Studying social norms with behavioral experiments (with oTree)

Day 1 - Cheating game and DG.

Philipp Chapkovski[1]

[1]HSE-Moscow

July 14, 2021

Plans for the week

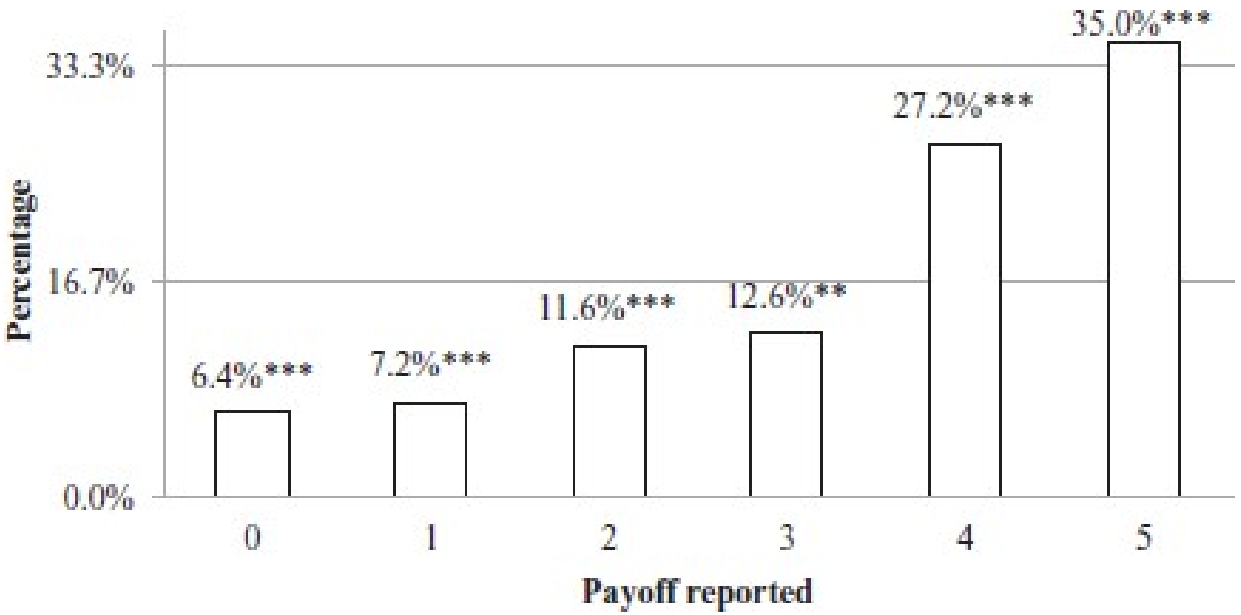
- Cheating game (Fischbacher and Föllmi-Heusi 2013)
- Dictator game (Kahneman, Knetsch, and Thaler 1986)
- Ultimatum game (Güth, Schmittberger, and Schwarze 1982)
- Trust game (Berg, Dickhaut, and McCabe 1995)
- Public good game (Fehr and Gächter 2000)

Code and demo:

- Code: https://github.com/chapkovski/spbss_code
- Demo app: https://spbss.herokuapp.com/
- Presentation (these slides:)
  https://github.com/chapkovski/spbss/tree/main/pres/ready

Cheating game

- Profit depends on the number reported on a die



Cheating game

Benefits:

- Super simple to understand
- Veil of anonymity
- Single player game

oTree - intro

Start new project: - Open a terminal (command line, power shell in Windows, Terminal on MacOS or Linux)

```
otree startproject NAME_OF_THE_PROJECT
```

