

oTree (2/3)

고려대 실험경제 세미나
2017.8.10 @ 강남토즈

준비물

- Python3 이 설치되어 있는 컴퓨터
 - Windows 10, bash 설치 권장
 - bash 설치: <http://sanghaklee.tistory.com/39>
 - Python3 설치: (bash상에서) `$ sudo apt-get install python3`
- 인터넷 가능해야 함
- 따라해보기 위한 환경임. 필수요건이 아님!

Public Good Game

첫번째 otree 프로젝트 만들어보기

PGG

- 3명이 한 그룹을 이룸
- 그룹내 각 참가자들은 100만원의 가상화폐를 지급받음
- $MPCR = 1 = 100\%$ (즉, 2배가 된다는 것을 의미)
 - $\text{efficiency factor} = 1 + MPCR = 2$
- 참가자들은 자신의 endowment를 자신이 보유하는 금액과 기여할 금액으로 쪼갬
 - 그룹내 모든 기여금을 합하여 efficiency factor 를 곱하여 $1/n$ 으로 나눠줌

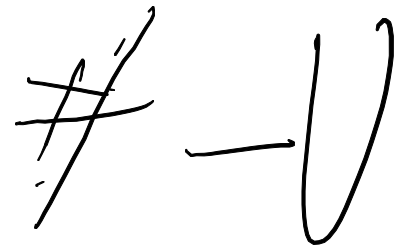
Big Picture

- Preparation
- models.py
- Template/*.html
- views.py

Reference

- <http://otree.readthedocs.io/en/latest/tutorial/part1.html>
- 위 링크를 따라서 step by step으로 따라해봅시다.

Upgrade oTree

- 실제 실험 진행중에는 신중할 것 (업그레이드 되어 의도치 않은 현상이 발생할 수 있음)
- `$ pip3 install --upgrade otree-core` 
- `$ otree resetdb`

앱 만들기

- 우선, 자신이 oTree 프로젝트를 설치할 폴더로 이동할 것
- `otree startproject [[프로젝트 이름]]`
 - `$ otree startproject otree_exercise`
 - Include sample games? \Rightarrow no!
- 프로젝트 폴더로 이동 (requirements_base.txt 가 있는 폴더임)
 - `otree startapp [[앱이름]]`
 - `$ otree startapp pgg`

.

models.py

models.py 편집

- 에디터에서 [[앱이름]]/models.py를 연다
- 필수 class들은 이미 적혀 있음
 - pass: 정의만 하고 아무 일도 하지 않는다는 의미
- BaseConstants, BaseSubsession..
- otree-core에 미리 정의되어 있는 기본 값들을 상속받음.

```
from otree.api import (  
    models, widgets, BaseConstant  
    Currency as c, currency_range  
)
```

```
author = 'Your name here'
```

```
doc = """  
Your app description  
"""
```

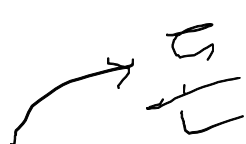
```
class Constants(BaseConstants):  
    name_in_url = 'pgg'  
    players_per_group = None  
    num_rounds = 1
```

```
class Subsession(BaseSubsession):  
    pass
```

class Constants

```
class Constants(BaseConstants):  
    name_in_url = 'pgg' # url에 보일 이름  
    players_per_group = 3 # 그룹당 플레이어수  
    num_rounds = 1 # 실행 총 라운드수  
    endowment = c(100) # endowment  
    efficiency_factor = 2 # MPCR
```

#유동적으로 하고 싶으면 1000정도로
하고 중간에 조건을 걸어 끝냄



Tip: 주석은 꼼꼼히 달아주는 것이 좋음

class Player

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

- contribution: 기여금의 양
 - 자신이 보유할 금액은 endowment - contribution 으로 계산하면 되기 때문에 별도의 변수를 둘 필요는 없음
 - 하지만 명시적으로 보고자 하면 두어도 됨.
- CurrencyField: 정수로 두어도 되지만 보상 지급을 위해 otree에서 정의한 게임화폐단위를 쓸 것임을 의미
 - min= .. max= ... 게임내 설정값. (method 정의에 포함되어 있을 때에만 작동함. 해당 method 참고)
 - Constants.endowment :: Constants class 의 member인 endowment 값을 부른 것임.

class Group [[1]]

```
class Group(BaseGroup):
```

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

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

↳ 들여쓰기

- 그룹 차원의 계산을 위해 필요한 변수들의 정의
- total_contribution : 그룹내 기여금의 총합
- individual_share : 그룹내 기여금 * efficiency_factor / n
- 이 역시 원한다면 별도의 변수 정의 없이 코딩 가능하지만 여기에 서는 명시적으로 보기 위해 별도 변수로 지정한 것이라 볼 수 있음.

class Group [[2]]

```
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 /  
                Constants.players_per_group  
        for p in self.get_players():  
            p.payoff =  
                (Constants.endowment - p.contribution +  
                 self.individual_share)
```

Templates

Define the Template

- 화면을 구성할 페이지의 레이아웃을 결정
- 값을 참조하는 것은 가능하지만..
 - 계산 등 실질적인 일들은 `views.py` 에서 해야 함
- PGG에서 필요한 화면 (**: 만들어야 하는 템플릿)
 - 게임설명 + 전략선택화면 ** (Page 1)
 - 그룹원들을 기다리는 중에 출력할 화면 → 사전에 정의되어 있음
 - 결과 및 보상 설명화면 ** (Page 2)

