Seaweed: A Web Application for Designing Economic Games

by

Lydia B. Chilton

S.B., Course XIV, 2006 and S.B., Course VI M.I.T., 2007

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

[JUNE]
May 2009

Author_____
Department of Electrical Engineering and Computer Science
May 19, 2009

Certified by_____
Robert C. Miller
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Seaweed: A Web Application for Designing Economic Games

by

Lydia B. Chilton

Submitted to the Department of Electrical Engineering and Computer Science on 19 May 2009,
in partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Seaweed is a web application for experimental economists with no programming background to design two-player symmetric games in a visual-oriented interface. Games are automatically published to the web where players can play against each other remotely and game play is logged so that the game's designer can analyze the data. Seaweed's interface draws heavily on interface features of Microsoft PowerPoint™ and Excel™, which experimental economists are familiar with. The design and implementation challenge in Seaweed is to provide an end user programming environment that creates games responsive to events and controlled by logic without the designer understanding programming concepts such as events and synchronization, or being burdened by specifying low-level programming detail. Seaweed achieves this by providing high-level visual representations for variables, control flow, and logic, and by automating behaviors for event handling, synchronization, and function evaluation. Seaweed contributes a end-user programming tool for economists, as well as generalizable designs for representing programming concepts visually. It also demonstrates that Amazon's Mechanical Turk is a viable platform for forming partnerships between people and paying them to perform cooperative tasks in real-time, cheaply and with high through put.

Thesis Supervisor: Robert C. Miller
Title: Associate Professor

Acknowledgements

I've been blessed with extraordinary good friends and good fortune. I want to thank God for all the gifts he gives us and the vast wells of strength he plants deep within us to combat insurmountable odds.

I thank my parents, Ed and Lisa Chilton, for the million things they've done for me. Here's a random sampling: correcting my homework, taking me tobogganing, sending me to public schools, pulling all-nighters with me for the science fair, making me go to church, sending me to MIT, and letting me call them while I'm waiting for the Saferide. Ma, this is your Mother's Day gift. I love you.

Big thanks to Clayton Tiberius Sims, Max Goldman, and Danny G. Little who contributed valuable designs to Seaweed and to Anna Dreber Almenberg, Erez Lieberman and David Rand at Harvard University's Program for Evolutionary Dynamics for the concept of Seaweed and for playing a key role in our participatory design.

I can't thank my advisor Rob Miller nearly enough. He did a very special thing for me – he let me show him what I do best, and then taught me to do it even better. And he's pretty stiff competition in a flame war. *Hantanyel órenyallo.*

Lastly, I want to say how much I love MIT. MIT is a tireless community of brilliant minds luring the future closer and closer. When I first walked through the gates of MIT I knew what world I was entering. The inscription above the gates read: *"Abandon sleep, all ye who enter here."*

Contents

List of Figures

Introduction

One of the fundamental problems in human-computer interaction is *end-user programming*: allowing people who aren't programmers to create novel systems as easily as they can describe them. One approach to this problem starts by making an end user programming platform for a specific domain, such as web customization[6], traffic simulation [4] and games for children [8][13][14][15][16]. Although there's no guarantee that a specific case will eventually solve the general case, the benefit is that by fully understanding the needs of one domain and building an end-to-end system for that domain, one can identify generalizable design patterns for important aspects of end user programming.

This thesis describes Seaweed, an end user programming platform specifically targeted to experimental economists for designing and deploying economic games. The history of economics is replete with theories based on the assumption that people behave rationally. Game theory, in particular, is a branch of economics that models how rational players interact directly in small groups rather than indirectly in large economies. Experimental economists are now aggressively challenging the relevance of theory by investigating how small groups of people actually behave in games. Some of the major results in experimental economics surround the notion that cooperation plays an important role in economic interaction – even though competitive strategies are more rational, cooperation actually pays off better[9].

Experimental economists do research by hypothesizing new games and offering subjects an incentive to play them. Typically, economists hire programmers to implement the games and then recruit subjects to play the games in a computer lab for an hour. The path from a hypothesis, to a game, to getting data from real subjects is long and slow. Economists have to find programmers, explain economics to them, and continually find new groups of people willing to come into the computer lab together at a given time. In order to remove these technical and logistical burdens from experimental economics, economists need an application where they can program their own games and then deploy them immediately to the web, where thousands of people can play them in a very short time.

All economic games involve players making choices that affect other players and then receiving feedback about the outcome of those choices. *Iterative* games repeat the choices over a series of rounds. Games may be *symmetric* or *asymmetric*. In symmetric games, players face the same choices and are given the same information, although their payoffs may be different. "Rock Paper Scissors" and

Prisoner's Dilemma are examples of symmetric games. Asymmetric games have players playing different roles with different choices. The prototypical example is the Ultimatum Game, where one player is the offerer, with the choice of how to split a sum of money, and the other player is the offeree, who chooses whether to accept or reject the offered amount.

Economic games tend to be simple, with limited choices for each player. The familiar children's game "Rock Paper Scissors" is perhaps the simplest economic game. Prisoner's Dilemma is also simple. In Prisoner's Dilemma, two players play a coordination game where they separately decide to cooperate or defect. Even in a simple game, however, making small changes can produce dramatically different results. Adding a new choice or new payoff structure to Prisoner's Dilemma results in significantly different outcomes [9], and changes as simple as adding an image to the interface of a game can substantially affect the results [12]. Tailoring a game and deploying it to a new culture has also brought new insights into human cooperative behavior [5][18].

Seaweed is a web application for designing two-player symmetric games with discrete choices. Seaweed has two parts: a designer interface for creating games, and a game engine for running them. The designer interface is similar to PowerPoint, allowing the user to draw the screens of a game (Figure 1). Button objects define the available choices on each screen, and the logic of the game is defined by a payoff matrix. Once designed, the game is stored in database tables on a server, where the game engine can retrieve it and generate instances of the game for pairs of players across the web.

Seaweed's approach is to take the programming concepts needed to design economic games and either represent them visually or automate their behavior so the designer never has to think about them. There are six key programming concepts involved in Seaweed: variables, control flow, events, logic, conditionals and synchronization.

1. **Variables.** In Seaweed, the designer never explicitly defines variables. Instead, variables are generated automatically when designer creates key objects they need to design a game. When the designer creates a screen, it is automatically associated with a variable. The name of the variable is the name of the screen and the possible values it takes are the labels of the button on the screen. Variables are also created when the designer defines a new payoff matrix through Seaweed's visual interface.

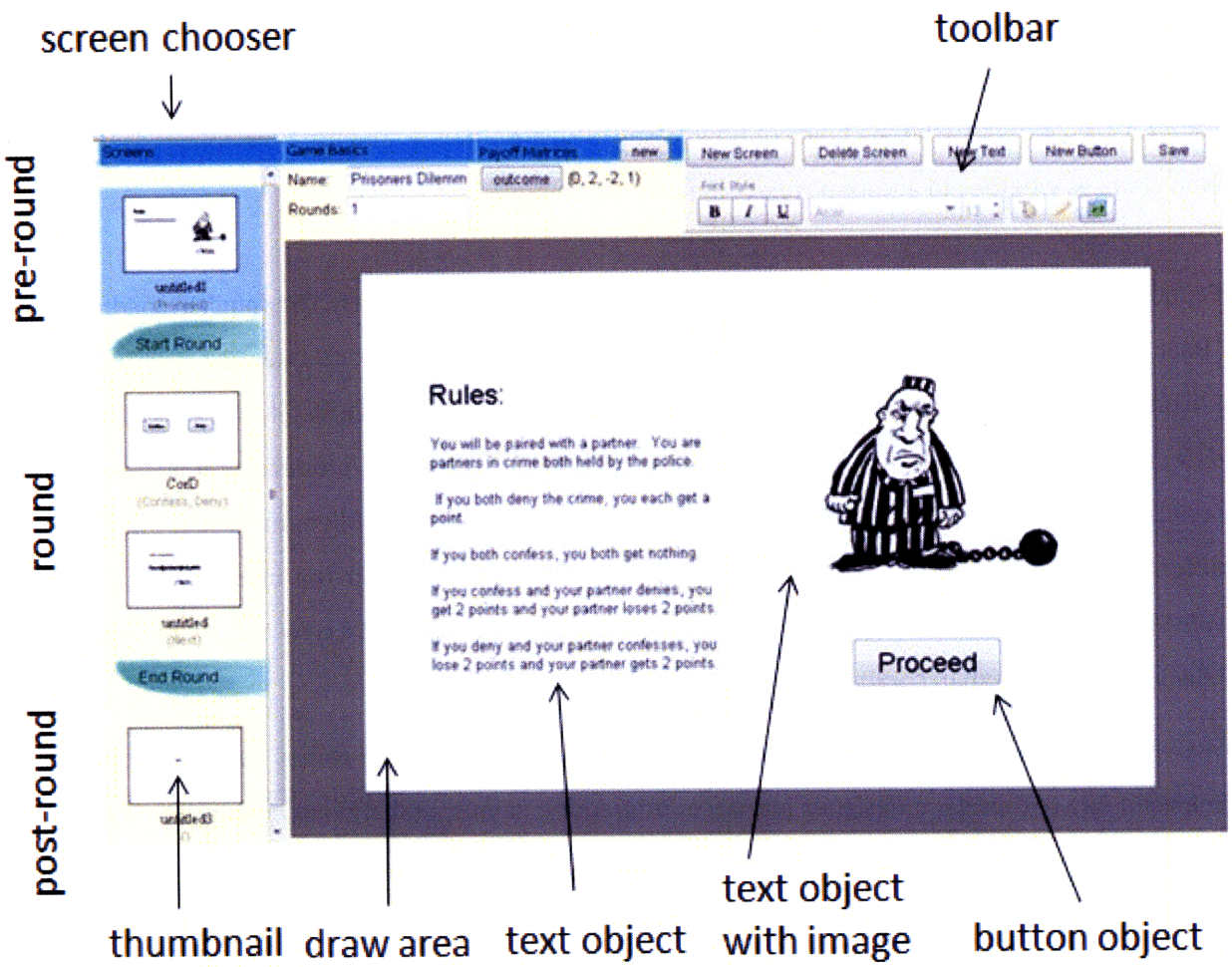

Figure 1: Seaweed's designer interface

2. **Control flow.** Seaweed's screen chooser dictates the order in which the states of the game (the screens) will be seen. In PowerPoint, the slides follow a very simple control flow, automatically proceeding to the next slide in linear order. In Seaweed, the control flow is more complicated because games are played in rounds, with some screens looping back to a previous screen if there are more rounds to play. To maintain consistency with PowerPoint as much as possible, Seaweed screens are seen in the same order as PowerPoint, but with a visibly marked looping structure built into the screen chooser.

3. **Events.** The only input widgets currently supported by Seaweed are buttons, so the user interface events that need handlers are all button clicks. Instead of requiring the designer to write explicit event handlers, however, Seaweed automatically handles button clicks. First it sets a variable in the database indicating which button was pressed, and then the player is taken to the next screen in the control flow.

4. **Logic.** Designers have to be able to specify the logic governing who wins the game. Economists have developed a visual representation for games, called a *payoff matrix*, which specifies the players, their choices (such as "Rock," "Paper" or "Scissors") and the outcome for every possible combinations of plays (Figure 2). A payoff matrix can be viewed as representing conditional logic with each cell being a different condition. The computation of a payoff matrix depends on the button that each player clicks, so it can't be evaluated until both players have clicked. Seaweed designers don't have to specify when the payoffs are evaluated, however. They are automatically evaluated as soon as the last player's click arrives. This behavior follows the dataflow model that has worked very successfully in other end user programming systems, such as spreadsheets.