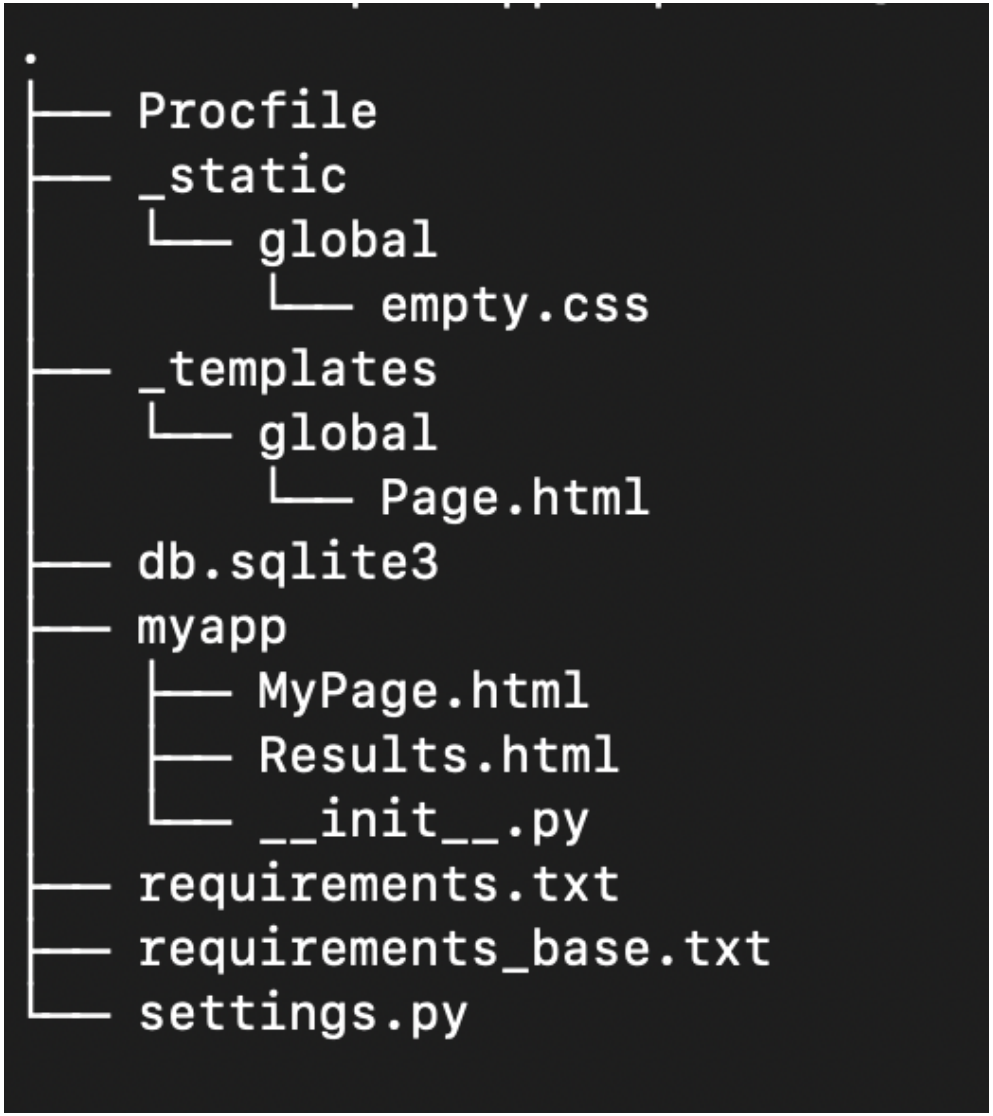
- Choose 'NO' when you are asked whether to add standard games.

oTree - intro

Change to the project folder:

```
cd NAME_OF_THE_PROJECT
otree startapp NEW_APP
```

Project structure



App structure

- main python file (`__init__.py`)
- HTML files for specific pages

Within `__init__` file:

- Constants
- Models
- Pages
- page sequence

App registration

App is just a folder with some files in it. To make it visible for oTree you need to add it to the `settings.py` file, to the `SESSION_CONFIGS` section:

```
SESSION_CONFIGS = [
    dict(
        name='dice',
        app_sequence=['dice'],
        num_demo_participants=1,
    ),
    ]
```

App registration

SESSION_CONFIGS is a right point for introducing treatments, like that:

```
SESSION_CONFIGS = [
    dict(
        name='dice',
        app_sequence=['dice'],
        num_demo_participants=1,
        beliefs=False
    ),
        dict(
        name='dice_belief',
        app_sequence=['dice'],
        num_demo_participants=1,
        beliefs=True
    ),
    ]
```

If you introduce treatments you can refer to the session.config later like that: `player.session.config['beliefs']`

Running the app:

`otree devserver` will make it running at the address http://localhost:8000

(You can run it on another port by adding port number after `devserver`)

Publishing the app

- Register at Heroku (you'll need your credit card info)
- Download GIT and Heroku CLI (here: https://devcenter.heroku.com/articles/heroku-cli)
- Terminal -> Project folder ->:
  - `heroku create APPNAME`
  - 'git add . & git commit -m 'my commit'
  - `git push heroku`

Building a Cheating game:

- Planning the page structure
- Planning the data structure
- Writing instructions
- Testing

Cheating game: page structure

- Instructions
- Decision (show them instructions again)
- Beliefs (if we are in `Beliefs` treatment)
- Results with payoff

Cheating game: data structure

We need to store only two pieces of info provided by a user:

- answer (integer number from 1 to 6)
- belief about an average answer of others (a float number from 1 to 6)

Cheating game: Writing instructions

We just borrow Follmi-Heusi instructions and adapt them:

https://academic.oup.com/jeea/article/11/3/525/2300098?login=true#58872029

We re-use instructions in several pages using `include` statement

```
{{ include 'dice/includes/instructions.html' }}
```

Cheating game: Testing

In addition to 'manual' testing we can write bots in oTree

```
class PlayerBot(Bot):
    def play_round(self):
        yield Intro,
        yield Decision, dict(answer=random.randint(1, 6))
        if self.session.config.get('beliefs'):
            yield Belief, dict(belief=random.randint(1, 6))
        yield Results
```

and run them either by `otree test dice` or by using browser bots. Include `use_browser_bots` option to the session config:

```
SESSION_CONFIG_DEFAULTS = dict(
    real_world_currency_per_point=1.00,
    participation_fee=0.00,
    doc="",
    use_browser_bots=False,
)
```

Anatomy of oTree pages

Each page (aka screen in z-Tree terminology) has three key elements:

- class
- html template
- position in a page sequence

Anatomy of oTree pages - two types of pages

- Page (aka *normal* page)
- WaitPage

Reaching the WaitPage in a page sequence, participants **wait** till all the members of their group or all the members of the entire experimental session reach the same page.

Anatomy of oTree pages - class

```
class Decision(Page):
    form_model = 'player'
    form_fields = ['answer']
    before_next_page = set_payoff


class Belief(Page):
    form_model = 'player'
    form_fields = ['belief']

    @staticmethod
    def is_displayed(player: Player):
        return player.session.config.get('beliefs', False)
```

Anatomy of oTree pages - class

Page has several built-in methods:

- is_displayed
- vars_for_template
- before_next_page

`WaitPage` has most of the methods above (with an exception of `before_next_page`) and a few others, the most important one is:

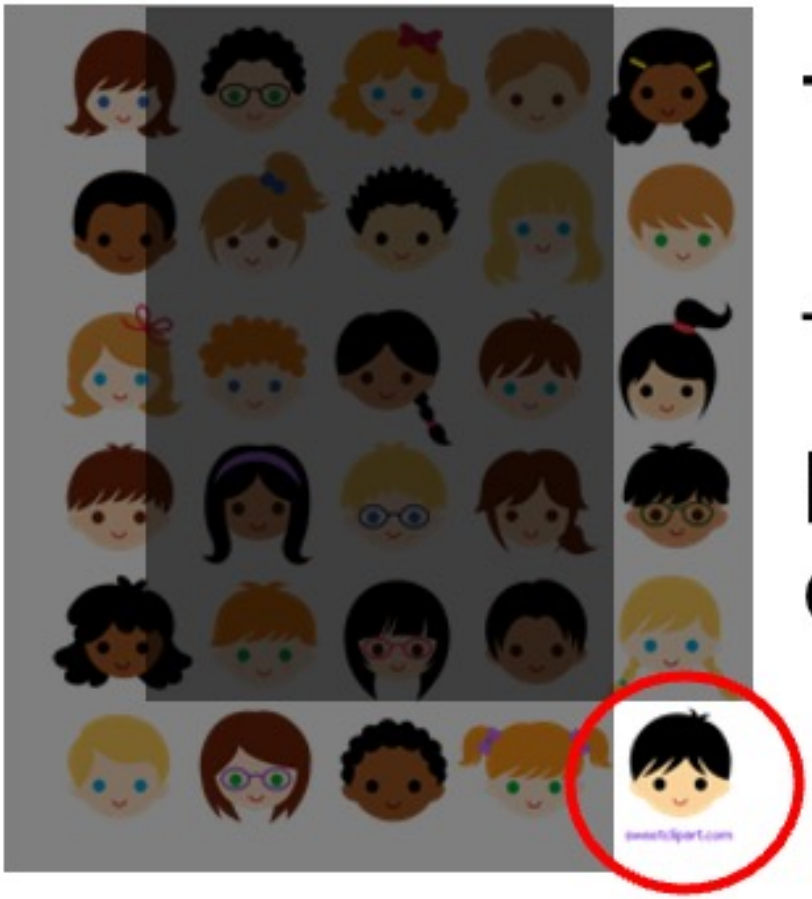
- after_all_players_arrive

Anatomy of oTree database - models

This is session.

The entire set of all participants in your lab or online

Anatomy of oTree database - models



This is participant.

The entire set of all participants in your lab or online

Anatomy of oTree database - models

Subsession is a set of all players in one round

Anatomy of oTree database - models

Player is an element of subsession

Anatomy of oTree database - models

One participant contains the info about all players who he/she 'owns'

Anatomy of oTree database - models

The group
is a set of players
in one particular
subsession

Anatomy of oTree database - models

One session can contain several apps

Chaining the apps:

```
SESSION_CONFIGS = [
    dict(
        name='dice_and_dg',
        display_name='Dice game + Dictator game',
        app_sequence=['dice', 'dg'],
        num_demo_participants=2,
    )
    ]
```

Anatomy of oTree database - models

Where to store the data? Subsession? Group? Player?

Rule of thumb: *put it to the level of the highest variability*

Example: Dictator game:

```
class Group(BaseGroup):
    kept = models.CurrencyField()


class Player(BasePlayer):
    age = models.IntegerField()
```

Anatomy of oTree database - models

Some field parameters may be set statically:

```
class Player(BasePlayer):
    age = models.IntegerField(min=18, max=100, label='How old are yo
    gender = models.StringField(choices=['Male', 'Female'],
                                widget=widgets.RadioSelect)
```

Anatomy of oTree database - models

The field parameters may be set dynamically:

```
import random
class Player(BasePlayer):
    voting  = models.StringField(label='Who will you vote for at 202

def voting_choices(player):
    choices = ['Republican', 'Democrat']
    random.shuffle(choices)
    return choices ## NB! this one will not store the order in the D
```

Other dynamically set options: `_max`, `_min`, `_error_message`, and PAGE `_error_message`

Anatomy of oTree - Constants

At the top of `__init__` file there is a `Constants` section. Use it as **SSOT**

SSOT (Single Source of Truth)

WRONG:

```
class Player(BasePlayer):
    send  = models.IntegerField(min=0, max=100,
label='How much you would like to send to charity (from 0 to 100)?')
```

RIGHT:

```
class Constants(BaseConstants):
    ...
    endowment = 100
class Player(BasePlayer):
    send  = models.IntegerField(min=0, max=Constants.endowment,
label=f'How much you would like to send to charity'
      f'(from 0 to {Constants.endowment})?')
```

Anatomy of oTree - Constants

Some built-in values of Constants include:

```
class Constants(BaseConstants):
    name_in_url = 'dictator'
    players_per_group = 2
    num_rounds = 1
    instructions_template = 'dictator/instructions.html'
    # Initial amount allocated to the dictator
    endowment = cu(100)
    dictator_role = 'Participant A'
    recipient_role = 'Participant B'
```

Dictator game - planning

- Two players game
- Asymmetric: two different roles (*Dictator* and *Recipient*)
- Payoffs:
  - Dictator decides how much to send out of endowment. The rest is his/her payoff

Dictator game - pages

- Intro
- Decision (for *Dictator*)
- Belief about Dictator's decision (for *Recipient&*)
- Waiting for Dictator's decision
- Results

Dictator game - pages

```
page_sequence = [
    Intro,
    Decision,
    Belief,
    ResultsWaitPage,
    Results
]
```

