SCIENTIFIC Publishing Limited

Digital Technologies Research and Applications

http://ojs.ukscip.com/index.php/dtra

Article

A Web-Based Multiplayer Common-Pool Resource Game Platform

Vasileios Tzimourtos

Department of Economics, University of Thessaly, Volos 38333, Greece

* Correspondence: vtzimourtos@uth.gr

Received: 15 May 2025; **Revised:** 6 July 2025; **Accepted:** 8 July 2025; **Published:** 22 July 2025

Abstract: The paper introduces a web-based platform for conducting multiplayer common-pool resource (CPR) games, replicating the experimental framework of Ahn et al. Built using technologies such as PHP, MySQL, HTML5, and JavaScript the platform facilitates remote and synchronous gameplay in structured groups with configurable settings. Players extract tokens from a shared resource over multiple rounds, with payoffs determined by token-order pricing and optional costly punishment stages. The system supports dynamic group formation, randomized player order, and round-based synchronization to ensure fairness. It includes real-time coordination via AJAX polling; a responsive Bootstrap interface; administrative controls, and live monitoring through Grafana dashboards. Data integrity is maintained using server-side validation, prepared S.Q.L. statements, and transactional database updates, with all actions logged in a normalized relational schema. The platform addresses technical challenges such as session timeouts, synchronization delays, and dropouts using heartbeat callbacks, waiting screens, and robust state checks. Designed for extensibility, it can support different game-theoretic designs and is suitable for both lab and remote experiments. Future enhancements include WebSocket-based real-time communication and Docker-based deployment for reproducibility. By offering a customizable, secure, and scalable CPR experiment environment with real-time feedback and comprehensive logging, this platform serves as a valuable tool for advancing research in experimental economics, particularly in cooperation, punishment mechanism, and resource governance. The platform demonstrates reliable performance with low-latency synchronization and minimal dropout-related issues. Compared to tools like oTree and LIONESS Lab, it offers a more lightweight and modular alternative optimized for small-scale deployment, remote teaching, and rapid experimental design.

Keywords: Common-Pool Resource; Experimental Economics; Web-Based Game; Multiplayer Coordination; Session Management; Penalty Mechanisms

1. Introduction

Common-pool resources (CPRs)–such as fisheries, forests, or irrigation systems – are characterized by being non-excludable yet rivalrous: individuals cannot be easily excluded from using them, but one person's consumption reduces availability for others [1]. Garrett Hardin's seminal essay *The Tragedy of the Commons* argued that, in the absence of regulation, rational self-interested users will overexploit such resources, leading to their eventual depletion [2]. In contrast, Elinor Ostrom's pioneering work challenged this view, showing that communities often succeed in self-organizing sustainable governance structures for CPRs [3]. In *Working Together*, Poteete, Janssen, and Ostrom further emphasized the importance of integrating behavioral experiments, case studies, and theoretical models to understand collective action problems in commons governance [4]. This principle has also been demonstrated in regional experiments, such as those conducted in Greece, where community-based institutional responses have been shown to support cooperation in CPR dilemmas [5].

Empirical laboratory and field experiments have since explored how communication, norms, and institutions affect CPR outcomes. For instance, Ahn et al. conducted experiments demonstrating that non-binding communication among participants in a finite-horizon token-extraction game could yield nearly efficient outcomes [6]. Punishment mechanisms, such as costly peer penalties, have also been shown to support cooperation in social dilemmas [7]. CPR games are essential tools in behavioral economics for studying cooperation, self-governance, and institutional responses to shared resource dilemmas. In real-world contexts such as fisheries, water distribution, or climate governance, these dynamics mirror collective action problems that require sustainable and equitable solutions. Experiments help evaluate which mechanisms—like communication or sanctioning—improve outcomes under scarcity and interdependence. Field studies also show that participants bring context-specific norms and shared experiences into CPR experiments, shaping cooperation in important ways [8].

Building on this tradition, we have developed a web-based platform that replicates the CPR experiment of Ahn et al. [6], allowing multiple remote participants to interact in a token-collection game with monetary incentives and optional sanctioning. The platform supports asynchronous joining, dynamic group formation, round-based decision-making, and real-time monitoring via an administrative interface.

Web-based experimental platforms have become increasingly important tools in behavioral economics. Beyond behavioral economics, such platforms have proven essential in collective behavior studies using web-based market simulations [9]. Notable examples include oTree, a Python/Django-based framework [10], and LIONESS Lab, an HTML5-based tool designed to reduce wait times and support flexible grouping [11]. Chan et al. surveyed dozens of experimental tools and found that oTree and SoPHIE offered the most comprehensive feature sets [12]. These studies highlight the growing need for accessible experimental platforms. Similarly, in fields such as sustainability and supply chain design, heuristic and AI-based modeling frameworks have gained traction [13–15], reinforcing the demand for adaptable digital tools that support strategic coordination. Most current platforms share common technological foundations, typically using languages like Python, PHP, or Java, and web frameworks such as Django or Zend with front-end tools like Bootstrap.

Our system follows a similar architecture, using a standard LAMP-like stack. It allows remote participants to be matched into groups and play synchronously through round-based decision screens, while an admin interface enables real-time experiment configuration and monitoring.

In this paper, we first review relevant literature (Section 2), then present the design and architecture of our system (Section 3), describe the implementation (Section 4), discuss key challenges and solutions (Section 5), and detail platform features (Section 6). We conclude with directions for future development (Section 7).

The main contributions of this work are:

- A fully functional web-based platform replicating a well-known CPR experiment with support for penalty mechanisms;
- A modular and extensible design compatible with both lab and remote use;
- Integration of real-time monitoring tools and data export functions for experimental control;
- Detailed documentation of design choices, implementation logic, and challenges encountered to support reproducibility and future development.

2. Design and Architecture

The system follows a client-server architecture using proven web technologies. On the server side, we use PHP 8.2 and MySQL 8. These open-source tools are common in web-based experiments; for example, Chan et al. note that many economics software platforms rely on PHP or Python backends with MVC frameworks [12]. They also allow integration with containerized workflows that improve replicability and version control in experimental platforms [16]. We chose PHP (with jQuery and Bootstrap) for easy deployment on standard Linux/Apache stacks (i.e., LAMP). On the client side, the interface is built with HTML5, CSS3, and JavaScript, communicating via AJAX requests. Using the Bootstrap CSS framework ensures that the UI is responsive and accessible on desktops, tablets, or phones without additional development effort. For instance, all pages adapt to screen size, and range-slider inputs provide immediate visual feedback. **Figure 1** (below) shows the initial registration and lobby screens.