→ Same class name view.py
in

Page1: Contribute.html

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

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

{% block content %}
```

{% ... %}: block
{{ ... }}: 참조
django 전용

<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 multiplier }}.

</p>

그 외에는 표준
HTML임

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

→ input in html

efficiency factor

To make Template

- Base: HTML, CSS 관련 지식 필요
 - 동적인 효과 (리프레시 없이 작동하는 부분) 를 만
들고자 하는 경우는 Javascript 필요
- Django 블록 등의 규약에 대해서는 django 문서 참
고
 - <https://docs.djangoproject.com/en/1.8/ref/templates/language/>
 - oTree Template manual: <http://otree.readthedocs.io/en/latest/templates.html#templates>

Page2: Result.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 %}
```

views.py

개요

- 실질적인 actions를 관장
- 앞에서 구성한 template를 이용하여 Page들을 정의하고 models.py 와 views.py 에서 정의한 함수들을 언제 어떻게 사용할 것인지를 규정해야 함.

views.Contribute

```
class Contribute(Page):  
  
    # model의 Player에 대한 form임을 선언  
    form_model = models.Player  
    # Player class의 contribution을 form  
    # 입력으로 받을 것임을 선언  
    # 따라서 models.Player에 contribution  
    # 이라는 이름의 변수가 정의되어 있어야 함.  
    form_fields = ['contribution']
```

- form
 - 입력을 받는 Page
 - 여기에서의 입력값을 model에 넣거나 다음 page로 넘기는 등의 진행과 관련된 action 수행
- class 이름은 Template의 file 이름과 동일해야 함.
- <http://otree.readthedocs.io/en/latest/forms.html#forms>

views.ResultsWaitPage

```
class ResultsWaitPage(WaitPage):  
    def after_all_players_arrive(self):  
        self.group.set_payoffs()
```

built-in



- `after_all_players_arrive()` 함수는 oTree에서 모든 플레이어 결정이 끝났을 때 자동실행되는 함수임
- 모든 플레이어 결정이 끝나고 나면 models의 Group에서 정의했던 `set_payoffs()`를 실행함을 의미
- 그 외 나머지 부분은 WaitPage라는 oTree에서 미리 만들어 둔 template를 사용
- <http://otree.readthedocs.io/en/latest/multiplayer/waitpages.html#wait-pages>

views.Results

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

- 실질적으로 결과를 보여주는 것 말고는 하는 일이 없으므로 pass 하는 것임

views.page_sequence

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

- 앞에서 정의했던 Page class
들의 순서를 정의
- 반드시 정의되어 있어야 함.

Global Setting

settings.py

```
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 ...  
]
```

- 세팅에 필요한 항목들을 정의
 - <http://otree.readthedocs.io/en/latest/settings.html>
 - app_sequence
 - 만든 앱들의 순서를 정의
 - 앱을 독립적으로 만들면 여러 개의 SESSION_CONFIG 들을 만들어 다양한 버전의 실험을 진행할 수 있음
- survey, payment_info app은 기본으로 저장되어 있음.

Demo Test

Reset the database & Run

- \$ otree resetdb
 - models 에 오류가 있을 경우 이 단계에서 에러가 나옴
- \$ otree runserver
 - 그 밖의 실행단계 오류가 있을 경우 이 단계에서 에러가 나옴
- —> 브라우저에서 localhost:8000 을 입력하면 볼 수 있음.

Debugging

- 문법적이거나 실행 단계까지 나아갔다 하더라도 오류가 발생할 가능성은 있음
- 오작동할 경우 브라우저는 충분한 정보를 내지 않음
 - 특히 HTML은 오류가 있을 경우 조용히 작동정지하도록 설계되어 있음.
- 따라서 의도대로 되고 있는지 확인할 필요가 있음.

Simple Debugging

```
def set_payoffs(self):  
    (...)  
    for p in self.get_players():  
        p.payoff = (...)  
        print('*****p.payoff is',  
p.payoff)
```

- 가장 간단한 디버깅 방법
- 제대로 계산하고 있는지 `print()` 함수를 사용하여 궁금한 변수들을 터미널에 출력하게 함.
- `runserver` 명령을 실행한 터미널에 `print()` 의 메시지가 출력됨
 - 브라우저의 console에 출력하게 하는 방법도 있으나 이 경우 client가 확인할 수 있으므로 권장하지 않음.

코드수정

- 심각한 문제가 없었다면, localhost:8000 에서 demo oTree page가 작동할 것임
- models, views를 수정할 경우 otree resetdb, otree runserver를 다시 해야 함
- 하지만 template를 수정할 경우에는 위 과정 거칠 필요 없음 (단순 출력 레이아웃일 뿐이므로)
 - 그냥 브라우저에서 refresh 하면 바로 반영됨

실습

- 조금씩 바뀌가며 변화가 어떻게 적용되는지 확인해 봅시다!

마지막 주제

- Production
- 실제 실험을 진행하기 위해 필요한 것들
 - server setting 하기
 - production server 준비하기
- Rooms
- 그 외..
 - 실험 진행과 관련한 실질적인 준비들
 - IRB
 - 모집절차

수고하셨습니다!