Groups

Remixed from material by Ali Seyhun Saral & Philipp Chapkovski

Groups

- Homogenous groups
  - Every group member will have the same role in the game
  - Examples: Market games, Public good games, Prisoners Dilemma etc
- Heterogenous groups
  - Players have different roles
  - Examples: Trust game, Games with asymmetries etc

Reminder: Groups in oTree

The group is a set of player in one particular subsession

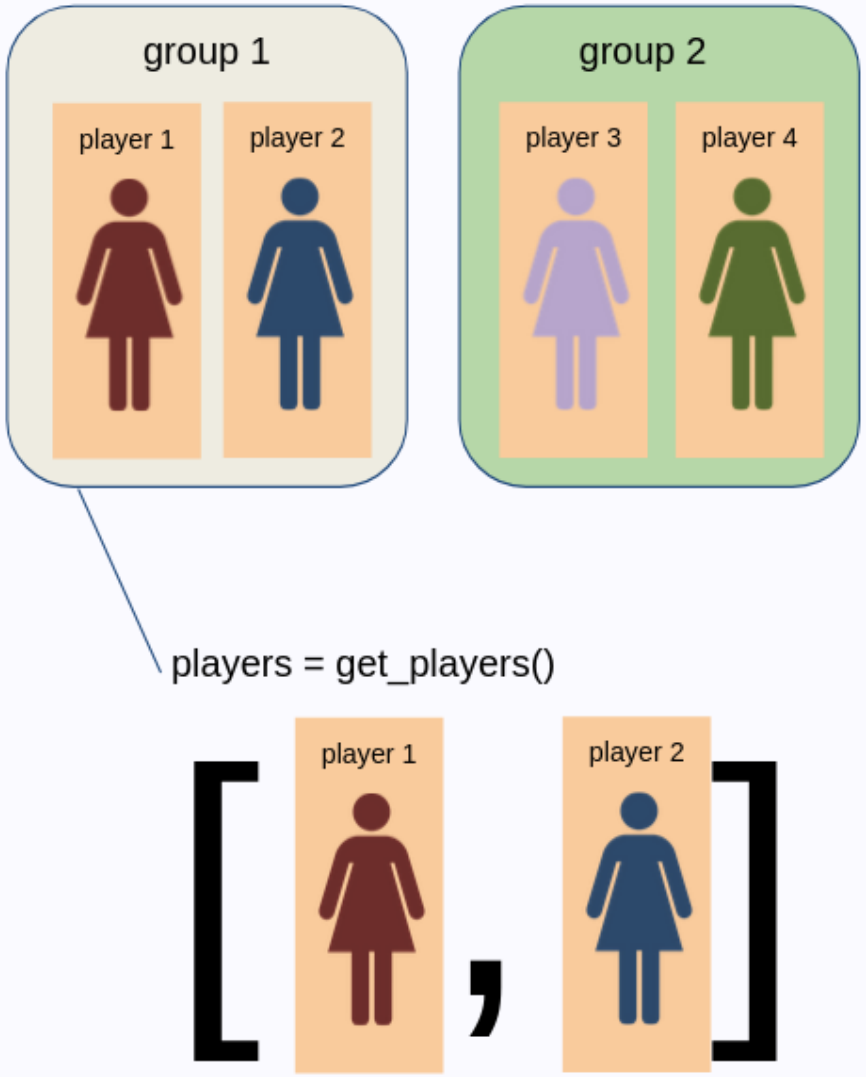
Group structure in oTree

```
class Constants(BaseConstants):
    name_in_url = 'some_name'
    players_per_group = None
    num_rounds = 2
```