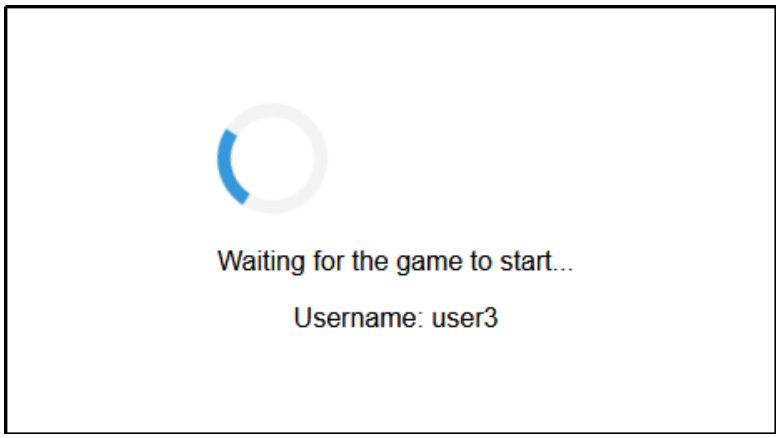


Figure 1. Waiting Screen Until Groups Assemble.

Players begin by registering a pseudonymous username (no personal data is collected). They then enter a lobby until enough players have joined. We form fixed-size groups (e.g., four players per group) either on a first-come/first-served basis or randomly. Once groups are full, the experiment session starts simultaneously for all ("lockstep" synchronization). This structure—borrowed from prior designs like Ahn et al.—ensures fairness: no subgroup progresses ahead of the others, and all group members make decisions together [6]. Within each group, players participate in a series of identical rounds. In each round, every player selects how many tokens to extract using a range slider. The token-order pricing rule then determines each player's payoff: higher token orders incur higher costs (based on a convex supply curve) and produce lower net returns. This replicates the CPR payoff structure used by Ahn et al. [6]. Similar incentive structures have been explored to compare institutional performance under sanctioning rules and cooperative pressures [17,18].

An optional penalty stage follows each round if enabled: players may assign costly punishment points to others, which reduces both the punished player's and the punisher's payoffs. This implements the well-known "costly punishment" mechanism, which has been shown to promote cooperation in social dilemmas e.g., Fehr & Gächter [7].

Administrators can configure session parameters (e.g., group size, number of rounds, penalty rate) through a password-protected admin panel. All parameters are stored in the database and dynamically accessed by the game logic.

Network and session flow are illustrated in **Figure 2**. All client-server communication is handled asynchronously. A key design choice was to implement real-time state monitoring using short polling via AJAX. For example, once a player submits their token choice, their browser enters a "waiting" screen and sends AJAX requests (via JavaScript) to the backend API (e.g., check_group_status.php) every few seconds. When the server confirms that all players have submitted, the client receives a redirect to the next round's input screen. While short polling is less efficient than WebSockets, it is easier to implement and works well over standard HTTP. Tools like Socket.IO could be used in future versions to enable push-based communication. However, in our tests, the current approach (with a 2–3 second polling interval) performed reliably and is compatible with shared hosting environments that lack WebSocket support. While AJAX polling is simple and widely supported, it introduces some latency and overhead, particularly in large-scale or fast-paced experiments. WebSocket-based communication could improve synchronization and responsiveness by enabling server-push updates, and is considered for future releases to enhance scalability and real-time performance [19].

Security and data integrity were key concerns throughout development. All client-server communication is encrypted via HTTPS. Server-side validation and prepared SQL statements are used to prevent injection attacks and ensure valid data. A complete audit log is maintained in the database: every token submission, payoff calculation, and penalty assignment are recorded. The schema (**Figure 3**) consists of five primary tables: users, tokenorders, penalties, instructions, and settings. Parameters such as maxRounds or penaltyRate are stored centrally in settings and can be modified during an experiment session if needed.

(**A**) (**B**)

Figure 2. Administrator's Grafana Dashboard (**A**) Token Metrics, (**B**) User Response and Request Metrics.

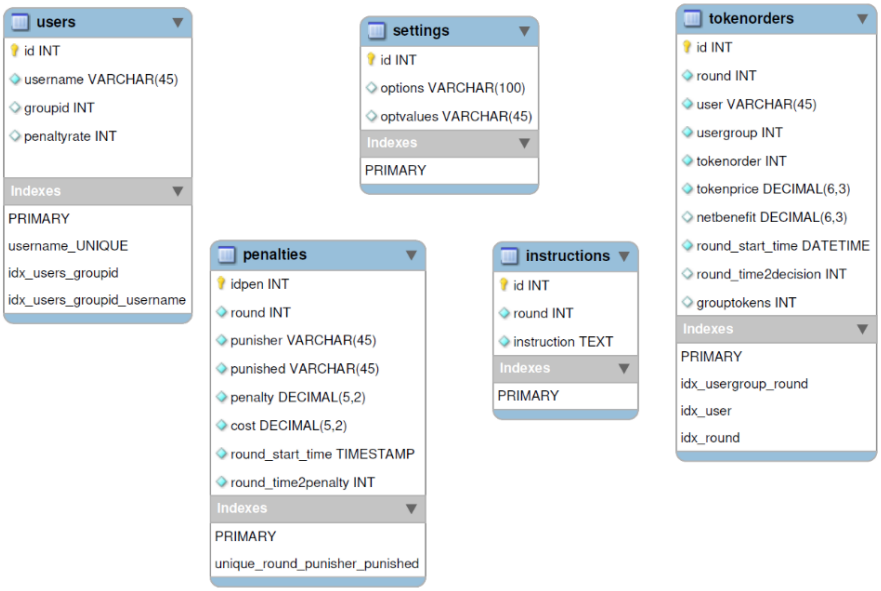


Figure 3. Database Schema.

The interface follows responsive design principles, including Bootstrap's grid system, accessible forms, and mobile-friendly layouts. We tested the platform across all major browsers and operating systems. These choices result in a flexible, extensible, and secure environment for conducting standard CPR experiments online. The system's logging design also aligns with reproducible research practices. All data is timestamped, exportable in machine-readable formats, and compatible with FAIR data management standards [20], facilitating post-experimental analysis, transparency, and long-term archival.

