# Continuous Game Experiments[1]

by

James Pettit[*], Daniel Friedman[*], Curtis Kephart[*], and Ryan Oprea[†]

August 27, 2012

**Abstract.** ConG is software for conducting economic experiments in continuous and discrete time. It allows experimenters with limited programming experience to create a variety of strategic environments featuring rich visual feedback in continuous time and over continuous action spaces, as well as in discrete time or over discrete action spaces. Simple, easily edited input files give the experimenter considerable flexibility in specifying the strategic environment and visual feedback. Source code is modular and allows researchers with programming skills to create novel strategic environments and displays.

Keywords: Experimental economics. Continuous time. Software for laboratory experiments.

JEL codes: C70, C88, C90

---

[1] This project was made possible by National Science Foundation Grant SES-0925039. We are grateful to the undergraduate programmers who contributed to this project: Joseph Allington, Pranava Adduri, Ashley Chard, Michael Cusack, Keith Henwood, Anthony Lim, Alex (Richard) Lou, Vadim Maximov, Jacob Meryett, Laker Sparks, and Sam Wolpert. Graduate researchers, including Jacopo Magnani, Luba Petersen, Jean Paul Rabanal, and undergraduate researchers Tamara Bakarian, Thomas Campbell, and Richard Shall also made significant contributions.

* Economics Department, University of California, Santa Cruz, CA, 95064. *jpettit@ucsc.edu, dan@ucsc.edu, and curtisk@ucsc.edu*

† Department of Economics, University of British Columbia, Vancouver, BC, V6T-1Z1. *roprea@gmail.com*

## 1. Introduction

How do people behave when they interact strategically? Social scientists have, for more than half a century, turned to game theory to answer such questions, but predictions offered by game theory are often ambiguous (e.g., multiple equilibria), vague (e.g., contingent on details of timing that are hard to interpret), or implausible. Not surprisingly, many game theorists have turned to laboratory experiments to test predictions and refine the theory (e.g., Thrall et al., 1954; Orwant and Rapoport, 1962; Kagel and Roth 1997; Goeree and Holt, 2001).

So far, however, laboratory environments have been rather restrictive. Most are either one-shot encounters between two players, or very simple sequential play, or repetition of some stage game in discrete time. Most offer each player a very limited range of feedback on what other players are doing. Outside the lab, strategic interaction occurs in a much wider range of environments.

In this paper we introduce software that expands the range of environments for conducting experiments inspired by game theory. ConG (short for Continuous Games) is a suite of programs for running experiments with human subjects who interact strategically in real time. It uses graphics intensively to create a variety of visual environments, and allows subjects to continuously change and adapt their strategies. ConG permits several different ways to specify payoff functions, and to display a wide variety of information in a compact manner. ConG can also run games in discrete time in a manner that closely parallels and can be easily compared to continuous time.

Two published papers illustrate some of ConG's flexibility and research potential. Friedman and Oprea (2012) find remarkably high rates of mutual cooperation in prisoner's dilemma experiments conducted in continuous time, and a steady transition from near zero cooperation in one shot settings to almost full cooperation in discrete time as the time intervals get shorter and more numerous. By contrast, Oprea, Friedman and Henwood (2011) find very little cooperation in populations of 10-12 human subjects playing a Hawk-Dove game in continuous time, and instead find support for some distinctive predictions of evolutionary game theory.

There already exists excellent general purpose software for one-shot and discrete time game experiments, and also for continuous double auction markets (e.g., zTree, MarketLink). But thus far, studies of near continuous time have relied on ad hoc software designed to address a specific research question. ConG strives to meet the standards of flexibility, stability, and ease of use set by packages like zTree, while advancing the state of the art in three important aspects.

- ConG is truly asynchronous and event driven – it does not simply run many discrete periods quickly (though the software allows the option of implementing discrete time decision making). The experimenter can run an entire continuous period, re-initialize, and run additional periods under stationary repetition.
- ConG's configuration files enable experimenters to quickly implement a multitude of game environments. These files allow experimenters to specify the payoff function, display options, lags, etc., , period by period, as explained below.
- ConG emphasizes real-time 2D and even 3D graphical displays to provide feedback that subjects can respond to quickly and effortlessly.

## 2. Basic Functionality

Experimenters specify the period-by-period environment of a ConG session via a Comma Separated Variables (.csv) configuration file, easily editable with any spreadsheet software. This file defines the number of periods, period lengths, whether or not periods are to be run in discrete or continuous time, the payoff function and its parameters, and other parameters specifying what subjects will see and how they interact in the game. Full details and documentation can be found at http://leeps.ucsc.edu/cong/.

For example, consider Figure 1. It shows the players' screen in ConG for a symmetric 2x2 matrix game played in discrete time. The configuration file seen in Table 1 shows that the experimenters have chosen prisoner's dilemma payoffs this period, and that the 60 second period is divided into 6 subperiods. For subsequent periods, additional lines of the configuration files could specify different period lengths, unpaid (practice) periods, different numbers of subperiods, and different payoff functions.

