Graphical Feedback using chart.js

Setting up some graphical feedback may sometimes be easier to understand for subjects than lines upon lines of text. There are lots of ways to set up a graph. I would recommend to have a look at chart.js, which is an open source library for charts. To include this library you need to load it in javascript. Head to the chartjs documentation and click on Chart.js CDN, which provides a stable link to the scripts. Look for chart.min.js and copy the link to the clipboard.

https://www.chartjs.org/

https://www.chartjs.org/docs/latest/

https://www.jsdelivr.com/package/npm/chart.js

Here is the direct link to the javascript file.

https://cdn.jsdelivr.net/npm/chart.js@3.9.1/dist/chart.min.js

This one is the same but a bit better readable in case you are interested.

https://cdn.jsdelivr.net/npm/chart.js@3.9.1/dist/chart.js

Let's set up a simple line chart in the feedback for the public goods game. On every feedback page subjects see the information how much they contributed and what total contributions were in all periods till now. Head to Feedback.html in the pgg app and add a scripts block first. Add a script and load the chart.js script by using the src option.

```
{{ block scripts }}
    <script
src="https://cdn.jsdelivr.net/npm/chart.js@3.9.1/dist/chart.min.js">
</script>
{{ endblock}}
```

Now we can start and set up a canvas on which we will draw our graph. In the content block add a canvas. I am going to set it up using a fixed width and height.

```
<canvas id="feedback-graph" width="600" height="400"></canvas>
```

Now that we have a canvas we can draw on it. Begin by setting up the config of how the graph should look like. Head back to the scripts block and set up a new script below the existing one. The config for a chart.js work as follows: Define a type (see the documentation for options), define the data, define the datasets within the data, define the options. For a line chart you have to provide the labels for the x-axis. Then, define two datasets with dummy data. Also define some colors so it's easier to distinguish between the lines. Below the config look up the canvas that we created in the html part and tell chart.js to create new chart on this canvas using the config that we specified.

```
<script>
    var config = {
        type: 'line',
        data: {
            labels: ['Period 1', 'Period 2', 'Period 3'],
            datasets: [
                {
                    label: 'Your Contribution',
                    data: [1, 2],
                    borderColor: 'rgb(75, 192, 192)',
                },
                {
                    label: 'Total Contributions',
                    data: [6, 6],
                    borderColor: 'rgb(192, 75, 192)',
                }
            ]
        }
    }

    var chartCanvas = document.getElementById('feedback-
graph').getContext('2d');
    new Chart(chartCanvas, config);

</script>
```

Run devserver and check if the graph loads. If it does, move on to the next problem. The basic graph is up, but we need to push the actual data to the graph. To do this, head back to init.py and look for the Feedback page. Add a new js_vars function to push data to javascript. To access a player's action in any previous round we have to know this player's identity in the previous round. Remember, players are only "incarnations" or identities of participants in each round and they change between rounds. Still, you can ask a player object what their previous player identity was in round x. So first create a list of player identities.

```
all_player_identities = [player.in_round(r) for r in range(1,
player.round_number+1)]
```

Now that you have this list of identities from the first to the current round you can simply create lists of these players' contributions and their group totals, and finally push it to javascript in a dictionary.

```
    all_player_contributions = [p.contribution for p in
all_player_identities]
    all_group_contributions = [p.group.total for p in
all_player_identities]
    return dict(
        all_player_contributions=all_player_contributions,
        all_group_contributions=all_group_contributions,
    )
```

In the end your function should look as follows.

```
@staticmethod
def js_vars(player: Player):
    all_player_identities = [player.in_round(r) for r in range(1,
player.round_number+1)]
    all_player_contributions = [p.contribution for p in
all_player_identities]
    all_group_contributions = [p.group.total for p in
all_player_identities]
    return dict(
        contributions=all_player_contributions,
        totals=all_group_contributions,
    )
```

Return to Feedback.html. Replace the datasets in the config with the new datasets that we just pushed.

```
            datasets: [
                {
                    label: 'Your Contribution',
                    data: js_vars.all_player_contributions,
                    borderColor: 'rgb(75, 192, 192)',
                },
                {
                    label: 'Total Contributions',
                    data: js_vars.all_group_contributions,
                    borderColor: 'rgb(192, 75, 192)',
                }
            ]
```

Now the graph should update after every round. You can add some new options like defining a minimum and maximum for the axes, for example. In the config, add options below the data entry. In the options, you can manipulate the axes by specifying a minimum and maximum for the y-axis.

https://www.chartjs.org/docs/latest/axes/

```
options: {
    scales: {
        y: {
            min: 0,
            max: 30
        }
    }
}
```

There are tons of other options and ways to change the graph. Take a look at the documentation, and if you don't know how to include it in your graph, you should be able to find examples by googling. Using chart.js you can even set up a graphical interface that registers when a subject clicked somewhere to make choices and record data.