All game actions—such as token submissions, penalty allocations, and timestamps—are logged in real time to the MySQL database. For live monitoring, we integrated a Grafana dashboard using a Prometheus exporter or MySQL plugin. This setup provides real-time statistics on active sessions, average payoffs, submitted tokens, and server performance. The administrator can thus monitor ongoing experiments (e.g., number of groups playing, response times) and detect potential issues. This architecture supports continuous data flows and periodic client updates, enabling synchronized gameplay across distributed users (see **Figure 2A,B**) [1,6].

3. Implementation

The core implementation consists of front-end HTML and JavaScript pages, back-end PHP API scripts, and a relational database schema built on MySQL. Both the codebase and the database design were originally developed by the author and are described in detail below. We deliberately selected a traditional LAMP-based stack (PHP, MySQL, JavaScript) to ensure maximum compatibility, ease of hosting, and simplicity for non-specialist deployers.

While modern frameworks (e.g., Node.js, Django, or cloud-native stacks) offer advanced modularity and scalability, they may introduce barriers to adoption for small labs or single-instructor classrooms. Our current architecture balances accessibility and functionality, though future iterations may explore decoupled or microservice-based implementations for greater flexibility and integration with real-time analytics or AI modules [16]. The following sections outline the key components and workflows that underpin the system's operation. **Figure 4** provides a visual overview of the sequential user flow, from registration to game completion.

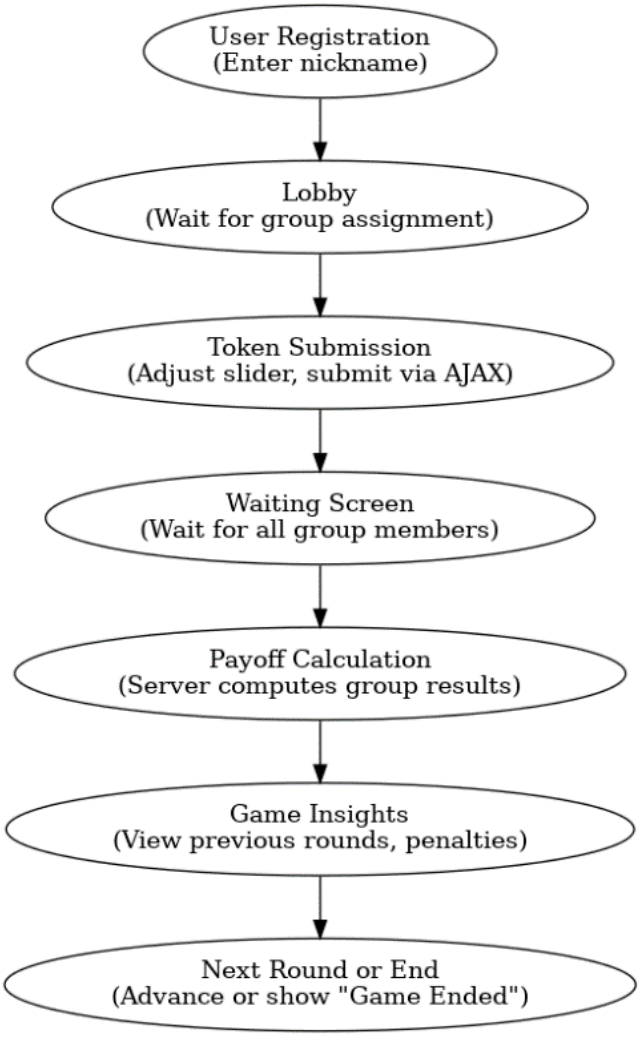


Figure 4. Illustration of the Sequential User Interaction Flow in the CPR Game Platform, from Registration to Game Completion.

3.1. User Interface and Token Ordering

Players interact through an HTML5 token request form with a range slider (<input type = "range">). As the player moves the slider, a JavaScript calcBenefits() function computes the expected benefit using the quadratic formula:

$$B = t \times (0.761 - 0.007t) \tag{1}$$

This value is updated in real time on the interface [6]. Real-time visual feedback has been shown to enhance decision-making and user engagement in online public goods experiments [21].

The current token request and calculated benefit are shown on the page (**Figure 5**). When the player submits the form, the client captures the selected value (rangeValue) and benefit, then sends it via fetchWithRetry

('process_token_request.php', {method:'POST'}). The form data includes: username, round number (currentRound from sessionStorage), group ID, tokens requested, and a timestamp of when the round started (pageLoadTime).

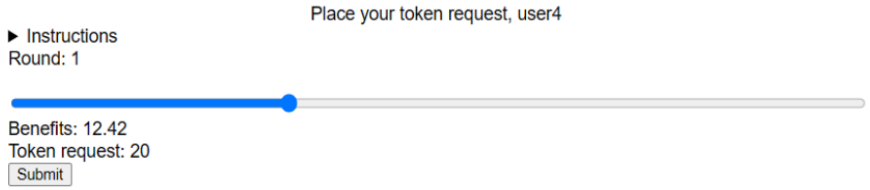


Figure 5. Token Request Interface with Dynamic Calculation of Benefits.

Upon receiving a token submission, the process_token_request.php script on the server performs the following steps (**Figure 6**):

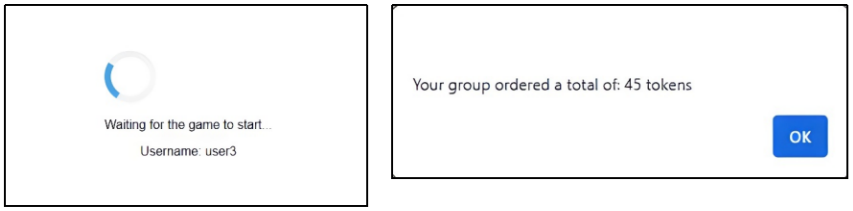


Figure 6. User Interface Displays for Managing Group Requests: (**A**) Waiting Screen for Group Requests, (**B**) Information Alert for Total Group Requests.

1. Store Token Order: The chosen token count (order) is saved in the tokenorders table. Each record includes fields (round, user, usergroup, tokenorder) plus placeholders for later fields (tokenprice, netbenefit, grouptokens);
2. Synchronize Group: The script checks if all group members have submitted for this round (by querying tokenorders for the group and round). If not, all have submitted, it marks the submission and returns status pending. If all are present, it proceeds (**Figure 6A**);
3. Compute Grouptokens: Sum all tokens in the group to get groupTokens. This value is updated in each row's grouptokens field for the current round (**Figure 6B**);
4. Compute Benefits and Costs: For each player in the group, the server calculates the average cost:

$$\text{avgCost} = 0.01 \times (\text{groupTokens} + 1)/2 \quad (2)$$

Then the token price (cost per player) is tokenorder × avgCost, and net benefit = (calculated benefit from formula) – (total cost). These values are written into a database table;

5. Advance Round: Once payoffs are computed, the PHP sets a success response. The client script then increments currentRound in sessionStorage and navigates players to a waiting-group page. This page polls the server until it signals that the next round can begin (handled via a small AJAX call to a next_round.php which checks if the admin hasn't blocked the next round or if all players are ready).