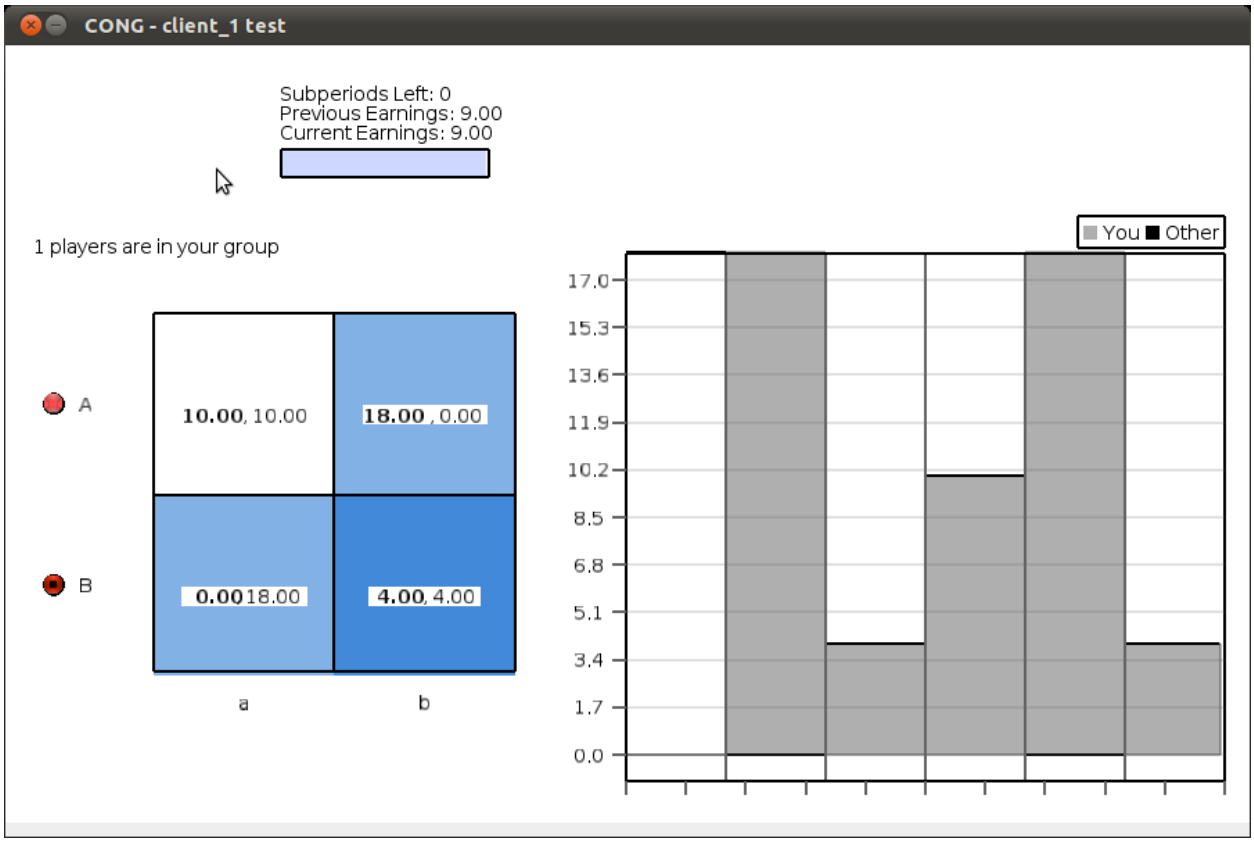


**Figure 1**: User interface for a simultaneous-move discrete-time game with ConG software. Once the period starts, subjects may freely choose their current action (here "B") by clicking on one of the two radio buttons on the far left of the window, or by using an arrow key, and the corresponding matrix row is shaded. The matrix column corresponding to the counterpart player's action choice last period (here "b") is also shaded, so the cell with corresponding payoffs is doubly shaded. The time series chart to the right shows payoffs in the subperiods completed so far in the current period. These payoffs are determined only by the action profile at the end of each subperiod. Counterpart action choices in the current subperiod are revealed only at the end of the subperiod. To indicate how much time remains in each subperiod, a progress bar (the rectangle just below previous and current earnings table) fills from white/empty to blue/full. The progress bar is fully blue here, indicating the subperiod is over.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | period | paid | length | subperiods | percentChangePerSecond | groupSize | payoffFunction | name | min | max | Aa | Ab | Ba | Bb | selector |
| 2 | 1 | TRUE | 60 | 6 | NaN | 1 | matrix2x2 | first | 0 | 17 | 10 | 0 | 18 | 4 | pure |

**Table 1**: Configuration file corresponding to Figure 1. Line 2 specifies the first period (column A) of a potentially multiple period session. Column B specifies that the period is paid, as opposed to an unpaid practice period. Columns C and D specify that the period lasts for 60 seconds and is broken down into 6 subperiods (each thus lasting 10 seconds). Column E is not applicable here, F specifies pairwise matching of players, and G invokes a payoff function in the form of a symmetric 2x2 matrix whose entries appear in columns K-N. Columns I-J define the vertical scale of the payoff chart, H allows for the experimenter to give period treatments different names, (allowing for ease of data analysis), and the "pure" in column O specifies the radio buttons subject use to choose their actions.

