CREED Experimenting Guide*

Updated Dec 2025

Check for latest document

Tell me if anything is unclear, wrong or missing so we can update it!

Contents

*P.S.: This is unofficial and merely what I jotted down so I would remember it next time I want to run. All mistakes are my own. J. Tait

1 Experimental Timeline

1.1 Overview

Before running an experiment at the CREED Lab or applying for ACBC funding, present your project at the CREED Lunch Seminar. Contact Julian Kirschner or Madgalena Wasilewska to schedule.

Before running, ensure you have each of the following:

1. **Pre-analysis plan:** AsPredicted
2. **UvA/EB Project:** Research Management Services
   - Ethics review
   - Data Management Plan
3. **Funding:** for example from ACBC

Pre-Analysis Plan Questions.

1. Has any data already been collected?
2. What is the main question or hypothesis?
3. Key dependent variable(s) and measurement method?
4. Number and nature of conditions?
5. Planned analyses?
6. Outlier handling and exclusion criteria?
7. Sample size or rule?
8. Additional info?

2 Running Experiments in the Lab

We use the CREED website to manage everything about participant sign-ups. In the lab you run your experiment from your oTree folder on the local server. You need to make sure you i) reserve the lab, ii) get the participants invited, and iii) your code works on all the computers in the lab. Find instructions on how to make sure this all works below.

2.1 Reserving the Lab

Coordinate with Arthur Schram to reserve lab access and receive credentials for CREEDExperiment.nl.

2.2 CREED Website - Managing Participant Sign-up

Getting Started with the Website.

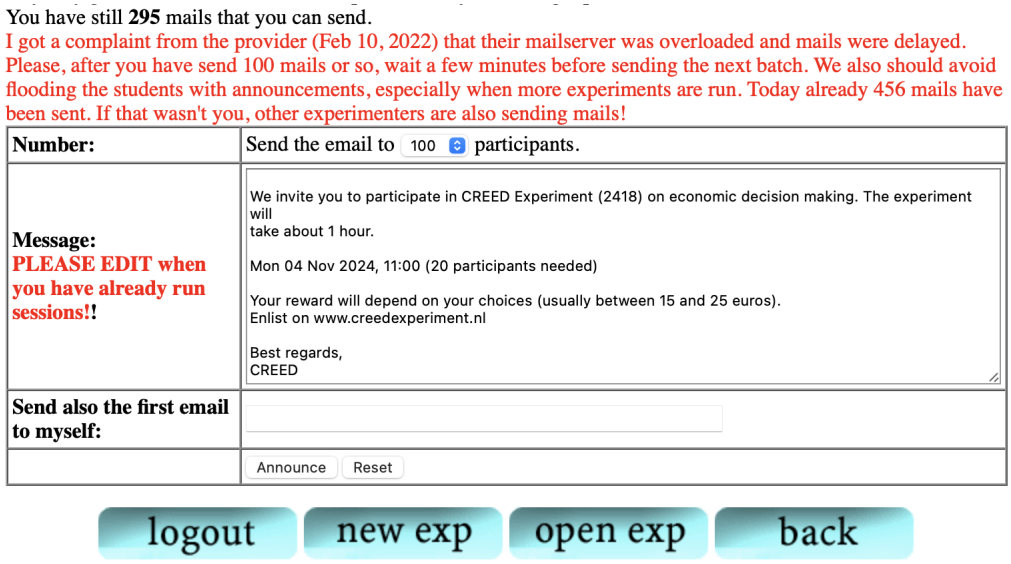
- Login details: from Joep
- Experiment number: from Ailko
- URL: `http://www.creedexperiment.nl/proftulp/loginexp.html`
- Create a new session via “new exp”

Managing Experimental Sessions.

- Modify session: `Show Session > Modify`
- Send reminder: `Show Session > Reminder`
- Finish session: `Show Session > Finish`

Inviting Participants.

1. click ’make a table with potential subjects with name and email address’
2. click ’send emails to a limited number of subjects (like 100)’
3. Prepare the customisable email template
4. Limit email sends (e.g., 100 per batch)
5. System tracks sent emails and removes people from the table to avoid re-sending to the same participant.
   Only create new tables the first time you send invites or when you don’t have enough ’mails to send’ in the screen attached below. Once you create a table it removes anyone you have already sent a mail to from the table, hence why the number decreases. This avoids sending duplicates.



Sending Reminders. Navigate to Show Sessions, select a session, and click `Reminder`.

- You can customise the reminder (subject, message).
- Add your email to test delivery.
- Double-check it's for the right session before sending.

Closing the Session. After running a session you need to log attendance. Navigate to Show Sessions and 'Finish' the relevant session. You can then select who showed up and who didn't as well as log how the session went.