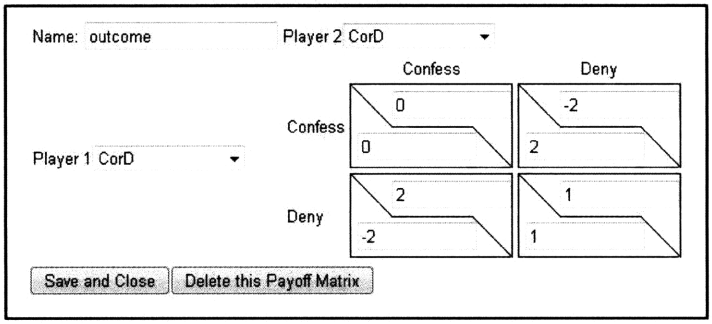

Figure 2 Payoff matrix interface

5. **Conditionals.** All games require giving players feedback about the outcome of the game. This typically involves making text on a screen conditionally dependent on the outcome of a payoff matrix. Seaweed allows designers to attach conditional logic to any text or button element on the screen. (Figure 3)
6. **Synchronization.** Games with more than one player usually have points where both players have to be in the same state at the same time. Synchronization can be challenging, even for trained programmers, so Seaweed handles it automatically. Before the game starts, Seaweed makes partners from the available players; players are synchronized when they start the game. After the game begins, players are only synchronized when they reach a screen that needs data from the other player. For example, in "Rock Paper Scissors" (Figure 4), the feedback screen shows what both players chose and who won the game. Displaying this screen requires both players to have clicked a choice on the previous screen. Players can play at their own pace until reaching a screen that requires synchronization, at which point they will be taken to a waiting screen until the other player's data is ready.

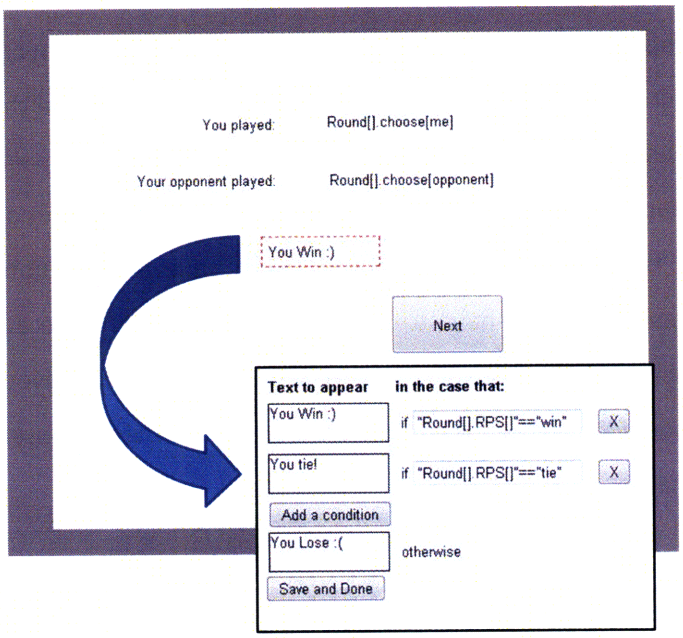

Figure 3 Conditional text interface

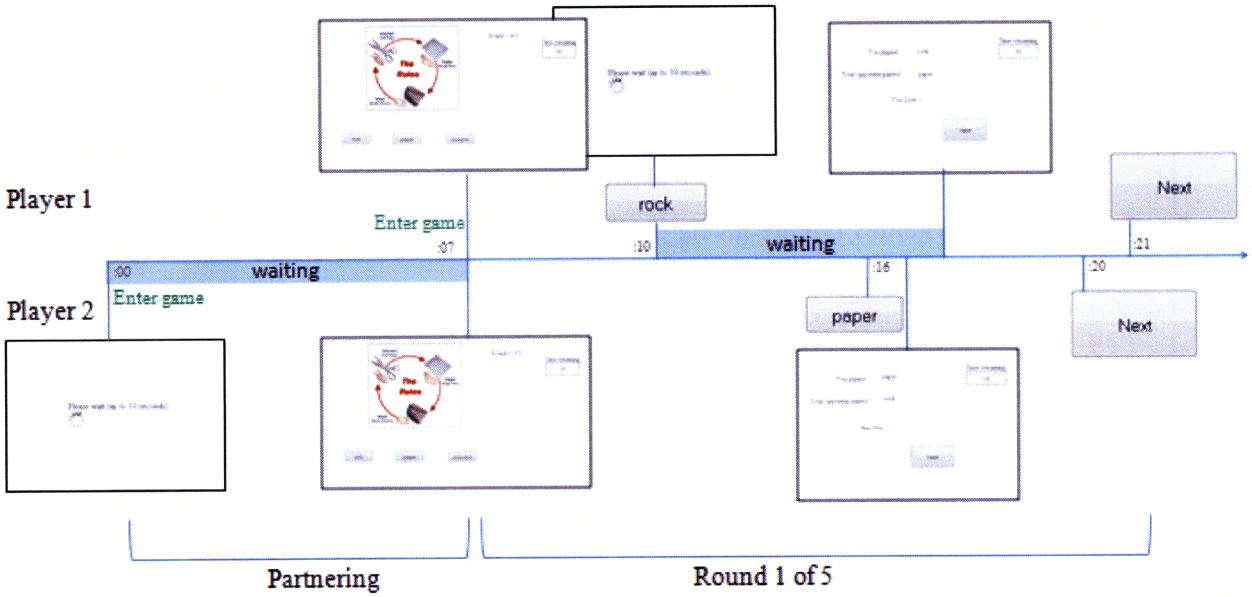

Figure 4 Waiting screen to synchronize players

Seaweed has two user classes: *designers*, who are the economists creating the games, and *players*, who are their subjects. Each user class has a different interface which must be evaluated in different ways. Designers use the design interface (Figure 1) which needs to be evaluated based on how well economists can discover its features and use them to create games without knowing programming concepts. Players use the game designed by the economist, running on Seaweed's game engine. We

need to evaluate if a Seaweed game can be deployed to players over the web in a way that can scale to thousands of subjects.

We evaluated the design interface by asking five economists to fix bugs in and augment a pre-existing game. We found that our visual approach was easy to learn. The consistency with PowerPoint made the design features very discoverable. Using payoff matrices was an effective way to represent game logic. The features that we automated (events, synchronization, logic evaluation) were indeed concepts that economists preferred to not think about.

We evaluated the game engine and justified claims about the scale of Seaweed games by posting a game of "Rock Paper Scissors" on Amazon's Mechanical Turk (MTurk). MTurk is an online labor market where buyers can post *human intelligence tasks* (HITs) for small amounts of money (typically $.01 to $.25) and sellers can accept those HITs, and get paid for their work. We were very successful at making partnerships. In under four minutes, we got 22 human-human partnerships and 12 human-computer partnerships to play all five rounds of the game for a total of $20.00. Usually, games are tested by advertising for subjects near a testing facility, paying them $5-10 dollars each and renting the facilities that only accommodate 30 people at a time. Seaweed demonstrates the high game-play through put of which MTurk is capable.

The design and implementation challenge in Seaweed is to provide an end user programming environment that creates games responsive to events and controlled by logic without the designer understanding programming concepts such as events and synchronization or being burdened by specifying low-level programming detail. Seaweed achieves this by providing high-level visual representations for variables, control flow, and logic, and by automating behaviors for event handling, synchronization, and function evaluation. Currently, Seaweed can design and deploy two-player, symmetric, games with discrete choices. Using the same general design techniques, Seaweed can be extended to cover more classes of games, as discussed in the future work section at the end of this thesis.

Seaweed makes the following contributions:

1. The design and implementation of a domain-specific end user visual programming system for economists to design and deploy economic games on the web.
2. The design of visual representations of key programming concepts (e.g. variables as screens and buttons, control flow as an augmented PowerPoint slide chooser, and logic as payoff matrices and conditional text).
3. Methods for simplifying distributed game design by automating event handling, synchronization and evaluation of logic.
4. An experiment demonstrating the efficiency in time and money of running partner-based economic games on Amazon's Mechanical Turk.
5. To our knowledge, Seaweed's evaluation demonstrates the first instance of real-time cooperative tasks completed on Amazon's Mechanical Turk.

The rest of this thesis is organized as follows. Chapter 2 covers related work. Chapter 3 details the design of Seaweeds interface for designing games. Chapter 4 covers the implementation of the design interface and the game engine for running games. Chapter 5 contains the evaluation both of the designer interface and of playing games on MTurk. Chapter's 6 and 7 are the conclusion and future work.

2. Related Work

2.1 End-User Programming

End-user programming is a technique for allowing people who aren't programmers to create novel systems as easily as they can describe them. In the introduction to "Watch What I Do"[7] Allen Cypher surveys existing end-user programming systems. Although his specific focus is programming by demonstration, he discusses three other approaches to end-user programming: soliciting preferences, scripting languages and macro recorders. Seaweed isn't a strict implementation of any of these approaches but it does draw insights from them.

Cypher defines *preferences* in a software interface as "predefined alternatives" to the common mode of operation. By specifying enough preferences a user can theoretically have a system tailored exactly to their needs. This is the concept behind wizards – dialogs which ask a series of questions to expedite the user's task. In practice, preferences aren't able to tailor a system to a user's exact needs because it's cumbersome to solicit the number of preferences needed to accommodate exact needs. For Seaweed, we decided against preferences and wizards. Designers need to be able to specify any part of their design in any order rather than have the entry process be guided by a wizard. This is best achieved by a visual, direct manipulation interface rather than one controlled by preference dialogs, for the sake of visibility.

Scripting languages are programming languages with a special vocabulary to operate on a specific domain. Although economic games constitute a specific domain with unique abstractions, such as payoff matrices, the learnability challenges of programming languages make them less than ideal. Scripting languages require the designer to use exact syntax and learn language-specific vocabulary. They also require knowledge of computer science concepts, such as variables, loops, conditionals, and events. For Seaweed's target users, learning the syntax and the subtleties behind these concepts (such as types, and guarded loops) would be a barrier to using the system. Making Seaweed a visual language eliminates the need for correct syntax and language-specific vocabulary, and helps support tasks that require control structures and other elements of programming with specialized widgets.

Macro recorders allow users to program without any syntax at all. They record the user's actions and allow them to be replayed, often with some amount of variation. Macro recorders are typically used to

automate repetitive tasks, which doesn't apply to Seaweed. Closely related, though, is *programming by demonstration,* where a user "refers to an action by performing the action." It avoids technical concepts by mapping actions to representations of objects. Seaweed doesn't employ macro recording; it has different strategies to elicit complex behaviors from its designers.

2.2 Microsoft PowerPoint and Excel

Although the main user interface challenge of Seaweed is to help non-programmers to essentially program, Seaweed also needs to be a place for designers to specify the layout of each screen of the game. PowerPoint is a widely successful and familiar designing environment for end users. PowerPoint uses direct manipulation to create, edit, move and resize text and images on a canvas representing the computer screen. PowerPoint represents a thumbnail of each slide in the slide chooser allowing the user to switch between slides. Seaweed has a similar interface, but for screens of the game instead of slides.

PowerPoint solves some difficult problems relating to direct manipulation of text and images. In PowerPoint 2007, when text is editable, a cursor is present in the text. Anytime the cursor is present, eight-way resize knobs and a dashed border that serves as a drag handle also appear. Drag and resize handles are also available when the text is not being edited, but the border is solid rather than dashed to indicate that the text is not yet in an editable mode. To create text, the user selects "Insert" and then "Text Box." One click on the screen will make a very small text box, a drag click will make a text box as wide as the drag operation, but with default height. PowerPoint users are very familiar with the details of how to manipulate text and image objects. Seaweed does not replicate them exactly, but uses very similar operations.