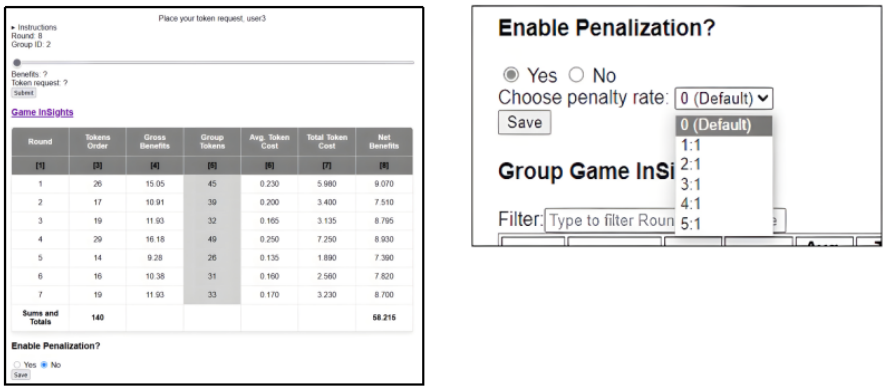
These steps implement the "token ordering" logic and round progression. The ordering of tokens (priority) is not explicitly determined when players submit identical values; in such cases, the system may assign order randomly for record-keeping purposes, although the underlying model assumes simultaneous decisions [4]. The database table tokenorders (see schema) records all of this information. Key columns are:

- round: round index;

- user: username of player;
- usergroup: group identifier;
- tokenorder: tokens requested by the user;
- tokenprice, netbenefit: calculated payoff fields;
- grouptokens: total tokens of the group in that round;
- Timestamps (round_start_time, round_time2decision) track timings.

3.2. Penalty Mechanism

An optional peer-penalty stage is supported in later rounds (**Figure 7**). If enabled by each player group after a threshold round (e.g., round ≥ 7), the platform shows a secondary form where each player can assign penalties to group members (**Figure 7A,B**). The penalty rate could act as the conversion factor; for example, if a player selects a penalty value for another, that amount is deducted from the other player's payoff and charged to the punisher at a defined cost.



(**A**) (**B**)

Figure 7. Game Player Interface: (**A**) Game Options to Enable Penalization, (**B**) Setup Options for Penalty Severity.

Costly punishment mechanisms have been shown to sustain cooperation in repeated social dilemmas by deterring free-riding behavior. Long-term experiments further demonstrate that punishment can yield sustained cooperation benefits over time, even when its application declines [22]. In our implementation, this dynamic can be explored by adjusting the penalty cost-to-impact ratio, enabling researchers to examine how varying punishment burdens affect cooperation over time. This makes the system useful not only for replicating past designs but also for exploring novel treatment variations in institutional design experiments.

Implementation details:

- The HTML includes a hidden with dropdowns for each other player. At the penalty stage, the page populates these <select> elements via createPenaltyInputs() by fetching current group members from get_group_members.php;
- When submitted, save_penalties.php receives a JSON map of {target: penaltyValue}. The script iterates through each pair: it inserts rows into the penalties table with round, punisher, punished, penalty (value set by punisher), and computes cost (punisher's actual cost, e.g. penalty × penaltyRate). It then updates the punished player's net benefit in tokenorders by subtracting the penalty. Concurrently, the punisher's own payoff is reduced by the cost. The UI then confirms with a "Penalties saved" message;
- A separate get_penalty_cost.php API is used to fetch combined penalty costs for display in the previous-rounds table: it returns how much a given player gave and received in penalties so that the interface can show final payoff = netBenefit – totalPenalties.

The penalty logic is a sample implementation following similar experiments where costly punishment has been shown to deter selfish over-withdrawal [7]. In code, the penalty entries form a unique key (round, punisher,

punished) to avoid duplicates.

3.3. Database Structure

The MySQL schema is simple and normalized. The five key tables include (**Figure 3**):

- users: (id, username, groupid, penaltyrate). Holds registered players. groupid is assigned when the game starts; penaltyrate settings;
- tokenorders: (id, round, user, usergroup, tokenorder, tokenprice, netbenefit, round_start_time, round_time2decision, grouptokens). Captures each player's action and payoff for every round in a group;
- penalties: (idpen, round, punisher, punished, penalty, cost, …). Records each penalty assignment between pairs in a group;
- instructions: (id, round, instruction). Stores per-round instructions or messages shown on the UI;
- settings: (id, option, optvalue). Stores game parameters (e.g., maxRounds, penaltyRate, groupSize) that the admin configures.

For example, when the administrator sets a parameter such as maxRounds = 6 through the interface, the corresponding value is stored in the settings table and retrieved by the client via an AJAX call to the getMaxRound() method [6]. The use of a relational database (MySQL InnoDB) with transactional support ensures atomic updates to player payoffs and maintains data consistency in the event of errors. All records are time-stamped, allowing for complete archival of each game session. Additionally, player data can be archived between sessions using the admin_archive_game.php utility.

3.4. Code Handling

All dynamic operations use AJAX (fetch or XMLHttpRequest). Key code elements include:

- Session Management and Dynamic Logic: All asynchronous client-server interactions are handled using AJAX, either via the Fetch API or XMLHttpRequest, depending on the page. To preserve per-client game state (e.g., the current round number), the system uses the browser's sessionStorage object. JavaScript functions such as getCurrentRound() and updateRoundNumber() initialize and manage the round state locally [6]. After each token submission, the round number is incremented. When the client detects that currentRound > maxRound, the interface disables input elements and displays a "Game Ended" message, preventing further interaction and signaling the session's completion;
- Client Callbacks: The slider's input event triggers an update of the tooltip and the calculated benefit in real time [6]. The form's submit handler sends the choice to process_token_request.php, handles the JSON response, increments the round, and navigates to waiting pages [6];
- Waiting Screens and Synchronization Logic: To coordinate players during the game, the system uses dedicated waiting pages—waiting_group.html and waiting_start.html (**Figure 6A,B**)—which display the message "Waiting for other players...". These pages implement JavaScript polling logic using setInterval to call the check_group_status.php API every few seconds. The server responds when all members of a group have submitted their choices, at which point the client automatically redirects back to the token submission interface for the next round. Similarly, before the game begins, waiting_start.html ensures that all players have registered and been assigned to groups before allowing the session to proceed (**Figure 1**). This mechanism ensures synchronized progression across players and preserves group integrity during gameplay [6];

