Carbon Emission Task

Release -

Noel Strahm

Sep 29, 2021

CONTENTS

Carbon Emission Task
A SOCIAL EXPERIMENT WITH REAL
CO₂ CONSEQUENCES

CHAPTER

ONE

ABOUT

- The Carbon Emission Task (CET) is a new paradigm grounded in experimental economics to assess actual pro-environmental behavior.
- The CET can be used in laboratory, online, or classroom studies.
- Participants face repeated tradeoffs between financial bonus opportunities paired with real carbon emissions and foregoing such opportunities while staying carbon-neutral.
- Measuring pro-environmental behavior using the Carbon Emission Task (Berger & Wyss, 2021) was published in the *Journal of Environmental Psychology*.

CHAPTER TWO

SUPPORT

For help, please contact noel.strahm@iop.unibe.ch

CHAPTER THREE

CONTENTS:

3.1 Set up the CET in oTree

This is how you import the Carbon Emission Task into your oTree framework.

3.1.1 Step 1: Download the CET

Download the CET folder from Github/NoelGit95/CET_doc.

3.1.2 Step 2: Add the CET to your oTree Project

Copy the CET folder into your oTree project directory like this:

— Your_oTree_Project

- otree_app1
- otree_app2

..

- cet_light
- manage.py
- requirements.txt
- settings.py

3.1.3 Step 3: Change session configurations in settings.py

Include the CET app in your settings.py file. To do this append the following code in the SESSION_CONFIGS list.

3.1: Dictionary

```
dict(
    name='cet_light',
    display_name="CET Light Version",
    num_demo_participants=40,
    app_sequence=['cet_light'],
)
```

Note: Name Matching: Like with any oTree app the specified name in the dict() must match the name of the project's folder. So make sure to change both the folder name as well as the `name=''` in the dict() in case you want to name the project any other way than *cet_light*.

3.2: Currency

The *CET Data (csv file)* provides the values for a player's bonus. The currency used is the Great British Pound. If you want to use a different currency, the data in the cet_data file has to be adjusted accordingly. If no changes are made to the cet_data file the currency parameters have to be specified as such:

```
# e.g. EUR, GBP, CNY, JPY
REAL_WORLD_CURRENCY_CODE = 'GBP'
USE_POINTS = False
```

3.1.4 Step 4: Install required packages

4.1: Download packages

The following packages that are not in the Python Standard Library are required: requests

4.2: Add packages to requirements.txt

Add the following line to the `requirements.txt` file:

```
requests~=2.25.0 #Or higher version if this one is outdated.
```

3.1.5 Step 5: Start the CET

All Done. You can start the cet app like any other otree app by going to your oTree project directory and starting the oTree devserver.

```
otree devserer
```

Click Next for further explanations of the cet_light folder's contents.

3.2 CET Data (csv file)

The cet_data.csv file provides the player's *Carbon*, *Car Miles* and *Bonus* values. There are 20 unique combinations of carbon, car miles and bonus values in the file. Those rows are repeated once, yielding a total of 40 data rows. Each data row supplies data for a CET round, hence the number of data rows is equal to the number of rounds in the CET (40).

Experimenters can extend the csv file with more data rows, which in turn increases the number of rounds in the CET. Furthermore, the data values can be modified, if experimenters want to explore different framework conditions for the CET. For example a different currency other than GBP, different carbon values, higher or lower bonuses or different analogies other than car miles to show the carbon footprint of each decision.

If experimenters want to modify the carbon and car mile values they are encouraged to use the *Car Distance Calculator* within the project folder.

3.3 Car Distance Calculator

The car_distance_calculator.xlsx file provides a usable calculator that determines how far an average UK car can drive, given a certain amount of CO2 emission. The calculator can be used if experimenters decide to choose a different weight unit other than pound to display the *Carbon* values or use kilometers instead of miles to display the *Car Miles* values.

The calculator provides a C02 emission to car miles / kilometers conversion for the following weight units: Milligrams (mg), Grams (g), Kilograms (kg), Tons (t), Pounds (lbs), Ounzes (oz) and Stones (st). The weight of the CO2 emission is then converted to miles and kilometers driven in an average UK car using petrol as fuel.

A more detailed description on how the calculator works is provided within the car_distance_calculator.xlsx file.

All carbon and car mile values must be changed in the *CET Data (csv file)* in order to change the values during the experiment.

Some further modifications to the CET can be made in the *Constants* class of the models.py file.