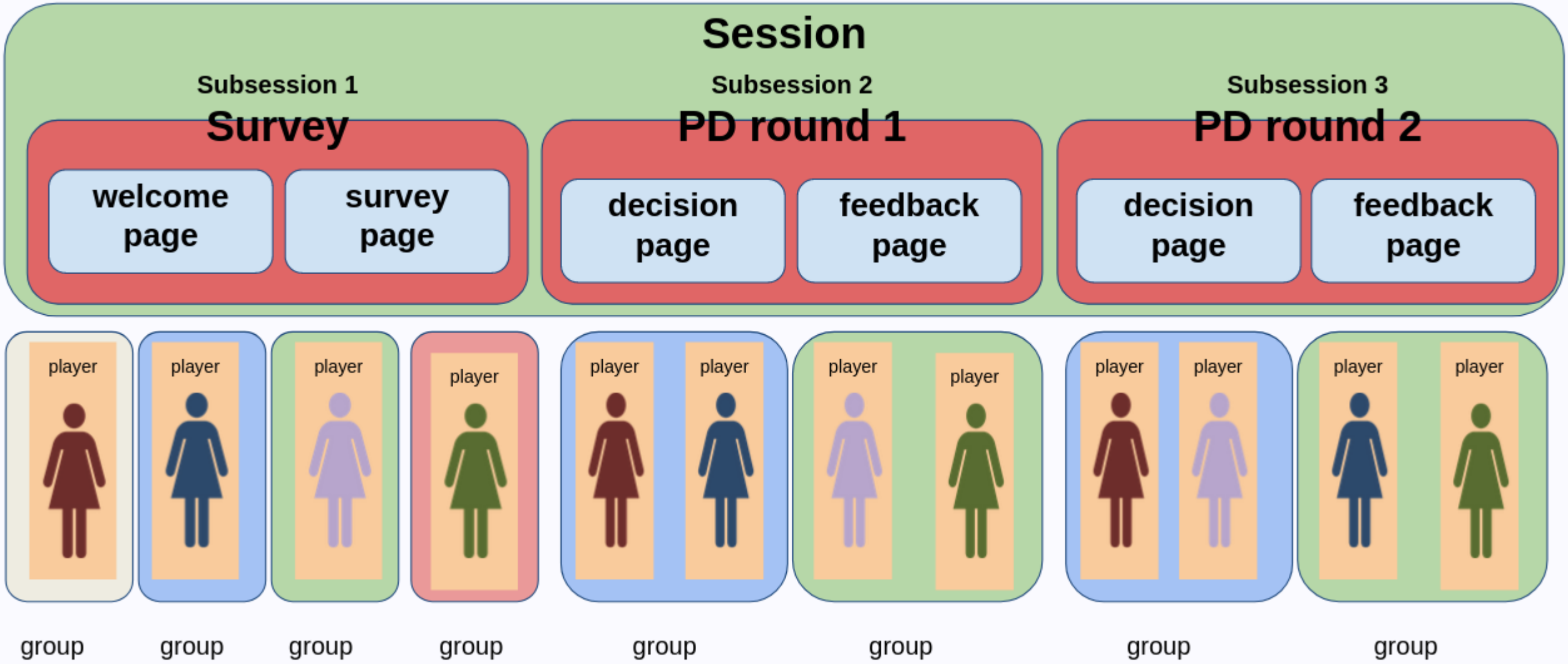
```
class Group(BaseGroup):
    pass
```

- Specific methods on group objects
  - get_players() and get_player_by_id()
- Specific methods on player object related to the group
  - get_others_in_group() and get_others_in_subsession()
- Methods on subsession level to handle the group creation/matching
- Waitpages in pages.py to process group information
  - after_all_players_arrive() and others

Group methods example



Groups can change within the app



Groups in oTree

Plan:

1. Homogenous groups
2. Heterogenous groups
3. Multiple rounds and matching

Homogenous groups

Running example: Cournot market game

- Fischer & Normann (2019) as a running example
  - Two firm market pick in each round quantity q_i
  - Linear production costs $C_i(q_i) = \theta_i q_i$
  - Price is given by $p(Q) = \max\{\alpha - Q, 0\}$ with $Q = q_1 + q_2$
  - Profit given by $\pi_i = q_i * P - C_i(q_i)$
- $\alpha = 91, \theta = 25, q_i \in \{0,1,..45\}$
- We consider only one round
  - Extension to multiple rounds in the next session