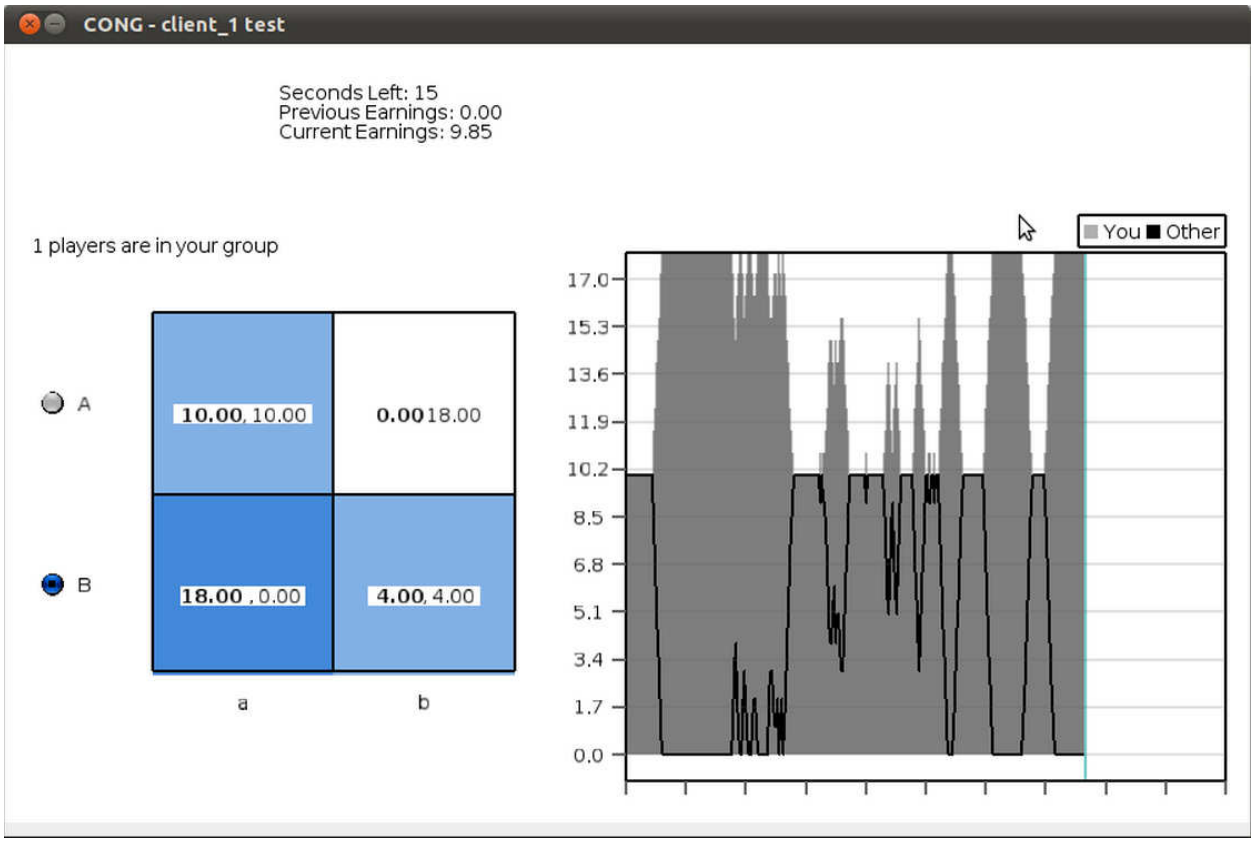


**Figure 2** – User interface for a continuous-time game with ConG software. Action selection and shading are as in Figure 1 (here the player is earning a payoff of 18 while the counterpart earns 0), but the payoffs here are continuous flows that immediately reflect changes in the action profile. The right-side time series chart shows the flow payoffs so far this period for both players. At the end of the period they are paid the integral of these flows, shown as the gray area for the player and as the area under the black line for the player's counterpart.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | period | paid | length | subperiods | percentChangePerSecond | groupSize | payoffFunction | min | max | Aa | Ab | Ba | Bb | selector | name |
| 2 | 1 | TRUE | 60 | 0 | NaN | 1 | matrix2x2 | 0 | 17 | 10 | 0 | 18 | 4 | pure | second |

**Table 2**: The configuration file corresponding to Figure 2. Column D is set to 0 so this period is continuous time, and column O specifies a different name for output files.

Figure 2 shows the user interface for the same game played in continuous time. Again players may change their action at any time and as often as desired, but now any change in the action profile is immediately reflected in users' display and in the flow payoffs for both players. The computer response time between hardware interaction and display update in labs that satisfy minimum hardware requirements is less than 50 milliseconds - faster than human reaction time - giving the experience of continuous action in real time.

Figure 3 shows a player's screen for a continuous time population game with a heatmap display. Clicking on the heatmap allows players to choose from a continuous action set, here consisting of all mixes of two pure actions in a symmetric 2x2 payoff matrix. The numbers at the corners of the heatmap show the matrix entries, which here constitute a hawk-dove game. The heatmap