Microsoft Excel is the most widely used end user programming environment [17]. It allows users to write complex functions aided in large part by referencing variables visually and applying some common programming syntax such as quotes around strings, equal signs to test equality, parentheses for function arguments and conditional statements, which its users have been willing to learn in order to use the software. Seaweed uses conditional statements to control what text and buttons to display. Expressing conditionals in Seaweed is more visual than Excel, but still relies on conventional syntax for the test expressions.

2.3 z-Tree

Currently, many experimental economists are using an application called z-Tree (Zurich Toolbox for Ready-made Economic Experiments) to make games for laboratory experiments [11]. z-Tree's strength is that it can design a wide variety of games, including multi-player games, symmetric or asymmetric games, and auction games. It is meant to be an end-user programming environment for economists, but it has two major limitations. First, games can only be deployed on lab computers, which means all the subjects have to be in a lab at a particular time to play each other, rather than playing online at any time from any environment. Second, it isn't as easy to use as the economists would like: there is very little freedom in the visual appearance of the screens, and it leans heavily on syntax and programming concepts.

z-Tree defines several novel concepts that part of mainstream economics or computer science literature[10]. Z-Tree games contain *stages*, each of which consists of two screens – an *active screen* where users can see data and enter data, and a *waiting screen* where players wait until all the other players are ready. Each z-Tree game has a *background* – data not specific to a particular stage, such as the number of players, the number of rounds, and the message displayed on the waiting screen. However, *parameters* are stored separately. z-Tree also allows users to create *items* which are representations of variables. Items belong to a *treatment* which is a part of the game during which data is collected.

Users design games in z-Tree almost entirely through dialog boxes. Dialogs make buttons, create variables, create new stages, and define the logic of games. A visual overview of the game is given in tree form. New elements can be created and added to the tree to extend the game. For example, creating a new button requires opening a menu for Treatments, selecting "New Button" and then giving it a name, a label, a color (red or grey) and saying whether it advances the user to the next screen.

The button dialog doesn't offer control over positioning, size, font face, font size, etc. A layout menu for screens handles positioning; size and appearance are not in the user's control (Figure 5).

The Layout menu can also be used to specify an item. It uses a syntax unique to z-Tree where the type of input, the parameters it takes, and the player's options are all specified in one expression. The exclamation mark indicates that the field does not contain simply a number (Figure 6).

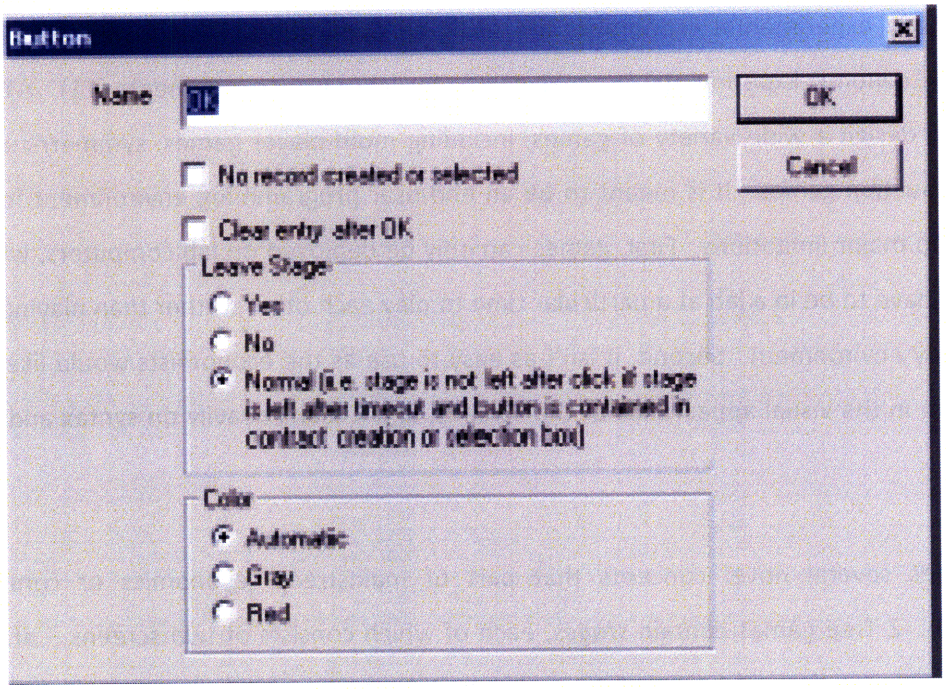

Figure 5 z-Tree dialog for customizing a button

Layout	input variable	output variable
2		
!radio: 1 = "86.8"; 24 = "102.8";		
!radioline: 0="zero";5="five"; 6;		
!slider: 0 ="A"; 100= "B"; 101;		
!scrollbar: 0="L";100= "R";101;		
!checkbox:1="check me";		
!text: 2 = "two"; 3 = "three"; 5 = "five"; 7 = "seven"; 11 = "eleven";		
!button: 1 = "accept"; 0 = "reject";		

Figure 6 z-Tree syntax for creating input widgets

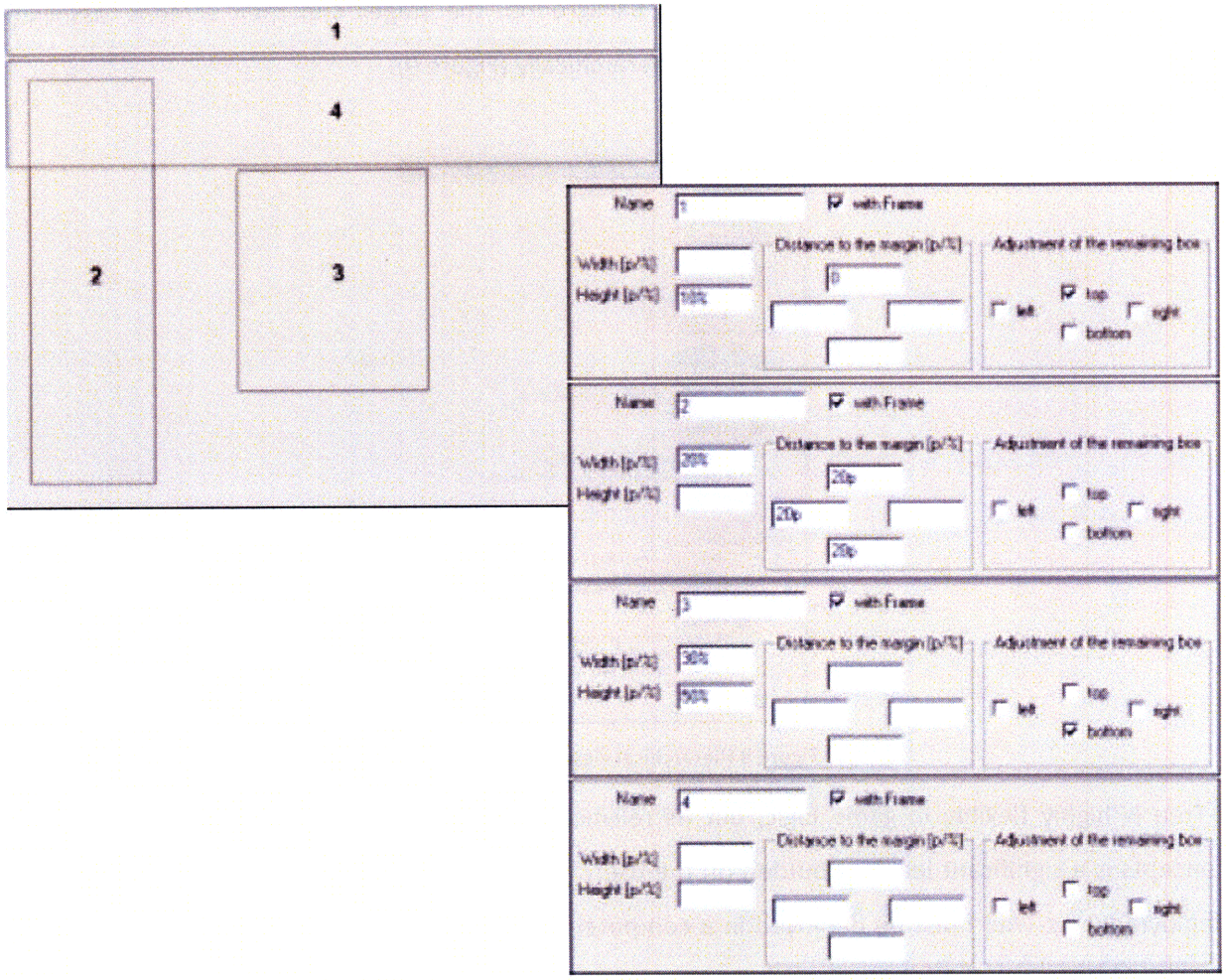

Figure 7 z-Tree layout options dialog

The screens are also designed with dialogs. Layout and positioning of "Boxes" is specified in a menu (Figure 7).

```
contracts.do{
    if ( Buyer == :Buyer & Seller == -1 ) {
        Seller = -2;
    }
    if (Seller == :Seller & Buyer == -1 ) {
        Buyer = -2;
    }
}
```

Figure 8 z-Tree do-statement syntax

The basic game logic of z-Tree is handled by programming a "do Statement." The user can assign variables, write control structures, in Java-like syntax (Figure 8).

The entire game is displayed in a tree. The background, the stages and their screens, associated variables and do Statements are all presented hierarchically (Figure 9).

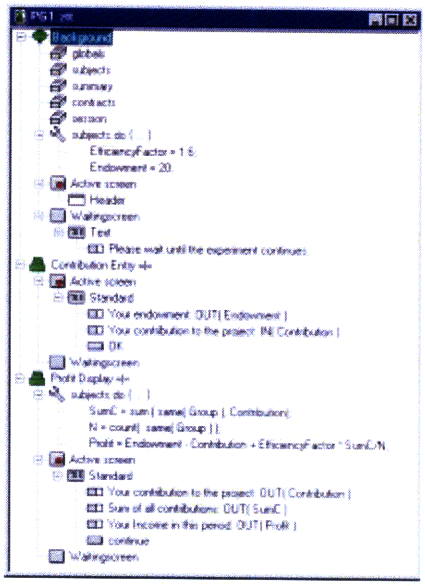

Figure 9 Hierarchical view of a z-Tree game

z-Tree is highly flexible in game logic, but its reliance on programming concepts and z-Tree-specific concepts is a significant learning burden on the user. z-Tree is fairly inflexible in graphic design and in deployment. z-Tree must be deployed in a computer lab setting and the design is constrained to the options in its layout dialog boxes.

2.4 Games with a Purpose

Seaweed is about designing games to be played on the web with the purpose of learning more about human economic behavior. In the field of human-computer interaction, web-scale games have already been used to collect data for useful purposes[1]. Luis von Ahn developed the ESP Game[2] and Peekaboom[3] which allow two randomly selected users to collaborate to solve a problem that is easy for humans to solve and difficult for computers to solve, such as image labeling. The ESP Game is a two-player, symmetric game similar to a cooperative auction where the agents involved are making bids not to compete with their partner but to converge on an agreement. Peekaboom is a two-player asymmetric game, also with a cooperative auction. The structure of these games is more complicated than Seaweed can design, but falls within the same general domain. The ESP Game and Peekaboom have been successful in that they are popular enough to partner players at any time, and that people will play them for free because they are fun.

We deployed Seaweed games on Amazon's Mechanical Turk, playing players small amounts of money for their time. However, it is worthwhile to note that people may be willing to play the games designed with Seaweed for other types of rewards such as fun, social capital or perhaps a charitable donation in their name.

3. Design

Seaweed's user interface design has four distinct elements: (1) designers need to make the graphic elements of the game; (2) they need to control what data is collected from the game; (3) they need to display data from the current game that is already logged in the database, and (4) they need to control the logic of the game.