Cournot game: models.py

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

    # Parameters from the paper
    alpha = 91
    theta_H = 25

    # Total production capacity of all players
    total_capacity = 90
    max_units_per_player = int(total_capacity / players_per_group)

    instructions_template = 'cournot/instructions.html'
```

Why?

- Player_per_group is a built-in constant
- Can also be used for calculations to make parts of the experiment more general

Cournot game: models.py

```
class Player(BasePlayer):

    units = models.IntegerField(
        min=0,
        max=Constants.max_units_per_player,
        label=f"How many units will you produce (from 0 to 45)?"
    )

    costs = models.IntegerField()

    def get_others_units(self):
        others = self.get_others_in_group()
        return [o.units for o in others]

    def calculate_costs(self):
        self.costs = Constants.theta_H * self.units
```

- Each player can choose a quantity
- Costs saved to the database and calculated according to cost function
- Additional helper function to retrieve the quantity of the other player(s) in the group
  - Note that player object does not have information by itself on the group members but it has to be retrieved from the “higher” object

Cournot game: models.py

```
class Group(BaseGroup):

    unit_price = models.CurrencyField()

    total_units = models.IntegerField()

    def set_payoffs(self):
        players = self.get_players()
        self.total_units = sum([p.units for p in players])
        self.unit_price = max(Constants.alpha - self.total_units, 0)
        for p in players:
            p.calculate_costs()
            p.payoff = self.unit_price * p.units - p.costs
```

- Group is the market level here:
  - Market price
  - Total industry production
- Payoff has to be calculate now on the group level as it depends on the choice if all group members
  - Where to run this method?

Cournot game: pages.py

```
class Introduction(Page):
  pass

class Decide(Page):
   form_model = 'player'
   form_fields = ['units']


class ResultsWaitPage(WaitPage):
   body_text = "Waiting for the other participant to decide."
   after_all_players_arrive = 'set_payoffs'


class Results(Page):
 def vars_for_template(self):
    return dict(other_player_units=self.player.get_others_units())


page_sequence = [Introduction, Decide, ResultsWaitPage, Results]
```

- Waitpages are necessary when one player needs to wait for others to take some action before they can proceed
  - Total quantity cannot be calculated unless all firms have decided
- oTree waits until all players in the group have arrived at that point in the sequence
- after_all_players_arrive lets you run some calculations once all players have arrived
  - Use a method that you define on Group class
- No template file but you can change the body_text

Cournot game: Templates Results.html

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

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

    {% block content %}

      <p>The results are shown in the following table.</p>

      <table class="table">

        <tr>
        <td>Your firm produced:</td>
        <td>{{ player.units }} units</td>
      </tr>


        {% for o_units in other_player_units %}
      <tr>
        <td> Other Firm {{ forloop.counter }} </td>
        <td> {{o_units}} units </td>
      <tr>
    {% endfor %}

        <tr>
      <td>Total production:</td>
      <td>{{ group.total_units }} units</td>
    </tr>
                ...
                ...
                ...
  </table>

      {% next_button %}

      {% include Constants.instructions_template %}
{% endblock %}
```

- Group variables easy to access in the same way as player variables
- Different way to display other group information in a flexible way

