Escuela de Invierno CESS 2017

Experimentos en oTree

CESS Santiago — Julio 25, 2017

David Klinowski

Centre for Experimental Social Sciences (CESS)
Nuffield College, University of Oxford
University of Santiago de Chile

Índice

1. ¿Qué es oTree?

2. El Shell y Python

3. Ejemplo: cuestionario simple

4. Ejemplo: juego del bien público

5. Bots de prueba

¿Qué es oTree?

¿Qué es oTree?

- plataforma para programar experimentos en las ciencias sociales
- basado en Django, un esquema para desarrollar aplicaciones web (apps que corren en el navegador, ej. ventas por internet, email por internet)
- software de código abierto, lenguaje Python
- corre en cualquier dispositivo con navegador (computador, tablet, celular)
- no requiere instalación de software en el dispositivo del participante
- se implementa por internet (sin red local) o por red local (sin internet)

¿Qué es oTree?

- excelente documentación: otree.readthedocs.io
- forum de discusión con +5 posts al día
- dado el uso de Python y Django en contextos más generales que experimentos, es posible que los problemas ya han sido solucionados y discutidos en internet

¿Qué es oTree?

- un experimento de oTree es una aplicación web creada bajo el esquema Django
- valor agregado de oTree: al momento de iniciar la programación, la estructura para el funcionamiento de la app ya está creada y solo debemos agregar los elementos particulares al experimento
  - lo que verán los participantes
  - la información que ingresarán los participantes
  - qué se hará con esa información

El Shell y Python

El Shell y Python

El Shell

- es un programa que recibe comandos de texto
- permite correr otros programas y navegar por el sistema de archivos del computador
- lo usaremos para instalar oTree, inicializar un experimento, actualizar base de datos y lanzar el servidor
- comandos últiles:
  - pwd: conocer ubicación actual
  - cd [*directorio*]: cambiar ubicación
  - flecha arriba: comando anterior
  - copiar-pegar directorio: escribir dirección más rápidamente

El Shell y Python

Python

- completaremos una adaptación del tutorial de python en documentación de oTree
- trabajaremos en un cuaderno interactivo Jupyter Notebook
- para esto debemos instalar Python 3 y Jupyter

El Shell y Python

- instalar Python 3
  - descargar Python 3 desde https://www.python.org/downloads/
- instalar Jupyter
  - abrir el Terminal (Mac) o Command Prompt (Windows)
  - $ *pip3 install –upgrade pip*
  - $ *pip3 install jupyter*

El Shell y Python

Para crear un Jupyter Notebook en el Desktop desde el Shell

- abrir el Terminal (Mac) o Command Prompt (Windows)
- navegar al Desktop
- $ *jupyter notebook*
- automáticamente se abrirá el navegador con el servidor de Notebook
- New/Python3

El Shell y Python

Alternativa: crear Jupyter Notebook en línea sin instalar software (suponiendo que el internet nos aguanta):

- ir a https://tmpnb.org/
- New/Python3

El Shell y Python

Alternativa: crear Jupyter Notebook desde PyCharm

- crear nuevo proyecto de Pure Python
- ir a Settings/Preferences $\rightarrow$ project:[nombre] $\rightarrow$ Project Interpreter $\rightarrow$ + $\rightarrow$ Jupyter
- File/New... $\rightarrow$ Jupyter Notebook
- ingresa código y presiona shift+enter
- dirígete al link en el mensaje de PyCharm, o posiblemente a http://127.0.0.1:8888/ en el navegador

Ejemplo: cuestionario simple

Para entender la estructura de un proyecto de oTree en general, hagamos un par de experimentos

Ejemplo: cuestionario simple

Ejemplo: cuestionario simple

Estructura del experimento:

- Página 1: participante ingresa su nombre y edad
- Página 2: se muestra su nombre y edad al participante

Ver *Demo Survey*.

Ejemplo: cuestionario simple

1. Inicializar proyecto oTree (directorio con esqueleto básico ya creado)

   - navegar en el Shell a la ubicación deseada
   - crear un proyecto llamado proyecto_cess
     - *otree startproject proyecto_cess*
     - Include sample games? (y or n): n

Ejemplo: cuestionario simple

2. Crear app que pregunta nombre y edad
   - ubicarse en el directorio creado
     - *cd proyecto_cess*
   - inicializar app llamada cuestionario
     - *otree startapp cuestionario*

Ejemplo: cuestionario simple

3. Abrir proyecto en PyCharm para comenzar a editar código
   - abrir PyCharm
   - Open
   - seleccionar carpeta proyecto_cess

Ejemplo: cuestionario simple

Para editar el código, cubriremos uno a uno los pasos en la *documentación oficial*.