2.3 Running in the Lab

I provide easy copy-paste code in Section A.1 to implement a super simple workflow when you enter the lab. With this configuration all you have to do to run is:

1. turn on all the computers
2. run *set_up_otree.bat* on the experimenter computer (download link)
3. open shortcut 'Experiment Room Large/Small Lab' on the participant desktop
4. confirm your session configs in the oTree monitoring page
5. create the session and run

Monitor the Session. *set_up_otree.bat* will automatically open the monitoring page. If not, visit `http://145.18.178.130:8000/rooms` for the large lab or `http://145.18.178.133:8000/rooms` for the small lab.

2.4 Saving Lab Session Data

Download it from oTree after each session is completed in the 'Data' tab. Save it in some data folder in your project folder as 'Session_XX.csv'. This ensures that whatever happens to the database on the experimenter computer you already have your data elsewhere.

3 Paying Participants

3.1 Cash

Ask your supervisor to email Ailko (ailko@xs4all.nl) with:

- Experiment number
- Total cash needed
- Preferred denominations

Bring them up one-by-one to pay them the cash in the correct amounts. Make sure they sign a receipt of payment so you can prove where the cash has gone.

3.2 Digital Transfer (UvA Administration)

Overview: Participants can be paid via bank transfer from the central UvA administration (SEPA only). If their IBAN is not in the SEPA region, arrange an alternative (e.g. cash to be collected from your office).

1. **Collect Payment Details**
   - Collect **IBAN** from all participants.
   - If IBAN is not Dutch, also collect the **BIC/SWIFT** code.
   - Check if IBAN belongs to a SEPA country.
   - If not SEPA, show: *"Please speak to the experimenter before leaving."*
2. **Handle Non-SEPA Participants**
   - Inform them that **cash payment** will be arranged later.
   - Record their contact details and agree on an office pickup time.
   - You can enter your bank details into the payment schedule and source the cash privately. This proved a very small share of total participants (6 of 350).
3. **Submit SEPA Bank Payments**
   - Compile valid IBANs in the required format (download link)
   - Email `servicedesk-ac@uva.nl` with the payment list and request processing.
4. **Track Payment Status** (& maintain a good reputation for CREED)
   - Look for two emails: *confirmation of processing* and *confirmation of payment.*
   - Keep a log of any **rejected IBANs** and the **session** that participant was in so you can identify people who reach out to CREED.
   - Email a list of the failed payments to experiment@creedexperiment.nl with subject 'Failed Transfers ExpXXXX' so the data is available to anyone checking the email (**ideally you!**). This should identify a participant with information on: i) amount, ii) session + date, iii) failed IBAN, and iv) experiment number

4 oTree Tips and Tricks

Some solutions to my oTree problems may be useful to you too. Here's a collection of ones I imagine most people can face.

4.1 Determining Experiment Payoff (Using a PayoffSummary Page)

Because oTree does not allow retrieving payoffs from past apps directly, each app must store its payoff information in a participant-level variable (e.g., `participant.payoff_vector`). Later apps (e.g., an outro/payment app) can then access this list to compute payments or randomly select paid rounds.

Step 1: Declare the Participant Variable in `settings.py`. Add `payoff_vector` to `PARTICIPANT_FIELDS` so it persists across apps.

```
PARTICIPANT_FIELDS = [
    ...,
    "payoff_vector",
]
```

Listing 1: Adding payoff_vector to Participant Fields (settings.py)

Step 2: Append This App's Round Payoffs at the End of the App. Instead of using a `before_next_page` hook on the final page, finish each app with a page that runs once on the final round and aggregates each participant's payoffs from all rounds in that app.

```
class payoff(Page):
    template_name = 'main/payoff.html'

    def vars_for_template(self):
        return {
            'payoff': cu(self.payoff),
            'round_count': self.round_number,
        }

    @staticmethod
    def before_next_page(player, timeout_happened):
        # On the final round, collect all payoffs into the participant vector.
        if player.round_number == C.NUM_ROUNDS:
            payoff_vector = [pr.payoff for pr in player.in_all_rounds()]
            if not hasattr(player.participant, 'payoff_vector') or player.participant.payoff_vector
     is None:
                player.participant.payoff_vector = []
            player.participant.payoff_vector.extend(payoff_vector)
```

Listing 2: PayoffSummary page to store per-app payoffs into participant.payoff_vector

Notes: Place `PayoffSummary` as the final page in your app's `page_sequence`. This ensures the payoffs are stored once the app is complete.