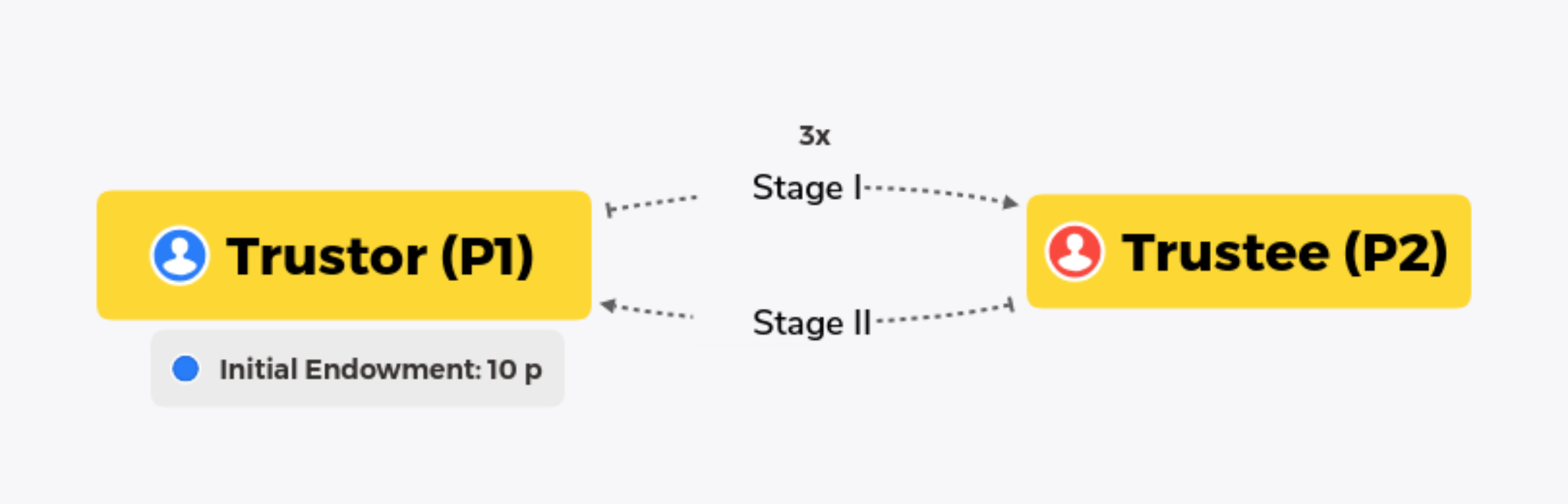
Cournot game

Show/Run the App

Show Database

Heterogenous Groups

Example: Trust game



Example adapted from oTree documentation

Trust game: models.py

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

    instructions_template = 'trust/instructions.html’

   # Initial amount allocated to each player
    endowment = c(100)
    multiplier = 3

    # {name}_role
    sender_role = 'Sender'
    receiver_role = 'Receiver'
```

- If each group has multiple roles, such as buyer/seller, principal/agent, etc., you can define them in Constants
- Roles defined by {name}_role
- oTree automatically assigns each role to a different player (sequentially according to id_in_group)
  - Show in DB
- Can be access by {{ player.role }} in template
- Alternative: Use directly id_in_group

Trust game: models.py

- Two key variables need to be defined:
  - “Sent” amount
  - “Sent back” amount
  - Where to define those?
- Naïve solution:
  - Define in player class
  - Problem: Data model is not accurate as each variables only applies to specific player role
- Better solution:
  - Define fields on the Group level

Trust game: models.py

```
class Group(BaseGroup):
    sent_amount = models.CurrencyField(
        min=0,
        max=Constants.endowment,
        label="Please enter an amount from 0 to 100:",
    )

    sent_back_amount = models.CurrencyField(min=c(0))

    def sent_back_amount_max(self):
        return self.sent_amount * Constants.multiplier