3.4 Constants

The following Constants can be set in the Constants class:

3.4.1 Random Payoff

- `random_payoff` is used to define your preferred payoff calculation for the `player.payoff` field.
- If set to `True` a player's payoff is only calculated in the `paying_round`. The payoff in all other rounds is 0. See *Paying Round*
- If set to `False` a player's payoff is calculated in all rounds.

3.4.2 Random Saved Emission

- `random_saved_emission` is used to define how the `Subsession.sum_saved_emission` field is calculated. See *Sum saved Emission*
- If set to `True` the sum is calculated by adding each player's saved emission in the `paying_round`.
- If set to `False` the saved emission for all rounds is added.

3.4.3 Bot Criteria

- `Bot_criteria` is used to define the criteria as to when a player is seen as a bot.
- `Bot_criteria` must be a value between 0-1 that represents the percentage of autonomous decisions a player has to make in order to be seen as human.
- Example: The definition below means a player must decide autonomously (No timeout happened) in at least 75% of all rounds. Otherwise the player is regarded as a bot.

```
Bot_criteria = 0.75
```

3.4.4 Num Rounds

- This is an oTree specific variable that defines the number of rounds of the CET.
- The number of rounds is defined as the length of the *CET Data (csv file)* that contains the carbon data. If no changes are made this equals to 40 rounds.

3.5 Subsession Class

Here's an overview of all fields and functions in the Subsession class:

3.5.1 creating_session()

- The `paying_round` is set as a random round between 1 and `num_rounds`.
- The order of questions is randomized for each player.
- The player's *Carbon*, *Car Miles* and *Bonus* fields are initialized for all rounds.

3.5.2 Paying Round

- The `paying_round` is the round where the player's payoff is calculated if `random_payoff = True`. See *Random Payoff*
- A player's saved emission in the `paying_round` is added to the `sum_saved_emission` field if `random_saved_emission = True`. See *Random Saved Emission*

3.5.3 Sum saved Emission

Field

- The `sum_saved_emission` field is the sum of the `saved_emission` player field for all players.
- The sum is either calculated across the `paying_round` or across all rounds depending on the *Random Saved Emission* field.
- The field is used as an input in the *Send Payment Mail* function.
- Excludes all players that ar seen as bots. See *Bot Criteria*

Function

- The `set_sum_saved_emission()` function sets the `sum_saved_emission` field.
- Checks if a player is a bot. See *Bot Criteria*
- If a player is not a bot then the total `saved_emission` of all players is added to the `sum_saved_emission` (Either across the paying round or all rounds).
- A player has to finish all rounds of the CET, so that the correct data is available. Therefore, this function is only called in the last round of the CET. See *Experiment Page*

3.5.4 All Players Finished

Field

- `all_players_finished` is a Boolean field that turns `True` once all players have finished the CET.

Function

- `set_all_players_finished()` calculates how many players in total have finished the CET (`sum_finished`).
- If `sum_finished` = Number of participants then the `all_players_finished` field turns `True`.
- This function is only called once (for each player): When the player hits the "Next" button on the *Results Page*.

3.5.5 Helpful prints

- The `helpful_prints()` function prints helpful information about the current state of many player and subsession fields to the terminal.
- The function can be extended at will and be used for bug fixing purposes, if a new field is added.
- The function is called in every round of the CET and when a player finishes the CET.
- The function is only useful if the number of participants is small.

3.5.6 Send Payment Mail

The `send_payment_mail()` function is used to automate carbon-emission certificate purchases for experiments with real-carbon externalities, such as the CET. For more information see ACO Documentation.

Requirements

- The modules requests and smtplib have to be imported at the top of the models.py file.
- A valid account from an SMTP service provider is needed. The credentials of the account have to be specified at the top of the function.

Parameters

- `weight_to_donate`: A float value used to pass the amount of carbon emission that is saved by the experimental participants. The *Sum saved Emission* is used for this.
- `unit`: A string value that defines the unit of the saved carbon emission. The following values are accepted: `["mg", "g", "kg", "t", "oz", "lbs", "st"]`
- `experiment_name`: A string value that specifies the name of the experiment (e.g. "Carbon Emission Task")
- `payment_e_mail_name`: A string that specifies the name of the person or team that receives the mail
- `payment_e_mail_to`: A list containing the mail addresses of all recipients . If the mail is only to be sent to one address then a single string can be passed to the function.

