Deployment Preparations

You need to add one thing in your oTree folder: Create a file called runtime.txt in your main oTree folder. This txt will tell the server provider heroku which Python version it should use. First, check the Python version that you are using

```
python --version
```

Note down the Python version and create runtime.txt in the main oTree folder. My Python version is 3.11.9 so I write

```
python-3.11.9
```

Finalizing settings.py

Go to settings.py to make some final settings. First, check the LANGUAGE_CODE. This setting is mainly relevant for the text on waitpages and the default buttons. The language of error messages is governed by the language of the browser. The language code could be set to de, fr, etc.

```
LANGUAGE_CODE = 'en'
```

Next, set a participation fee for the experiment in SESSION_CONFIGS. That's the amount of money participants will start with when they enter the experiment. If you do not set it in your oTree will default to the SESSION_CONFIG_DEFAULTS, which is by default zero. Note that the participation fee is in currency units, not experimental points!

```
dict(
    name='course_baseline',
    display_name='WZB oTree course Baseline',
    ## I'm omitting all the other lines of the dictionary here
    participation_fee=5.00
),
```

Settings for experiments in the lab

In settings.py you can find the ROOMS setting. Rooms are very useful for lab experiments because they are pre-defined waiting rooms. Clients can connect to a waiting room and in the server administration panel you can see how many clients have already connected. You have to set up a name and a display name at minimum. For lab experiments you can create a txt file in the _rooms folder and call it using

participant_label_file. In this text file you can set up the labels for the participants which are useful for payments. Let's create a new room called lab.

```
ROOMS = [
    dict(
        name='lab',
        display_name='Lab',
        participant_label_file='_rooms/lab.txt',
    ),
]
```

Then, add lab.txt in the _rooms folder. Add one name per line. Then start the server and check out the new room. Participants can join the room by a participant-specific URL. Alternatively, you can open the room-wide URL and enter the label manually. But then the label that is entered has to match one of the labels in the file.

Prolific Integration

I won't go into all the details of how to set up a prolific study and focus on the most important parts: how to let subjects join directly from prolific and how to send them back to prolific after the study. The first part is very easy. Simply create a session and look for the session-wide link. You can simply add ?participant_label=blablabla to the session-wide link and anybody who clicks this link will receive the label blablabla in the game. Also, this lets the same subject continue a study if they close the window and click the link again.

You can use this participant label command to use subjects' prolific ID as their label in your experiment. Prolific lets you include the {{%PROLIFIC_PID%}} command that prolific automatically translates into the subject's unique prolific ID. You can use the labels for payment later.

```
http://localhost:8000/join/SESSIONID?participant_label={{%PROLIFIC_PID%}}
```

After that the experiment proceeds as normal. At the end of the experiment you have to send subjects back to prolific with a completion code so they can prove that they completed the study. This completion code will change for every session so include an option for it in the SESSION_CONFIGS config.

```
dict(
    name='course_baseline',
    display_name='WZB oTree course Baseline',
    app_sequence=['consent', 'pgg'],
    num_demo_participants=3,
    multiplier=0.4,
    prolific_completion_link='https://app.prolific.co/submissions/complete?
cc=I2PWSFRG',
),
```

To send subjects back to Prolific it is best to create a new final page in your app on which the participant is sent back to prolific. On this page push the link to javascript. Don't forget to include BackToProlific in your app sequence.

```
class BackToProlific(Page):
    @staticmethod
    def js_vars(player: Player):
        return dict(

prolific_completion_link=player.subsession.session.config['prolific_complet
ion_link']
        )
```

Create BackToProlific.html and create a scripts block

```
{{ block scripts }}
<script>
var backtoprolific = js_vars.prolific_completion_link;
window.onload = function(){
    window.location.href=backtoprolific;
}
</script>
{{ endblock }}
```

As soon as a participant arrives on the last page they are redirected to the prolific website with their completion code.

Final documentation and testing

Finally, we should document which versions of Python and oTree we programmed our experiment with. I like to create a new file called README.md. Right-click on the oTree folder in the Explorer tab and select New File. Document the current Python and oTree versions that we installed. If you are unsure, open the terminal and type

```
pip show otree
```

to see the current version of otree that was installed.

Run otree devserver and check that everything works. Check out the Sessions and Rooms tabs. If you want to run a single session you can just create a Session but if you want to use the Rooms features create a room. Click through an experiment and go to the Data tab. Here you can download the data for any experimental session. Download some of those files and check that all the data is recorded correctly.

Finally, check the Server Check page. On this page you can check the setup. oTree tells me that I run the latest version of oTree but there are three red warnings. Right now debug mode is on (the little info

windows on the bottom of the experiment page), there is no password protection of the admin panel and we are not using a robust PostgresQL but a flimsy SQLite database. The first two things we can test easily on our local system but the third is more difficult to test locally. Close the server and set two environment variables in the terminal. These environment variables are only active as long as the terminal is open and they are deleted as soon as the terminal is closed. If you close the terminal you have to re-set them.

On Windows type

```
set OTREE_PRODUCTION=1
set OTREE_AUTH_LEVEL=STUDY
set OTREE_ADMIN_PASSWORD=chooseapassword
echo %OTREE_PRODUCTION%
```

On Linux/Mac type

```
export OTREE_PRODUCTION=1
export OTREE_AUTH_LEVEL=STUDY
export OTREE_ADMIN_PASSWORD=chooseapassword
echo $OTREE_PRODUCTION
```

Run otree devserver and access the server. You will be asked for a user name and password. The user name is admin and the password is the one we just set. Run and experiment and you should see that the debug window is now gone.