En general para todo experimento son 4 las áreas a trabajar:

- *models.py*: define la base de datos
- templates: contiene cada página que mostraremos al participante
  - en html, no Python
  - no permite cálculos ni operaciones del tutorial Python
- *views.py*: define la lógica de presentación de las páginas
- *settings.py*: define la configuración general de la sesión

Ejemplo: cuestionario simple

models.py

Define models.py

Open `models.py` and scroll to the line that says `class Player(BasePlayer):`. Here we can define what fields will be stored in the database for each player. Let's add 2 fields:

- `name` (which is a `CharField`, meaning text characters)
- `age` (which is a positive integer field)

```
class Player(BasePlayer):
    name = models.CharField()
    age = models.PositiveIntegerField()
```

Ejemplo: cuestionario simple

Templates

Define the template

This survey has 2 pages:

- Page 1: players enter their name and age
- Page 2: players see the data they entered on the previous page

In this section we will define the HTML templates that users see.

So, let's make 2 HTML files under `templates/my_simple_survey/`.

Ejemplo: cuestionario simple

Templates

Let's name the first page `MyPage.html`, and put these contents inside:

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

{% block title %}
    Enter your information
{% endblock %}

{% block content %}

    {% formfield player.name with label="Enter your name" %}

    {% formfield player.age with label="Enter your age" %}

    {% next_button %}

{% endblock %}
```

Ejemplo: cuestionario simple

Templates

The second template will be called `Results.html`.

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

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

{% block content %}

    <p>Your name is {{ player.name }} and your age is {{ player.age }}.</p>

    {% next_button %}
{% endblock %}
```

Ejemplo: cuestionario simple

views.py

Define views.py

Now we define our views, which contain the logic for how to display the HTML templates.

Since we have 2 templates, we need 2 `Page` classes in `views.py`. The names should match those of the templates (`MyPage` and `Results`).

First let's define `MyPage`. This page contains a form, so we need to define `form_model` and `form_fields`. Specifically, this form should let you set the `name` and `age` fields on the player.

```
class MyPage(Page):
    form_model = models.Player
    form_fields = ['name', 'age']
```

- cada página es una clase distinta (Page o WaitPage)

Ejemplo: cuestionario simple

views.py

Now we define `Results`. This page doesn't have a form so our class definition can just say `pass`.

```
class Results(Page):
    pass
```

If `views.py` already has a `WaitPage`, you can delete that, because WaitPages are only necessary for multi-player games and more complex games.

- Waitpages también son útiles para controlar el paso de los participantes, aún en juegos no multi-player

Ejemplo: cuestionario simple

views.py

Then, set your `page_sequence` to `MyPage` followed by `Results`. So, all in all, `views.py` should contain this:

```
from otree.api import Currency as c, currency_range
from . import models
from ._builtin import Page, WaitPage
from .models import Constants


class MyPage(Page):
    form_model = models.Player
    form_fields = ['name', 'age']


class Results(Page):
    pass


page_sequence = [
    MyPage,
    Results
]
```

Ejemplo: cuestionario simple

settings.py

Define the session config in settings.py

Now we go to `settings.py` in the project's root directory and add an entry to `SESSION_CONFIGS`.

```
SESSION_CONFIGS = [
    {
        'name': 'my_simple_survey',
        'display_name': "My Simple Survey",
        'num_demo_participants': 3,
        'app_sequence': ['my_simple_survey'],
    },
    # other session configs go here ...
]
```

'app_sequence': ['cuestionario']

Ejemplo: cuestionario simple

Reset the database and run

Enter:

```
otree resetdb
otree runserver
```

Then open your browser to `http://127.0.0.1:8000` to try out the survey.