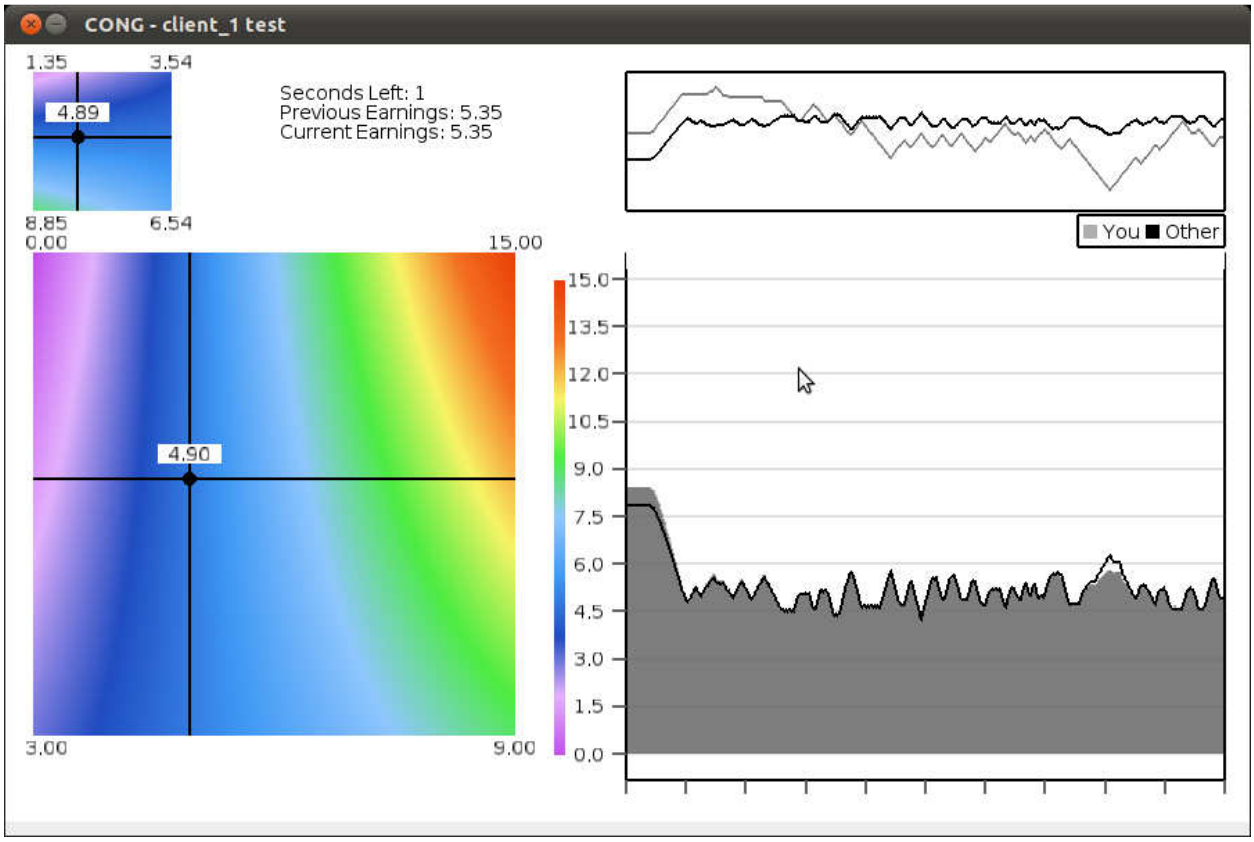


**Figure 3**: Heatmap selector in a continuous time population game with a continuous action space. Players click (or use arrow keys) along the vertical axis of the large heatmap to select the desired action, which is displayed as a horizontal black line at the chosen level; the counterparts' current action is indicated by the vertical black line. The intersection of these lines determines the current flow payoff for the player (currently 4.90). The thermometer scale to the right of the large heatmap translates the colors to numerical flow payoffs. Realized flow payoffs are graphed in the large time chart on the right. The small time chart above it shows player action choices over time. The small heatmap to the top left shows the counterparts' heatmap, including their current flow payoffs.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | period | paid | length | subperiods | numGroups | matchType | percentChangePerSecond | payoffFunction | min | max | Aa | Ab | Ba | Bb | selector | mixed | matrixDisplay | showHeatmapLegend | name |
| 2 | 1 | TRUE | 60 | 0 | 1 | pair | 1 | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | heatmap2d | TRUE | HeatmapBoth | TRUE | hd |

**Table 3**: The configuration file corresponding to Figure 3. Columns E and F specify a single population game in which each player is matched with the average choice of all other players. Column G sets the "speed limit" for strategy adjustment so that it would take 1 second to traverse the strategy spectrum, e.g, from the top of the heatmap to the bottom. Columns K-N define this game as Hawk-Dove, column P enables continuous strategy adjustment (if the entry were FALSE then subjects could select only corner actions). Columns O, Q and R respectively call for the display of the player's own heatmap, the counterpart heatmap, and the thermometer scale.

displays all potential flow payoffs, with warmer colors indicating higher values. Although the same information is conveyed in two other ways (hovering the cursor over an action profile pops up a text bubble reporting the numerical payoff at that profile, and a numerate player can linearly interpolate payoffs from the numbers shown at the heatmap's corners), the heatmap offers a very accessible overview of current and all counterfactual payoffs. Even colorblind subjects have reported that the heatmap (in conjunction with the text bubbles) is quite helpful. The configuration file permits the experimenter to suppress the counterpart's heatmap and/or the thermometer scale and/or the heatmap itself.

Figure 4 shows another interface option: the bubbles selector. Subjects select their actions by clicking to move the black rectangle slider along the x-axis. The vertical height of the player's "bubble" corresponds to his or her flow payoff. This selector is a very simple way to allow subjects to pick from a continuous action space. The Figure shows a public goods game (Charness et al. 2012), but the bubbles selector has also been used in Bertrand games, Cournot games and Hotelling spatial competition, among other games played in continuous or discrete time.

A number of additional features may be implemented via edits to ConG's .csv configuration file. Players may be grouped and matched period-by-period in intricate ways, creating one-population games, two-population games and any other arbitrary counterpart matching or block randomization. Players may be permitted to chat with one another via a freeform chat window. When a player clicks to a new action target in a continuous time game, a configuration file setting determines whether that action is realized instantaneously or the adjustment occurs at a specified finite rate. All of these features are discussed below and on ConG's documentation website.

## 3. Extending ConG

Experimenters with limited computer skills can implement a wide variety of laboratory games simply by judicious editing of the configuration file. Experimenters who have Java programming skills (or who can hire a programmer) can create new payoff functions and novel visual display environments that significantly extend the scope of the software. The source code is modular and

open-ended, so researchers can design and run new visually-intensive and/or continuous games without having to bear heavy development costs.

