
- or the frames somehow occlude the eye-image in the camera (a simple adjustment of the angle of the tracker can usually remedy this).

The eye tracker will not cope with bi- or varifocal lenses and this is because the lens distorts the shape of the pupil which causes detection errors in the eye tracking software. So, when recruiting, it's important that you always include questions about glasses in the screening and *ensure participants are wearing the appropriate pair for the task BEFORE calibrating*. Oh, and yes, offer them a lens cloth before you start calibration if the glasses look a bit grubby!

Contact lenses also come in bi-focal/varifocal form these days. So, whilst contact lenses are usually OK with all kinds of eye-trackers, the multi-prescription lenses CAN cause problems, so it's always worth checking!

Tips for facilitating eye tracking studies with people who wear corrective lenses – and even those who don't…

Corrective glasses should be worn during calibration!!

If you see your participant squinting, it's almost certainly going to cause issues for the tracker and it suggests their vision is not corrected, so **keep an eye on your participant during data collection.**

If you have access to one, use a Snellen or LogMAR chart at the distance of your stimuli to test your participant's vision before you start. You're not an optician and you're not going to diagnose their condition, but this is a great way to check if their vision is as good as they claim it to be and so they should be wearing their lenses whilst doing this!

Show them some example images/text on screen and get them to describe them to you or read out loud and you'll soon know if you're going to have a problem – because they'll squint, lean in to the screen or simply not be able to do it. If you can, have the eye tracker on for this to test how well the eyes are being detected.

If you're using a desktop tracker, have a range of cheap reading glasses, covering different corrections – ideally from -6 to +6 diopters. When the inevitable happens and someone forgets their glasses you can offer them as a temporary solution and avoid having to discard the participant. These are typically available from large supermarkets or chemists.

Important Reminders

Remind your participants for the day of the eye tracking study:

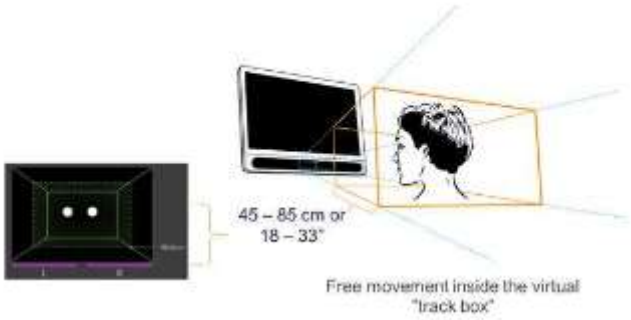
- Avoid wearing heavy make-up, mascara, false eye lashes or colored contact lenses.
- If doing 'intercept' recruitment, keep an eye out for the above in addition to physical features (such as unusually droopy eyelids) that may prevent the eye tracking sensors from 'seeing' the eyes.
- Check and confirm that all participants have read and agreed to the consent form!

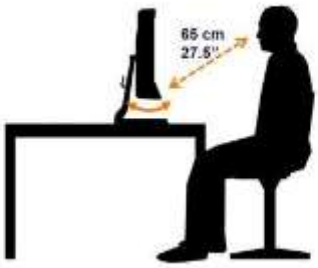
Don't forget to inspect your equipment:

- Your eye tracker is cleaned and ready with any accessories.
- Your computer or tablet is up to date, with all required software functioning as expected.
- You have additional accessories such as power cables, chargers, extensions, cleaning cloth and sanitary wipes, external storage drives, and any required dongles or adapters.
- Perform a test run of the data collection, exactly like you would for the participants, to make sure your setup and environment are up to expectations.

Positioning Participants

- Make sure your participant is comfortable, has a good position relative to the screen, and can reach the keyboard/mouse if used.
- You can check the participant's head's range of movement using the track box guide. It is available on the top left before starting a recording in Tobii Pro Lab.
- Use of a chair without wheels and swiveling is recommended as it can limit how far the participant can move from the track box.





Running the Test

1. After positioning your participant in front of the screen, start a recording in Tobii Pro Lab.
2. Since all eyes are different, you will first run a calibration.
3. During the calibration, ensure the participant remains focused; avoid conversation or laughter.

Interpret the results

- Measurement points should be within the cross' boundaries for best accuracy
- The bigger the distance of the points to the cross' center, the larger the potential measurement error
- The further the elements on your stimulus are separated, the larger error can be accepted
- For target groups with difficult calibration, consider modifying the stimulus

Recording

After successful calibration, you may start the recording.

- Explain the task slowly and thoroughly to the participant and make sure they know what they are supposed to do.
- During the task, you as a data collector should only observe. Let the participant complete the task in a natural way and only interrupt if necessary. Take notes of relevant behavior and observations during the test - this is not only be useful for post-interviews but also analyzing and interpreting your data.
- After the presented tasks are complete, stop and save the recording.
- Perform the post-interview (optionally with retrospective think aloud)
- Don't forget the consent form!

