NodeGame: Real-Time Social Experiments in the Browser

Stefano Balietti[1]

1 Computational Social Science, ETH Zürich, Switzerland
∗ **E-mail:** sbalietti@ethz.ch

1 Abstract

NodeGame is a software framework for conducting behavioral experiments online and in the lab directly in the browser window. It allows both real time and discrete time experiments, and with zero-install it can run on a great variety of devices, from desktop computers to laptops, smartphones, and tablets. The source code is modular and the programming language is HTML5 / JavaScript. Extensive documentation is available on the wiki pages about how to configure the platform, and how to create new strategic environments.

Keywords: experimental economics - software - real time - browser - online - open source.

Online Material: http://nodegame.org/

2 Introduction

Proponents of online vs laboratory experiments sometimes "quarrel" with each other about the pros and cons of the two approaches. What follows below, is the reconstruction of a fictitious encounter in the imagined arena of academic debate between two staunch exponents of opposite sides.

Online behavioral scientist (O): "Online research brings about several advantages over traditional laboratory experiments, such as: lower costs, shorter times for data collection, access to a broader subject pool, and the possibility to test participants in a more natural environment (Musch and Reips, 2000; Birnbaum, 2004; Paolacci, Chandler, and Ipeirotis, 2010; Reips and Krantz, 2010). Moreover, online experiments can also be used for didactic purposes in the lecture hall (Rubinstein, 1999), or for conducting field-research (Levitt and List, 2009)."

Laboratory behavioral scientist (L): "Research performed in the lab, even if more constrained than online research in the dimensions that you just highlighted, has the great advantage of taking place under rigorously controlled conditions. This allows to exclude many possible confounding explanations, minimizing the presence of noise in the data, and, therefore, also facilitating the replication of experimental results (Friedman and Cassar, 2004; Guala, 2005)."

O: "Actually, the voluntary nature of online participation can often lead to the production of better quality data than laboratory experiments, because in the laboratory subjects feel obliged to stay in the lab even when they have stopped paying attention to the task at hand (Reips, 2002)."

L: "This is exactly the point. Online research experiences significant rates of dropouts, and this undermines the degree of randomness in the assignment of participants to treatment conditions. Moreover, it can also be a source of self-selection bias (Kraut et al., 2004)."

O: "That is true, but there are effective measures to handle dropouts, such as 'warm-up phases,' and 'seriousness checks.' As for the self-selection that you mentioned, it can be controlled for using the 'multiple-site-access' technique (Reips, 2002; Reips and Krantz, 2010)."

L: "Fair enough, but there are also other issues. For example, the enhanced anonymity of online experiments can generate more deviant behavior in the data (Kraut et al., 2004). Moreover, participants

could purposely try to damage the experiment, or trying to tamper with its code in order to advance quicker through the answers."

O: "I would disagree. I see the enhanced anonymity of the web as a valuable asset because it reduces the experimenter bias in the first place, and furthermore it permits to do research on more sensitive topics or on individuals with rare conditions of interest (Mangan and Reips, 2007). Moreover, multiple submissions and other form of hacking are not so common, according to mine and other online experimenters' experience (Birnbaum, 2004)."

L: "I see, but you need anyway to take into account that people might exhibit a different behavior online and in person to a certain extent (Bargh and McKenna, 2004)."

O: "That should not be an issue, in fact the results from Internet-based investigation are often qualitatively comparable to other traditional methods (Horswill and Coster, 2001; Buchanan, Johnson, and Goldberg, 2005; Luce et al., 2007)."

L: "But some studies simply cannot be conducted on the web; for example, when some physiological parameters need to be measured directly, or when specialized hardware is required. Moreover, on the web there is no possibility to explain the instructions to the subjects, in case they have doubts (Reips, 2002)."

O: "That is true, but on the other hand, on the web it is possible to perform studies which would not be possible to conduct in the laboratory; for example studies testing large groups for an extended period of time (Butler, 2001; Bos et al., 2002; Salganik, Dodds, and Watts, 2006; Centola, 2010)."

L: "..."

O: "..."

While the two sides keep debating about their favorite approaches, the rest of the paper introduces nodeGame,[1] a new software framework that tries to blur the boundary between lab and online experimentation, by making behavioral experiments directly available in the browser window. An experiment – or a game[2] – implemented in nodeGame can run in any device equipped with a browser, be it personal computer, or a computer in the university laboratory.

2.1 You and nodeGame: Two Exemplary Use Cases

The biggest methodological advantage offered by nodeGame is the ease of switching between the two settings: lab and online. In fact, either of the two can be used to calibrate, validate or refine the results of the other. Consider the following two scenarios.

From Online to Lab

Researcher You[3] is developing a new theory about a known behavioral phenomenon. His or her hypotheses have not been previously examined in the literature. Therefore, as it is usual in the process of building a new theory, researcher You needs to make important choices regarding the selection and calibration of the model variables. You could proceed with the investigation as follows: (i) use intuition based on personal observations, (ii) develop an additional computational model to test the plausibility of the various model parameters in a (usually agent-based) computer simulation, or (iii) perform a certain number of pilot laboratory experiments to empirically verify the preliminary assumptions. The latter would be

[1] The name nodeGame comes from the technology used for its implementation: Node.js. For details about Node.js and architecture of nodeGame refer to Sec. 6. The name nodeGame has also affinity with the fact that each client of the game is also a separate computing node.

[2] There exist fundamental differences between an experiment and a game, but the two concepts are not totally disjoint. In fact, many experiments in the behavioral sciences come in the form of games that involve strategic interactions among players. nodeGame is a flexible environment that permits to implement scientific experiments, decision tasks and combinations thereof. In this article, the words game and experiment will be used mostly interchangeably.

[3] Using a Korean / Chinese family name for representing an hypothetical social science researcher is a rhetorical artifact to create empathy with this fictitious character.

the most methodologically-sound way of proceeding, but also the most expensive, both in monetary terms, and in terms of time needed to perform the recruitment and running the pilots. However, having recently discovered nodeGame, You decided to run the pilot experiments using one of the many online labor markets.[4] With nodeGame, You could fast-prototype his or her theory with a number of targeted behavioral experiments, scanning the space of model parameters online, and then move to the lab to study the relevant parameters under well-controlled conditions. This methodological approach allows You to get the best of the two worlds, lab and online: (i) getting a more precise behavioral calibration of the model parameters of new theories still under development, and (ii) saving time and money as well.

From Lab to Online

Researcher You has completed his or her study in the laboratory and submitted a paper for review to a prestigious behavioral economics journal. Notwithstanding the though competition in this field, You was lucky enough to raise the editor's interest and have the work sent out to referees. However, during the evaluation phase a meticulous reviewer suggests an alternative explanation for the social mechanism studied by You. The explanation is incompatible with You's theory, and cannot be excluded by the data of the original experiment. Therefore, further empirical evidence is needed to put the alternative hypothesis under test. However, it is exams period in You's university, and it is unlikely for You to recruit the students for a new experiment in a short amount of time. Therefore, You decides to answer the reviewer's concerns with an additional online experiment. In fact, in nodeGame, the same experiment that You designed for the lab can run *as it is*[5] also online. In a short amount of time and for a little cost, You could test the alternative hypothesis raised by the scrupulous reviewer, discard it with the aid of empirical evidence, and secure the publication.[6]

2.2 Outline of This Paper

The scenarios depicted in the introduction are just two examples of the possibilities offered by a tool like nodeGame. In fact, nodeGame brings about several methodological advances, as it will be illustrated in the next sections. The remainder of the paper is organized as follows. Section 3 gives an overview of the nodeGame software, the design principles, and the domain of applicability. Section 4 describes how to create a new experiment in nodeGame, trying to avoid the most technical details, which are available in the online wiki on the project website. Section 5 highlights some of the main features that make nodeGame particularly suited to carry out online experimentation. Section 6 gives a bird-eye view on nodeGame technical architecture, and might be as well skipped if the reader is not a developer. Finally, Sec. 7 summarizes the advantages and the current limitations of nodeGame, concluding the paper with some general considerations about the use of open source software in scientific research, and about the need of putting the experimental method into an even more central position in the social science disciplines.