Admin Panel: The administrator controls the experiment through a dedicated settings interface (adminpages.html, **Figure 8**). Each configurable option—such as adjusting group size or user count—triggers an AJAX POST request to backend scripts like save_value_to_settings.php, which update the settings table in real time. Toggle switches and dropdowns allow quick changes without disrupting the live session. The panel also displays a list of current users using admin_fetch_users.php and enables direct interaction with individual player data. Additional buttons provide access to core administrative actions: admin_export_data.php allows the experimenter to download a complete CSV of the session results, while admin_reset_game.php clears the database to prepare for a new session. This separation of administrative functions ensures that the researcher can monitor and man-

age the experiment without interfering with the user experience [6]. Such administrative flexibility aligns with best practices in experimental economics software, as emphasized in recent comparative evaluations [23]. While the admin interface allows full customization of session parameters, future development will include pre-configured templates (e.g., "Basic CPR game", "CPR with penalties") to simplify setup. This enhancement would help researchers without technical backgrounds configure and launch sessions more quickly, improving accessibility across disciplines.

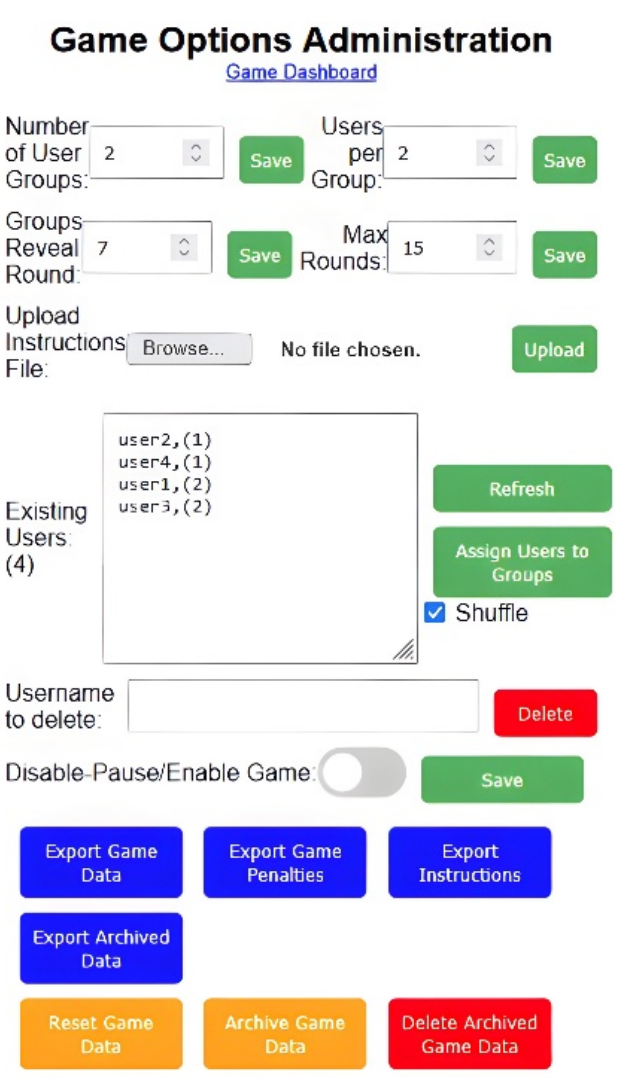


Figure 8. Admin Panel (Settings Page): The Researcher Sets Game Parameters (e.g., Group Size, Max Rounds) and Can Control User Groups, Data Export, and Resets.

Overall, the implementation tightly couples the client and server with asynchronous calls. The database schema underlies both game logic and data retrieval (for scoreboards, Grafana metrics, etc.), while the JavaScript front end ensures a smooth, responsive user experience with immediate feedback on actions.

4. Challenges and Solutions

Several technical and experimental challenges arose in building a reliable and robust online CPR game. Key issues and their solutions are:

- Session Timeout and Connectivity: Internet connections can be unreliable. In web experiments, users may

experience disconnects or leave idle. We implemented a retry mechanism (fetchWithRetry) to mitigate transient failures [6]. On the server side, PHP sessions have long timeouts and a heartbeat (via AJAX) can keep the session alive. If a user disconnects permanently, the admin panel allows resetting or archiving the game state. In practice, if one group member times out, the waiting-room logic can stall. To address this, we included a manual "reset game" option in the admin User Interface;

- Player Waiting and Synchronization: It is crucial that all group members advance rounds together. After each submission, players go to a waiting page and cannot proceed until the server confirms everyone is ready. This strict lock-step ensures no one gains an advantage. We gave clear feedback (e.g., "Round complete, waiting for others...") to mitigate impatience. The client polls the server every few seconds to check if the round result is ready. This ensured fairness (no player can proceed before others) but introduced idle wait times. We minimized unnecessary polling by using moderate intervals (e.g., 2–3 seconds) and only updating visible status;
- Group Fairness: We ensured groups were balanced by size and wait times. Upon registration, users may have to wait until enough players arrive (**Figure 6A**). We considered adjustable group sizes. The system can form groups either sequentially (first-come, first-served) or randomly; we implemented first-come grouping. To avoid long waits, the admin can adjust parameters or launch the game with partial groups. In case of uneven final group, the researcher can omit the last incomplete group or merge with another;
- Fairness in Play Order: In a common-payoff structure, the order of token extraction can introduce unfairness, as earlier extractors may face lower marginal costs than those who move later. To mitigate this bias and promote fairness, the system applies one of two approaches: (a) randomizing the order of players' token submissions each round, or (b) treating all submissions as simultaneous by applying a uniform pricing formula based on the group's total token requests. In the current implementation, the method used is fixed in the back-end logic. Exploring decision order and timing effects has been crucial in studying strategic adaptation in social dilemmas [24]. However, future versions of the platform may expose this choice as a configurable option in the administrative interface. These mechanisms ensure that no player benefits from a fixed "first-mover" position, thereby enhancing the experimental validity of the platform [6];
- Data Consistency and Recovery: All critical database operations—such as payoff calculations and group token totals—are executed within SQL transactions to ensure atomicity and prevent partial updates in the event of a script failure. Every game action is time-stamped and logged to preserve the complete state of the experiment. The administrative panel includes "Archive" and "Export" functions, enabling experimenters to save raw data for analysis or replicate sessions. These tools allow the researcher to snapshot the full database—such as downloading a CSV of all token decisions—at any point during or after a session. In the case of a server crash or restart, the most recent state persists in the database, and unsubmitted form data is minimal, allowing for a straightforward recovery or controlled reset. To monitor system health and performance, we integrated a Prometheus exporter with a Grafana dashboard. This setup provides real-time metrics such as CPU usage, memory load, active users, and request latency (see **Figure 9**). Alerts can be configured (e.g., when response times exceed a threshold), allowing the researcher to identify issues and intervene if needed. Grafana and Prometheus are widely adopted open-source tools for real-time system monitoring and alerting [1];

Figure 9. Server Monitoring Dashboard.