Section Sources

This entire section mirrors the content of the following websites:

1. https://connect.tobii.com/s/field-guide-screen-based?language=en_US&t=1693606963655
2. https://www.tobii.com/blog/eye-tracking-study-recruitment-managing-participants-with-vision-irregularities

App settings

The HUMANS app accesses a list of parameters located in the '/bin/settings.ini' configuration file (INI). Below is a detailed explanation of what each parameter does and where it is accessed in the source code.

Parameter	Section	Type	Location	Description
host	postgresql	str		The host computer's IPv4 address or domain name. If HUMANS is running on the host computer, enter the word localhost.
database	postgresql	str		The name of your PostgreSQL database.

Parameter	Section	Type	Location	Description
port	postgresql	int		The host computer's port-forwarded port for servicing database queries (PostgreSQL defaults to port 5432).
user	postgresql	str		The PostgreSQL username. Keep in mind that the account that the HMDA uses must have database read-write privileges.
password	postgresql	str		The password tied to the account identified by the 'user' parameter.
data_table	app_settings	str		The name of the data table structure defined in the PostgreSQL database. Data generated by the HUMANS app will be stored here. If a data table has not already been created, refer to the 'auto_create_table' parameter below.
auto_create_table	app_settings	bool		If the table specified above does not exist in the database, the app may be able to create it using the participant's data. If this parameter is set to 1, the app will attempt to create the target data table once it has all the required data from the user If set to 0, the app will not upload data to the database.

Parameter	Section	Type	Location	Description
enable_consecutive_users	app_settings	bool		Enables back-to-back use of the app without the need to close and re-run startup.bat (which restarts the flask server). If set to 1, the app will reset all global app and user parameters to default when loading the landing portal (setup_session.html, routed as '/'). If this option is not enabled (set to 0), the flask server must be closed and re-run. Any changes to the source code or templates are not affected by this option.
data_upload	app_settings	bool		Ensures that data is automatically uploaded to the database after each session if set to 1. When set to 0, the write_trial_to_database routine is silently skipped.
unique_ids_from	app_settings	str		Controls where IDs are read from when trying to generate a unique ID for each user. Possible values are "database" and "local". ● "database" will look into the database and find all the unique IDs stored there. ● "local" will only look to the "/data/" directory to find all the taken IDs. The app will then generate a new random ID that does not exist in the retrieved list of existing IDs.

Parameter	Section	Type	Location	Description
next_story_from	app_settings	str		Controls where the next story index is referenced from. Possible values are "database" and "local". ● "database" will look into the database to find the next story the user will see by counting all the unique entries of 'tasktypedone'. ● "local" will look into the user's local demographic info record in '/data/' and read the 'next_story_index' entry.
timestamp_timezone	app_settings	str		The time zone that timestamps are collected in. 'UTC' is recommended. If this parameter is not set, the app will default to 'UTC'.
minimum_topics	app_settings	int		The minimum amount of topics that the subject must select. Replaces 'min_stories_to_choose' from version 30.
questions_per_story	app_settings	int		The number of questions selected to show the user **per story**.

Parameter	Section	Type	Location	Description
ignore_legacy_story_data	app_settings	bool		How the app handles story data ('pref_stories' and 'story_order') from legacy app versions. If this is set to 0, the app will NOT ignore legacy story data from returning users and continue the remainder of the users' sessions with only 'approach_avoid' task types, keeping their preferred stories and previously calculated story order. If set to 1, the app will discard this information and ask the user to restart their sessions to add the new task types to their story data.
randomise_relation_levels	app_settings	bool		FOR SOCIAL TASK ONLY: Whether to randomly choose a random relationship keyword and replace it into the text of the social task stories. If set to 1, a random relationship keyword will be picked from the list under the 'relation_levels' list parameter and replaced into every snippet of text throughout the entire story.
relation_levels	app_settings	list		FOR SOCIAL TASK ONLY: The words to look for and replace in story text. The app will randomly sample a word and replace it throughout the entire story text.
relation_level_stories	app_settings	list		FOR SOCIAL TASK ONLY: Stories where replacing the relationship level can be done

Parameter	Section	Type	Location	Description
validate_stories	app_settings	bool		Validates that the stories described in the 'Human DM Topics' relationship table are actually contained in '../stories/task_types/'. Stories that don't exist in that directory will be deleted from the pool of stories that can be selected for each subject. Setting this parameter to 1 can cause errors if the validated story pool ends up being smaller than the sample size for each task type (see below). If this is the case, check that there are enough stories for each task type in '../stories/task_types/'. Conversely, not validating the stories increases the odds of trying to access a story that does not exist, causing a different error.
approach_avoid	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.
benefit_benefit	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.
cost_cost	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.

Parameter	Section	Type	Location	Description
moral	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.
multi_choice	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.
probability	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.
social	app_settings	int		The amount of stories of the corresponding task type to sample when creating a user's story order.
manager_install_path	eye_tracker	str		The install path to the eye tracker's management program. Tobii eye trackers will install the manager in C:\Users\{HOST USERNAME}\AppData\Local\Programs\TobiiProEyeTrackerManager\TobiiProEyeTrackerManager.exe
subscriptions	eye_tracker	list		A list of data streams to subscribe to. Different eye tracking devices may offer different data streams. The app's EyeTracker class considers only 'gaze', 'openness', and 'position' as valid options.