3 nodeGame

As outlined in the previous section, online experiments complements laboratory experiments on several dimensions. However, interactive online experiments are still a rare commodity nowadays (Horton, Rand,

[4] Common online labor markets include Amazon Mechanical Turk, oDesk, Freelancer, Elance, Guru, etc. Depending on the choice of the online labor market, some restriction to the use of the recruitment platform may apply. In such a case, it is always possible to recur to crowd-sourcing intermediary companies such as CrowdFlower, Innocentive or SurveySampling, however costs might be slightly higher. If monetary incentives are not required by the experimental design is simply possible to upload a link on websites like "Psychological Research on the Net" (`http://psych.hanover.edu/research/exponnet.html`) or "The Web Experiment List" (`http://www.wexlist.net`) or "Online Social Psychology Studies" (`http://www.socialpsychology.org/expts.htm`) listing experiments currently active on the Internet.

[5] Minimal adjustments might still be needed, for example to apply adequate browser requirement checkings.

[6] The great impact of the newly developed theory earned You a tenured position.

and Zeckhauser, 2011). One of the reasons has certainly to do with the lack of a widely established software for conducting synchronous online experiments, and with the technical difficulty of copying with the challenge of creating one. Z-Tree (Fischbacher, 2007) has proven to be an excellent software for conducting lab research, but does not scale well to the web, or to a large number of participants. Some attempts have been made to fill the gap, however, none of them has yet become a standard. Examples are: EconPort (Cox and Swarthout, 2005), ConG (Pettit et al., 2014), oTree (Chen, Schonger, and Wickens, 2014), Wextor (Reips and Neuhaus, 2002), FactorWiz (Birnbaum, 2000), and the MIT Seaweed project (http://dspace.mit.edu/handle/1721.1/53094).[7]

nodeGame is designed and implemented taking into account the features and limitations of current experimental software, aiming to accomplish the following goals:

1. use only open source and free software technologies
2. realize a robust and fault-tolerant application
3. run discrete- and real-time experiments
4. support hundreds of simultaneous participants,
5. grant flexibility and fine-grained control to the experimenter.

To the best of our knowledge, no other experimental software besides nodeGame has so far achieved all 5 design goals simultaneously.

3.1 nodeGame Overview

nodeGame provides a programming framework, which is easy to extend and customize. This is accomplished via the nodeGame API (Application Programming Interface). The language of the framework is JavaScript both on the client and on the server. JavaScript is the scripting language of the browser, and it is also an efficient language for dealing with asynchronous I/O (Input/Output) operations in the back-end, such as accessing a database or the file system. Through the nodeGame API, the experimenter obtains a fine-grained control over both experimental variables and technical settings. However, most of the complexity regarding the latter is hidden away by the choice of a number of pre-configured default options. In this way, for common experimental designs, the experimenter does not need to care about technical details and can concentrate on the development of his or her own experiment.

nodeGame is entirely based on open web technologies (HTML5, CSS, JavaScript). Therefore it allows to conduct more complex experiments than what is usually done with traditional software available to experimentalists. For example, nodeGame can go beyond traditional turn-based experiments, and, by exploiting the power of HTML5 WebSockets,[8] it also allows to conduct real-time games. Moreover, nodeGame integrates seamlessly with popular web development third-party libraries, such as jQuery and D3,[9] to create stunning visualizations and rich user interfaces.

Tapping into the array of web development technologies provides extra-freedom in the creation of scientific experiments. On the other hand, it requires the skills to make use of them. The traditional know-how of behavioral experimentalists might not include an education in web development techniques. Those experimenters lacking the time or the programming skills to implement experiments in nodeGame can easily recruit web developers from the job market at a reasonably low cost. However, those experimenters

[7] For more platforms refer to the list of available experimental software maintained by the UAA Experimental Economics Laboratory (http://econlab.uaa.alaska.edu/Software.html

[8] WebSockets represent a major improvement in the history of web data communication. For more information see http://en.wikipedia.org/wiki/WebSocket or https://www.websocket.org/.

[9] jQuery (http://jquery.com) is a cross-browser JavaScript library designed to simplify the client-side manipulation of an HTML page. D3 (http://d3js.org) is a JavaScript library for creating data-driven interactive visualizations.

willing to enrich their programming skill set can find many good books and plenty of accessible online tutorials to get started with the JavaScript language and related technologies.[10] Furthermore, to facilitate the learning and usage of nodeGame, the website of the project offers additional resources. For example, a one-click automated installer script is available to download and install the latest stable or development version of nodeGame on Linux, Mac OS X, and Windows machines. Moreover, the online wiki can guide the users step by step in configuring the platform, and in the creation of new experiments.

nodeGame allows to design and implement experiments online and in the lab directly in the browser window. It is completely cross-browser and cross-platform. Desktop devices (Windows, Mac OS X, and Linux) and mobile devices can connect to the nodeGame server and join an experiment at the same time. Every new release of nodeGame is rigorously tested against an array of different browsers to guarantee its correct functioning.

nodeGame can be used to run experiments on online labor markets. One of those, Amazon Mechanical Turk (AMT), has been extensively used for conducting online psychological, sociological and economic research (Paolacci, Chandler, and Ipeirotis, 2010; Mason, 2012). The integration between AMT and nodeGame is based on a shared list of authorization codes, which are assigned to online workers upon accepting a task and are checked by nodeGame as a requirement for participation. The upload and the management of the work task on AMT must be done by the experimenter outside of nodeGame.

nodeGame is under steady development, but it has already been used by several research teams to run experiments in the laboratory and online, such as: ultimatum games, public-goods games with and without assortative matching, prisoner dilemma games, burden-sharing games, art-exhibition games, and congestion games (Balietti, Goldstone, and Helbing, 2015; Anderson et al., 2015; Balietti, Jäggi, and Axhausen, 2015). Fig 1 contains screenshots of the nodeGame interface for public-goods games with noisy assortative matching.

Finally, in addition to online and laboratory behavioral experiments, nodeGame can also be used to perform other types of data collection, such as surveys and field experiments, as well as for didactic purposes in the lecture hall.

4 Creating Experiments with nodeGame

The purpose of this section is not to be a thorough user manual for nodeGame, but rather to present the main concepts behind the creation of experiments in nodeGame. Technical details and complex code examples will be avoided because they might change from one version to another of the software, and because they can be found on the online wiki of the project (`http://nodegame.org/wiki/`).

To start the creation of a new experiment, open a terminal and type:

```
nodegame create-game myexperiment
```

This command will automatically add a new folder in the nodeGame games directory containing all the necessary files to run a default experiment. When the server is started, it will open the "myexperiment" *channel*, that is an URL of the type http://myserver/myexperiment/ where participants are able to join the experiment. It is possible to add as many games and channels as needed. By default, each channel contains a *waiting room* that will automatically start a new *game room* with a given treatment as soon as enough participants have connected.

To run an experiment in nodeGame the experimenter will have to implement: (i) the game sequence, (ii) the client types, (iii) the game configuration (including treatments), and optionally (iv) the requirements and authorization rules.

[10] For example, see the JavaScript guides at the addresses `http://www.w3schools.com/js/` and `http://javascript.info/`.