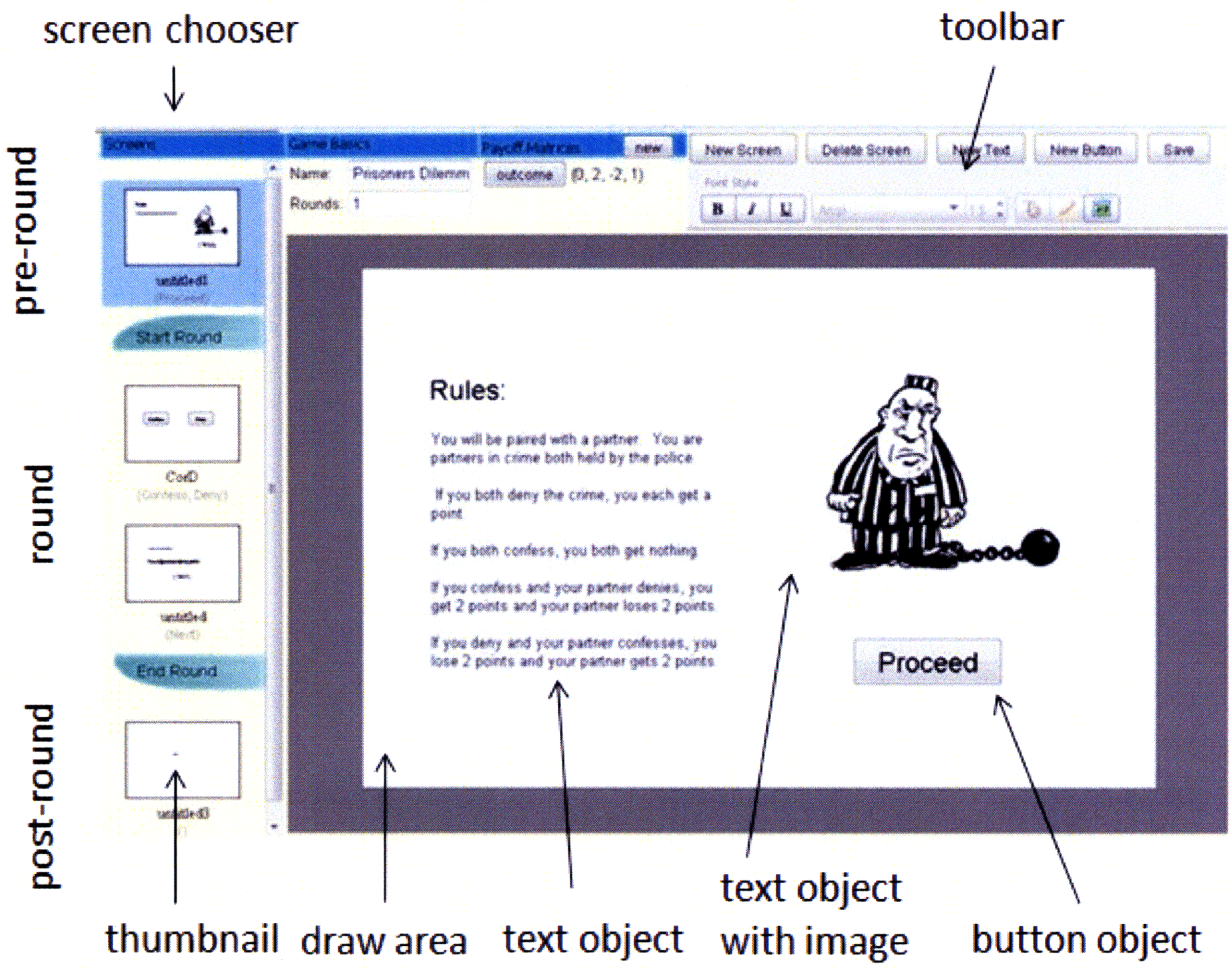

Figure 10 Seaweed's design interface

3.1 Graphical Interface tools for designers

Seaweed allows designers to make the graphic interface of its two-player, symmetric games by creating the screens and putting text and button objects on them through direct manipulation. The screen in the draw area is the currently selected screen (Figure 10).

When the "New Screen" button is clicked a new screen is created and inserted after the currently selected screen. The "Delete Screen" button deletes the currently selected screen and makes the next screen the currently selected screen.

The screen chooser draws on designers' familiarity with PowerPoint's interface for selecting slides. To be consistent with PowerPoint, the order that screens appear determines the order in which the players will play them. A key difference is that the Seaweed screen chooser is broken into three sections: pre-round, round and post-round screens. Screens within the round section repeat in a loop. The screen chooser contains thumbnail views of all the screens, displayed at 1/7 size. When a screen is selected, its thumbnail is highlighted in blue. Screens can be dragged and dropped within the screen chooser to change the order. Screens are given a unique default name, but names can be edited by double-clicking the current name. Seaweed may amend the name, if it is not unique. Names of the buttons on a screen appear below the screen name in grey for easy reference.

While a screen is selected, designers create text and buttons on the draw area to populate that screen in a direct manipulation interface similar to many existing design-oriented applications. Text widgets and button widgets can be dragged and resized in eight directions, and their text can be edited, including inserting images. To edit a text or button, the designer clicks on it to show the drag and resize handles and to enable the text to be edited.

3.2 Data Collection

The purpose of economic games is to collect data. Seaweed has two ways of collecting data: button clicks and payoff matrix evaluations.

Button clicks:

Although there are many types of data that designers could want to collect from players, Seaweed currently restricts player input to discrete choices in the form of clicking one button per screen. Each screen is listed as a variable in the database and the value of that variable is the text on the button that the player presses on that screen. One consequence of this is that screens with one button – such as a "Ready to play?" button – are variables that always have the same value logged to them. Although this

may seem unusual, an advantage is that the time the user clicks that button is logged. This allows the timing of players' actions to be recreated from the data.

Payoff matrices:

Payoff matrices allow the designer to define a new variable whose value is dependent on the actions two players. For both players - "Player 1" and "Player 2" - the designer picks a variable, and Seaweed creates a grid with the possible values of the chosen variables on the axis. In the matrix the designer can enter an outcome for both players.

In "Rock, Paper, Scissors," a payoff matrix describes who wins and who loses for the nine possible combinations of moves (Figure 11). To create a payoff matrix, the designer specifies its name (which becomes the name of its output variable) and the input variables for each player. The interface automatically makes a grid with all the possible values of the variables chosen on the axes. Changing the input variables automatically updates the grid to have the correct axes.

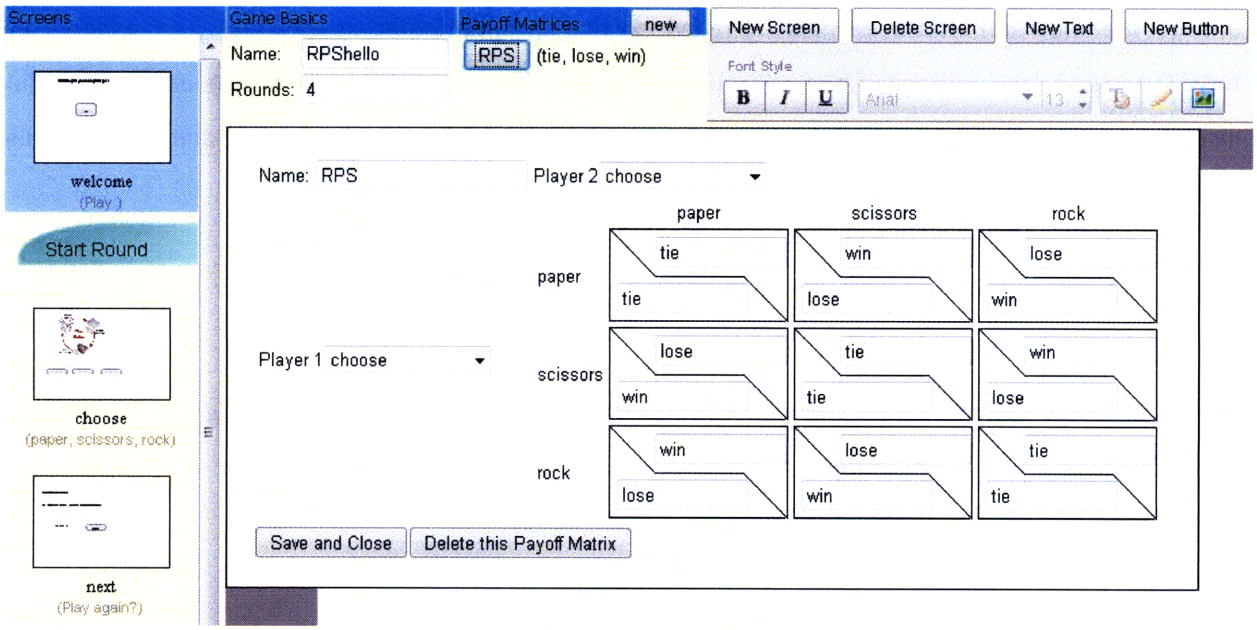

Figure 11 Payoff matrix widget in "Rock Paper Scissors"

The "Rock, Paper, Scissors" payoff matrix happens to be symmetric – it considers the same options ("rock", "paper" or "scissors") for both players. Payoff matrices do not need to be symmetric. For example, in a hypothetical game where players play "Rock Paper Scissors" and also decide to punish or

reward the other player, a payoff matrix could specify the outcomes of one player having chosen "rock," "paper," or "scissors" on the "Choose" screen and his opponent having chosen "Punish partner" or "Reward partner" on a different screen. (Figure 12)

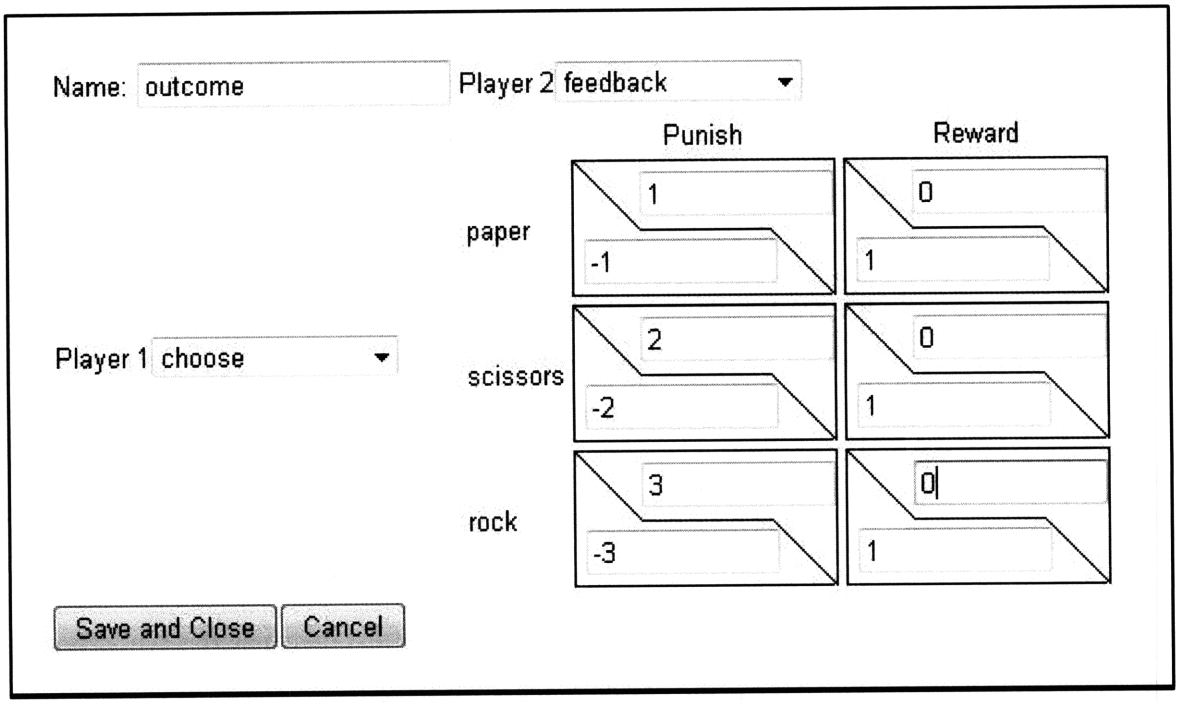

Figure 12 Example asymmetric payoff matrix

3.3 Retrieving and Displaying Values from the Database

Feedback on the results of a player's actions is essential to games. For example, in "Rock, Paper, Scissors," players need to know what their opponent played and whether they won or lost (Figure 14).