    def set_payoffs(self):
        p1 = self.get_player_by_role(Constants.sender_role)
        p2 = self.get_player_by_role(Constants.receiver_role)
        p1.payoff = Constants.endowment - self.sent_amount + self.sent_back_amount
        p2.payoff = self.sent_amount * Constants.multiplier - self.sent_back_amount
```

- Define fields on Group level
- Maximum of the *sent_back_amount* determined dynamically
  - Like {field}_choices in RPM Example
- Payoff calculated on group level
  - Use get_player_by_role()/ get_player_by_id() method to retrieve sender/receiver
  - We use the built-in payoff field to store the earnings
  - Set_payoff() must be used on a waitpage again

Trust game: models.py

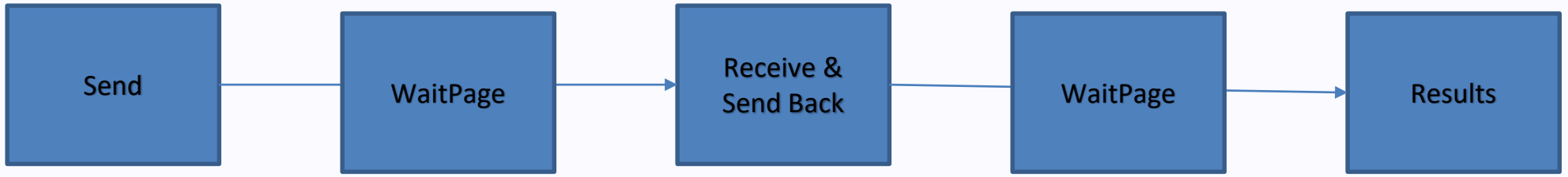
```
class Player(BasePlayer):
    pass

class Subsession(BaseSubsession):
    pass
```

- Everything defined on group level here
- No repeated interaction or different treatments
  - Else add it to create_session() method of the Subsession

Trust game: Pages

- SendPage should only be visible for type=Sender
- ReceivePage only relevant for type=Receiver
  - Use is_displayed() method
- ReceivePage can only be completed after sender made a choice
- Results should be shown to both types but only after SendPage and ReceivePage have been completed by the respective type

Trust game: WaitPages



Trust game: pages.py

```
class Send(Page):
    """This page is only for P1
    P1 sends amount (all, some, or none) to P2
    This amount is tripled by experimenter,
    i.e if sent amount by P1 is 5, amount received by P2 is 15"""

    form_model = 'group'
    form_fields = ['sent_amount']

    def is_displayed(self):
        player = self.player
        return player.role == Constants.sender_role


class SendBackWaitPage(WaitPage):
    pass
```

- Note that we use form_model = ‘group’ as the fields have been defined on the group level
- Player.role was assigned automatically by id_in_group
  - It is NOT a method, e.g. **don’t do player.role()**
  - Note that we have to do self.player.xyz
    - Remember object hierarchy from the introduction
- SendBackWaitPage as experiment can only progress after Sender made choice

Trust game: pages.py

```
class SendBack(Page):
    """This page is only for P2
    P2 sends back some amount (of the tripled amount received) to P1"""“

    form_model = 'group'
    form_fields = ['sent_back_amount']

        def is_displayed(self):
        player = self.player
        return player.role == Constants.receiver_role

        def vars_for_template(self):
        tripled_amount = self.group.sent_amount * Constants.multiplier

            return dict(
            tripled_amount=tripled_amount,
        )

class ResultsWaitPage(WaitPage):
    after_all_players_arrive = 'set_payoffs'

class Results(Page):
    """This page displays the earnings of each player"""“

    def vars_for_template(self):
        return dict(tripled_amount=self.group.sent_amount * Constants.multiplier)

page_sequence = [
    Introduction, Send, SendBackWaitPage, SendBack, ResultsWaitPage, Results,
]
```

```
class Group(BaseGroup):
…

 def set_payoffs(self):
        p1 = self.get_player_by_role(Constants.sender_role)
        p2 = self.get_player_by_role(Constants.receiver_role)
        p1.payoff = (Constants.endowment -
      self.sent_amount + self.sent_back_amount)

        p2.payoff = (self.sent_amount * Constants.multiplier -
       self.sent_back_amount)
```

Trust game: Templates

- No Template for Waitpages
- Else everything as before

Cournot game

Show/Run the App

Show Database