Parameter	Section	Type	Location	Description
eyetracker_index	eye_tracker	int		If multiple devices are connected to the system, this number will indicate which device to use. If only one device is set up, use 0.
use_eyetracker	eye_tracker	bool		Activates or deactivates the subroutines involving the eye tracker. ● If 1, the app will attempt to make use of any eye trackers connected to the host. ● If 0, all subroutines will be skipped.
use_external_app	hr_tracker	bool		Determines whether an external program will be called as a subprocess for collection of heart rate data. ● 1, an external program is called for data collection (see below). ● 0, a thread will be spawned by the main application process (see last four params in this section).
external_app_install_path	hr_tracker	str		The external heart rate-capturing program's location in disk. Any program may be used as long as it is callable from this path.
hrtracker_index	hr_tracker	int		If multiple devices are connected to the system, this number will indicate which device to use. If only one device is set up, use 0.
use_heartracker	hr_tracker	bool		Activates or deactivates the subroutines involving the heart rate tracker. ● If 1, the app will attempt to make use of any heart rate monitor devices connected to the host. ● If 0, all subroutines will be skipped.

Parameter	Section	Type	Location	Description
emulate_device	hr_tracker	bool		Whether to emulate a heart rate device for the heart rate monitor thread. This is mainly useful for development. Use this if you're testing the app and don't have a heart rate monitor device connected to the computer. Emulated data is also packed and sent to the database for troubleshooting purposes.
run_thread_as_daemon	hr_tracker	bool		Whether to run the heart rate monitor thread as a daemon. If set to 1, the heart monitor thread will run alongside the app as a daemon. This ensures that the thread will exit along with the app, however, this could cause problems if the thread is not manually stopped
verbose	hr_tracker	bool		Toggles verbose output of the hr monitor thread. This has little impact on functionality, but is useful when debugging the heart rate monitor. Setting this to 1 will allow the hr monitor thread to display messages on the same console window the app is running on.
test_on_startup	hr_tracker	bool		Whether to test the heart rate monitor when the app starts. • 1, the device will be tested by initialising and collecting data for 3 seconds. • 0, device will not be tested on startup.

Glossary

Story: A scenario. In our application, stories present a subject with a conflict resolution problem. A story encompasses the story context, the reward and cost preferences, and a series of trial questions. Each story belongs to a particular task type, and the context of each story surrounds a singular topic.

Story order: The list of stories assigned to a participant. The list is generated based on the participant topic preferences and the task types chosen, then it is randomised and recorded in the local participant file. The *story index* parameter dictates which story will be accessed next.

Story index: An integer that points to a particular position in the *story order* list. This parameter starts at 0 (pointing to the first element in the list) and is incremented by 1 after all 16 trial questions have been answered.

Unicode: A text encoding standard which allows the use of emoji and other regional characters for various writing systems.

Demographic data: All data taken from the user including but not limited to age range, sex, gender identity, education, hobbies, and visual media interests. Demographic information on the user is stored both locally in a file named '*demographic_info.txt*' inside '/data/{SUBJECT_ID_NUMBER}', and in the remote or local PostgreSQL database.

Root directory: The main working directory for the HUMANS application. The root directory will typically be represented with the leftmost '/' in a file path throughout this document.

Directory: A 'folder' in the host computer's filesystem. Directories follow a hierarchy where the leftmost directory in a path represents the *root directory*. Any subsequent directories in the path are contained within each other. For example the file path '*C:\Windows\System32*' is a path that points to a Windows machine's system 32 directory.

Command line: A software program which provides direct communication between the user and the operating system or an application. In Windows, there are two command line shells, the Command shell (cmd) and the Power Shell.

Console: Can also be used as shorthand for *command line*. The console is a type of user interface which displays an application's output.

ANT+ protocol: The wireless communication protocol used for communicating heart rate information from the heart rate monitor to the host computer.

HDMA (acronym): Human Decision-Making App

HUMANS (acronym): Human Model for Analyzing Neuroeconomic Situations

Host: The computer on which an application runs.

Client: A computer which communicates with a *host* computer to request data.

Integer (int, data type): A data type which includes whole numbers only. Integers are, simply put, numbers to do mathematical or logical operations on or with.

Boolean (bool, data type): Is evaluated in conditional statements. A boolean can either be True (1), or False (0).

List (list, data structure): An ordered succession of values in memory. Each element of a list can be accessed with an *integer* value denoting a value's position in the list. An index of 0 points to the first element in the list.

Dictionary (dict, data structure): An ordered structure of values where each value is identified by a key. Dictionaries can easily be interpreted as tables where each key is a column name.

String (str, data type): A data type containing numerical, ASCII, and/or *unicode* characters. Strings are typically used to represent non-numerical information and can be of variable length, spanning from a single character, to a whole book's worth of characters.

ASCII (acronym): American Standard Code for Information Interchange. A standard data-encoding format for data exchange between computers. It assigns numerical values to letters, numbers, punctuation marks, and special characters.