Seaweed has a text-based mechanism for accessing values from the database. In order to retrieve a value, Seaweed needs to know the round number, the variable and a designation of the player. A full specification would use the syntax :

Round[***n***].***VariableName***[***player***]

Where ***n, VariableName,*** and ***player*** are filled in by the designer (Figure 13).

n	Where n is an integer specifying the round number. 0 is the current round; positive values of n are indicated round numbers and negative values of n mean that the round you want is the "current round number + **n**". If you don't use Round[**n**], Seaweed will use the current round. Using Round[] will also use the current round.
VariableName	"***VariableName***" is simply the name of an existing variable that you want the value of.
player	"***player***" can be one of four things. The designer can specify a player by name - "Player 1" or "Player 2." Or they can use "me" or "opponent" to specify a player relative to which player is seeing the screen.

Figure 13 Syntax for retrieving data on a screen

Any text appearing in a text widget, button widget or in conditional text matching this syntax is replaced by the value in the database when the game engine renders the screen.

3.4 Game Logic

Logic is a common source of software bugs. Seaweed designers are largely not programmers and accordingly, much of the game logic has been taken out of the designer's control and replaced with sensible defaults.

Partnering

As soon as a player indicates that they want to play, Seaweed starts looking for a partner. Partners are made on a first-come, first-serve basis, regardless of any other factors, such as whether or not they have been partners before. A partnership remains intact until the end of the game. Players cannot change partnerships mid-game.

Screen Transition

In the screen chooser, there are three sections: the pre-round, during-round and post-round screens. Consistent with PowerPoint, the screens advance in the order that they are shown, with the exception

that the screens in the round section are repeated. The number of rounds is specified by the designer under "Game Basics." Both players must click through every round – the actions of one player cannot terminate the round for another player. Screens only advance when a button has been clicked. This means every screen except the last screen must have a button on it. If the player does not click a button within thirty seconds, they are logged out of the game and their partner is informed that they will be starting a new game as soon as another partner has been found.

Payoff Matrix

The designer does have to specify an event to trigger the evaluation of a payoff matrix – payoff matrices are evaluated as soon as the data they depend on is available. If the payoff matrix depends on a variable that occurs every round, the payoff matrix is reevaluated every round.

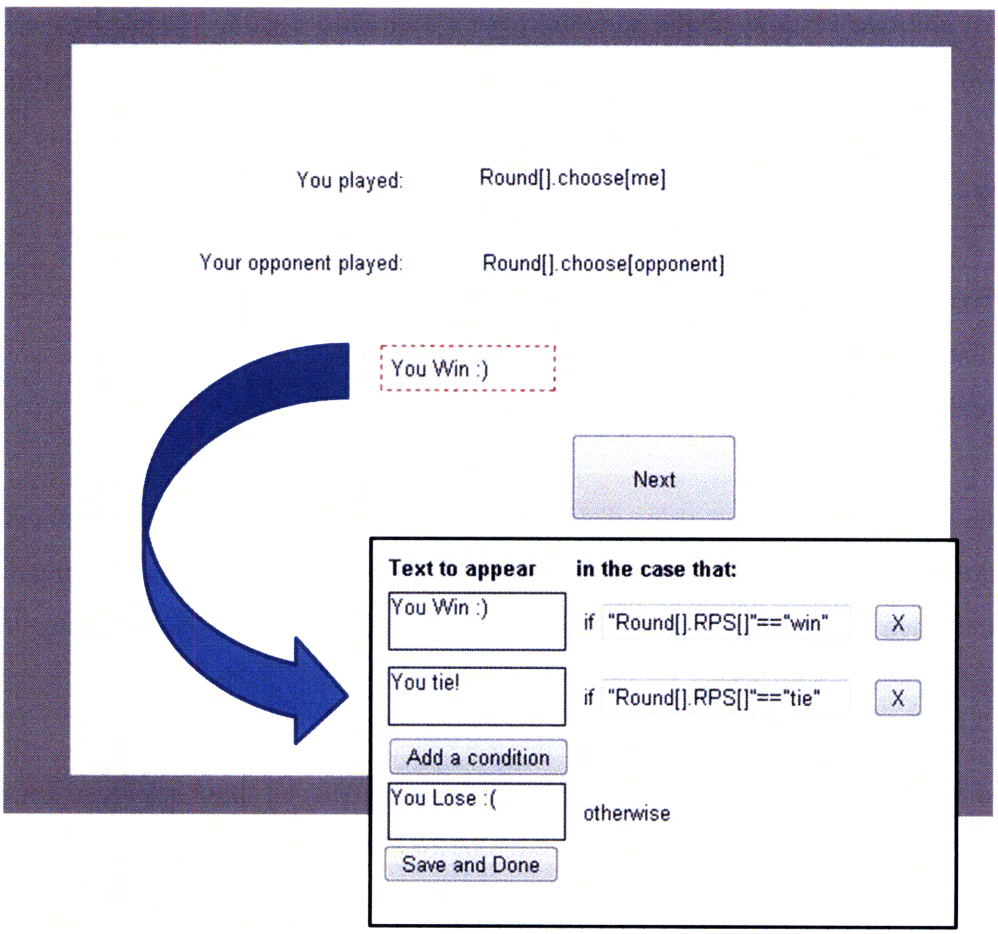

Figure 14 Conditional text in "Rock Paper Scissors"

Conditional Text

Designers have the ability to display text conditional on data in the database. The designer specifies which text they want to display and under what condition. The condition will be evaluated in JavaScript and uses JavaScript syntax as well as Seaweed's syntax for retrieving data from the database (Figure 14).

Waiting for Variables

When the text or buttons on a screen are dependent on values in the database, the first player to advance to that screen may have to wait for the next screen. When the player advances, Seaweed looks for which values are needed for the new screen, and will display a waiting screen until all the variables are ready.

3.5 Design Decisions

Seaweed underwent four distinct cycles of iterative design: two low-fidelity prototypes (simulated on a whiteboard) and two versions of the software over a 16-month period. The big design challenge of Seaweed was the interface for designing games without programming. We assume that our users are economists experienced in specifying economic games on paper and with programming ability on the level of basic Microsoft Excel skills.

The path from the first version to the current design involved two main challenges:

1. Reduce the redundancy of user input.
2. Eliminate the need for designers to understand programming concepts.

Our first design was very flexible: using dialog boxes, designers could specify how partners were to be made, create arbitrary variables, jump to screens in any order, increment or decrement the round number, and leave screens and enter screens triggered on a number of different events, including the other player's actions (Figure 15).

The immediate drawback was that the designer had to be meticulous about specifying the game logic. The designer was in charge of incrementing the round when it was appropriate, sending a player back to the first screen in the round, and creating variables logging data in to those variables in database. In the first paper prototyping session, the economists grew tired of entering the same data over and over: for every button, they had to make an event that would take an action when it was clicked; for almost every

button they had to create a variable in the database; and for the last screen in the round, they had to instruct Seaweed as to what screen to send the player to next. The concepts of event handling and goto's were not natural to them. Moreover, entering all the low-level steps was redundant. Approximately one step in five would differ between specifying the actions of one button and another.

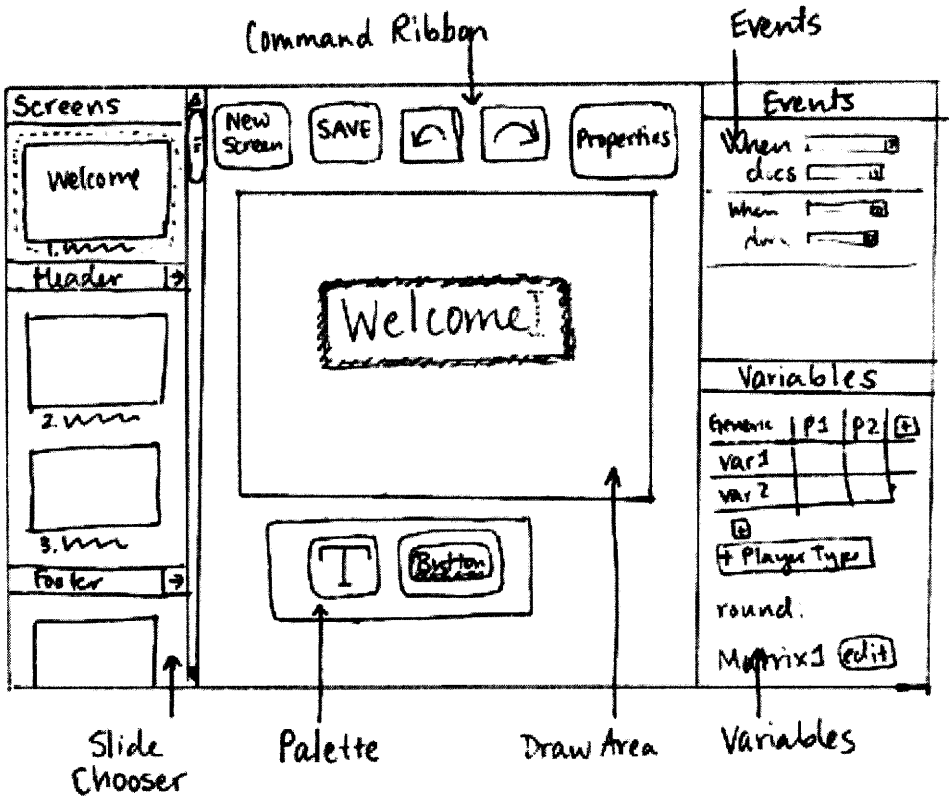

Figure 15 Early low-fidelity prototype of the design interface

After user testing the low-fidelity prototype of this design with three experimental economists, the design took a different direction. Seaweed's game design interface would be presented at a higher level, even at the expense of flexibility of design. Flexibility could be introduced later through escape hatches for programmers to add their own behavior to a game such as defining new partnering mechanisms or introducing custom widgets and store the data the widgets log. Replacing interfaces that relied on programming concepts with automatic behavior and sensible defaults played a large part in reducing the redundancy of user input.

We introduced the following simplifications to Seaweed:

1. **Automatic partnering**. In evaluating the low-fidelity prototypes, we realized that our designers didn't even think about the steps of partnering, rather they took partnering for granted. By sacrificing the ability to match players creatively (by gender, by age, experience, etc.) we were able to automate partnering and completely remove the partnering process from the interface.

2. **Screens are variables and buttons are value.** Creating variables was fairly natural to our designers, from their experience in Excel or statistical econometric packages. However, the process of creating variables was redundant. The largest simplification step we made was to not allow the user to create variables; rather, every screen was automatically a variable. We restricted input to discrete choices and made buttons on each screen the only way to enter a value into the database for the screen variable.

3. **Screens transition in order**. Goto's are empirically problematic to use, and their specification was redundant. We removed the option of specifying a screen's successor, and simply let the sequence of screens in the screen chooser determine order. The approach is familiar to PowerPoint users.

4. **Rounds are defined by repeating a sequence of screens**. In the screen chooser, Seaweed has three defined sections – pre-round, round, and post-round. Instead of defining goto's on screens to indicate successors, the screens inside the round automatically repeat. Although this is different from the slide chooser in PowerPoint, it allows the user to clearly visualize the start and end of the round, and is a step in eliminating goto's. By embedding a loop structure into the screen chooser, we are making it resemble a state diagram. This is an idea we will explore further in future work.