Step 3: Retrieve the Stored Payoffs Later (e.g., in the Payment/Outro App). In the payment-determining app, retrieve the accumulated payoffs safely via `getattr`.

```
payoffs_vector = getattr(p.participant, 'payoff_vector', [])
```

Listing 3: Retrieving the accumulated payoff vector in the payment app

Step 4: Example: Randomly Select Paid Rounds from the Stored Vector. Once you have `payoffs_vector`, you can compute totals or draw a random subset of rounds for payment.

```
import random

payoffs_vector = getattr(p.participant, 'payoff_vector', [])

if payoffs_vector:
    paid_amount = random.choice(payoffs_vector)
else:
    paid_amount = 0
```

Listing 4: Example: select one random payoff from the stored vector

This approach mirrors the pattern used in common oTree outro/payment templates (e.g., `oTree-template/outro`) and avoids relying on cross-app payoff retrieval.

5 oTree Template

There is an oTree template app with a lot of functionality needed for running experiments in the lab (download link). To use it, download the file and rename it from `.otreezip` to `.zip`, then unzip. It consists of four parts:

- **before** (welcome + consent)
- **intro** (instructions + quiz)
- **main** (main experimental game/task)
- **outro** (demographics + payment)

How to use and edit this template

- **Instructions content:** edit `intro/instructions_text.html`. Each `<div class="instruction-block">` is shown as one instruction page to participants. Add, edit, reorder, or remove blocks there to change instruction pages. (This means you typically only need to update instructions in *one* file.)
- **Quiz questions:** edit `intro/quiz_items.py`. Define `QUIZ_ITEMS` entries (e.g., `field`, `prompt`, `choices`, `answer`). The quiz reads directly from this file and updates dynamically.
- **Treatment assignment:** edit `before/treatment_assignment.py`. Treatments are assigned when the session is created (via `creating_session` in the `before` app). Adjust `assign_treatments` to set the treatment groups you need.
- **Experimental payoff rule:** edit `outro/payment_rule.py` to determine how participants are paid (e.g., which rounds are payoff-relevant and how totals are computed).

before

This includes a welcome screen and a consent page. Participants cannot go beyond the start/welcome holding page, so the experimenter can start the session when everyone is ready. Key files:

- `before/startpage.html`: waiting/holding page; usually leave as-is unless you want different text.
- `before/welcome+consent.html`: welcome and consent language; edit to match approved consent wording.
- `before/treatment_assignment.py`: treatment assignment logic (called at session creation).

intro

This includes an instructions sequence with a quiz at the end. Instructions are paginated automatically: each instruction block is shown as one page. The template dynamically updates with the available quiz questions; you only edit the instruction text file and the quiz item list. If participants cannot answer the quiz, they can return to the instructions. Key files:

- `intro/instructions_text.html`: edit instructions here; each `instruction-block` is one page.

- `intro/quiz_items.py`: edit questions, choices, and correct answers; the quiz reads from this file.
- `intro/templates/` (e.g., `instructing.html`, `quiz.html`): template shells; typically do not edit.

main

This folder contains the core logic for your experimental task. Place your main game code, task logic, and the files that control the experiment's core flow here.

outro

This section collects demographics (e.g., age, gender, IBAN + BIC), computes payment, and displays participants' payment for the experiment. Key files:

- `outro/Demographics.html`: demographics and payment details collection; edit if you want different questions/validation text.
- `outro/Results.html`: built-in results summary (per-round payoffs and total); generally leave untouched.
- `outro/payment_rule.py`: controls how payment is calculated (recommended place to change payoff logic).

Other files

- `set_up_otree.bat`: script to start oTree on the experimenter's PC in the lab.
- `format_session_data.py`: turns raw *sensitive* data into (i) a payment CSV, (ii) an anonymised experiment-data CSV, and (iii) sends the anonymised file to `experimentdata@gmail.nl`.

6 FAQ

6.1 oTree/Heroku

Q: How many participants can Heroku handle at once?. Based on experience, standard setups can support around **20–50 participants** simultaneously. Going beyond this increases the risk of server crashes.

Q: How can I scale to more participants (e.g., 80–120)?. A setup that successfully handled up to 120 participants used the following Heroku configuration:

- **Heroku Postgres Standard 0** ($50/month)
- **Heroku Redis Premium 2** ($60/month)
- **Heroku Dynos:**
  - Web dynos: 2X with 4 dynos ($200/month)
  - Worker dynos: 2X with 2 dynos ($100/month)

Tip: Dynos and Redis can be scaled up before a session and down afterward to reduce costs. However, the Postgres database must remain active to retain session data.

Q: Wait pages don't refresh automatically. What should I do?. If you encounter this issue, make sure to run:

```
heroku config:set PGSSLMODE=require
```

Listing 5: Heroku SSL Configuration

Without this, you may get the error: `FATAL: no pg_hba.conf entry for host`, and wait pages won't auto-update.

Q: Is there a workaround to prevent stuck wait pages?. Yes. You can add a fallback script that reloads the page every 10 seconds to avoid session breakdowns:

```
setInterval(function(){ location.reload(); }, 10000);
```

Listing 6: Auto-refresh fallback

Note: This helped recover from issues during high-load sessions where the auto-update failed.