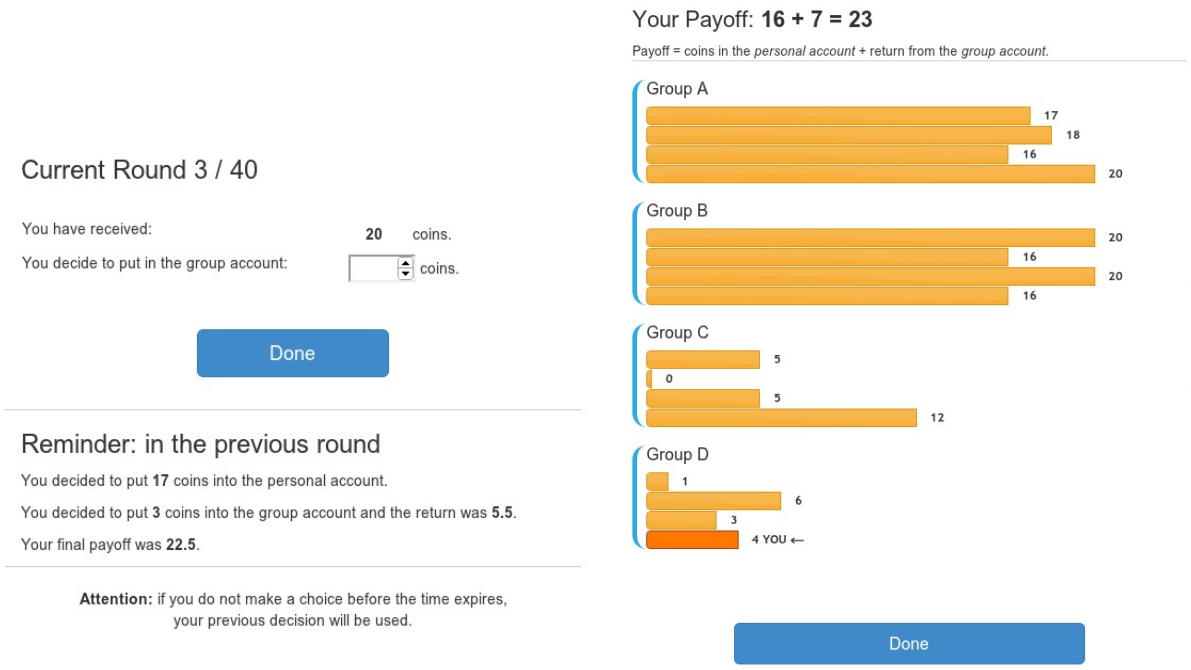

Figure 1. Screenshot of the nodeGame interface of a public-goods game. Left: the interface allows to place a contribution between 0 and the full endowment, and displays the previous contribution decision and consequent outcome to the participant. Right: the interface shows the groups formed through a noisy assortative matching based on the subjects' initial contributions. Within each group payoffs realize.

4.1 The Game Sequence: Stages, Steps, Rounds, and Blocks

The game sequence is the set of *stages* that will be executed sequentially after the experiment begins. The stages of an hypothetical ultimatum-game experiment could the following: (i) instructions, (ii) quiz, (iii) game, (iv) questionnaire. Each stage must have at least one *step* inside, consisting in a unique name and an step callback function. Inside this function, the experimenter defines the page that will be loaded, how to handle user input, and what information should be sent to server (more details in the next subsections).

Steps are useful for a number of reason. Firstly, they can be used to break down big chunks of text across several pages. In our case, the instructions stage is, in fact, further subdivided in three steps: instruction_1, instruction_2, instruction_3. Moreover, steps can be used as a synchronization checkpoint for all clients connected to the same game. For example, the assignment of clients into sub-groups could be done into a designated step. Furthermore, in turn-based games, steps naturally correspond to the turns as they are usually defined in classic game-theoretical problems. In our example, the steps of the game stage could consist in: (i) making an offer / waiting for an offer, (ii) accepting or rejecting it, (iii) displaying the results of the actions of the players. The same outcome could be achieved by condensing all the actions in a single stage, but separating them across individual steps has several advantages. Firstly, the duration of each step is automatically calculated, and a time interval is stored on the server. Secondly, it is possible to skip or jump to a specific step from the administration panel. Thirdly, upon disconnection and subsequent reconnection of the same player, steps guarantee a more fine grained recovery point. Finally, steps permit to have a clearer and more maintainable code.

An entire stage can then be repeated multiple times, each of those constituting a different *round*. The number of rounds can be predefined before the experiment starts, or it can be determined at runtime, upon fulfillment of certain criteria, e.g. players reaching an agreement on their decision task. In our example, the game stage will be repeated for three rounds, during which players will be randomly assigned to the role of bidder or respondent.

Fig.2 A shows the stages and steps described so far in a diagrammatic form. In the nodeGame programming language they would correspond to the following code:

```
stager.stage("instructions")
      .step("instructions_1");
      .step("instructions_2");
      .step("instructions_3");

stager.stage("quiz");

stager.repeatStage("game", 3)
      .step("offer");
      .step("respond");
      .step("display_results");

stager.stage("questionnaire");
```

Finally, for more advanced setups, it is possible to group stages or steps in *blocks*. The position of a block in the game sequence, as well as the order of the elements inside each block, can be randomized, granting the experimenter a lot of freedom in the definition of the game sequence.

4.2 Client Types: Different Implementations of the Game Sequence

Once the sequence of stages and steps in a game is defined, what happens in each step must be implemented. Multiple implementation of the same game sequence are allowed, and each one is defining a

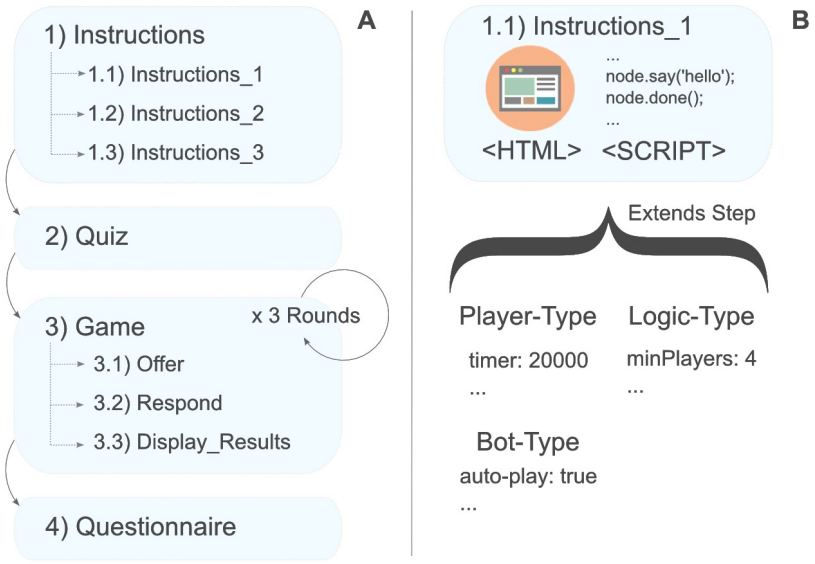

Figure 2. Creating an Experiment with nodeGame. (A) Simplified representation of the stages and steps of an ultimatum game implemented in nodeGame. (B) Simplified representation of a fictitious Instructions_1 step. The step contains a unique name, an execution callback, and an HTML page. In this example, three client types (Player, Logic, and Bot) extend the step by adding further properties to the step object.

different *client type*. What a client type does in the corresponding step of the sequence can be radically different depending on what the purpose of the client type is.

Two client types are mandatory: (i) the *player* type, and (ii) the *logic* type. The player type is assigned by default to every newly connecting browser, and it has the purpose of displaying the pages of the game to human participants and handle their interaction with the screen. In general, the step callback functions of a player client type should also include an HTML page to be loaded in the browser and some code to respond to user generated events on the page (the movement of the mouse, or the click on a button) and to exchange messages with the server. When the step ends, the client state must be set to "done." If conditions are met, e.g. all players in the game are "done," then the game will proceed forward. More details about different synchronization options are provided in Sec. 5.6.

The logic type is executed on the server and controls the flow of operations of a game room, such as creating sub-groups, accessing external datasets, handling player disconnections, etc.