5. **Automatic payoff matrix evaluation** - Payoff matrices in Seaweed are calculated based on data collected from screens. Therefore, payoff matrices can be calculated as soon as that data is ready. The user doesn't have to define an event handler that recalculates a payoff matrix.

6. **Automatic waiting screens** – Seaweed's testing designers in the first paper prototype did not realize that players might have to wait for a screen to be ready because the other player had to make their move. The concept of synchronization was too low-level for them to be conscious of. In a paper-prototyping session where designs were meant to control the waiting and synchronization, our subjects didn't discover the synchronization features, despite the fact that they were prominently displayed with

other features that *were* used (in the same drop down menu). Subjects also couldn't recover from the error of not having used synchronization and waiting correctly (they just didn't know why the system wasn't responding). Even after being instructed about how and why to use synchronization, subjects couldn't remember to do it. These were all failings not of our users but of our design. In response, we automated the process of waiting by creating a screen that the players would see anytime they were between screens.

By automating payoff matrix evaluation, screen transitions, round repetition and by making screens variables and buttons their values, we were able to eliminate the concept of events entirely from Seaweed.

After making these design decisions, we built a fully-functional computer prototype. Because we had automated so many of the variables and events, we wanted to have a place in the interface of Seaweed that would expose the system by showing what variables exist, what values they take on for which players, and what kind of variable they are (screen variable, payoff matrix variable or a general parameter such as the number of rounds).

We created a variables panel to house this information (Figure 16).

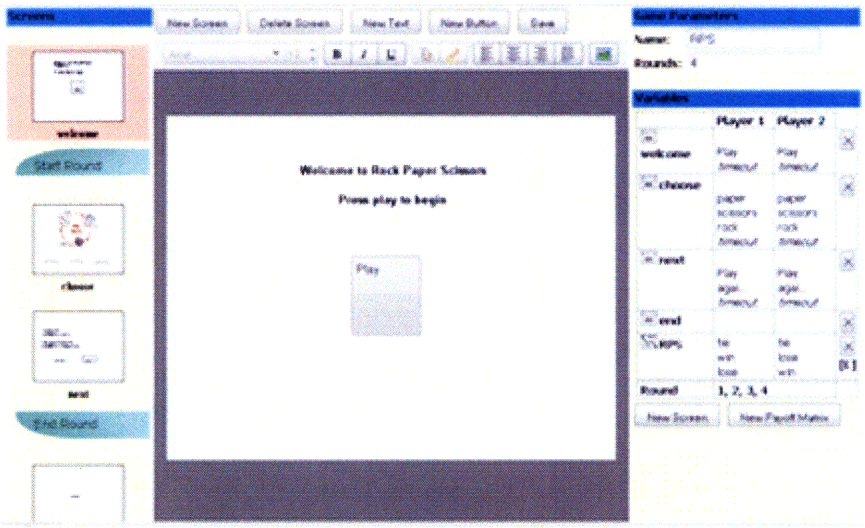

Figure 16 Version of design interface with the variables panel

After the evaluation of the first working prototype, we discovered that our test designers didn't use the variables panel at all. It wasn't clear they understood what it was; the symbols for payoff-matrix and screen variable were not clear; the list of values a variable could take on were not labeled, and designers didn't make inferences about what they were. When prompted, subjects guessed correctly that "Player 1" and "Player 2" referred to the two players and that the rows were variables, but weren't sure what the data in the table was (it was list of values for a given player and variable.)

The variables panel was also designed to be editable. Designers could change the names of their screens, delete screens, and edit payoff matrices. Although designed to be convenient, all the actions were duplicated elsewhere in the interface, and the variables panel contained nothing new.

The current design eliminates the variables panel. In order to retain some of the system exposure that the variables table provided, the design retains the list of values of a variable but relocates it. For a screen variable, the possible values appear in a comma-separated list under the screen name (Figure 17). For payoff matrices, the possible values appear next to the name of the payoff matrix.

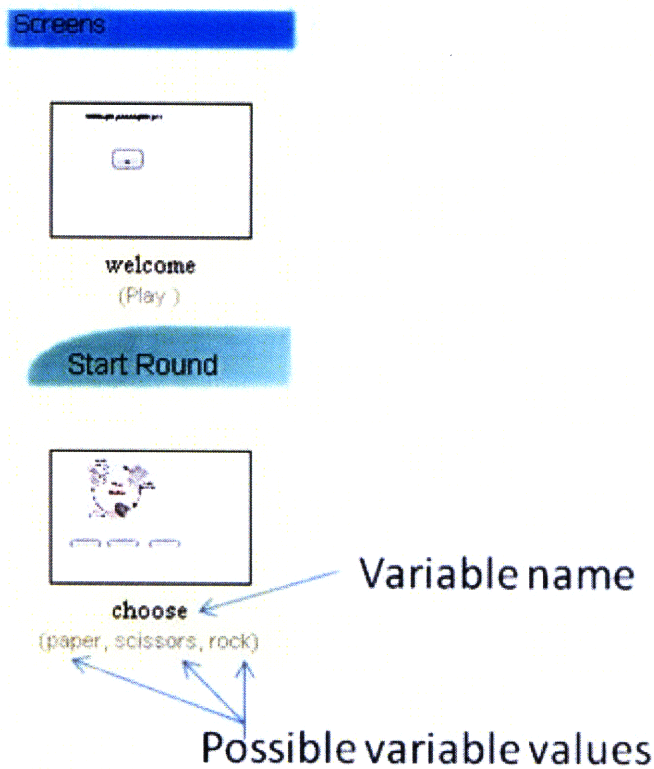

Figure 17 Details in the screen chooser indicating variable name and possible values of that variable

4.Implementation

Seaweed is written in HTML, CSS, JavaScript, SQL and PHP, and is hosted by the Linux-Apache-MySQL-PHP (LAMP) configuration available to all MIT affiliates at http://scripts.mit.edu. Seaweed uses JavaScript widgets from the Yahoo User Interface (YUI) toolkit; mostly notably, the Resize, Drag-and-Drop, Custom Events, AJAX and Rich Text Editor packages. Seaweed stores games in a MySQL database and uses its own game engine to generate the games dynamically as players are playing them.

4.1 Client Side Implementation: the Designer (HTML, JavaScript)

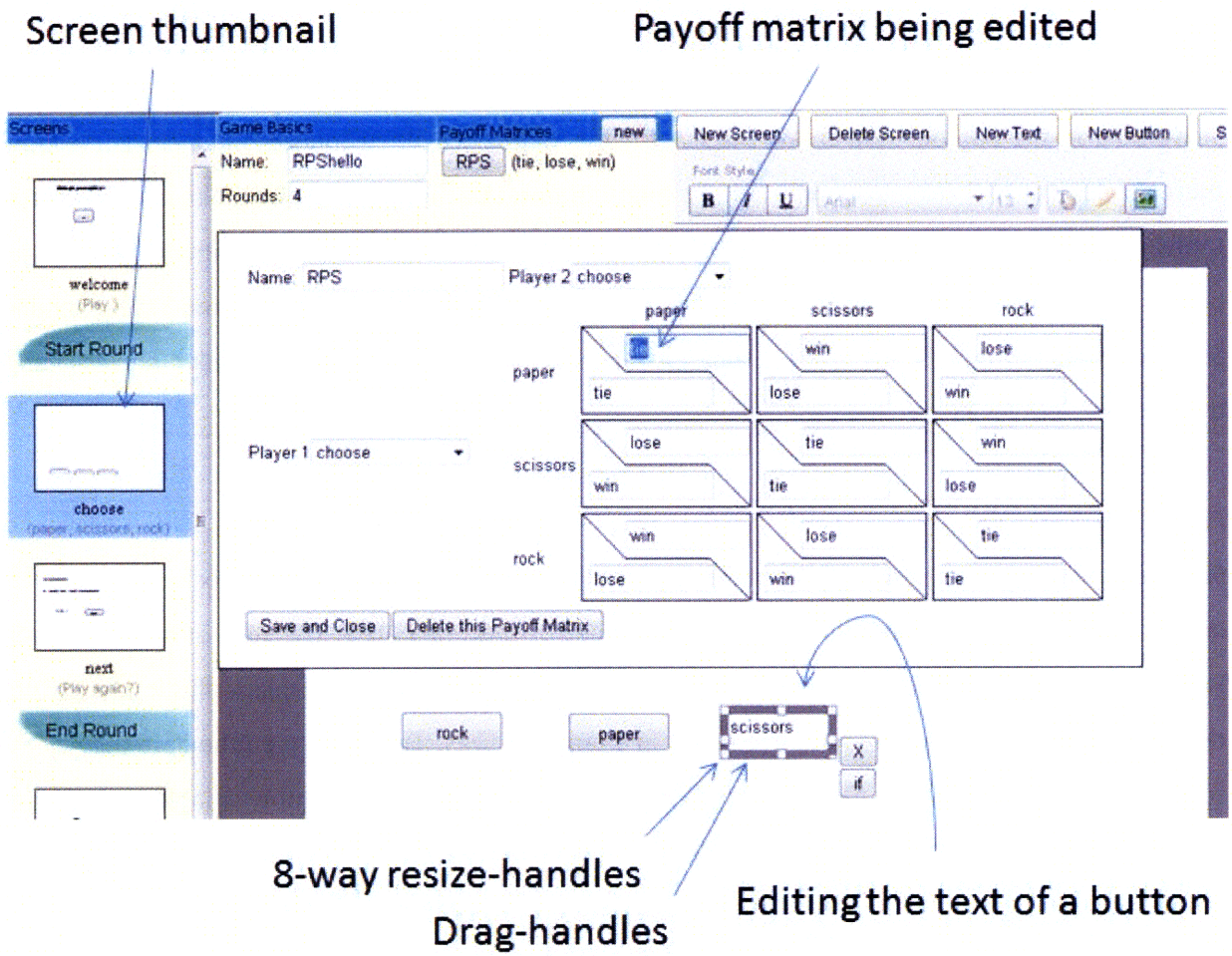

Figure 18 Screen thumbnails, payoff matrix editing, and text editing

Screen Elements: Text and Button

The two types of HTML elements designers can put on the screen are text and buttons. Text and buttons are normally displayed on the draw area with a div with a child that is either text or a button.

When the element is being edited however, the div is replaced with an iframe which serves as a rich text editor with 8-way resize handles and drag handles around its perimeter. The handles allow the user to resize and position the element. The rich text editor allows the designer to edit the text of the text or button element as users do in Microsoft Word or PowerPoint – they create the markup for text size, font and background color, font-face, and justification with a toolbar fixed in the upper right of the page. They can also insert images into the text (Figure 18).

Since only one object can be edited at a time, Seaweed only has one instance of a rich text editor. When no screen object is being edited, the rich text editor is hidden. When a screen object needs to be edited, the rich text editor is moved to the place of the screen object, set to the right height and width, and its HTML content is set to the HTML of the screen object.

Seaweed uses the YUI toolkit to create the screen objects, but heavily customizes them all. For example, the code for the YUI rich text editor comes as an iframe attached to a toolbar; Seaweed places the toolbar in a div on the screen which never moves, but sets the position of the iframe where the user places it. Seaweed also changes the CSS and some of the behavior of the drag and resize handles . Seaweed does not modify any of YUI's code. After the YUI object is rendered, Seaweed makes changes dynamically with JavaScript and CSS.

Payoff Matrices

Seaweed uses a custom-designed widget to specify the payoff matrix. A payoff matrix is displayed in the design interface as a div fixed in the top right corner of the draw area. The cells of a payoff matrix, including the piecewise-diagonal line, are drawn with individually placed 1x1 black divs in JavaScript so that they can extend to accommodate content in a range of sizes. Designers type the data they want in to textboxes in the cells. The payoff matrix lays out all the possible values of one variable against all the possible values of another variable. The designer picks the two variables to compare and the payoff matrix looks up their possible values, and creates a matrix of the right dimensions to compare each pair of values dynamically with JavaScript. Anytime the designer changes either of the variables, the widget is updated (Figure 18).