- User Interface Robustness: To ensure the integrity of gameplay and prevent exploitation through browser manipulation, we implemented several security-focused interface and server-side controls. The browser's back button is disabled during play to prevent repeated form submissions or state inconsistencies. All user inputs—such as token requests and penalty assignments—are validated on the server, even if constraints are already enforced on the client side. For example, the token slider is restricted to a valid range (e.g., 0–60 tokens) via HTML attributes, and the server independently verifies that submitted values fall within acceptable bounds. Penalty input fields are dynamically shown only for valid group members and only when the penalty phase is active, minimizing user-side tampering. On the administrative side, form validation prevents invalid configurations—for instance, ensuring that penalty thresholds align with the number of game rounds, or that incompatible options are not selected simultaneously. These measures collectively help maintain consistency across sessions and ensure that the recorded data is clean, complete, and trustworthy.

By addressing these challenges, we established a stable and reliable experimental environment that allows participants to focus on the decision-making task rather than technical issues. This, in turn, improves data quality, enhances experimental validity, and supports reproducibility.

5. Features

The platform provides several features to streamline CPR experiments:

- A simple registration form collects unique usernames from participants, who are then placed in a lobby to await the start of the game (**Figure 10**). This approach accommodates late arrivals and ensures that all players begin simultaneously. Once the required number of players has joined, the administrator triggers the session, and all players advance together;

Figure 10. User Registration Form to Enter the Game.

- Graphical Interfaces: The game interface is built using responsive design principles and styled with the Bootstrap framework to ensure a clean and consistent appearance across devices. All game screens—including the registration, token submission, and feedback pages—automatically adapt to different screen sizes, supporting both desktop and mobile browsers. The token-submission form (**Figure 5**) is centered on the page and features a range slider that updates the calculated payoff in real time, providing immediate feedback and helping keep players engaged. After each round, players are presented with a "Game Insights" summary table (**Figure 11**), which displays all group members' token choices, total group tokens, individual benefits, and any penalties received or given. This transparent feedback mechanism, inspired by Ahn et al. [6], allows players to learn from previous outcomes and refine their strategies over time;
- Admin Controls: The administrator panel (**Figure 8**) provides a flexible interface for configuring and managing game sessions. It allows on-the-fly adjustments to key parameters such as group size, number of rounds, penalty activation round, and penalty rate. Features like costly punishment can be enabled or disabled with immediate effect. The admin can also manage players directly—for example, by force-assigning a latecomer to a group or removing an idle participant to prevent delays. All settings changes are written to the settings table and persist throughout the session. The panel also supports exporting data at any point during or after the game. With a single click, the administrator can download a complete CSV of the session data for analysis.

Additionally, the "Reset" function clears the game state, enabling quick setup of new sessions or experimental conditions. This control interface significantly simplifies the process of running repeated experiments or testing different treatment configurations [6];

Group Game InSights

Filter: Type to filter Round or Username

Round	User	Tokens Order	Gross Benefits	Group Tokens	Avg. Token Cost	Total Token Cost	Net Benefits	Penalties Cost (Given + Taken = Total)	Net benefits after punishment
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
1	user3	26	15.05	45	0.230	5.980	9.070	0+0=0	9.070
2	user3	17	10.91	39	0.200	3.400	7.510	0+0=0	7.510
3	user3	19	11.93	32	0.165	3.135	8.795	0+0=0	8.795
4	user3	29	16.18	49	0.250	7.250	8.930	0+0=0	8.930
5	user3	14	9.28	26	0.135	1.890	7.390	0+0=0	7.390
6	user3	16	10.38	31	0.160	2.560	7.820	0+0=0	7.820
7	user3	19	11.93	33	0.170	3.230	8.700	0+0=0	8.700
8	user3	21	12.89	45	0.230	4.830	8.060	0+0=0	8.060
9	user3	19	11.93	40	0.205	3.895	8.035	0+0=0	8.035
10	user3	33	17.49	60	0.305	10.065	7.425	0+0=0	7.425
11	user3	23	13.8	38	0.195	4.485	9.315	0+0=0	9.315

Figure 11. Sample of Game Insights Section.

- Real-Time Monitoring: A Grafana dashboard connects to the database (via Prometheus MySQL exporter). It displays real-time metrics: current number of active users, response times and aggregate game statistics (e.g., average tokens per round). Experimenters can thus monitor the experiment remotely and identify issues (e.g., stalled groups) (**Figure 2A,B**) [1]. Alerts can be set for critical events like high latency on the server (**Figure 9**);
- Data Logging and Export: All game events—including token order, payoffs, and penalty assignments—are time-stamped and logged. The system exports the complete dataset (CSV format) upon conclusion, capturing all intermediate states and archiving past games for comparative analysis. This archive enables seamless analysis and replication, with direct compatibility for statistical software (e.g., using R to compute summary metrics or generate plots offline);
- Security and Privacy: All client-server communications are encrypted via HTTPS. Participant anonymity is preserved through pseudonymous identifiers, with no collection of personally identifiable information. Server-side input validation and sanitization mitigate injection attacks (e.g., SQLi) and cross-site scripting (XSS) vulnerabilities. These measures ensure both data integrity and confidentiality while complying with standard web security practices for behavioral research platforms.

The system provides a turnkey solution for CPR experiments, combining a polished participant interface with robust administrative controls. Researchers benefit from real-time monitoring capabilities and rapid experimental reconfiguration, eliminating the need for low-level coding or manual data management. This integrated design ensures both experimental flexibility and operational efficiency while maintaining rigorous data collection stan-

dards. The broader validity of online behavioral experiments has also been established in digital labor markets and distributed experimental labs [25].

