Trust game

Chris Wickens

October 27, 2016

Contents

Let's create a Trust game, and learn some more features of oTree.

This is a trust game with 2 players. To start, Player 1 receives 10 points; Player 2 receives nothing. Player 1 can send some or all of his points to Player 2. Before P2 receives these points they will be tripled. Once P2 receives the tripled points he can decide to send some or all of his points to P1.

The completed app is here. Create the app

```
$ otree startapp my_trust
```

1 Define models.py

2 Constants

First we define our app's constants. The endowment is 10 points and the donation gets tripled.

```
class Constants(BaseConstants):
    name_in_url = 'my_trust'
    players_per_group = 2
    num_rounds = 1

    endowment = c(10)
    multiplication_factor = 3
```

3 Fields

Then we think about how to define fields on the data model. There are 2 critical data points to capture: the "sent" amount from P1, and the "sent back" amount from P2.

Your first instinct may be to define the fields on the Player like this:

```
class Player(BasePlayer):

    sent_amount = models.CurrencyField()
    sent_back_amount = models.CurrencyField()
```

The problem with this model is that $\text{sent}_{\text{amount}}$ only applies to P1, and $\text{sent}_{\text{backamount}}$ only applies to P2. It does not make sense that P1 should have a field called $\text{sent}_{\text{backamount}}$. How can we make our data model more accurate?

We can do it by defining those fields at the Group level. This makes sense because each group has exactly 1 $\text{sent}_{\text{amount}}$ and exactly 1 $\text{sent}_{\text{backamount}}$:

```
class Group(BaseGroup):

    sent_amount = models.CurrencyField()
    sent_back_amount = models.CurrencyField()
```

Even though it may not seem that important at this point, modeling our data correctly will make the rest of our work easier.

Let's let P1 choose from a dropdown menu how much to donate, rather than entering free text. To do this, we use the choices argument, as well as the $\text{currency}_{\text{range}}$ function:

```
sent_amount = models.CurrencyField(
    choices=currency_range(0, Constants.endowment, c(1)),
)
```

We'd also like P2 to use a dropdown menu to choose how much to send back, but we can't specify a fixed list of choices, because P2's available choices depend on how much P1 donated. I'll show a bit later how we can make this list dynamic.

Also, let's define the payoff function on the group:

```
def set_payoffs(self):
    p1 = self.get_player_by_id(1)
    p2 = self.get_player_by_id(2)
    p1.payoff = Constants.endowment - self.sent_amount + self.sent_back_amount
    p2.payoff = self.sent_amount * Constants.multiplication_factor - self.sent_back_amount
```

Define the templates and views

4 We need 3 pages:

- P1's "Send" page
- P2's "Send back" page
- "Results" page that both users see.

It would also be good if game instructions appeared on each page so that players are clear how the game works. Instructions.html

To create the instructions, we can define a file Instructions.html that gets included on each page.

```
{% load otree_tags staticfiles %}

<div class="instructions well well-lg">

    <h3 class="panel-sub-heading">
        Instructions
    </h3>
<p>
    This is a trust game with 2 players.
</p>
<p>
    To start, participant A receives {{ Constants.endowment }};
    participant B receives nothing.
    Participant A can send some or all of his {{ Constants.endowment }} to participant B.
    Before B receives these points they will be tripled.
    Once B receives the tripled points he can decide to send some or all of his points to A.
</p>
</div>
```

5 Send.html

This page looks like the templates we have seen so far. Note the use of {% include %} to automatically insert another template.

```
{% extends "global/Base.html" %}
{% load staticfiles otree_tags %}

{% block title %}
    Trust Game: Your Choice
{% endblock %}

{% block content %}

    {% include 'my_trust/Instructions.html' %}

    <p>
    You are Participant A. Now you have {{Constants.endowment}}.
    </p>

    {% formfield group.sent_amount with label="How much do you want to send to participant B?"

    {% next_button %}

{% endblock %}
```

We also define the view in views.py:

```
class Send(Page):

    form_model = models.Group
    form_fields = ['sent_amount']

    def is_displayed(self):
        return self.player.id_in_group == 1
```

The {% formfield %} in the template must match the $\text{form}_{\text{model}}$ and $\text{form}_{\text{fields}}$ in the view.