Ejemplo: juego del bien público

Ejemplo: juego del bien público

El juego:

- Grupos de 3 miembros
- Cada miembro recibe 100
- Cada miembro decide contribuir $0 \leq g_i \leq 100$ simultáneamente
- Cada miembro termina con $2G/3 + 100 - g_i$

Ejemplo: juego del bien público

Estructura del experimento:

- Se forman grupos de 3 miembros
- Página 1: participante decide cuánto contribuir
- Se calcula el pago final basado en contribuciones del grupo
- Página 2: participante es informado de su pago final

Ejemplo: juego del bien público

Como antes, trabajaremos en 4 aspectos:

- *models.py*
- templates
- *views.py*
- *settings.py*
- esta vez agregaremos bots de testeo, que nos simplificarán la vida infinitamente al momento de probar el programa

Ejemplo: juego del bien público

Crearemos una (otra) app dentro del proyecto ya inicializado *proyecto_cess*

- ubicarse en el directorio creado
  - *pwd* para asegurarnos de estar en proyecto_cess
- inicializar app llamada bien_publico
  - *otree startapp bien_publico*
- en PyCharm deberá aparecer la carpeta *bien_publico*

Editaremos el código cubriendo los pasos en la *documentación oficial*.

Ejemplo: juego del bien público

models.py

Define models.py

Open `models.py`. This file contains the game's data models (player, group, subsession) and constant parameters.

First, let's modify the `Constants` class to define our constants and parameters – things that are the same for all players in all games. (For more info, see Constants.)

- There are 3 players per group. So, change `players_per_group` to 3. oTree will then automatically divide players into groups of 3.
- The endowment to each player is 100 points. So, let's define `endowment` and set it to `c(100)`. (`c()` means it is a currency amount; see Money and Points).
- Each contribution is multiplied by 2. So let's define `efficiency_factor` and set it to 2:

Ejemplo: juego del bien público

models.py

Now we have:

```
class Constants(BaseConstants):
    name_in_url = 'my_public_goods'
    players_per_group = 3
    num_rounds = 1

    endowment = c(100)
    efficiency_factor = 2
```

Ejemplo: juego del bien público

models.py

Now let's think about the main entities in this game: the Player and the Group.

After the game is played, what data points will we need about each player? It's important to know how much each person contributed. So, we define a field `contribution`, which is a currency (see Money and Points):

```
class Player(BasePlayer):
    contribution = models.CurrencyField(min=0, max=Constants.endowment)
```

What data points are we interested in recording about each group? We might be interested in knowing the total contributions to the group, and the individual share returned to each player. So, we define those 2 fields:

```
class Group(BaseGroup):

    total_contribution = models.CurrencyField()
    individual_share = models.CurrencyField()
```

Ejemplo: juego del bien público

models.py

Now let's define a method that calculates the payoff (and other fields like `total_contribution` and `individual_share`). Let's call it `set_payoffs`:

```
class Group(BaseGroup):

    total_contribution = models.CurrencyField()
    individual_share = models.CurrencyField()

    def set_payoffs(self):
        self.total_contribution = sum([p.contribution for p in self.get_players()])
        self.individual_share = self.total_contribution * Constants.efficiency_factor / Constan
        for p in self.get_players():
            p.payoff = Constants.endowment - p.contribution + self.individual_share
```

self.individual_share = self.total_contribution * Constants.efficiency_factor / Constants.players_per_group

Ejemplo: juego del bien público

Templates

Define the template

This game has 2 pages:

- Page 1: players decide how much to contribute
- Page 2: players are told the results

In this section we will define the HTML templates to display the game.

So, let's make 2 HTML files under `templates/my_public_goods/`.

Ejemplo: juego del bien público

Templates

The first is `Contribute.html`, which contains a brief explanation of the game, and a form field where the player can enter their contribution.

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

{% block title %} Contribute {% endblock %}

{% block content %}

<p>
    This is a public goods game with
    {{ Constants.players_per_group }} players per group,
    an endowment of {{ Constants.endowment }},
    and an efficiency factor of {{ Constants.efficiency_factor }}.
</p>


{% formfield player.contribution with label="How much will you contribute?" %}

{% next_button %}

{% endblock %}
```

Ejemplo: juego del bien público

Templates

The second template will be called `Results.html`.

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

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

{% block content %}

<p>
    You started with an endowment of {{ Constants.endowment }},
    of which you contributed {{ player.contribution }}.
    Your group contributed {{ group.total_contribution }},
    resulting in an individual share of {{ group.individual_share }}.
    Your profit is therefore {{ player.payoff }}.
</p>

{% next_button %}

{% endblock %}
```

Ejemplo: juego del bien público

views.py

Define views.py

Now we define our views, which contain the logic for how to display the HTML templates. (For more info, see Views.)