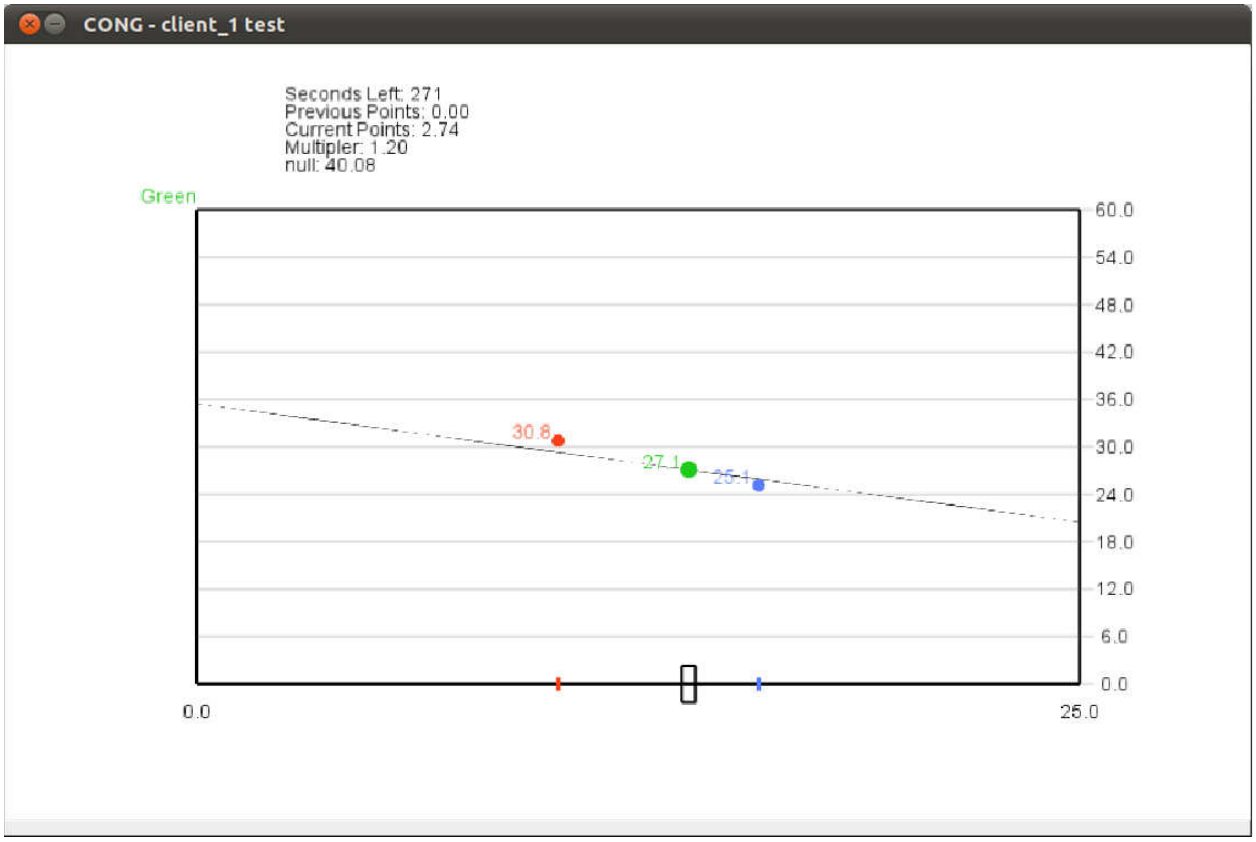


**Figure 4:** Bubbles selector in a continuous times experiment. Subjects select their action via the black rectangle slider along the bottom horizontal axis. The vertical height of player's "bubbles" indicate current flow payoffs, with the "Current Earnings" field accumulating flow payoffs as the period advances. Other bubbles in the display indicate current actions (i.e. current horizontal positions) and payoffs (vertical heights) of counterpart players. The thin black "payoff landscape" line shows the payoff the player would earn at all possible actions choices, given counterparts' current profile.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | period | paid | length | subperiods | groupSize | matchType | payoffFunction | type | name | A | smin | smax | min | max | mixed | selector | initial | potential |
| 2 | 6 | TRUE | 300 | 0 | 3 | self | sum | public_goods | pf | 1.2 | 0 | 25 | 0 | 60 | TRUE | bubbles | 0 | TRUE |

**Table 4:** The configuration file corresponding to Figure 4. It specifies a 3 player public goods game with multiplier 1.2 during a 300 second paid period in continuous time, using the bubble selector. A FALSE entry in column R would suppress the payoff landscape line.

Some background on the software libraries and architecture will be helpful for experimenters thinking about extending ConG. Casual readers, or those who intend to use only

new configuration files for existing capacities, can skip the rest of this section with no loss of continuity.

Over the last two decades the LEEPS lab at UC Santa Cruz has developed software libraries that ease the creation of new programs for conducting laboratory experiments. In particular, FIRE (Framework for Interactive Real-time Experiments) collects utilities for general real-time networked client-server applications, specifically targeted towards Experimental Economics applications. FIRE provides utilities that enable networked experiments with "soft" real-time latency, typically less than 50 milliseconds round-trip. It also offers access to high quality graphics using the OpenGL and Processing libraries. OpenGL enables 2D and 3D graphics rendering using a graphics co-processor and thus smooth animation. FIRE is quite flexible and accessible to programmers. For example, in Winter 2008, two undergraduate students with heavy course loads elsewhere took less than four weeks to design, code, and run a continuous time game for a game theory class project.

**3.1 ConG Architecture.** Funded by NSF grant SES-0925039 ("Continuous Games in the Laboratory," 2009-2012), LEEPS lab has used FIRE to construct ConG, a suite of programs whose imputs consist of four main elements:

- **Subjects and Actions:** The experimenter first defines *n*, the number of subjects in the experiment and the actions available. Each subject, labeled $i$, has a current action vector $\bar{s}_i \in \mathbb{R}^k$, where the number of action dimensions $k$ is often 1, but can be more.
- **Time:** The experimenter defies the length of time T each period lasts, and whether or not the period is to be run in continuous time or discrete. Discrete time periods also require a number m of subperiods, which implies the subperiod grid size T/m.
- **Grouping and Matching:** Subjects are partitioned into groups $g_1, \ldots, g_r,\ r > 1$. Each subject in group $g_i$ are matched with all subjects in group $m(g_i)$, and the experimenter specifies whether or not matching with own-actions is permitted. For example, in the simplest prisoner's dilemma game -- a two-player single-population game with no self-matching -- a subject's match group is their own group, $g_1 = \{\bar{s}_1, \bar{s}_2\} = m(g_1)$. This grouping structure also enables multi-population games, e.g., $g_1 = \{\bar{s}_1, \bar{s}_2, \bar{s}_3\}$ matched

with $m(g_1) = g_2 = \{\bar{s}_4, \bar{s}_5\}$, and $m(g_2) = g_1$ . ConG's configuration file may be extended to accommodate complex participant grouping, matching and silos.

- **A Payoff Function** is a mapping of players' current and previous action choices to a single flow payoff number. Flow payoffs are aggregated over the entire period to determine per-period payoffs, and over all paid periods to determine subject's total points. Individual payoffs are a function of one's own group's action choices and the choices of their matched group. Payoff functions have four inputs: (1) subject, $i$, the player receiving the payoff, (2) the set of current action vectors in the subject's group, $g$, of which one component may be subject $i$'s own current action vector, (3) the set of current action vectors in the matched group, $m(g)$, which could be $g$, and (4) time settings that aggregate flow payoffs. The payoff function, $\Pi_i\big(s_i, t, g_i, m(g_i)\big) \rightarrow (R)$, returns a single, floating-point number, which is added to the player's accumulated earnings.

Continuous and discrete time flow payoffs are handled similarly, but distinctly. In continuous time, flow payoffs are calculated and applied for the duration that subjects remain in a fixed action profile – the amount of time $s_j$ remains constant for every $j \in r$. In discrete time, actions are fixed in each subperiod, therefore flow payoffs are applied for the length of the subperiod grid.

Figure 5 abstracts the general process by which the client and server control programs continuously update each other and respond to user input. At initiation of ConG, the software queries the experimenter the number of subjects in the session, and allows the experimenter to load a .csv configuration file. This file establishes the the payoff function, action selection interface, how subjects are grouped and matched, how each period is timed, and other details that fully characterize the experiment to be run.