Also, we use $\text{is}_{\text{displayed}}()$ to only show this to P1; P2 skips the page. For more info on $\text{id}_{\text{in group}}$, see Groups and multiplayer games. SendBack.html

6 This is the page that P2 sees to send money back. Here is the template:

```
{% extends "global/Base.html" %}
{% load staticfiles otree_tags %}

{% block title %}
```

```
    Trust Game: Your Choice
{% endblock %}

{% block content %}

    {% include 'my_trust/Instructions.html' %}

    <p>
        You are Participant B. Participant A sent you {{group.sent_amount}}
        and you received {{tripled_amount}}.
    </p>

    {% formfield group.sent_back_amount with label="How much do you want to send back?" %}

    {% next_button %}

{% endblock %}
```

7 Here is the code from views.py. Notes:

We use $\text{vars}_{\text{fortemplate}}()$ to pass the variable $\text{tripled}_{\text{amount}}$ to the template. Django does not let you do calculations directly in a template, so this number needs to be calculated in Python code and passed to the template. We define a method $\text{sent}_{\text{back amount choices}}$ to populate the dropdown menu dynamically. This is the feature called $\{\text{field}_{\text{name}}\}_{\text{choices}}$, which is explained here: Dynamic form field validation.

```
class SendBack(Page):

    form_model = models.Group
    form_fields = ['sent_back_amount']

    def is_displayed(self):
        return self.player.id_in_group == 2

    def vars_for_template(self):
        return {
            'tripled_amount': self.group.sent_amount * Constants.multiplication_factor
        }

    def sent_back_amount_choices(self):
        return currency_range(
            c(0),
            self.group.sent_amount * Constants.multiplication_factor,
            c(1)
        )
```

8 Results

The results page needs to look slightly different for P1 vs. P2. So, we use the {% if %} statement (part of Django's template language) to condition on the current player's $\text{id}_{\text{in group}}$.

```
{% extends "global/Base.html" %}
{% load staticfiles otree_tags %}

{% block title %}
    Results
{% endblock %}

{% block content %}

{% if player.id_in_group == 1 %}
    <p>
        You sent Participant B {{ group.sent_amount }}.
        Participant B returned {{group.sent_back_amount}}.
    </p>
{% else %}
    <p>
        Participant A sent you {{ group.sent_amount }}.
        You returned {{group.sent_back_amount}}.
    </p>

{% endif %}

    <p>
    Therefore, your total payoff is {{player.payoff}}.
    </p>

    {% include 'my_trust/Instructions.html' %}

{% endblock %}
```

Here is the Python code for this page in views.py:

```
class Results(Page):

    def vars_for_template(self):
        return {
            'tripled_amount': self.group.sent_amount * Constants.multiplication_factor
        }
```

9 Wait pages and page sequence

This game has 2 wait pages:

P2 needs to wait while P1 decides how much to send P1 needs to wait while P2 decides how much to send back

After the second wait page, we should calculate the payoffs. So, we use $\text{after}_{\text{allplayersarrive}}$.

So, we define these pages:

```
class WaitForP1(WaitPage):
    pass

class ResultsWaitPage(WaitPage):

    def after_all_players_arrive(self):
        self.group.set_payoffs()
```

10 Then we define the page sequence:

```
page_sequence = [
    Send,
    WaitForP1,
    SendBack,
    ResultsWaitPage,
    Results,
]
```

11 Add an entry to $\text{SESSION}_{\text{CONFIGS}}$ in settings.py

```
{
    'name': 'my_trust',
    'display_name': "My Trust Game (simple version from tutorial)",
    'num_demo_participants': 2,
    'app_sequence': ['my_trust'],
},
```

12 Reset the database and run

```
otree resetdb
otree runserver
```

Then open your browser to `http://127.0.0.1:8000` to play the game.

Note: You need to run resetdb every time you create a new app, or when you add/change/remove a field in models.py. This is because you have new fields in models.py, and the SQL database needs to be re-generated to create these tables and columns.