Since we have 2 templates, we need 2 `Page` classes in `views.py`. The names should match those of the templates (`Contribute` and `Results`).

Ejemplo: juego del bien público

views.py

First let's define `Contribute`. This page contains a form, so we need to define `form_model` and `form_fields`. Specifically, this form should let you set the `contribution` field on the player. (For more info, see Forms.)

```
class Contribute(Page):

    form_model = models.Player
    form_fields = ['contribution']
```

Ejemplo: juego del bien público

views.py

Now we define `Results`. This page doesn't have a form so our class definition can be empty (with the `pass` keyword).

```
class Results(Page):
    pass
```

Ejemplo: juego del bien público

views.py

We are almost done, but one more page is needed. After a player makes a contribution, they cannot see the results page right away; they first need to wait for the other players to contribute. You therefore need to add a `WaitPage`. When a player arrives at a wait page, they must wait until all other players in the group have arrived. Then everyone can proceed to the next page. (For more info, see Wait pages).

When all players have completed the `Contribute` page, the players' payoffs can be calculated. You can trigger this calculation inside the the `after_all_players_arrive` method on the `WaitPage`, which automatically gets called when all players have arrived at the wait page. Another advantage of putting the code here is that it only gets executed once, rather than being executed separately for each participant, which is redundant.

Ejemplo: juego del bien público

views.py

We write `self.group.set_payoffs()` because earlier we decided to name the payoff calculation method `set_payoffs`, and it's a method under the `Group` class. That's why we prefix it with `self.group`.

```
class ResultsWaitPage(WaitPage):

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

Now we define `page_sequence` to specify the order in which the pages are shown:

```
page_sequence = [
    Contribute,
    ResultsWaitPage,
    Results
]
```

Ejemplo: juego del bien público

settings.py

Define the session config in settings.py

Now we go to `settings.py` in the project's root directory and add an entry to `SESSION_CONFIGS`.

```
SESSION_CONFIGS = [
    {
        'name': 'my_public_goods',
        'display_name': "My Public Goods (Simple Version)",
        'num_demo_participants': 3,
        'app_sequence': ['my_public_goods', 'survey', 'payment_info'],
    },
    # other session configs ...
]
```

'app_sequence': ['bien_publico']

Ejemplo: juego del bien público

Reset the database and run

Enter:

```
$ otree resetdb
$ otree runserver
```

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

Ejemplo: juego del bien público

Para combinar apps cuestionario y bien_publico:

- En *settings.py*:
  - 'app_sequence': ['cuestionario', 'bien_publico']

Bots de prueba

Bots de prueba

- un bot de prueba simula a los participantes
- permite testear la sesión automáticamente en segundos
    - testear manualmente es una piña bajo el brazo
- útil para chequear que la sesión termina sin errores y para verificar que la data se genera de manera esperada de acuerdo a decisiones programadas

Bots de prueba

Pasos:

- en *tests.py* agregar una acción para cada campo que el participante debe proveer
- en *settings.py* agregar *'use_browser_bots': True*
- *$ otree resetdb*
- *$ otree runserver*

Bots de prueba

tests.py de la app *cuestionario*

```python
class PlayerBot(Bot):

    def play_round(self):
        yield (views.MyPage, {'name': 'El Bot', 'age': 18})
        yield (views.Results)
```

Bots de prueba

.py de la app *bien_publico*

```python
class PlayerBot(Bot):

    def play_round(self):

        if self.player.id_in_group == 1:
            yield (views.Contribute, {'contribution': c(100)})

        elif self.player.id_in_group ==2:
            yield (views.Contribute, {'contribution': c(50)})

        else:
            yield (views.Contribute, {'contribution': c(0)})

        yield (views.Results)
```

Bots de prueba

settings.py

```
SESSION_CONFIGS = [
    {
        'name': 'primer_cuestionario',
        'display_name': 'Mi primer cuestionario',
        'num_demo_participants': 3,
        'app_sequence': ['cuestionario', 'bien_publico'],
        'use_browser_bots': True
    }
]
```