How it works (basic)

1. The `weight_to_donate` value is converted to metric tons. The conversion is based on the `unit` value.
2. The current CO2 price per ton for emission certificates is fetched from a price endpoint that is provided by Compensators.
3. The price of the carbon-emission certificate is calculated.
4. A mail is sent to all addresses within the `payment_e_mail_to` list. The mail includes the total weight of carbon-emission saved, the current price per ton for carbon-emission certificates, as well as the link to Compensators donation form with the correct price to make the carbon-emission certificate purchase. These contents can be changed at will.

When is send_payment_mail() called?

- The function is called when all players have finished the CET. For an example see *Results Page*

3.6 Player Class

Here's an overview of all fields and functions in the Player class:

3.6.1 Fields

Carbon

- The `player.carbon` field specifies the carbon value in lbs for each round of the CET.

Car Miles

- `player.car_miles` field specifies the equivalent of the carbon value in miles driven in a standard car.
- This value serves as a real-life example for the players to understand the carbon footprint of the decision.

Bonus

- `player.bonus` is a Currency field that specifies the monetary compensation a player receives, if the environmentally unfriendly Option A is picked.

The `player.carbon`, `player.car_miles` and `player.bonus` value are initialised for all rounds before the start of the experiment. See *creating_session()*

Decided

- `player.decided` is a Boolean field that states whether a player has made an autonomous decision or not.
- `True`: The player has decided on his/her own.
- `False`: A timeout happened.
- A player has a `decided` value in each round of the CET.

Choice

- `player.choice` specifies the choice of the player for each round.
- `0`: Option B: The player decides to forfeit the bonus (Environmentally friendly decision).
- `1`: Option A: The player decides to take the bonus (Environmentally unfriendly decision).

Choice Practice

- `player.choice_practice` is needed to produce an error if player tries to click through practice rounds.
- This field serves no other purpose and can be disregarded in the *Export*.

Total Emission

- `player.total_emission` is the sum of the `player.carbon` values over all rounds the player has progressed so far.
- This value increases every time a player progresses to a new round.
- This field can be split into two sub_fields: `player.chosen_emission` and `player.saved_emission`.

Chosen Emission

- `player.chosen_emission` is the sum of all carbon values if Option A was clicked.
- This value increases everytime the player chooses Option A.

Saved Emission

- `player.saved_emission` is the sum of all carbon values if Option B was clicked.
- This value increases everytime the player chooses Option B.

Is Bot

- `player.is_bot` is a Boolean field that states if the player is seen as a Bot or not.
- A player is regarded as a bot if the amount of autonomous decisions falls below the *Bot Criteria*.
- The bot status of a player can vary from round to round depending on the player's percentage of autonomous decisions.

Is Dropout

- `player.is_dropout` is a Boolean field that states if the player is considered a dropout.
- A dropout player is a player that mathematically cannot become "human" again.
- **Example:** The CET has a total of 40 rounds. Let's assume the Bot Criteria is 0.75 so the player has to decide autonomously in 75% of all rounds. Hence, the player must make a decision in 30 of the total 40 rounds. So the player can miss a total of 10 rounds and can still avoid being considered a bot if he makes a decision in the remaining 30 rounds. However, once the player misses a total of 11 rounds, then the player can mathematically not become "human" again. As soon as this happens the `player.is_dropout` field turns `True` and the player is considered a dropout.
- If the player is considered a dropout, then the *Timeout* for every remaining page is set to 0 seconds. Thus, the dropout player is automatically progressed through each round until the CET is finished. This drastically reduces the total experiment time, since experimenters don't have to wait on dropout players. This is espacially important because every player (also dropouts) have to finish the CET so the `send_payment_mail()` function is triggered. See *When is send_payment_mail() called?*

Is Finished

- `player.is_finished` is a Boolean field that states if the player has finished the CET or not.
- A player is considered finished when the "Next" button on the Results page is pressed. See *Results Page*

Payoff per Round

- The `player.payoff_per_round` field contains the player's (hypothetical) payoff for each round.
- If the player chooses environmentally-friendly Option B, the payoff is 0.
- If the player chooses environmentally-unfriendly Option A, the payoff is `player.bonus`.
- `player.payoff_per_round` is used as a helper field and does NOT define the actual `player.payoff` field. The actual payoff is calculated in *Set Payoff*.

3.6.2 Functions

Current Question

- `current_question` is used to help initialize `player.carbon`, `player.car_miles` and `player.bonus`.
- This function is called in *creating_session()*.