6. Future Work

To maintain alignment with evolving research standards, several platform enhancements may be planned for future implementation. These prospective developments will focus on: (1) expanding experimental paradigms through additional modular components, (2) incorporating advanced data visualization tools for real-time analysis, and (3) implementing additional security protocols to address emerging cybersecurity requirements. Such improvements will further reduce technical barriers while increasing the platform's versatility for complex experimental designs:

- WebSocket or WebRTC Upgrade: Currently the game uses AJAX polling for synchronization. We could integrate WebSocket (e.g., via Socket.IO) so the server can push updates instantly when all players are ready, reducing latency and server load. This would allow a truly real-time experience and simpler code for wait synchronization;
- Containerized Deployment with Docker: A practical enhancement to the platform involves packaging the entire application using container technologies such as Docker. This would allow for easier deployment, scalability, and portability across different server environments. A containerized setup ensures consistent behavior regardless of the underlying operating system, simplifies updates, and facilitates reproducibility; key concerns in experimental economics. Additionally, container orchestration (e.g., with Docker Compose or Kubernetes) could be used to manage multiple parallel game sessions for large-scale experiments or classroom use;
- Extended Experimental Modules: Beyond the replicated Ahn et al. game, the platform can be adapted for other economic games. Future adaptations might include: public goods games with different contribution functions, bargaining games, or market experiments. The penalty mechanism can be generalized to any multi-round game where peer punishment is relevant;
- Mobile Optimization: While the platform currently supports desktop browsers, future development will prioritize mobile optimization to accommodate diverse participant devices. Planned enhancements include implementing responsive design with touch-friendly controls (e.g., larger buttons, improved spacing), refining touch-event handling for smoother interactions, and rigorous cross-device testing on phones and tablets. Additionally, we will integrate accessibility features such as keyboard navigation and colorblind-friendly palettes to ensure inclusive participation. These improvements aim to maintain experimental fidelity while expanding accessibility and ecological validity in online behavioral research.
- Enhanced Analytics: Integrating Python or R for real-time statistical analysis on the backend (e.g., calculating Gini coefficients of token usage, running reinforcement learning updates). This real-time analysis can help the experimenter identify emerging free-riders or exploiters, compare inequality trends across treatment groups and assess the effectiveness of penalties or other mechanisms [6]. Visualizations (charts) could be embedded in the Grafana dashboard for quick insights during the experiment;
- User Experience: While following Ahn et al.'s non-communication paradigm as our baseline, the platform could support modular addition of communication features (e.g., text chat lobbies in later rounds) to study deliberation effects. These optional toggles—which could include structured feedback prompts or real-time chat—enable controlled investigation of communication dynamics and behavioral nudges while preserving core CPR game mechanics. Researchers can thus balance methodological consistency with novel hypothesis testing.
- Bot players: One could simulate players by writing a script that mimics the AJAX calls of a real user. For example, a PHP or Python bot could POST to register.php, wait in check_game_started.php until a group forms, then submit random (or strategy-driven) token requests to process_token_request.php. Because the game logic is encapsulated in these APIs, bots can be inserted without changing core code;
- Alternative benefit curves: Currently, the benefit formula is hard-coded in process_token_request.php. To allow different curves, the code could be refactored so that the quadratic parameters (0.761 and 0.007) come from the settings table. For example, adding fields like benefit_a and benefit_b and reading them (via get_value_from_settings.php) would let the admin experiment with different functions without modifying code;
- Enhanced logging: Right now, game data is stored in MySQL tables. For better debugging or analytics, one

could add a logging layer (e.g., writing key events to a log file or using a PHP logging library). For instance, adding log statements in process_token_request.php or in the polling scripts could record timestamps and event flows. This would complement the existing Grafana integration by providing server-side logs of errors or performance metrics;

- Participant diversity: Another promising extension involves localizing the interface into multiple languages to support cross-cultural experiments. Combined with mobile optimization, this would enable researchers to reach broader participant pools, including underrepresented regions, and conduct comparative behavioral studies in a more inclusive and globally accessible format;
- Modularize frontend/backend: The current implementation uses many individual PHP scripts and inline JS. To improve maintainability, common code (like the database connection or session handling) could be centralized. For example, using a single db.php include, or converting to a simple MVC framework. On the frontend, repeated JS functions (AJAX utilities, session helpers) could be moved into separate .js files rather than embedded in HTML. This would reduce duplication (notice, e.g., fetchWithRetry() appears in multiple pages). Such refactoring would make it easier to add features like a chat interface or a WebSocket server later.

By pursuing these development pathways, our platform will maintain its position at the forefront of online experimental methodology. These advancements will yield a more powerful research tool capable of supporting richer behavioral investigations in CPR and related domains, while fully leveraging modern web infrastructure. The resulting enhancements will enable researchers to address increasingly complex questions with improved methodological rigor and ecological validity. Emerging research in reinforcement learning and AI-enhanced simulation suggests potential extensions in adaptive experiment design [26].

In future research, the platform can be used to simulate alternative governance regimes or collective choice settings relevant to public policy. For example, it could support classroom simulations of carbon budgeting, fisheries management, or land use decisions. Its flexibility enables comparative experiments across cultures or policy environments, offering insights for institutional design and real-world coordination challenges. The platform could support dynamic goal-alignment experiments, as studied in recent CPR cooperation work [27].

7. Conclusions

We have presented the design, implementation, and operation of an open-source web platform for conducting online common-pool resource (CPR) experiments. Faithfully replicating the canonical Ahn et al. [6], paradigm while augmenting it with researcher-focused features—including real-time group dynamics, sanctioning algorithms, and remote monitoring—our system provides a turnkey solution for multiplayer economic games. The technical exposition (covering database architecture, client-server callbacks, and administrative controls) offers implementable insights for developing similar systems using standard web technologies (HTML5/JS, PHP, MySQL). Designed for reproducibility and scalability, the platform reduces technical barriers to rigorous CPR research while addressing key web-experiment challenges. Future enhancements in real-time communication and containerization will further support large-scale studies. The platform's transparency, reproducibility, and adaptability make it a valuable methodological contribution to experimental economics. It enables researchers to test institutional mechanisms under controlled yet scalable settings and offers a practical bridge between theoretical modeling and real-world economic behavior. By combining methodological fidelity with operational flexibility, this work advances the experimental economics toolkit, enabling both laboratory and field investigations of resource dilemmas through a robust, accessible interface. A pilot trial of the platform is envisioned for future implementation in a classroom environment, where it will be used to gather initial user feedback and explore its real-world applicability.