Once the server connects to each client, subjects are queried their name -- information linked to the subject’s final payoff account -- and client windows create the display environment chosen for the first period.

When the experimenter clicks to start period one, the server instructs each client window to unlock the selector, freeing subjects to edit their current action profile. The display begins charting current and past actions, payoffs and anything else selected in the configuration

file. The period clock counts down: in discrete time periods time is designated by a "Subperiods Left" counter, in continuous time periods the clock counts the number of seconds left.

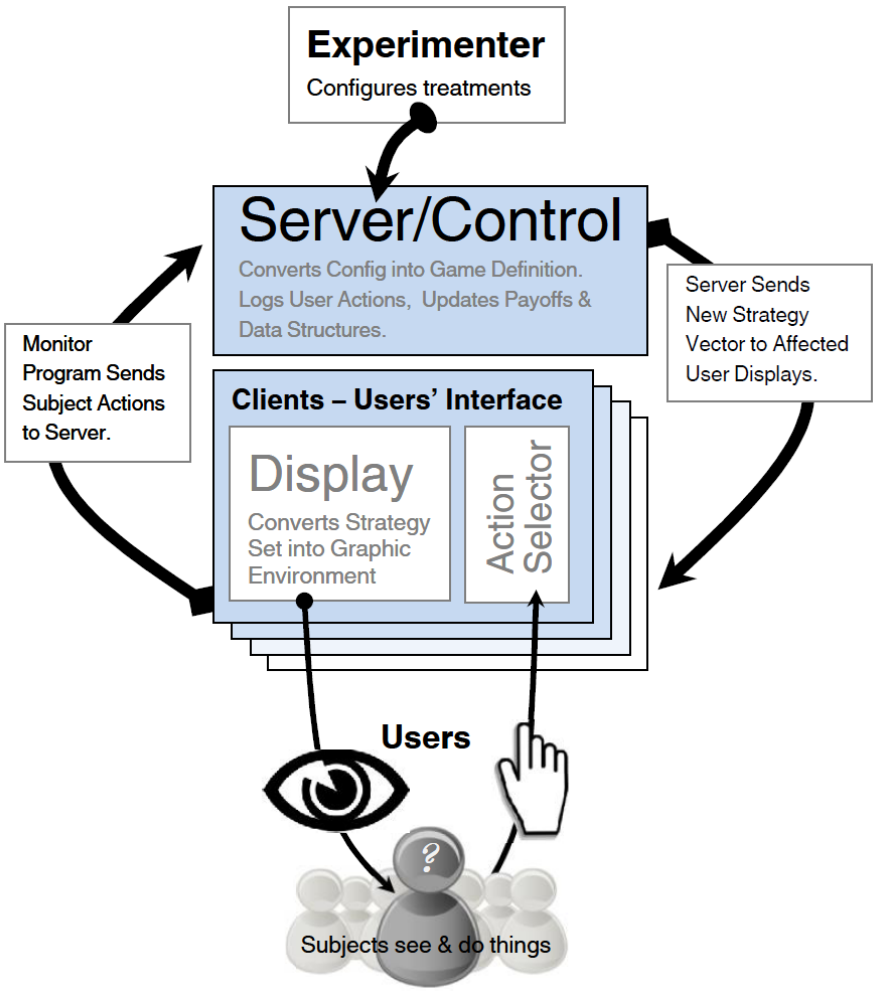


**Figure 5** – A visual abstraction of a ConG Experiment.

The user interface allows players to keep or change their current action choices, and provides useful information about previous, current, and potential action choices and payoffs. Each combination of configuration file "selector", "payoffFunction" and other fields will produce a different display environment and interface for experiment subjects.

After the period has begun, whenever a subject clicks on the selector to make a change, a monitor process (a thread running alongside the client, watching for any subject activity) notifies the server that the action vector has been modified. The server picks up this notification, queues it, logs it, calculates the duration of the previous action profile, multiplies this by each

relevant player's flow payoffs for that profile, and adds it to each player's accumulated payoff accounts. (The relevant players are determined by the grouping and matching procedure, e.g., in a 4 player public goods game with 8 subjects, only half of them will be affected by any individual player's change of action.) The server then sends subjects in the affected group the new action vector, which the client programs incorporate into their graphic displays.

This loop is continuously repeated throughout the period's duration: the client threads send subject action changes to the server and, after time enough for a number of calculations and changing of data structures, client programs update users' screens. Given minimal hardware specifications, this loop is timed at about 50 milliseconds (0.05 seconds). That is, it takes about 50 milliseconds between when a user clicks a new action to when all users see the effect of this change on-screen. Since even the fastest human reaction time is significantly longer than 50 milliseconds, subjects are unable to react to and unlikely to even notice this feedback latency. This means that users interacting via ConG perceive that the computer accepts and incorporates all clicks instantaneously.

**3.2 Graphic Display.** ConG leverages the Processing library (processing.org) to build graphical widgets for displaying data and selecting from possible actions in the game. Each widget is given a portion of the screen that it may draw to. All widgets have access to the current game state, including the following: (1) the points earned by the subject in the current period, updated at least once per second, (2) the total points earned by the subject in all periods, (3) a time-indexed list of the actions chosen by all subjects in the group, and (4) the current payoff function, a Java function that takes the current actions played and returns a floating point number as flow payoff.

Widgets also have access to mouse and keyboard input, and can use these to send action updates to the server. Currently, all widgets draw in 2D, but the library translates this to a plane in 3D, that is rendered using the 3D co-processor when present. This offloads most of the rendering to hardware, and gives frame-rates of around 30 frames per second even on low-end hardware (for example, the LEEPS lab currently uses $300 machines with a build-in 3D processor). On mid-range hardware, 60+ fps is common and easily achievable. In addition, extension to displaying 3D models of data will be straightforward, requiring only a new widget that draws in 3D.