Set Total Emission

- `set_total_emission` sets the `player.total_emission` field.
- This function is called in every round of the CET. See *Experiment Page*

Set Chosen Emission

- `set_chosen_emission` sets the `player.chosen_emission` field.
- This function is called in every round of the CET. See *Experiment Page*

Set Saved Emission

- `set_saved_emission` sets the `player.saved_emission` field.
- This function is called in every round of the CET. See *Experiment Page*

Set is Bot

- `set_is_bot` sets the `player.is_bot` and the `player.is_dropout` field.
- This function is called in every round of the CET. See *Experiment Page*

Set Payoff per Round

- `set_payoffs_per_round` sets the `player.payoff_per_round` field.
- This function is called in every round of the CET. See *Experiment Page*

Set Payoff

- `set_payoff` sets the `player.payoff` field.
- The payoff is a built-in player field that does not have to be initialised and is used to determine the player's payoff.
- This function is dependent on the *Random Payoff* constant.
- If `random_payoff = True` then the `player.payoff` = `player.payoff_per_round` in the paying round and 0 in every other round.
- Else `player.payoff` = `player.payoff_per_round` for every round.
- This function is called in every round of the CET.

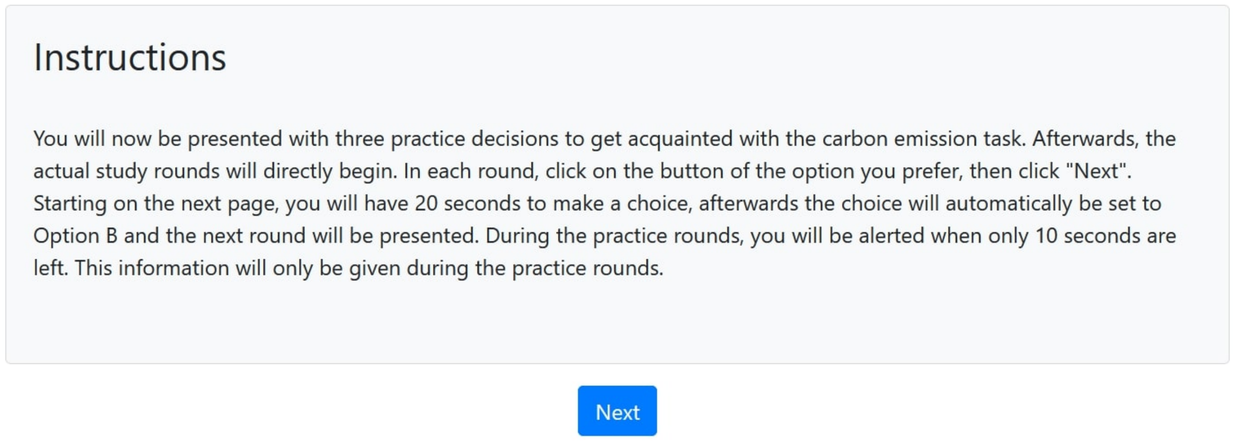
3.7 Pages

Overview of all pages of the Carbon Emission Task

Note: Timeouts: There is a timeout on every page of the experiment to account for dropped out players. In case a player begins the experiment and leaves, the player is automatically progressed through the rest of the experiment.

3.7.1 Instruction Page

The following instructions are presented to the player:



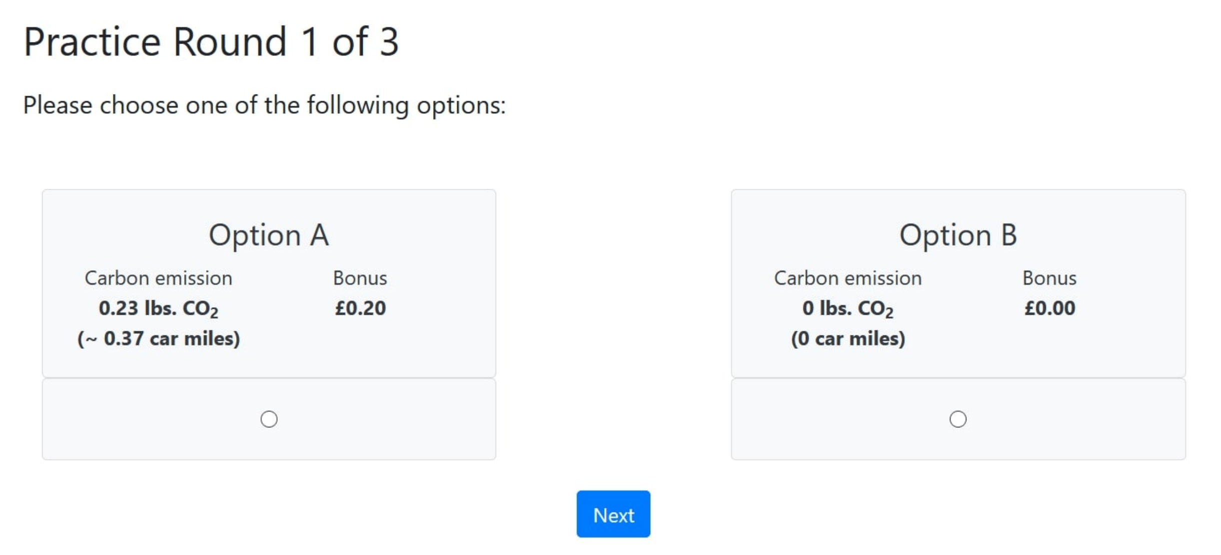
Instructions

You will now be presented with three practice decisions to get acquainted with the carbon emission task. Afterwards, the actual study rounds will directly begin. In each round, click on the button of the option you prefer, then click "Next". Starting on the next page, you will have 20 seconds to make a choice, afterwards the choice will automatically be set to Option B and the next round will be presented. During the practice rounds, you will be alerted when only 10 seconds are left. This information will only be given during the practice rounds.

Timeout

There is a Timeout of 120 seconds on this page.

3.7.2 Practice Pages


Practice Round 1 of 3

Please choose one of the following options:

Option A

Carbon emission
0.23 lbs. CO_2
(~ 0.37 car miles)

Bonus
£0.20

Option B

Carbon emission
0 lbs. CO_2
(0 car miles)

Bonus
£0.00

There is a total of three practice pages before the actual CET starts. The practice pages don't visibly differ from the actual experiment pages and serve the sole purpose of familiarising the participants with the task. The following `carbon`, `car_miles`, `bonus` values are used in this order for the three practice pages:

- 0.23 lbs. CO2, 0.37 car miles, 0.2£
- 4.46 lbs. CO2, 7.24 car miles, 0.6£
- 19.85 lbs. CO2, 32.22 car miles, 1£

These values can be changed in the following code block of the corresponding practice page class:

```
def vars_for_template(self):
    return dict(
        practice1_carbon=0.23,
        practice1_carmiles=0.37,
        practice1_bonus=0.2
    )
```

These values should match an existing combination of carbon, car miles and bonus values provided in the *CET Data (csv file)*. No data is logged for the practice pages.

Forms:

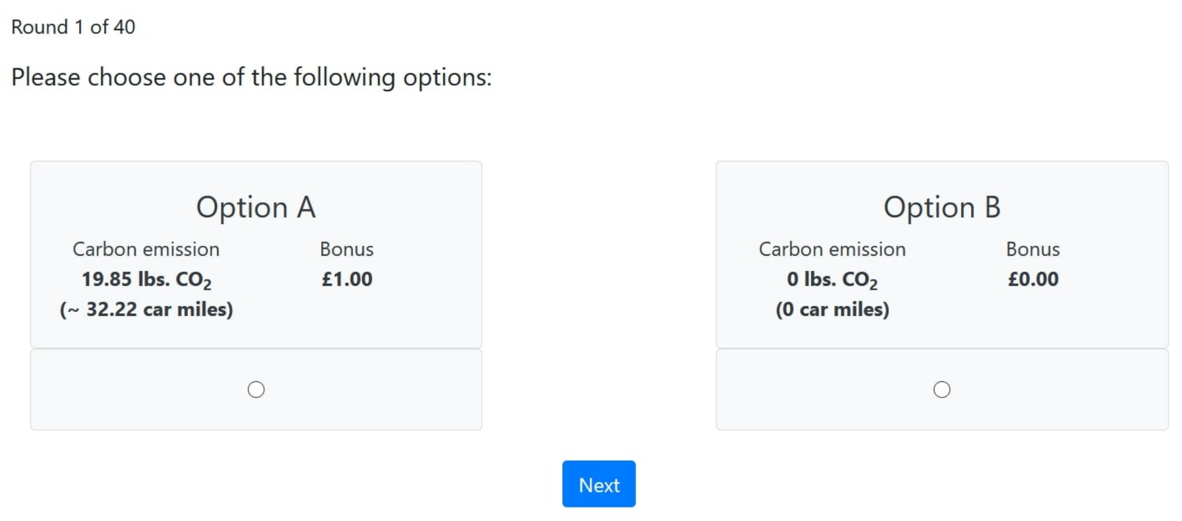
The following forms are used for the practice page class: The `choice_practice` field is needed to produce an error if a player clicks the "Next" Button before choosing an Option.

```
class Experiment_page(Page):
    form_model = 'player'
    form_fields = ['choice_practice']  # == player.choice_practice
```

Timeout:

The timeout for practice pages is set to 20 seconds.