Funding

This work received no external funding.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

No new data were created or analyzed in this study. The paper focuses on the system's design and implementation for online CPR experiments.

Acknowledgments

I would like to thank Dr. Paschalis A. Arvanitidis, Professor of Institutional Economics at the Department of Economics, University of Thessaly, and Director of the Laboratory of Economic Policy and Strategic Planning (L.E.P.S.PLAN), for bringing the common-pool resource (CPR) game and the publication by Ahn et al. [6], to my attention.

Conflicts of Interest

The author declares no conflict of interest.

References

1. Stephen, M.; Rosie, W.; Christian, S. 15.2: Common Pool Resource Theory. In *International Relations*. Social Science LibreTexts: Davis, CA, USA, 2021. Available online: https://socialsci.libretexts.org/Bookshelves/Sociology/International_Sociology/Book%3A_International_Relations_(McGlinchey)/15%3A_The_Environment/15.02%3A_Common_Pool_Resource_Theory (accessed on 1 May 2025).
2. Hardin, G. The tragedy of the commons: the population problem has no technical solution; it requires a fundamental extension in morality. *Science* **1968**, *162*, 1243–1248. [CrossRef]
3. Ostrom, E. *Governing the Commons: The Evolution of Institutions for Collective Action*; Cambridge University Press: Cambridge, UK, 1990.
4. Poteete, A.R.; Janssen, M.A.; Ostrom, E. *Working Together: Collective Action, the Commons and Multiple Methods in Practice*; Princeton University Press: Princeton, NJ, USA, 2010.
5. Arvanitidis, P.; Nasioka, F. From Commons Dilemmas to Social Solutions: A Common Pool Resource Experiment in Greece. In *Institutionalist Perspectives on Development*; Vliamos, S., Zouboulakis, M., Eds.; Palgrave Macmillan: Cham, Switzerland, 2018; pp. 125–142. [CrossRef]
6. Ahn, T.K.; Ostrom, E.; Walker, J. A common-pool resource experiment with postgraduate subjects from 41 countries. *Ecol. Econ.* **2010**, *69*, 2624–2633. [CrossRef]
7. Fehr, E.; Gächter, S. Cooperation and punishment in public goods experiments. *Am. Econ. Rev.* **2000**, *90*, 980–994. [CrossRef]
8. Cárdenas, J.C.; Ostrom, E. What do people bring into the game? Experiments in the field about cooperation in the commons. *Agric. Syst.* **2004**, *82*, 307–326. [CrossRef]
9. Salganik, M.J.; Watts, D.J. Web-based experiments for the study of collective social dynamics in cultural markets. *Top. Cogn. Sci.* **2009**, *1*(3), 439–468. [CrossRef]
10. Chen, D.L.; Schonger, M.; Wickens, C. oTree—An open-source platform for laboratory, online, and field experiments. *J. Behav. Exp. Finance* **2016**, *9*, 88–97. [CrossRef]
11. Chan, S.W.; Schilizzi, S.; Iftekhar, M.S.; et al. Web-based experimental economics software: How do they compare to desirable features? *J. Behav. Exp. Finance* **2019**, *23*, 138–160. [CrossRef]
12. Brandt, D.J.; Megiddo, I.; Head, J.W.; et al. OGUMI—An Android-based open-source mobile application to conduct common-pool resource experiments. *PLoS One* **2017**, *12*, e0178951. [CrossRef]
13. Rezaei Zeynali, F.; Parvin, M.; ForouzeshNejad, A. A.; et al. A heuristic-based multi-stage machine learning-based model to design a sustainable, resilient, and agile reverse corn supply chain by considering third-party recycling. *Appl Soft Comput.* **2025**, *174*, 113042. [CrossRef]
14. Tajally, A.; Babakhani, B.; Jeyzanibrahimzade, E.; et al. Sustainable supplier selection and order allocation problem considering the agility and resilience dimensions: A novel multi-stage data-driven decision-making approach. *Int. J. Syst. Sci. Oper. Logist.* **2025**, *12*, 2458756. [CrossRef]
15. Pazouki, S.; Jamshidi, M.; Jalali, M.; et al. Artificial intelligence and digital technologies in finance: a comprehensive review. *J. Econ. Finance Account. Stud.* **2025**, *7*, 54–69. [CrossRef]

16. Boettiger, C. An introduction to Docker for reproducible research. *ACM Oper. Syst. Rev.* **2015**, *49*, 71–79. [CrossRef]
17. Gürerk, Ö.; Irlenbusch, B.; Rockenbach, B. The competitive advantage of sanctioning institutions. *Science* **2006**, *312*, 108–111. [CrossRef]
18. Yamagishi, T. The provision of a sanctioning system as a public good. *J. Pers. Soc. Psychol.* **1986**, *51*, 110–116. [CrossRef]
19. Liu, Q.; Sun, X. Research of Web Real-Time Communication Based on WebSocket. *Int. J. Commun. Netw. Syst. Sci.* **2012**, *5*, 797–801. [CrossRef]
20. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef]
21. Fischbacher, U.; Gächter, S. Social preferences, beliefs, and the dynamics of free riding in public goods experiments. *Am. Econ. Rev.* **2010**, *100*, 541–556. [CrossRef]
22. Gächter, S.; Renner, E.; Sefton, M. The long-run benefits of punishment. *Science* **2008**, *322*, 1510. [CrossRef]
23. Giamattei, M.; Yahosseini, K.S.; Gächter, S.; et al. LIONESS Lab: a free web-based platform for conducting interactive experiments online. *J. Econ. Sci. Assoc.* **2020**, *6*, 95–111. [CrossRef]
24. Isaksen, E.T.; Brekke, K.A.; Richter, A. Positive framing does not solve the tragedy of the commons. *J. Environ. Econ. Manage.* **2019**, *95*, 45–56.
25. Horton, J.J.; Rand, D.G.; Zeckhauser, R.J. The online laboratory: Conducting experiments in a real labor market. *Exp. Econ.* **2011**, *14*, 399–425. [CrossRef]
26. Sandholm, T.; Crites, R. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems* **1996**, *37*, 147–166. [CrossRef]
27. Tu, C.; D'Odorico, P.; Li, Z.; et al. The emergence of cooperation from shared goals in the governance of common-pool resources. *Nat. Sustain.* **2023**, *6*, 139–147. [CrossRef]