Finally, client types can also be used to create automated players (see Bots and Phantoms in Sec. 5) that interact with humans, or which test the code of the experiment before launching it. Fig 2 B shows schematically different client types extending the same game step.

4.3 Experimental Variables, Treatments and Waiting Room

The settings of the experiment are all specified in the same file: `game.settings`. This file contains all the variables needed to define the functioning of the experiment, such as: how many monetary units are assigned to players, how long a timer run in a step, the conversion rate from experimental currencies to real currencies, etc. These variables can be grouped together under a common label to define a *treatment*.

The number of players needed to start the game is instead defined in the settings of the waiting

room, file `waitroom.settings`. The file allows to configure the waiting room with other options such as: maximum waiting time, maximum number of spawned game rooms, and how treatments are assigned to game rooms; alternatively it also possible to set a specific date in the future when the game will be started.

4.4 Authorization and Requirements

Writing authorization and requirements rules is not mandatory, and usually not necessary, for laboratory experiments, but it constitutes a good-practice for every online experiment. More on this topic is found in the description of the Authorization and Requirements feature in Sec. 5.

5 Features

In this section, a list of the most interesting features offered by nodeGame is reported in alphabetical order. Given that nodeGame is constantly being developed, such a list is not to be considered exhaustive.

5.1 Authorization and Requirements

The web is a great source of anonymity. Some online experiments actually requires it, but for some others, probably the majority of them, the ability to identify the client machines of the experimental participants is actually key. In fact, this allows to bar the same participant from entering the same game room multiple times, or to exclude those who have already played the experiment from taking part in subsequent repetitions. nodeGame provides a default authorization system which stores authorization tokens in the browser.[11] Moreover, it is possible to define a custom authorization function which can accept or reject incoming connections based on properties such as ip, browser type, etc.

However, even if authorized to take part in the experiment, clients might not actually possess the technical requirements to go through it and complete it safely. In fact, in the Internet there are hundreds, or even thousands, of different browsers and browser versions, each one equipped with a slightly different implementation of the HTML and JavaScript standards. In many cases, this does not represent an issue, but there are some situations to keep under control. For example, Internet Explorer browsers below version 10, are notoriously famous for being non standard-compatible. They can generate glitches in the visualization of the page, or in the communication with the server. Therefore, it is necessary to test the correct functioning of nodeGame on every client before letting it into an experiment. Most people participating in online labor markets usually have multiple browsers installed in their machine, and they can be invited to retry connecting to the experiment with another one.

Clients connecting from mobile devices, such as tables or smartphones, also require special attention. Differences in the size of the displays should be taken into account, and clients with requirements below compliance should not be accepted. Making use of CSS frameworks helps obtaining a consistent cross-device visualization of the experiment.[12] Therefore, they should always be used when performing online experiments. Furthermore, clients connecting via mobile devices might not only experience visualization issues due to smaller size of the display, but also encounter connectivity troubles due to a sudden failure of the mobile network they are connected to. If that is an issue for the design of the experiment, mobile clients should be excluded.

[11] Authorization tokens can be stored as cookies or in local storage of a browser. Cookies have more limitations, but are widely supported. Local Storage is available only on recent browsers. For further information see http://en.wikipedia.org/wiki/HTTP_cookie and http://en.wikipedia.org/wiki/Web_storage.

[12] A CSS-framework is a collection of HTML- and CSS-based design templates and JavaScript modules for developing responsive, mobile compliant web interfaces. nodeGame makes use of CSS-framework Twitter Bootstrap (http://getbootstrap.com/).

By default, in nodeGame the following requirement checkings are performed: whether the client supports JavaScript at all, if it provides some sort of persistent storage (cookie or local storage), if it can load new HTML pages in a dedicated iframe, and if it can communicate with the server successfully, and without an excessive delay. However, the experimenter can add any other checking, as he or she thinks it is required by the game. For example, it is possible to impose restrictions based on the location of the client. Some online labor markets allow to specify similar constraints when uploading the task, but this does not always lead to accurate results. HTML 5 Geolocation API can localize clients even at the level of single streets in dense urban areas.[13] However, clients must allow to be geolocated, and this could be introduced as a requirement for participation in the experiment.

5.2 Bots and Phantoms

Bots and phantoms are two computer-controlled client types executed in the server (see Sec. 4.2). The only difference between them consists in the fact that phantoms are executed in a headless browser, while bots are not. A headless browser is a normal web browser without a graphical user interface. It behaves exactly as a normal computer browser, with the only exception that it does not show the rendered pages to any human being. This means that phantoms are able to actually load a full HTML page like human players would do with their "headed" browser. This makes phantoms particularly suited for testing and debugging an experiment before it actually launching it. Bots, on the other hand, cannot load HTML pages, and therefore are much more light-weighted processes. This means that an experimenter can create a large number of them without affecting memory and CPU load too much. Bots can be used to replace a disconnected human player during an experiment which requires a minimum number of participants, or to play alongside humans in an interactive environment.

5.3 Disconnections and Reconnections

In the online world there is no guarantee that a client will stay connected until the end of an experiment. Participants can leave a previously joined experiment for any reason. For example, a network failure could cause the disconnection. Alternatively, participants might have simply miscalculated the time available for completing the whole experiment, therefore ending their commitment before reaching the end. Finally, they could get bored and go find another more interesting or more remunerative online task. Whatever the reason is, it is necessary to take action when a disconnection happens. nodeGame makes it possible to specify a "disconnection handler" associated with a minimum number of players that need to stay connected in order for the experiment to continue. This handler can be global, i.e. throughout the whole experiment, or it can be attached to single stages or steps. In this way, the minimum-players-connected condition is verified only when really needed. In fact, it is common that some parts of an experiment can be executed with a variable number of players, while others have stricter requirements. For example, if at the end of a collective behavior experiment participants are presented with a final questionnaire, a single disconnection in this stage should normally not affect any other player.

nodeGame has implemented a default behavior for handling disconnections: it immediately pauses the experiment displaying a notice to all connected clients, and simultaneously starts a countdown of sixty seconds, at the end of which the experimental session is canceled and the remaining clients are redirected to an exit stage, which can be a questionnaire or a payment stage, as specified by the experimenter.

5.4 Monitor Interface

Every experimenter needs a "control room" to monitor the correct progressing of the experiment and to fight back the impatience of having a first glance at the direction that the experimental results are

[13] The HTML 5 Geolocation API is available on all major modern browsers. Since it can compromise user privacy, it cannot be used unless the user approves it. For info see http://en.wikipedia.org/wiki/W3C_Geolocation_API.

taking. Fig. 3 shows the web-based administration interface of the nodeGame server.

The monitor interface includes the list of all games currently available, the list of game rooms spawn by the waiting rooms, and the list client connected. The state of every client is reported, including the client type, and the stage of the game they are currently in. Moreover, the interface allows to send group or individual game-commands to pause, resume, advance or restart the game. Finally, a chat window can be opened to communicate with participants in need of assistance. A separated tab lists the configuration of each game, for the experimenter's convenience.

5.5 Resources Caching

Static resources, such as HTML pages, JavaScript scripts, and Cascading Style Sheet (CSS) files, can be explicitly cached at any time during an experiment. Usually, this is done in the initialization phase of the experiment, together with the setting of other constants and environment variables. Caching provides a twofold advantage. Firstly, it helps decreasing the requests load on the nodeGame server machine (specially when an experiment includes stages that are repeated for many rounds). Secondly, it guarantees a smoother game experience, by bringing down the transition time between two subsequent stages or steps.

Caching might not available in older browser (e.g. IE8) with particular restrictions on iframes manipulations.[14] To detect and handle those situations in advance, a pre-caching test is performed automatically upon loading nodeGame, and the results are saved in a global variable.