3.7.3 Experiment Page



The experiment page class is responsible for the experimental rounds of the CET.

Forms

The following forms are used for the experiment page class: If a player presses either Option A or B the choice is automatically logged in the `player.choice` field.

```python
class Experiment_page(Page):
    form_model = 'player'
    form_fields = ['choice']   # == player.choice
```

Timeout

In contrast to the Practice Pages' timeout, the timeout for experiment pages is set dynamically. Before the timeout is set, it is checked whether or not the player has dropped out of the experiment. See *Is Dropout*

- If `player.is_dropout = True`, the player has dropped out and the timeout is set to 0. All subsequent pages are submitted instantly and the player is automatically progressed through the rest of the experiment.
- Else the timeout is set to 20 seconds.

Before_next_page()

The code below is executed once a player hits the “Next” button of a given round. The order of the code below is very important because some functions depend on results that are calculated in previous functions. This order must not be changed unless you absolutely know what you are doing.

```
def before_next_page(self):
    # Timeout check
    if self.timeout_happened:
        self.player.decided = False
        self.player.choice = 0
    else:
        self.player.decided = True

    #Payoff functions:
    self.player.set_payoff_per_round()
    self.player.set_payoff()

    #Emission functions:
    self.player.set_chosen_emission()
    self.player.set_total_emission()
    self.player.set_saved_emission()

    #Bot check
    self.player.set_is_bot()

    #Last round check
    if self.round_number == Constants.num_rounds:
        self.subsession.set_sum_saved_emission()

    # Helpful prints
    self.subsession.helpful_prints()
```

Sequence of events (basic):

1. Timeout check

   - If a timeout happened the `player.decided` field is set to `False` and `player.choice` is set to 0 (Option B).

   - Else the `player.decided` field is set to `True`.

2. Payoff functions

   - `set_payoff()` depends on `set_payoff_per_round()`.

3. The emission functions are called.

   - `set_chosen_emission()` and `set_saved_emission()` depend on the `player.choice` field that is set in step 1.

   - `set_saved_emssion()` depends on `set_total_emission` and `set_chosen_emission()`.

4. Bot check

   - `set_is_bot()` is called and evaluates whether the player is a bot and/or a dropout.

   - This function depends on the `player.decided` field that is set in step 1.

5. Last round check

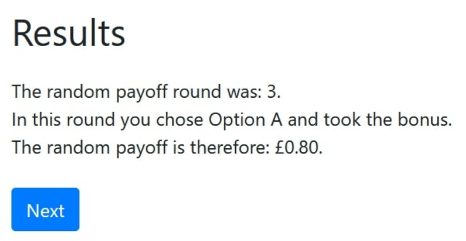
   - If a player is in the last round of the CET, `set_sum_saved_emission()` is called.

   - This function depends on `player.choice` (step 1), `set_saved_emision()` (step 3) and `set_is_bot()` (step 4).

6. Helpful prints
   - The helpful print functions is called and the state of most player and subsession fields are printed to the terminal.
   - This function depends on most of the above steps.

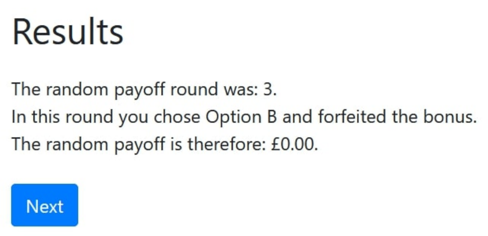
3.7.4 Results Page

The results page is displayed after the player has finished all rounds of the CET. The pages looks different depending on the *Random Payoff* constant and the Player's *Choice* in the paying round. From left to right: `random_payoff = True` and player chose Option A; `random_payoff = True` and player chose Option B; `random_payoff = False` so the total payoff is shown.



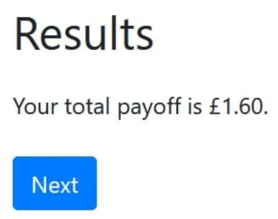
Results

The random payoff round was: 3.
In this round you chose Option A and took the bonus.
The random payoff is therefore: £0.80.



Results

The random payoff round was: 3.
In this round you chose Option B and forfeited the bonus.
The random payoff is therefore: £0.00.