Here is an example of game stat action profile data ConG uses to draw users' graphic display.

```
{{time:t1, action{1: [0.5, 0.5]   , 2: [0.2, 0.8]}},
 {time:t2, action{1: [0.75, 0.25] , 2: [0.2, 0.8]}},
 {time:tN, action{1: [0.8, 0.2]   , 2: [0.2, 0.8]}}}
```

At time $t_1$, subject 1 was playing [0.5, 0.5]. At time $t_2$, he or she switched to [0.75, 0.25]. Subject 2's action choice as remained unchanged. At time $t_N$, subject 1 switched had a strategy of [0.8, 0.2], while subject 2's action was still [0.8, 0.2].

**3.3 An Example Extension.** ConG's extensibility feature has been used to implement a version of Hotelling's model of spatial competition. In this game, players compete over a bounded, finite, single-dimension spectrum of differentiation. Flow payoffs are a function of player market share, i.e., the portion of the action space the player's chosen "location" is nearest. Since existing configuration file settings do not allow for the Hotelling model, a new file is needed to define what players see, how they interact, and how payoffs are defined.

ConG uses an interface named `PayoffScriptInterface` to ensure consistency in the extension of the software. The .java file that implements the new experiment must contain a number of methods in a consistent way in order to function correctly. The Hotelling extension relies on a display very similar to the bubbles display seen in Figure 4, players choose their locations via a horizontal slider, which defines their location. Players see their own and counterpart player's payoffs in the form of vertical heights of colored bubbles.

The code below is an abbreviated excerpt of the payoff function implementing the Hotelling game. User input is collected into a vector of current actions currently selected by players. Since edge players payoffs are calculated differently than center players, actions are first sorted from lowest to highest. Then for each player's action, "`s`" below, flow payoffs, "`u`", are a function of the following:

```
if (left == 0) {
    u = s + 0.5f * (right - s);
} else if (right == 1f) {
    u = 0.5f * (s - left) + (1 - s);
```

```
        } else {
            u = 0.5f * (s - left) + 0.5f * (right - s);
        }
        // Payoff function for Hotelling
        return config.get("Alpha") * 100 * (u / shared);
    }
```

Current flow playoffs are charted on screen, and added to subjects' point earnings. The full .java file that defines the Hotelling game discussed here is over 500 lines in length, of which the payoff function is only ten-percent. Most of the file describes the draw function, the fine details of the lines and rectangles subjects see and interact with their screen.

With some Java programming expertise, one can use ConG to implement any payoff function and take the current game state, (i.e. the full account of current and previous action profiles) to draw novels display environments. ConG's documentation website offers several examples of how a programmer might extend the existing code.

### 4. Configuration File Fields.

Experimenters about to create new configuration files, and programmers considering new extensions, may benefit from a closer look at the structure and syntax of configuration files. Again, more casual readers can skip this section with no loss of continuity, and more details can be found on the documentation website http://leeps.ucsc.edu/cong.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | period | paid | length | subperiods | groupSize | matchType | payoffFunction | min | max | Aa | Ab | Ba | Bb | mixed | selector | matrixDisplay | showHeatmapLegend |
| 2 | 1 | FALSE | 20 | 0 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | FALSE | pure | | |
| 3 | 2 | TRUE | 20 | 0 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | FALSE | pure | | |
| 4 | 3 | TRUE | 20 | 1 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | FALSE | pure | | |
| 5 | 4 | TRUE | 20 | 6 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | FALSE | pure | | |
| 6 | 5 | TRUE | 20 | 0 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | TRUE | heatmap2d | HeatmapSingle | TRUE |
| 7 | 6 | TRUE | 20 | 0 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | TRUE | heatmap2d | Corners | TRUE |
| 8 | 7 | TRUE | 20 | 0 | 1 | pair | matrix2x2 | 0 | 20 | 10 | 0 | 18 | 4 | TRUE | heatmap2d | HeatmapBoth | TRUE |
| 9 | 8 | TRUE | 60 | 0 | 6 | pair | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | TRUE | heatmap2d | HeatmapBoth | TRUE |
| 10 | 9 | TRUE | 60 | 0 | 6 | pair | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | TRUE | heatmap2d | HeatmapBoth | TRUE |
| 11 | 10 | TRUE | 60 | 0 | 12 | self | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | TRUE | heatmap2d | HeatmapBoth | TRUE |
| 12 | 11 | TRUE | 60 | 0 | 6 | pair | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | TRUE | heatmap2d | HeatmapSingle | TRUE |
| 13 | 12 | TRUE | 60 | 0 | 6 | pair | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | TRUE | heatmap2d | HeatmapBoth | TRUE |
| 14 | 13 | TRUE | 60 | 0 | 6 | pair | matrix2x2 | 0 | 15 | 0 | 15 | 3 | 9 | TRUE | heatmap2d | corners | TRUE |

**Table 5**. An example ConG .csv configuration file designed for twelve human subjects. Periods 1-7 run prisoner's dilemma with a number of treatment variations. Periods 8-14 run a hawk-dove game.

**Time:** Each session consists of a number of periods, each of which is specified in a line of the configuration file. As noted earlier, the specification includes the length of each period in seconds, whether or not the period is to be run in discrete time (as in periods 3 and 4 in Table 5) or continuous time, and in discrete time the number of subperiods. In discrete time players simultaneously make a single action choice in each subperiod: at the end of the subperiod, ConG collects and logs all subjects actions and updates subject displays with the new set of own and counterpart actions to be applied for the subsequent subperiod. This is, in effect, a finitely repeated simultaneous move game, one-shot in period 3 and repeated 6 times in period 4 of Table 5.

ConG software treats continuous time architecturally differently than discrete: current actions are changed asynchronously, whenever a user clicks to a new action choice. ConG has a reaction lag of sorts, the delay between when the user makes a change and when the computer reveals the effect of that change to all users graphically. With minimum hardware requirements met, that reaction lag is far below any human's ability to perceive it. Thus from the perspective of a human user, play is continuous in a ConG experiment. It approximates (but is not identical to) a very fine time grid with each second divided into more than twenty sub-periods.

The overarching goal in constructing ConG's software has been to allow participants to experience (in the display, counterpart action updates, action logging, etc.) minimal latency and the feel of continuous response. Thus, subject to the constraints of the hardware (and the laws of physics), ConG distributes as quickly as possible all action updates to grouped and matched population subject displays. It succeeds insofar as the response time is much faster than human reaction time.