Dictator game - pages

```
class Decision(Page):
  form_model = 'group'
  form_fields = ['send']

  @staticmethod
  def is_displayed(player: Player):
      return player.role == Constants.dictator_role


class Belief(Page):
  form_model = 'group'
  form_fields = ['belief']

  @staticmethod
  def is_displayed(player: Player):
      return player.role == Constants.recipient_role


class ResultsWaitPage(WaitPage):
```

Dictator game - role assignment and other constants:

```
class Constants(BaseConstants):
    name_in_url = 'dictator'
    players_per_group = 2
    num_rounds = 1
    # Initial amount allocated to the dictator
    endowment = cu(100)
    dictator_role = 'Participant A'
    recipient_role = 'Participant B'
```

Dictator game - models:

```
class Group(BaseGroup):
    send = models.CurrencyField(
        doc="""Amount dictator sends to a Recipient""",
        min=0,
        max=Constants.endowment,
        label=f"I will send to {Constants.recipient_role}",
    )
    kept = models.CurrencyField()
    belief = models.CurrencyField()
```

Dictator game - payoff calculations:

```
def set_payoffs(group: Group):
    dictator = group.get_player_by_role(Constants.dictator_role)
    recipient = group.get_player_by_role(Constants.recipient_role)
    group.kept = Constants.endowment - group.send
    dictator.payoff = group.kept
    recipient.payoff = group.send
```

HTML templates and fields from database:

```
class Decision(Page):
    form_model = 'group'
    form_fields = ['send']
```

and in the html template:

```
{{ formfields }}
```

OR:

```
{{ formfield player.send }}
```

HTML templates:

```
{{ block title }}
  Decision
{{ endblock }}
{{ block content }}
  <div class="alert alert-info">You are {{player.role}}}</div>
  {{ formfields }}
  {{ next_button }}
  {{ include 'dg/includes/instructions.html' }}
{{ endblock }}
```

HTML templates:

Instead of rendering all the fields we can render a single one:

```
{{ block content }}
  <div class="alert alert-info">You are {{player.role}}</div>
  <h5>Try to guess how much <b>{{Constants.dictator_role}}</b> will
  {{ formfield player.belief label=''}}
  {{ next_button }}
  {{ include 'dg/includes/instructions.html' }}
  {{ endblock }}
```

HTML templates:

We can render any variable from the database or from `Constants`:

```
Your endowment was {{Constants.endowment}}.

{{Constants.dictator_role}} has sent you {{group.send}}.
```

HTML templates:

You can use loops to render lists and arrays:

```
class Constants(BaseConstants):
    payoffs = {1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 0}
```

```
<table class="table">
  <tr>
      {{ for i in Constants.payoffs.keys }}
      <td>{{i}}</td>
      {{endfor}}
  </tr>
  <tr>
       {{  for j in Constants.payoffs.values }}
      <td>{{j}}</td>
      {{ endfor }}
  </tr>
</table>
```

HTML templates:

You can use `if` conditions in templates:

```
<div class="my-3">
    {{ if player.role == Constants.dictator_role }}
    You decided to keep <strong>{{ group.kept }}</strong> for yourse
    {{ else }}
    {{Constants.dictator_role}} decided to keep <strong>{{ group.kep
    you got <strong>{{ group.send }}</strong>.
    {{ endif }}
</div>
```

References I

Berg, Joyce, John Dickhaut, and Kevin McCabe. 1995. "Trust, Reciprocity, and Social History." *Games and Economic Behavior* 10 (1): 122–42.

Fehr, Ernst, and Simon Gächter. 2000. "Cooperation and Punishment in Public Goods Experiments." *American Economic Review* 90 (4): 980–94.

Fischbacher, Urs, and Franziska Föllmi-Heusi. 2013. "Lies in Disguise—an Experimental Study on Cheating." *Journal of the European Economic Association* 11 (3): 525–47.

Güth, Werner, Rolf Schmittberger, and Bernd Schwarze. 1982. "An Experimental Analysis of Ultimatum Bargaining." *Journal of Economic Behavior & Organization* 3 (4): 367–88.

Kahneman, Daniel, Jack L Knetsch, and Richard H Thaler. 1986. "Fairness and the Assumptions of Economics." *Journal of Business*, S285–300.