Results

Your total payoff is £1.60.

Timeout

The timeout logic works the same way as in the experiment pages.

- If a player has dropped out: Timeout = 0 seconds
- Else: Timeout = 60 seconds

Before_next_page()

This code is executed once the player hits the "Next" button of the results page. The code below is used to send the mail for carbon-emission certificate purchases.

```python
def before_next_page(self):
    #Is Finished fields and functions
    self.player.is_finished = True
    self.subsession.set_all_players_finished()

    # Helpful prints
    self.subsession.helpful_prints()

    # All finished check and send mail
```

(continues on next page)

(continued from previous page)

```
    if self.subsession.all_players_finished:
        self.subsession.send_payment_mail(self.subsession.sum_saved_emission,
                                          "lbs",
                                          "Carbon Emission Task",
                                          "John Doe",
                                          "john.doe@cet.com")
```

Sequence of events:

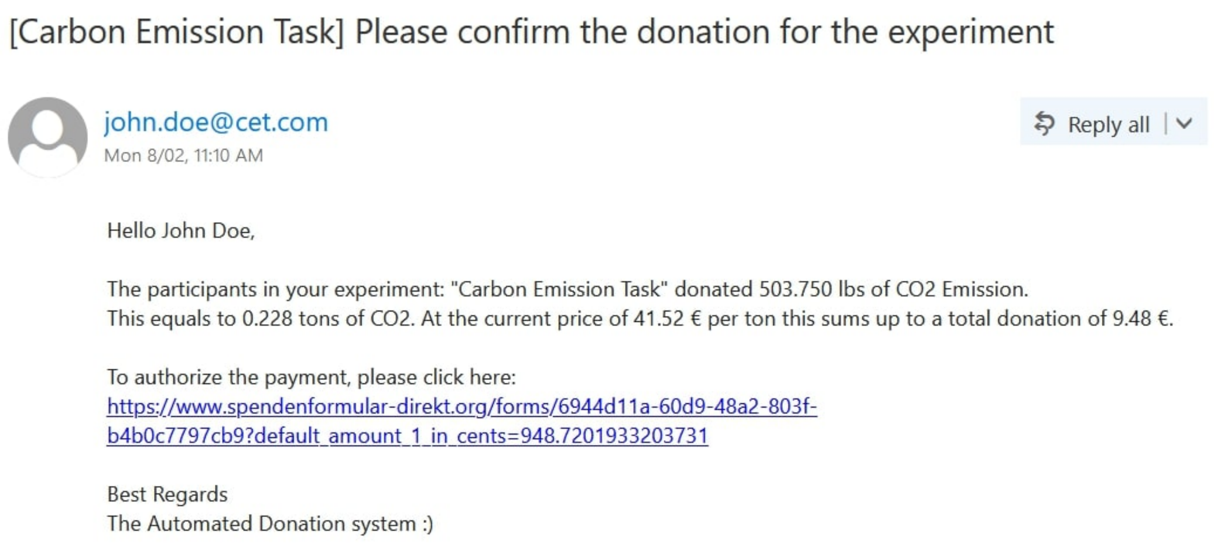
1. Once a player hits the "Next" button the `is_finished` field of the player is set to `True`
2. The `set_all_players_finished()` function checks if every player has finished the CET.
3. Helpful information is printed to the terminal (including the number of players that have finished the CET).
4. If all players have finished the CET, the `send_payment_mail()` function is called.

Mail parameters:

- The *Sum saved Emission* field is the total weight of CO2 emission that was saved by participants.
- The unit of the weight is lbs.
- The name of the experiment is Carbon Emission Task.
- The name of the recipient is John Doe.
- the recipient's email address is john.doe@cet.com. (Multiple addresses have to be specified in a list e.g. ["john.doe@cet.com", "jane.doe@cet.com"].

Contents of send_payment_mail():

This is an example of a generated email:


[Carbon Emission Task] Please confirm the donation for the experiment

john.doe@cet.com
Mon 8/02, 11:10 AM

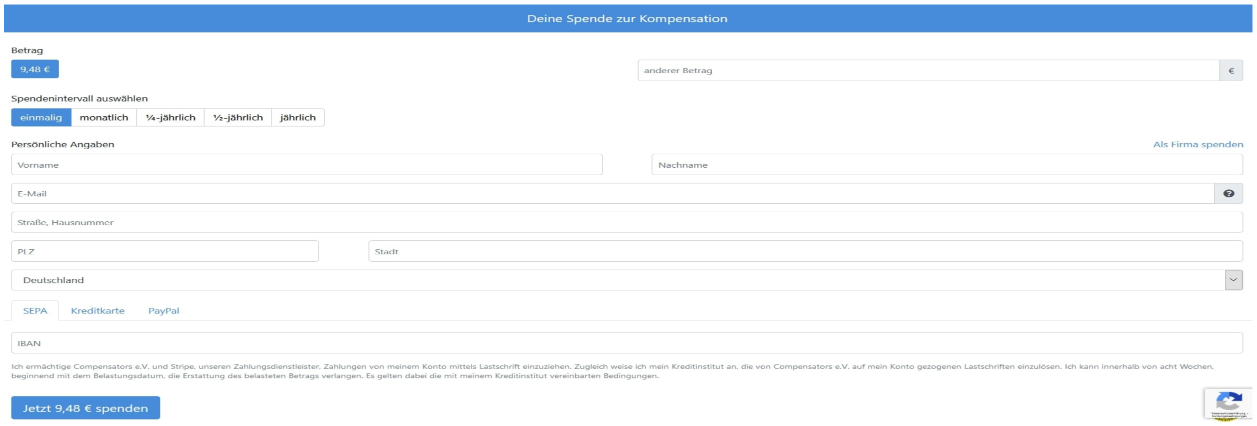
Hello John Doe,

The participants in your experiment: "Carbon Emission Task" donated 503.750 lbs of CO2 Emission.
This equals to 0.228 tons of CO2. At the current price of 41.52 € per ton this sums up to a total donation of 9.48 €.

To authorize the payment, please click here:
https://www.spendenformular-direkt.org/forms/6944d11a-60d9-48a2-803f-b4b0c7797cb9?default_amount_1_in_cents=948.7201933203731

Best Regards
The Automated Donation system :)

The link directs you to the donation form, where the carbon-emission certificate purchase can be made. The donation form looks like this:



Ich ermächtige Compensators e.V. und Stripe, unseren Zahlungsdienstleister, Zahlungen von meinem Konto mittels Lastschrift einzuziehen. Zugleich weise ich mein Kreditinstitut an, die von Compensators e.V. auf mein Konto gezogenen Lastschriften einzulösen. Ich kann innerhalb von acht Wochen, beginnend mit dem Belastungsdatum, die Erstattung des belasteten Betrags verlangen. Es gelten dabei die mit meinem Kreditinstitut vereinbarten Bedingungen.

3.8 Templates

Overview of the general structure of all html templates for the CET.

3.8.1 Block Styles

```
{% block styles %}
...
{% endblock %}
```

In this section you can define graphical options for the elements displayed in *Block Content*.

Error message

```
.otree-form-errors {
visibility: hidden;
display: none;
}
```

`visibility: hidden` hides a possible error message, but the element will still take up space on the page. In contrast, `display: none` completely removes the element. It is possible to show a different error message should participants fail to choose one of the necessary options to advance (e.g., one of the two options in the CET). The message can be modified within the `{% block content %}` inside this block:

```
{% if form.errors  %}
<div class="alert alert-danger" role="alert">
    <p>Please choose one of the required fields.</p>
</div>
{% endif %}
```

The expression `{{ form.choice.errors }}` is inserted in both lines in `{% block content %}` which define the radio buttons to be displayed on the page. See *Radio buttons*

Timer

This section also includes options to apply and, if necessary, show the timer indicating the remaining time on a given page. If a timeout is defined (on the page class), the display property can be specified under `.otree-timer` (e.g., "none" or "block"; click here for more information). Furthermore, it is possible to specify that the timer only appears if a certain amount of seconds remains before the next page is automatically displayed. Change the number in `if (event.offset.totalSeconds === 10)` according to your preferences.

The remaining code in the block style section is about graphical specifications (see information on "class" below).

3.8.2 Block Content

```
{% block content %}
...
{% endblock %}
```

In this section, you define what is to be displayed to the user. The Section is split into two container classes. The Container for Option A and the Container for Option B. Further customizations of the two containers can be done here.

Radio buttons

```
<input type="radio" id="Option A" name="choice" value="1"> {{ form.choice.errors }}
```

This defines the radio button which is linked to the player's *Choice* field. The `value` defines what is saved in the data if this button is chosen. The `id` attribute only serves as clarification in this case to explicitly show which value is chosen for which option above, since the code is somewhat difficult to oversee at first glance. (Generally, it could be styled in a similar way like the "class" attributes.)

3.9 Export

How to export the data that is generated from the CET can be found here