Thumbnails

Thumbnails are made using JavaScript and HTML in the same way that screens are made when they are initially populated – a div with text placed at a location relative to the upper left corner of the screen. The thumbnails are displayed at a 1:7 ratio to the screen (Figure 18). Each thumbnail looks at the model of for the representation of the screens it represents. It takes all of its objects (text and button) and scales down their visual properties including: font size, div width and height, x and y positions, text line height and image size. Each thumbnail listens for changes to its screen in the model and updates immediately – every change to any of the screen objects – movements, resize events, keystrokes and other changes associated with the rich text editor.

Conditional text

The text of text objects and button objects can be dependent on conditional logic. The interface is text area to write the condition (for example, "Round[]==1")and a rich text editor to enter the text they want to appear. Conditions can be added or removed but the "otherwise" condition will always remain. The size of the rich text editors is controlled by the size of the text or button whose text the designer is editing and cannot be controlled in the conditional text interface (Figure 19).

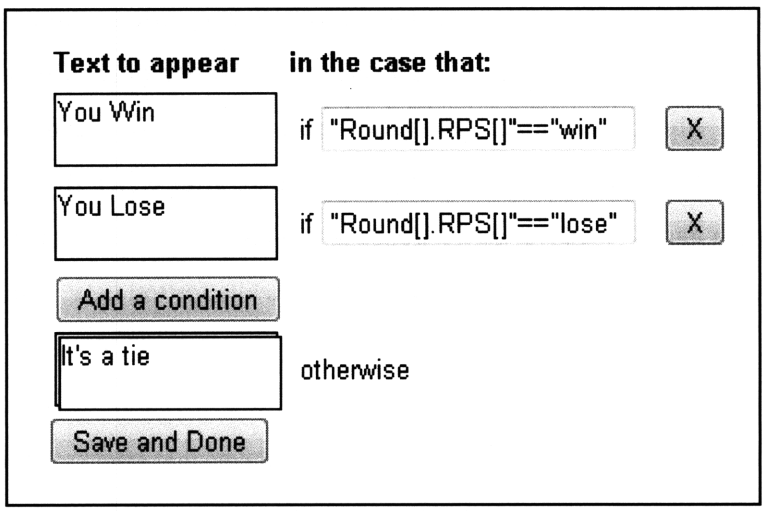

Figure 19 Conditional text for "Rock Paper Scissors"

4.2 Client-Side and Server –side Game Storage (JavaScript and SQL, respectively)

A designer's game is modeled in two places – in JavaScript objects on the client side and in database tables in the database on the server side. The JavaScript model is continually updated as the designer makes changes to the game: every time a payoff-matrix is saved, a condition changes on some text, text

is edited, screen objects are moved or resized, or the order of screens changes. When the designer saves the game, the entire JavaScript model is transferred in JSON to the server and is stored in the database. When a designer wants to edit a game stored in the database, the JavaScript model is repopulated from the database.

The database has a table for games which stores the game's name and a unique identifier. It has a table of screens that keeps track of what game id the screen is associated with, the order that screen is in the game and its name. There are separate tables for buttons, text and conditionals that store the position and HTML of the element and well as the screen id it is associated with. Payoff matrices are also represented in the database.

4.3 Game Engine (PHP, JavaScript, HTML)

Games are stored in tables in the database. Anytime a player wants to play a game, Seaweed generates the game from the database. At a high level, the game has a partnering phase and a game play phase. During the partnering phase, the player waits up to a certain amount of time to find another player looking for a partner. If he doesn't find a partner, he will play against a computer player that makes choices at random. During the game phase, the players move through every screen in order, waiting for their partner to move, when necessary. All of their button clicks are logged and they are given a timeframe in which to make a button click on any given screen, after which they are logged out. There are three ways to end a game. A player either reaches the last screen, forfeits the game by being idle for too long or the game is forfeited to them if their partner is idle for too long. (See Figure 20)

Partnering

Partnering is done on a first-come-first-served basis. A new player first checks if there is anyone already waiting for a partner. If there is, the two players are partnered and the game starts immediately. If there isn't, he puts his name in the queue waiting for a partner. He checks back every 2 seconds to see if he has been given a partner. After 15 seconds have passed with no partner, he is assigned a computer player. As soon as he has a partner – human or computer –the game starts.

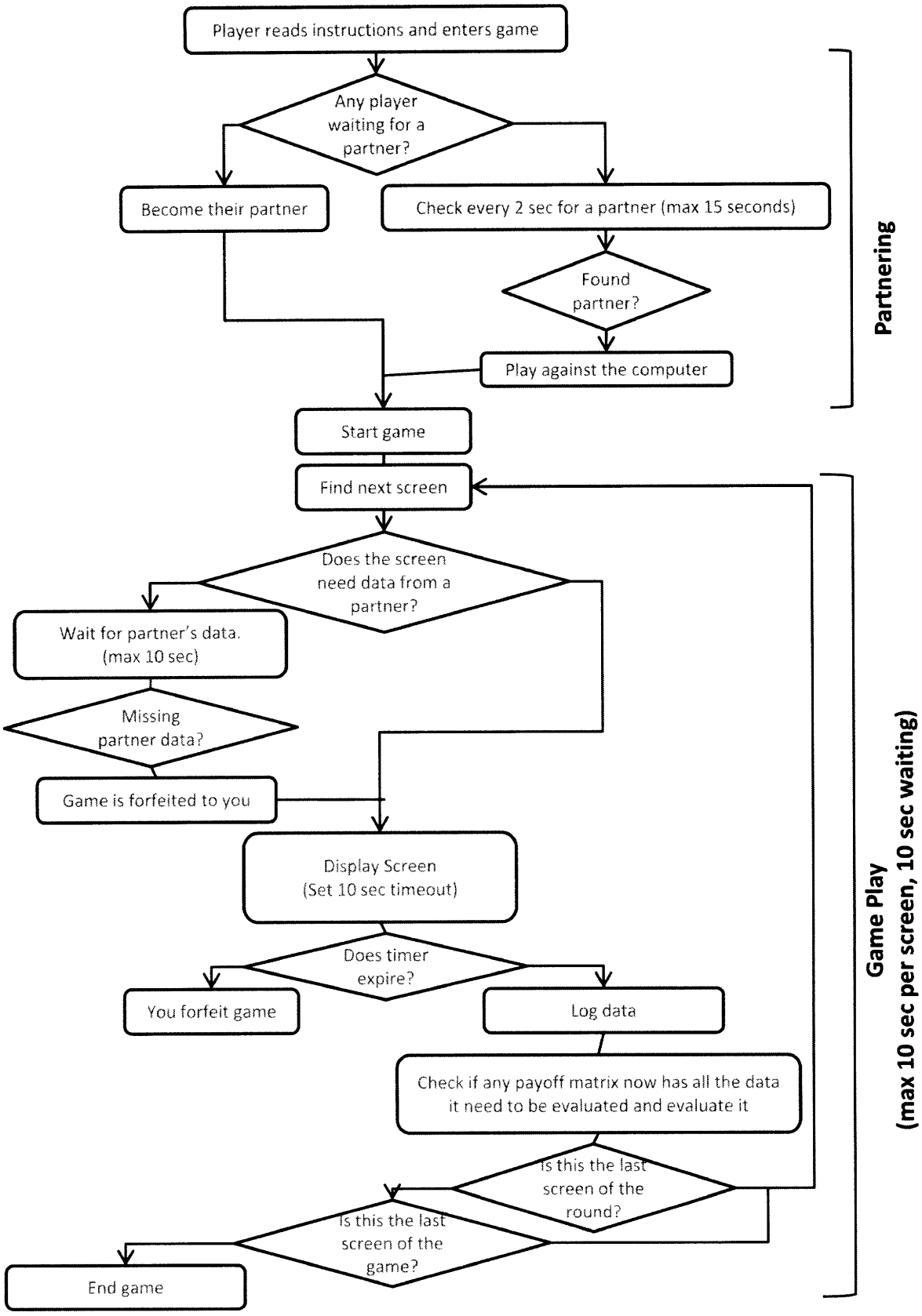

Figure 20. Flow chart illustrating the main actions of the game engine

Sometimes players put their name in the partnering queue and then leave the site. To help prevent making partnerships with players who have left, players without partners are removed from the queue after 15 seconds by the next player looking for a partner. By that time, they should have been assigned to a computer player, so they clearly have left the game.

Finding and Displaying the Next Screen

The screens of the designer are represented (along with the automatically generated waiting screens) the states of a state machine. In between each of the designer's screen will potentially be a waiting screen that will appear if the server is slow or a player is waiting on his partners' response. The first thing Seaweed does in the game play phase is find the first screen. When a player leaves a screen, the game will then find the next screen until there are no screens left. At the last screen of every round, Seaweed checks if there are more remaining rounds. If so, it increments the round and sends the player back to the first screen of the round. (Figure 20)

When the client asks for a new screen, it is expecting to either get back an array of text, buttons and conditionals affecting the text of text and buttons or to be told to wait because the screen requires data is not available yet.

Designers may use Seaweed's specialized syntax for displaying data from the current game within the game. Before returning the text, buttons and conditions needed for displaying a screen, we query the database for the next screen text, buttons and conditionals and use regular expressions to find any strings matching Seaweed's syntax for retrieving the current game's database values. If there are no matches, the screen information is sent to the client and the screen is displayed. If there are matches, Seaweed looks queries the database for that value. If all the values are present, the server sends the client the screen information with the values inserted in places of the Seaweed syntax. If there are values that are missing, the client is told to keep waiting.

If the server returns any conditionals to be applied to text or buttons, the logic of the conditions are evaluated by the client's browser in JavaScript when the screen is dynamically generated. We avoid evaluating logical conditions on the server side as a security precaution.

Any buttons that are created have an event listener attached to them that does three things: it stops the timer on screen, it logs data about the button press (the screen, round, player , and button, along with a timestamp), and it starts the process of finding the next screen.

Sometimes a player will see a waiting screen before they can proceed to the next screen (Figure 21). This happens when the server or connection is slow, but the more important case is when the player is waiting for their partner to respond. Although all Seaweed games are played with a partner, not all games will require a player to wait for a partner. A game of "Rock Paper Scissors" that just asks its players to choose rock paper or scissors over and over again will not need to wait for a partner. The actions of those two doesn't need to be synchronized. Actions only get synchronized when there is a screen that requires displaying data dependent on a partner. In "Rock Paper Scissors" there is typically a feedback screen after the choice screen where you learn what you partner played and whether you won or lost. To display this, Seaweed needs the partner's data and will wait for it (and any other data it need) before displaying the screen. At this point, the players are synchronized.

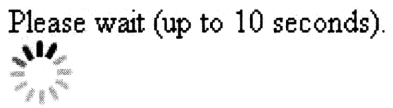

Figure 21 Waiting Screen image

Timeouts (JavaScript, PHP)

When Seaweed creates a screen, it puts a timer on the screen that counts down seconds starting at a given time limit (Figure 22). If one player exceeds the time limit before making a choice, that player forfeits the game and is taken to a screen saying so. The timer is implemented in JavaScript on the client side and is thus not secure. Malicious users could disable the timer and then have as much time as they want. To get around this, we also put a timer on the server. If the time a player takes between entering a screen and clicking one of its buttons is more than 2 seconds beyond the time limit allows for, he forfeits the game to his partner. As a partner, after you have been waiting for your next screen more than the duration of the time limit, you start to check whether you partner has forfeited to you. As a player, before you log any data, you are forced to check whether you have actually forfeited the game by taking too long.