5.6 Synchronization

One of the main challenges of large scale online experiments is achieving synchronization in an efficient way. In nodeGame, synchronization is achieved by the division of the game sequence into stages and steps, and by deciding which *step-rule* applying for each of them. A step-rule is a function specifying the condition for entering into the next game stage or step. A number of predefined rules are available. For example, *SYNC_STEP* waits for all the players to have terminated the current step; *SYNC_STAGE* lets players advance freely trough the steps of the same stage, but blocks them from entering into a new stage; *SYNC_GROUP_STEP* and *SYNC_GROUP_STAGE* achieve synchronization at the level of the step or the stage, only among the clients that were assigned to the same sub-group. If none of those suits the need of the experimenter, it is always possible to define custom step-rules.

nodeGame can make use of different synchronization strategies, to best adapt to different situations. By default, the logic client type is orchestrating the synchronization of all connected clients. However, it also possible that clients synchronize themselves and step automatically. The latter require more messages to be exchanged, because every stage update needs to be propagated to all the other connected clients as well. Therefore, leaving the game room logic controlling the stepping of every client represents a safer choice under normal conditions.

5.7 Templates and Internationalization

nodeGame supports the dynamic creation of HTML pages from templates which are rendered upon request from a connected client. Templates allow to write modular web pages that can be filled with blocks of content depending on the actual configuration of the experiment. As it will be explained below, the use of templates presents multiple advantages, and therefore they should always be preferred over static HTML pages whenever possible.

[14] Iframes are special HTML tags which are used to visualize entire HTML documents inside of a nested HTML page. nodeGame uses iframes to render the pages of the different stages of an the experiment. For more information about iframes see http://en.wikipedia.org/wiki/HTML_element#Frames.

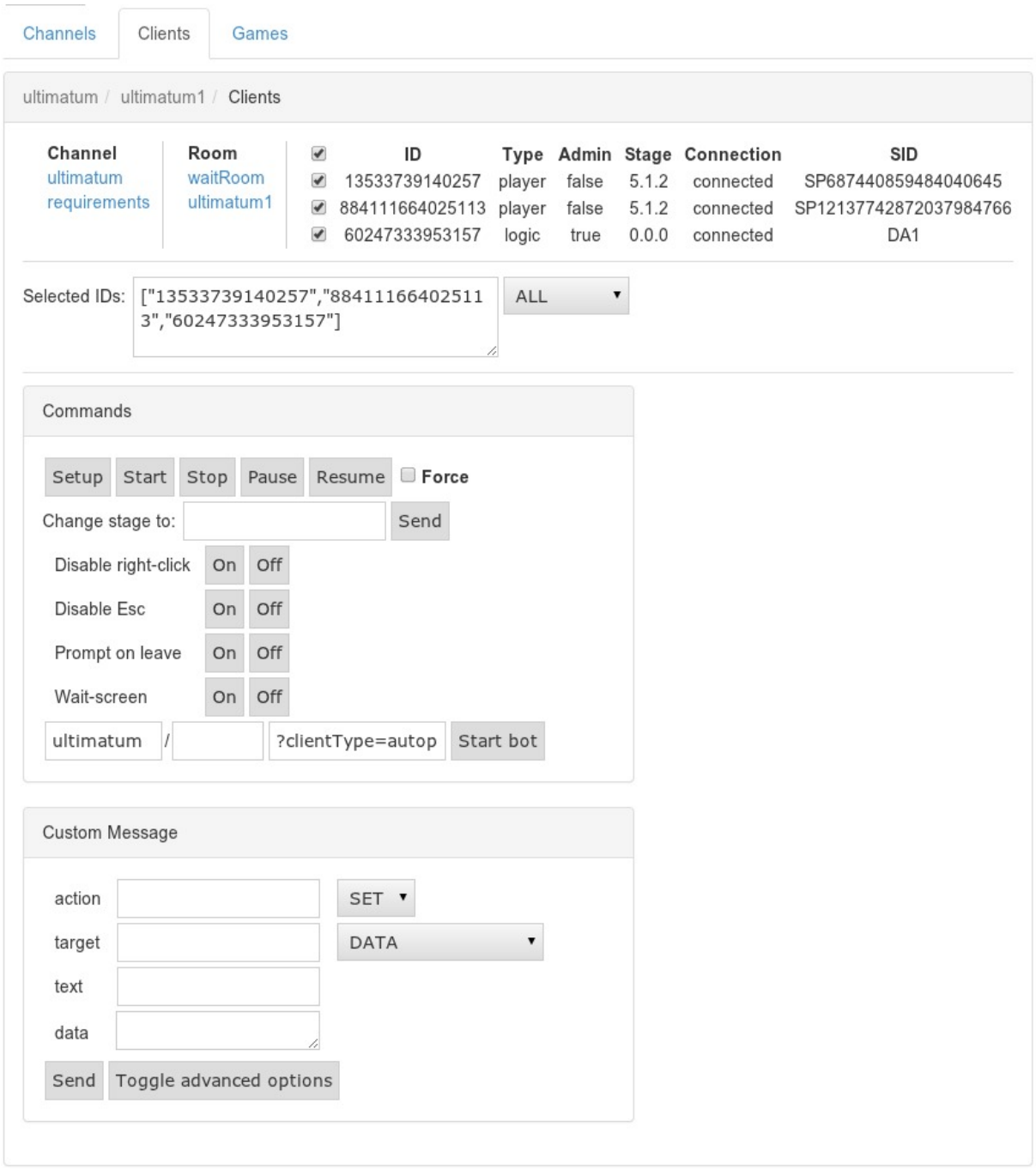

Figure 3. Screenshot of the monitor interface. At the top of the panel, it is possible to browse the game rooms currently active in the channel. The clients connected to the selected room ("Ultimatum1") are shown: two players and one logic. The middle panel allows to send a game command to the clients selected in the upper panel. Furthermore, it allows to performs other actions, such as manually connecting a computer-controlled client. The lower panel permits to create and send a custom game message to the clients selected in the top panel.

Firstly, templates introduce a clearer separation between changeable and static parts of the user interface. For example, in an ultimatum-game-like experiment, participants would divide a certain budget B of monetary units between a bidder and a respondent. An experimenter could hard code the actual value of B in every page, but this would make the code very hard to maintain whenever the value of B is updated. Instead, by using a template, the value of B would be automatically inserted in every page by the template engine. In this way, updating the value of B in the settings of a treatment does not require further modifications in other parts of the code.

Secondly, templates allow to create nested layout structures which reduce the complexity of the markup of the single components and the load on the server upon requesting them. A nested layout usually consists in a fixed outer frame and a variable number of interchangeable blocks. With this configuration, instead of requesting a whole new page to the server every time, only the block actually being updated will be downloaded.

Finally, templates allows to achieve the internationalization of an experiment, e.g. the display of the text of the experiment in different languages. This can be obtained separating the translation files in different directories (contexts) corresponding to the different prefixes of the requested languages, e.g. "en", "de", "it", etc. They will be automatically matched upon receiving a new request, and the properly localized page will be rendered and sent to the client.

5.8 Timers

Response times can be used as an indicator of the type of internal reasoning process used by decision makers. Its systematic analysis can reveal a personality trait that makes use of heuristics vs iterative, rationale thinking (Rubinstein, 2013), or whether a person is expected to play more cooperatively or more selfishly in interactive games (Rand, Greene, and Nowak, 2012). To support research investigating these questions, nodeGame has a dedicated API for defining new timers, and measuring time intervals between specific events. Most importantly, such intervals are measured directly on the client machines, meaning that they already exclude the network latency from the calculations. The precision of the timers is in the order of hundreds of milliseconds. Lower intervals can still be measured, but less reliably depending of the load on the client machine.

Furthermore, timers are by default synchronized with the flow of game commands, such as pausing or resuming a game, so that they are automatically paused and resumed as the game goes along. Finally, the value of a timer can be easily visualized on the screen in different formats using the appropriate widget (see Sec. 5.10).