**Payoff Function**: Simple edits to the configuration file support arbitrary symmetric 2x2 matrix games (Figures 1-3 and Table 5 above) and aggregative games. In aggregative games, the payoffs are a function of own actions and the sum of counterpart player actions. These include the public goods game featured in Figure 4 above, as well as threshold coordination games and most Cournot games.

**Selector, User Interface:** A user interface in an economics experiment will give subjects a way to enter their chosen action, and will display feedback about payoffs and counterpart actions. Popular experimental economics software usually requires subjects to use their keyboard to type

in numbers, and displays are usually text-intensive. ConG emphasizes mouse clicks, radio buttons and slider drags for choosing actions; except for chat room options, players rarely use the keyboard.

In ConG, players select from a set of possible actions via a "selector". The radio button selector, as in Figure 1, restricts actions to a specified finite discrete set of pure strategies. Selectors for continuous action spaces include both the heatmap and bubbles. The heatmap selector (seen in Figure 3), combines a mixed current-strategy slider selector for own actions displayed on the vertical axis of the heatmap, with a counterpart strategies on the horizontal axis. The heatmap itself displays the payoffs for each possible action profile. The bubbles selector produces the display seen in Figure 4; subjects choose actions via a slider along the bottom horizontal axis of the payoff chart.

Certain payoff functions work well, or only work at all, with certain selectors. For example the "2x2matrix" payoff function only functions with the radio buttons and heatmap selectors; it would have to be adapted to work with the bubbles selector. Likewise, the "sum" payoff function functions well with the bubbles selector but would need some adaptation to work (in special cases where it is relevant) with radio buttons.

**Matching and Grouping:** ConG's configuration file allows for intricate grouping and matching of participants, creating the potential for many types of population games. The configuration file lets an experimenter assign subjects to specific groups and defines how those groups are to be matched with one another. For example, in periods 1-7 the configuration file in Table 5 creates a two player prisoner's dilemma game in which an individual player is matched with other individual (note groupSize=1 and matchType=pair) with random rematching after each period. In one-population games, as in period 10 in Table 5, players are mean-matched against everyone in their own group (including self: note the matchType=self). In ConG's two-population matching protocol (periods 8 and 11-13: note the larger groupSize and matchType=pair), an individual's flow payoff is a function of her action choice played against the average in a different group, the one her group is matched with. ConG's configuration file also allows for finely detailed, subject-by-subject period-by-period group and matching assignments, as explained in documentation website.

**Data Collection and Error Logging:** As experiments are run via ConG and subjects interact with one another, "tick events" (i.e. data logged and organized into a report available to the experimenter) are generated by the server once every $N$ milliseconds. Each tick event contains time, (or subperiod in the case of the discrete time setting), payoff, action and other pertinent information generated by the experiment. A separate logging program receives tick events and updates a .csv file as quickly as possible. Should a catastrophic event occur that ends the experiment abruptly, the .csv tick file will lose only a few of the most recent events, saving as much data as possible.

The current version of ConG has ticks events logged once every second, $N = 1000$ (and in discrete time settings, once every subperiod), but this may be adjusted without greatly hindering processing resources. It should be noted that ConG does not currently log continuously, i.e. there is no log of each action change made by subjects during an experiment. Implementing continuous, asynchronous action-change logging and handling the potential deluge of experimental data is not trivial, but statistical inference techniques used to analyze ConG experiments thus far have not required it.

## 5. Conclusion.

Laboratory tests of game theory have become increasingly important over the last 60 years. Recently researchers have started to study games that either have continuous action spaces or are played in continuous time, or both. ConG is a flexible software suite intended to facilitate the laboratory study of such games.

Of course, continuous games are more cognitively demanding than games with simpler strategy spaces. For that reason, ConG emphasizes highly visual feedback and player inputs. This leverages human cognitive strengths in visual perception and processing. It also leverages advances in computer hardware, catalyzed by videogames, that build tremendous computing power into cheap graphics processors.

ConG software has been designed from the outset to minimize latency with generic hardware and to give players the look at feel of a continuous video game. As documented above and in more detail online, ConG allows quick implementation of a wide variety of game experiments by experimenters with limited programming skills. A file editable in any

spreadsheet program allows a high degree of control, period by period, over the player matchings, payoffs and feedback. Games studied this way so far include various two player bimatrix games, population games, public goods games, coordination and threshold games, Cournot oligopolies. For example, Cason et al (2012) use heatmaps that change in real time to study continuous time Rock-Paper-Scissors games.

ConG is an open-ended and modular software suite. Experimenters with Java programming skills can use it to create payoff functions, matching procedures, and visual feedback beyond those ever conceived by its creators.

**References**

**Cason, T., D. Friedman, and E. Hopkins** 2012. “Cycles and Instability in a Rock-Paper-Scissors Population Game: a Continuous Time Experiment.” Working paper.

**Fischbacher, U.** 1998. “z-Tree: Zurich toolbox for ready-made economic experiments. Instruktionen fur Experimentatoren." *mimeo,* University of Zurich.

**Oprea, R., D. Friedman, and K. Henwood** 2012. "Separating the Hawks from the Doves: Evidence from Continuous Time Laboratory Games." *Journal of Economic Theory* , 146:6, 2206-2225.

**Friedman, D. and R. Oprea.** 2012. "A Continuous Dilemma." *American Economic Review*, 102:1, 337-363.

**Goeree, J. K., and C. A. Holt.** 2001. "Ten Little Treasures of Game Theory and Ten Intuitive Contradictions." *American Economic Review*, 91(5): 1402-22.

**Kagel, J.H. and A.E. Roth.** 1997 "Handbook of Experimental Economics.", *Princeton University Press*

**Thrall, R. M., C. H. Coombs, and R. L. Davis.** 1954. "Decision Processes." *New York: Wiley*.

**Rapoport, A. and Orwant, C.** 1962. "Experimental games: a review." *Behavioral Science* vol. 7, pp. 1-37.