A Appendix

A.1 Copy-Paste Code Snippets

Configuration of oTree settings.py. Insert this code into your `settings.py` file. Ensure you don't have other rooms and don't hard-code the ADMIN_USERNAME or ADMIN_PASSWORD elsewhere.

```
ROOMS = [
    dict(
        name='experiment',
        display_name='Experimental Session',
    ),
]

# Admin credentials from environment
import os
ADMIN_USERNAME = os.environ.get('OTREE_ADMIN_USERNAME')
ADMIN_PASSWORD = os.environ.get('OTREE_ADMIN_PASSWORD')

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': os.environ.get('DB_NAME'),
        'USER': os.environ.get('DB_USER'),
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': os.environ.get('DB_HOST'),
        'PORT': os.environ.get('DB_PORT'),
    }
}
```

Listing 7: settings.py configuration

Displaying Participant Computer Terminal as Player Labels. Insert this code into the __init__.py file of your first app.

```
class Player(BasePlayer):
    participant_label = models.StringField()

    def set_participant_label(self):
        self.participant_label = self.participant.label
```

Listing 8: First app init.py

Start Session Script. Use this `.bat` script to automate the server and environment start-up in the lab (download link):

```
@echo off

REM FYI: REM means that row is commented out

REM === Database Setup (only needed if you want to set up your own DB; this requires the postgres
    password to be entered into the terminal) ===
REM psql -U postgres
REM CREATE DATABASE experimentname;
REM CREATE USER moi WITH PASSWORD '1234';
```

```
REM GRANT ALL PRIVILEGES ON DATABASE experimentname TO moi;

REM === Database Management (at CREED, you can use DB 'experimenting' without issue) ===
set DB_NAME=experimenting
set DB_USER=moi
set DB_PASSWORD=ExperimentAllDay
set DB_HOST=localhost
set DB_PORT=5432
set DATABASE_URL=postgres://%DB_USER%:%DB_PASSWORD%@%DB_HOST%:%DB_PORT%/%DB_NAME%


REM === OTree variables ===
set OTREE_ADMIN_USERNAME=admin
set OTREE_ADMIN_PASSWORD=1234
set OTREE_PRODUCTION=1
set OTREE_AUTH_LEVEL=STUDY

REM === Starting the OTree project up on the server ===

REM === !!! ADOPT PATH TO YOUR OTREE PROJECT FOLDER!!! ===
cd "C:\Users\Admin\Desktop\TAIT\my great experiment"
echo Type 'y' to reset the database (advised)
otree resetdb

REM Start the server in a new terminal
start "oTree Server" cmd /k otree prodserver

REM === Open oTree Monitoring Page ===

REM === !!! SELECT LAB !!! ===
REM SMALL LAB: 145.18.178.133
REM LARGE LAB: 145.18.178.130

REM Wait for prodserver to boot up & open oTree monitoring page
timeout /t 5 >nul
start http://145.18.178.130:8000/rooms

echo Your postgres server is up and running :)
echo This terminal will close now
timeout /t 10 >nul
```

Listing 9: set_up_otree.bat

A.2 Custom PostgreSQL Setup

If you want to set up your own PostgreSQL instead of using section 2.3 then you need to follow these steps. If you follow the steps above set_up_otree.bat you will automatically run with **PostgreSQL**. If not, you should know that oTree defaults to a basic database called SQLite. This can lag more and does not back up your data in case the computer crashes. I recommend you ensure you're using PostgreSQL when running sessions to avoid problems.

If you download your session data after each session and are happy with other wiping the database after every session (ensuring you already have it saved, see 2.4) you can use an existing PostgreSQL database to read and write your data throughout the session (this is already specified to start in set_up_otree.bat). Alternatively, you can also create your own database using i) the code below, or ii) simply opening 'pgAdmin 4' on the experimenter computer and following the steps below.

Using *pgAdmin 4*. This is the simplest way with a visual interface.

```
1. Create a new user and password:
    - Open pgAdmin.
    - Right-click Login/Group Roles $\rightarrow$ Create $\rightarrow$ Login/Group Role.
    - Set:
    - Name: newname
    - Password tab: enter newpassword
    - Privileges tab: check all boxes`

2. Create the database:
    - Right-click Databases $\rightarrow$ Create $\rightarrow$ Database
    - Name it experimentname
    - Set owner to newname
```

Using the terminal. The master password should be '1234'.

```
psql -U postgres
CREATE DATABASE experimentname;
CREATE USER moi WITH PASSWORD '1234';
GRANT ALL PRIVILEGES ON DATABASE experimentname TO moi;
```

Then set the 'DATABASE_URL' environment variable accordingly in the set_up_otree.bat script.