5.9 Waiting Rooms and Game Rooms

Behavioral experiments usually take place with a fixed number of participants, according to the experimental design. In the laboratory, participants arrive to the experimental facility more or less all at the same time, therefore there are no major problems in scheduling the begin of an experiment. Unfortunately, the situation online is a bit more complicated. Clients usually arrive to the experimental server a few at a time, meaning that those who connected first have to wait in a landing page until all the required slots for participating are filled. This is why every online experiment needs to make use of a so-called *waiting room*.

The waiting room has the purpose to start a new experiment when certain criteria are met, e.g. the number of clients connected simultaneously, or the total waiting time passed. Before the next game begins, participants in the waiting room usually receive information about how much time they have been waiting, how many other players are still needed, etc.

nodeGame provides a default type of configurable waiting room that spawns a new *game room* whenever a certain number of players is available. The desired pool of players can be larger than the group size required by a game room, and in that case participants will be randomly selected. Treatments are

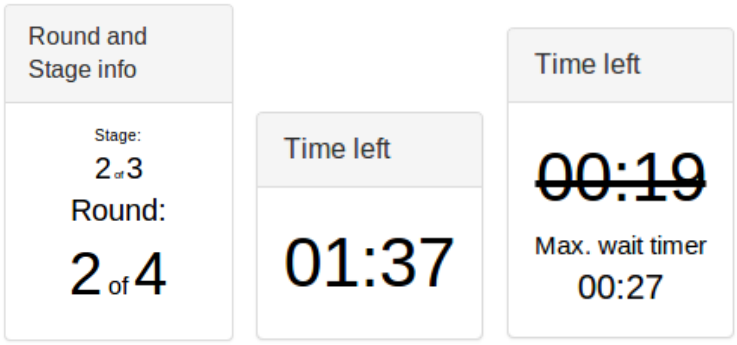

Figure 4. Illustration of the VisualRound and VisualTimer widgets for nodeGame. On the left, the VisualRound widget visualizes the stage and round counts. In the middle, the VisualTimer widget shows the default time countdown. On the right, a participant completed the stage and this triggered the VisualTimer widget to stop; the timer now shows the maximum time left until all other participants complete the stage.

assigned to game rooms randomly, sequentially, or according to custom criteria. What is most important, every new game room created by the waiting room is completely independent by the others. Therefore, it is possible to run several repetitions of the same experiment in parallel, with the same or different treatments.

The waiting room can also operate in different execution modes. For example, it is possible to fix a date and a time in the future when a new game room will be created, or to manually start the experiment from the monitor interface (particularly useful in the lab environment). For each execution mode, several additional options are available to give the experimenter a fine control over the conditions for dispatching new games. Finally, in the case of more complex experimental setups, it is also possible to specify the use a completely customized waiting room, in place of the standard one.

5.10 Widgets

Widgets are reusable user interface components that are available to be loaded dynamically in any nodeGame experiment. They naturally serve multiple purposes. For example, they can display the values of internal timer objects, the number of remaining rounds in a stage, the amount of monetary rewards gained by a player so far, etc. They can also offer a chat window to communicate between the experimenter and a participant, or simply display debugging information useful during the development and testing of the experiment. Fig 4 shows some examples of widgets displaying timers and counting the number of rounds.

6 Architecture

nodeGame is a modular framework, entirely implemented in JavaScript / Node.js. JavaScript is the standard programming language of the browser, it is completely event-driven and permits to respond to users' actions, such as the click of a button, and to fully manipulate the content of an HTML page. Node.js is its server side equivalent. Based on the V8 JavaScript engine used by Chromium and Google Chrome, Node.js can execute CPU intensive tasks with performances comparable or even superior to other interpreted programming languages (Bezanson et al., 2014). However, the greatest advantage of Node.js is that it is entirely event-based, exactly like JavaScript. This means that Node.js deals with I/O (Input/Output) requests asynchronously, making access to resources such as file system and databases

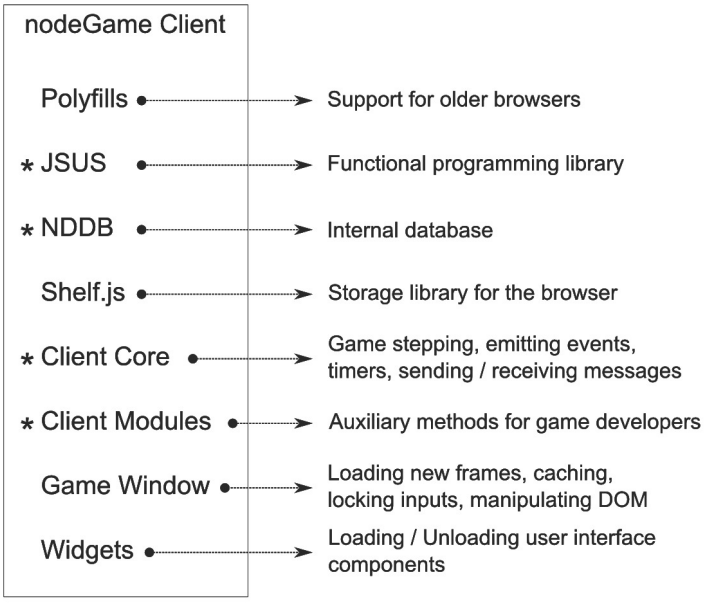

Figure 5. Schematic representation of the nodeGame client architecture. nodeGame client is composed by a collection of modules serving different purposes. In the browser, all of them are usually loaded, while on the server only those modules marked with an asterisk are used by logics and bots.

"non-blocking." Let us consider the following example to clarify the concept. When a client connects to the server for the first time, its credentials need to be verified. Usually, this implies an access to the database, a task that Node.js delegates to the operating system. In this way, while the credentials are being retrieved from the database, Node.js can serve another request. After a certain amount of time, the operating system returns the results of the database operation to Node.js, which processes and serves them to the requester as soon as possible. This feature makes Node.js particular suited to handle a large number of simultaneous connections, optimizing application's throughput and scalability.[15]

Beside performance and scalability, using Node.js as the server side language introduces another important advantage: *code re-use*. In fact, nodeGame follows the Client-Server paradigm, and shares the same classes and data structures between its two main architectural components: (i) nodeGame client and (ii) nodeGame server. This means that an instance of nodeGame client can be executed on the browser to respond to a player's action, or on the server machine to control a game room (logic) or as an automated player (bot). As Fig. 5 shows, nodeGame client itself is a modular application, and nodeGame server imports only those components that are actually needed from it.

The rest of the nodeGame infrastructure includes a number of popular software packages as dependencies to handle non-core tasks such as implementing the network transport layer or the logging system. As shown in Fig. 6, nodeGame uses Winston[16] for logging, Express[17] for answering HTTP requests, and Socket.io[18] as a multi-transport messaging library. In particular, Socket.io guarantees fast delivery of messages over the network across a broad range of devices and environments. In fact, it implements a

[15] As reported on the website of the project (http://nodejs.org): "Node.js building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices."
[16] See https://github.com/winstonjs/winston.
[17] See http://expressjs.com/.
[18] See http://socket.io/.

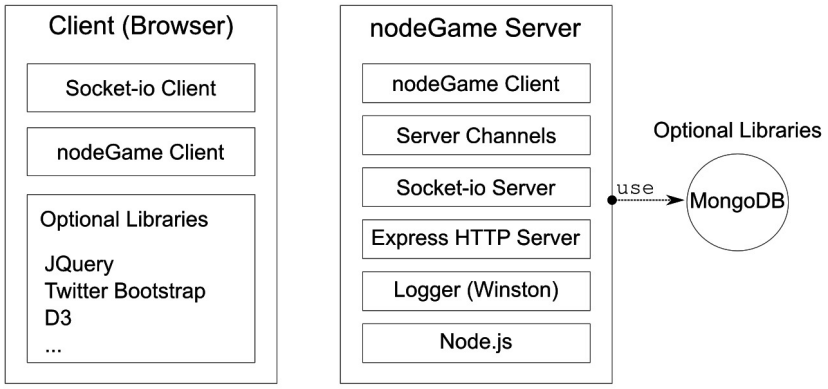

Figure 6. Schematic representation of the nodeGame architecture. The two main components, server and client, are modularized in nested components. Notice how nodeGame client is shared among client and server. For more details about the internal components of nodeGame client refer to Fig. 5.

number of different message-transport protocols, and automatically negotiates with incoming clients the one achieving the best results. Socket.io supports HTML5 WebSockets, which represent a major improvement in the history of web data communication and constitute the royal road for the implementation of large-scale real-time interactive games in the browser.[19]

A central component of the nodeGame server architecture is the server *channel*. A server can have as many channels as needed, and generally as many as the number of games currently installed. As illustrated in Fig. 7, a channel is composed by several nested objects: the waiting room, the clients registry, the collection of game rooms, and two independent game servers, one for the players and one for the administrators. A channel is internally split into two separated game servers in order to guarantee different privileges of access to its internal methods. Each of the two internal game servers is equipped with a different set of event handlers. Event handlers are supposed to reply to incoming messages, either by returning another message, or by triggering an internal operation. The Admin game server has an extended set of event handlers which includes methods, for example, to redirect and remotely setup any connected client. Such privileged operations are not allowed by the Player game server, whose task is limited to pass along data, text, log and synchronization messages. Each of the two internal game servers contains a dedicated socket manager, that is an abstraction of the actual type of connection used by the server to communicate with the clients based on their location of execution. For example, Socket.io is the type of socket used to communicate with clients located on the browser of remote machines; Socket Direct, instead, is used to exchange messages with logics and bots running as separated processes within nodeGame server. The channel registry stores information about all connected and disconnected clients, such as their game state and the game room they belong to. The registry is accessed by the other channel components whenever needed. For example, the channel waiting room consults the registry before starting a new game room, and updates it consequently. Each game room represents a wrapper around a group of clients which can exchange messages in a dedicated space. One of those, the *logic*, has the purpose of controlling the advancement of the game. The logic is implemented by the experimenter, as explained in Sec. 4.

nodeGame is an open source project under active development. A public organization containing its source code is hosted at Github[20] at the address `https://github.com/nodeGame/`. Therein, it is possible to access the source code of all the individual components of the nodeGame architecture: client,

[19] For more information see `http://en.wikipedia.org/wiki/WebSocket` or `https://www.websocket.org/`.
[20] Github is a collaborative platform for code review and code management of software projects.

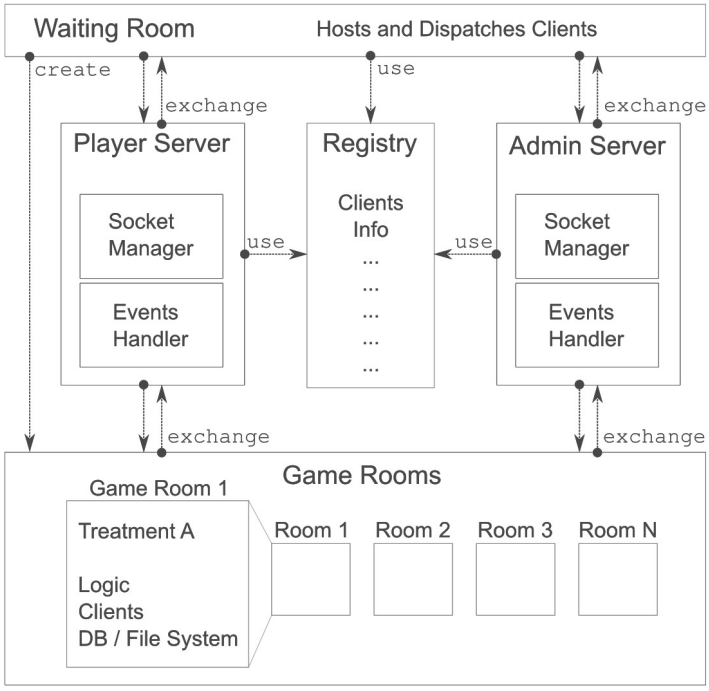

Figure 7. Schematic representation of the server channel architecture. The server channel is composed by the following interactive components: the waiting room, the clients registry, the collection of game rooms, and two independent game servers, one for the players and one for the administrators.

server, and all the supporting libraries. People interested in joining the development of nodeGame are very welcome, and are encouraged to follow the guidelines for developers on the project wiki (http://nodegame.org/wiki/), and to sign up to the nodeGame mailing list for announcements, help or features requests.

7 Conclusions

This paper introduced a new software framework called nodeGame (http://nodegame.org) for conducting behavioral research in the laboratory and online. This framework permits to conduct both discrete- and real-time experiments, and suits the execution of large-scale experiments up to hundreds of simultaneous participants, depending on the actual experimental settings.

This concluding section briefly recapitulates the main advantages and limitations brought about by nodeGame, and ends with some final considerations about the importance of open source software in scientific research, and the need of giving experimentation an even more prominent methodological role among the social science disciplines.

7.1 Advantages

Already illustrated in Sec. 4 and 5, the main advantages of nodeGame are also summarized in the list below.

- nodeGame can be executed in different settings, such as laboratory, online, field, and even lecture-hall, and makes easy switching among those.
- nodeGame grants great flexibility to the experimenters in the definition of experimental setups.
- nodeGame supports both turn-based and real-time experiments.
- nodeGame's scalable architecture can support large-scale experiments with hundreds of players even with a standard laptop machine.
- nodeGame lowers the barrier for online participation because clients can join an experiment without the need of installing any additional software.
- nodeGame can reduce the costs of maintaining a laboratory pool, since only one computer needs to be updated with future releases of the software.
- In nodeGame, it is possible to realize rich and interactive user interfaces with the use of standard web technologies (HTML, CSS, JavaScript).
- Being nodeGame open source, it is relatively easy to find good web developers able to implement experiments at a reasonably low cost.

7.2 Current Limitations

nodeGame offers new solutions to known challenges of online research. However, being it a relatively young project, it does not cover all the use cases implemented in traditional experimental software yet. Here is a list of nodeGame's currently known limitations.

- Experimenters need some programming skills to implement new experiments in nodeGame. This is a design feature, because programming an own experiment grants more flexibility, and can increase the general performance of the application, a vital goal for achieving large-scale experiments.

- There is only limited support for matching algorithms, and mapping roles to participants. This feature is currently under development, and should be available in the future. However, with average computer programming skills is possible to write custom code achieving the same goals in a relatively straightforward way.
- There is only limited support for integration with external databases. Currently, only MongoDB is supported.
- nodeGame does not do the recruitment of participants. This is unlikely to change in the future given the broad ecosystem of online labor markets available.
- nodeGame does not provide a direct integration with any online labor market. Experimenters must have their own employer accounts, and upload their tasks outside of nodeGame.
- The online documentation for nodeGame covers most of the use cases, but not yet all of them. 100% coverage will be reached in the near future, hopefully with the help of a community of users and contributors. As pointed out by Horton, Rand, and Zeckhauser (2011), we do not only need better tools, but also better documentations.

7.3 Some Final Considerations

nodeGame is entirely open source and free software. As science increasingly relies on computational technologies to achieve its goals, it is tremendously important that scientific research is performed making use of open source and free software as much as possible (Sonnenburg et al., 2007; Schwarz, 2010). Choosing open and free software ensures full replicability to scientific results obtained with computer-mediated methods. Moreover, it guarantees that errors in the source code can be easily spotted and quickly fixed by the same community of users (Von Krogh, Spaeth, and Lakhani, 2003).

The philosophy of nodeGame is to bring together the advantages of both laboratory and online research. In fact, online experimentation can significantly lower the cost of testing competing behavioral hypotheses, producing results in a shorter amount of time, which then laboratory research can replicate under extremely controlled conditions. Interestingly, when experimentation costs are lowered, it is not only possible to *test* more theories, but also to *generate* more. This can lead to a cyclical process of elimination of wrong theories and generation of new hypothesis. This process should be able to produce theories of greater explanatory power over time, which in turn should allow to make faster progress in the behavioral sciences. nodeGame aims at being a valuable instrument freely available to social scientists in such a process.

8 Acknowledgements

The author is grateful to all the persons who have contributed to the development of nodeGame over the years: Philipp Küng, Benedek Vartok, Sebastien Arnold, Lionel Miserez, Jan Wilken Dörrie and Nicole Barbara Lipsky. The author is also indebted for insightful and strategic discussions with Stefan Wehrli, Michael Mäs, Dirk Helbing, and Ryan O. Murphy. The author's work on nodeGame was gratified over the years by the financial support from the Professorship of Computational Social Science (COSS) at ETH Zürich, the ETH Decision Science Laboraty (DeSciL), and the seed grant PEER by the Institute for Science Technology and Policy (ISTP).

References

Anderson, B. et al. (2015). *Burden Sharing in Climate Negotiations: Experimental Evidence*. In preparation.

Balietti, S., R. Goldstone, and D. Helbing (2015). *Competition and Peer Review in Creative Production Enviroments*. submitted.

Balietti, S., B. Jäggi, and K. Axhausen (2015). *Human Coordination in Binary Transportation Choice Task with Flexible Time Departure*. In preparation.

Bargh, J.A. and K.Y.A. McKenna (2004). "The Internet and Social Life". In: *Annual Review of Psychology* 55, pp. 573–590.

Bezanson, J. et al. (2014). "Julia: A Fresh Approach to Numerical Computing". In: *Arxiv.org*. arXiv:1411.1607 [cs.MS]. URL: http://arxiv.org/abs/1411.1607.

Birnbaum, M.H. (2000). "SurveyWiz and FactorWiz: JavaScript Web Pages That Make HTML Forms for Research on the Internet". In: *Behavior Research Methods, Instruments, & Computers* 32.2, pp. 339–346.

— (2004). "Human Research and Data Collection via the Internet". In: *Annual Review of Psychology* 55, pp. 803–832.

Bos, N. et al. (2002). "Effects of Four Computer-Mediated Communications Channels on Trust Development". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 135–140.

Buchanan, T., J.A. Johnson, and L.R. Goldberg (2005). "Implementing a Five-Factor Personality Inventory for Use on the Internet". In: *European Journal of Psychological Assessment* 21.2, pp. 115–127.

Butler, B.S. (2001). "Membership Size, Communication Activity, and Sustainability: A Resource-Based Model of Online Social Structures". In: *Information Systems Research* 12.4, pp. 346–362.

Centola, D. (2010). "The Spread of Behavior in an Online Social Network Experiment". In: *Science* 329.5996, pp. 1194–1197.

Chen, C., M. Schonger, and C. Wickens (2014). *oTree: An Open-Source Platform for Laboratory, Online, and Field Experiments*. http://www.otree.org/.

Cox, J.C. and J.T. Swarthout (2005). "EconPort: Creating and Maintaining a Knowledge Commons". In: *Andrew Young School of Policy Studies Research Paper* 06–38.

Fischbacher, U. (2007). "z-Tree: Zurich Toolbox for Ready-Made Economic Experiments". In: *Experimental Economics* 10.2, pp. 171–178.

Friedman, D. and A. Cassar (2004). *Economics Lab: An Intensive Course in Experimental Economics*. London and New York: Routledge.

Guala, F. (2005). *The Methodology of Experimental Economics*. Cambridge Books 9780521618618. Cambridge University Press.

Horswill, M.S. and M.E. Coster (2001). "User-Controlled Photographic Animations, Photograph-Based Questions, and Questionnaires: Three Internet-Based Instruments for Measuring Drivers' Risk-Taking Behavior". In: *Behavior Research Methods, Instruments, & Computers* 33.1, pp. 46–58.

Horton, J.J., D.G. Rand, and R.J. Zeckhauser (2011). "The Online Laboratory: Conducting Experiments in a Real Labor Market". In: *Experimental Economics* 14.3, pp. 399–425.

Kraut, R. et al. (2004). "Psychological Research Online: Report of Board of Scientific Affairs' Advisory Group on the Conduct of Research on the Internet." In: *American Psychologist* 59.2, pp. 105–107.

Levitt, S.D. and J.A. List (2009). "Field Experiments in Economics: The past, the Present, and the Future". In: *European Economic Review* 53.1, pp. 1–18.

Luce, K.H. et al. (2007). "Reliability of Self-Report: Paper Versus Online Administration". In: *Computers in Human Behavior* 23.3, pp. 1384–1389.

Mangan, M.A. and U.-D. Reips (2007). "Sleep, Sex, and the Web: Surveying the Difficult-to-Reach Clinical Population Suffering From Sexsomnia". In: *Behavior Research Methods* 39.2, pp. 233–236.

Mason W.and Suri, S. (2012). "Conducting Behavioral Research on Amazon's Mechanical Turk". In: *Behavior Research Methods* 44.1, pp. 1–23.

Musch, J. and U.-D. Reips (2000). "Psychological Experiments on the Internet". In: ed. by M.H. Birnbaum. Academic Press. Chap. A Brief History of Web Experimenting.

Paolacci, G., J. Chandler, and P.G. Ipeirotis (2010). "Running Experiments on Amazon Mechanical Turk". In: *Judgment and Decision making* 5.5, pp. 411–419.

Pettit, J. et al. (2014). "Software for Continuous Game Experiments". In: *Experimental Economics* 17.4, pp. 631–648.

Rand, D.G., J.D. Greene, and M.A. Nowak (2012). "Spontaneous Giving and Calculated Greed". In: *Nature* 489.7416, pp. 427–430.

Reips, U.-D. (2002). "Standards for Internet-Based Experimenting". In: *Experimental Psychology* 49.4, p. 243.

Reips, U.-D. and J.H. Krantz (2010). "Advanced Methods for Conducting Online Behavioral Research". In: ed. by S.D. Gosling and Johnson J.A. American Psychological Association. Chap. Conducting True Experiments on the Web.

Reips, U.-D. and C. Neuhaus (2002). "WEXTOR: A Web-Based Tool for Generating and Visualizing Experimental Designs and Procedures". In: *Behavior Research Methods, Instruments, & Computers* 34.2, pp. 234–240.

Rubinstein, A. (1999). "Experience from a Course in Game Theory: Pre- and Post- class Problem Sets as a Didactic Device". In: *Games and Economic Behavior* 28. Revised edition at http://arielrubinstein.tau.ac.il/99/gt100.html, pp. 155–170.

Rubinstein, A (2013). "Response Time and Decision Making: An Experimental Study". In: *Judgment and Decision Making* 8.5, pp. 540–551.

Salganik, M.J., P.S. Dodds, and D.J. Watts (2006). "Experimental Study of Inequality and Unpredictability in an Artificial Cultural Market". In: *Science* 311.5762, pp. 854–856.

Schwarz M.and Takhteyev, Y. (2010). "Half a Century of Public Software Institutions: Open Source as a Solution to Hold-Up Problem". In: *Journal of Public Economic Theory* 12.4, pp. 609–639.

Sonnenburg, S. et al. (2007). "The Need for Open Source Software in Machine Learning". In: *Journal of Machine Learning Research* 8, pp. 2443–2466.

Von Krogh, G., S. Spaeth, and K.R. Lakhani (2003). "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study". In: *Research Policy* 32.7, pp. 1217–1241.