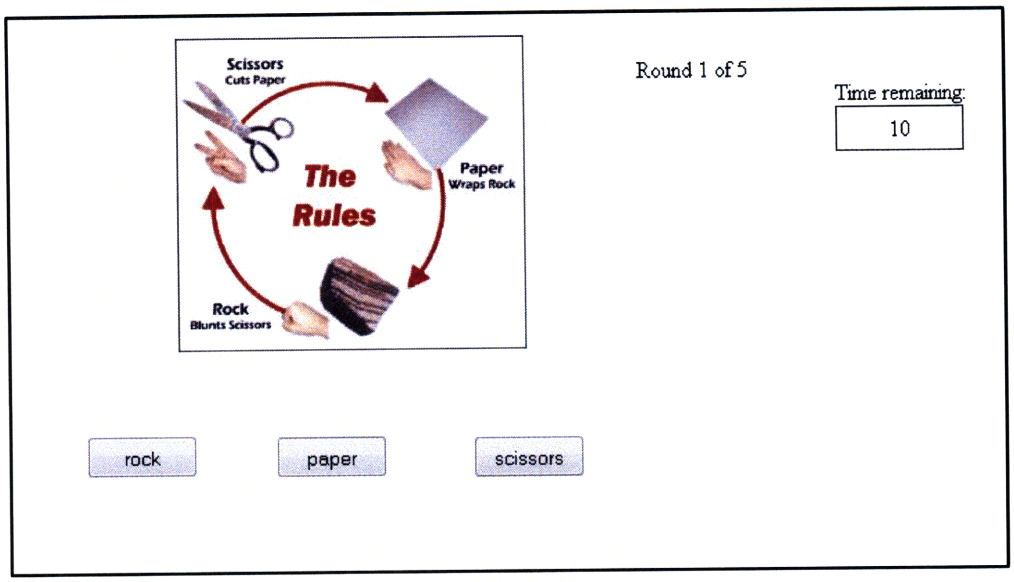

Figure 22 Screen of "Rock Paper Scissors" showing the timer with 10 seconds remaining

Logging Data(SQL)

Seaweed is very liberal about logging data. We would like to store enough data to recreate the game later. We log the partnerships that are made, every time a player enters a screen, every button clicked, the result of payoff matrix computation, and when a player times out (forfeits). Every logged event is timestamped, so that the timing of the game can also be recovered.

The most important data to log is the players' choices (corresponding to button clicks) and the results of the payoff matrices. Choices are logged immediately when the player clicks a button, but the evaluation of payoff matrices aren't by definition associated with any user action. Seaweed evaluates a payoff matrix as soon as all its input data is ready. Since payoff matrices are always dependent on data from a screen, the data will be available after either a player or his partner clicked a button on a particular screen. After a player logs his screen data, we check to see if any payoff matrix is now ready to be evaluated, and evaluate it if so.

Payoff matrices are really just a representation of a long list of conditionals – one conditional for every cell of the grid. For example, the Prisoner's Dilemma matrix has four conditionals: both players confess, both players deny, one player confesses while the other one denies, and one player denies while the other one confesses (Figure 23).

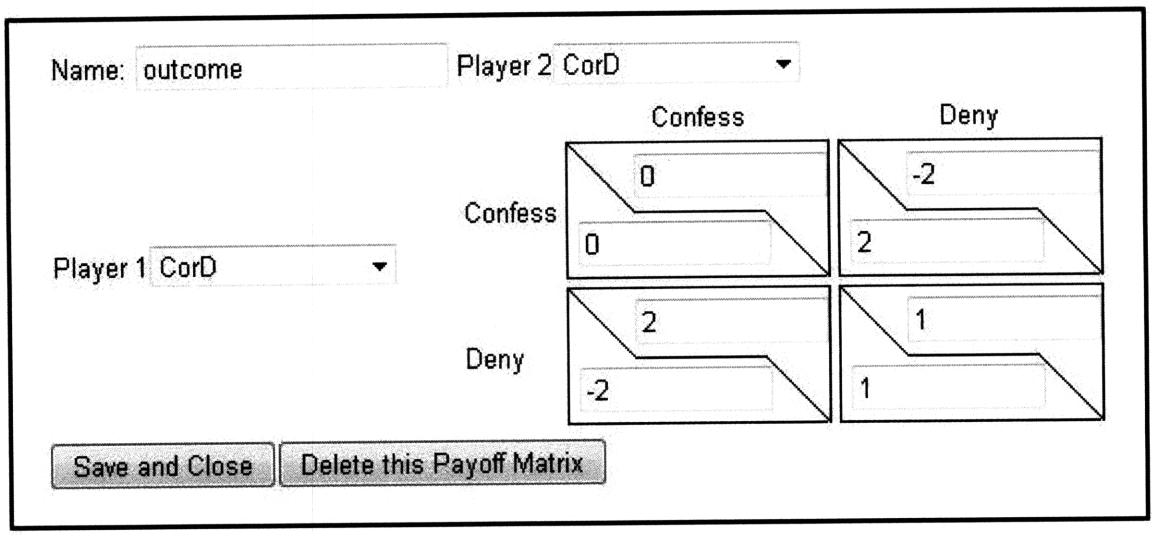

Figure 23 Prisoner's Dilemma payoff matrix

All data is logged to a single table in the database which keeps track of the player, the partnership that player is in, the type of variable (whether it is from a payoff matrix, a button click associated with a screen, entering a new screen, etc.) and the value of the variable, if it takes values ("rock", "paper" or "scissors" for the button click variables. The data that is logged is timestamped.

5.Evaluation

Seaweed has two parts: a design interface for economists, and an interface for players. The two parts are evaluated separately. The design interface is evaluated qualitatively by its target users, experimental economists. The playing interface is evaluated by demonstrating the feasibility of posting a game as a human intelligence task (HIT) on Amazon's Mechanical Turk (MTurk) and getting pairs of workers to play them quickly .

5.1 Designer Interface Evaluation

The usability principles that are most important to the design interface of Seaweed are user control and freedom, efficiency, and learnability. We have already set limits on the type of games the designer can make. However, of the things that the system allows, the designer should have the control they need to make them happen. Game logic should be efficient to specify. Specifically, data input such as payoff matrix output and conditional text should not be redundant to the point of annoyance or causing mistakes. Seaweed uses a visual language that designers are meant to learn by trying and by seeing examples. Some design elements are more obvious than others due to consistency with familiar design-oriented systems. All things considered, we want to be able to assess learnability given minimal instruction.

There are two design scenarios we expect to see. First is the creation of a completely new game. The second is the creation of a variation of a game that already exists. The second case is what we decided to test because it is more common and tasks less time than starting from scratch. We expect starting from scratch to be a very rare scenario because novel game ideas are rare. The economists who gave us input as to what sort of game design system they needed attested to the fact that creating variations to an existing game is more common. Even in the case of making a novel game, we expect users to draw on some existing game structure.

We implemented a version of "Rock Paper Scissors" for the purposes of testing (Figure 24). The game had a Welcome screen in the pre-round section, two screens in the round section, and an exit screen in the post-round section. Within each round, the first screen had an image reminding the player the game was played with three buttons representing the choices "rock", "paper" and "scissors." The second screen displayed the player's own choice, the other player's choice and said whether the player

won or lost, with a button prompting them to play again. The game had a payoff matrix that described who won and who lost for the nine possible combinations of players' actions.

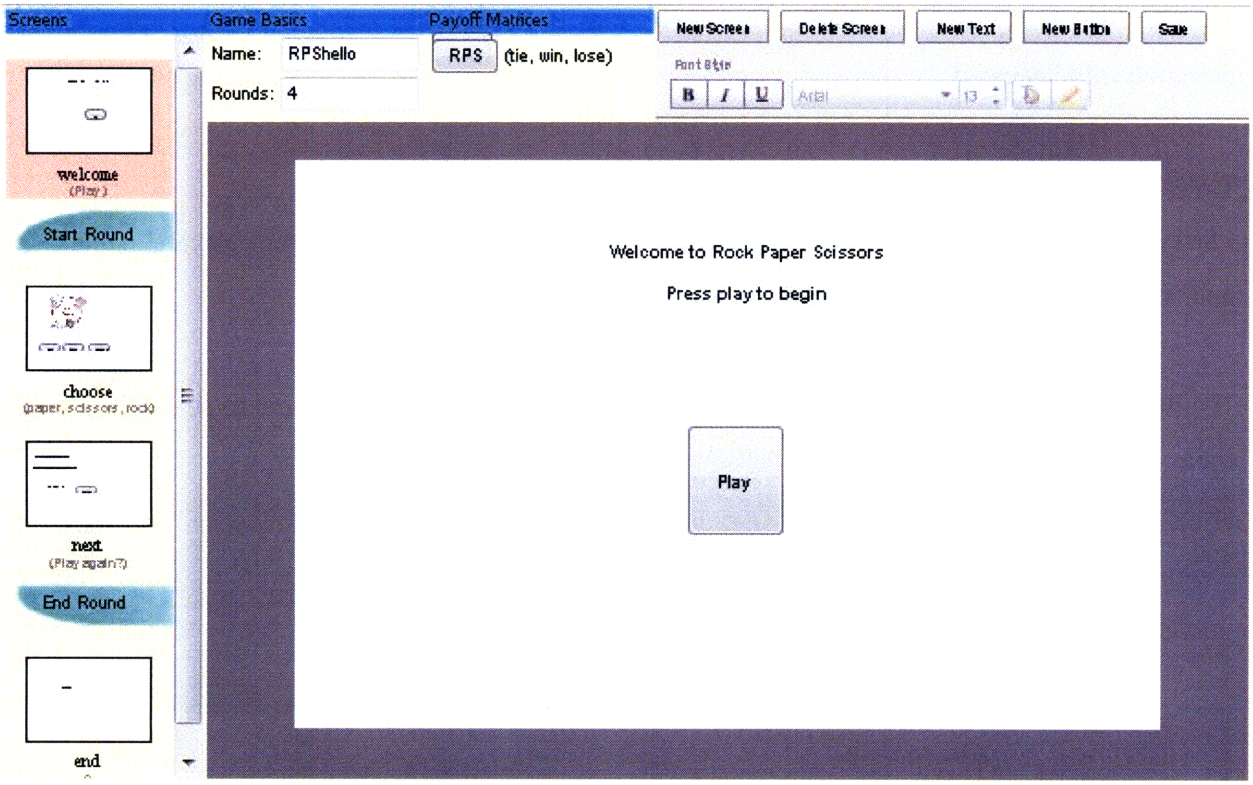

Figure 24 "Rock Paper Scissors" game used for evaluation

The game used for testing was intentionally seeded with two bugs. One bug is that in the event of a tie, the player's screens would say "You lost." The other bug is that every round had a button with the text "Play again?" on it. In the last round, however, even when the player clicked on "Play again?" it would take them to the final screen, rather than giving them a final round.

We had five designers test Seaweed: 1 female, 4 male – all affiliated with the Harvard University Program for Evolutionary Dynamics as either researchers or doctoral candidates. Three designers had no programming experience beyond Excel and command-line statistical packages. One had some Python background and another had background in C doing simulations, but no interactive interface programming. The testers all performed the tasks in Firefox 3 on a Windows Vista PC.

5.1.1 Tasks

Traditionally, in order to evaluate the learnability of an interface, a user is presented with the interface and a set of tasks. In testing Seaweed, we started slightly differently. To introduce the designers to Seaweed, we first had them play both players' roles in the existing game of "Rock, Paper, Scissors" designed with Seaweed.

Task#1: *Play both players' roles in "Rock Paper Scissors"*

I first explained the rules of "Rock Paper Scissors." I then entered two different players' names: "p1" and "p2" and then started the game (Figure 25).

The designers were not told about either of the bugs. I observed the designers to see if they noticed them.

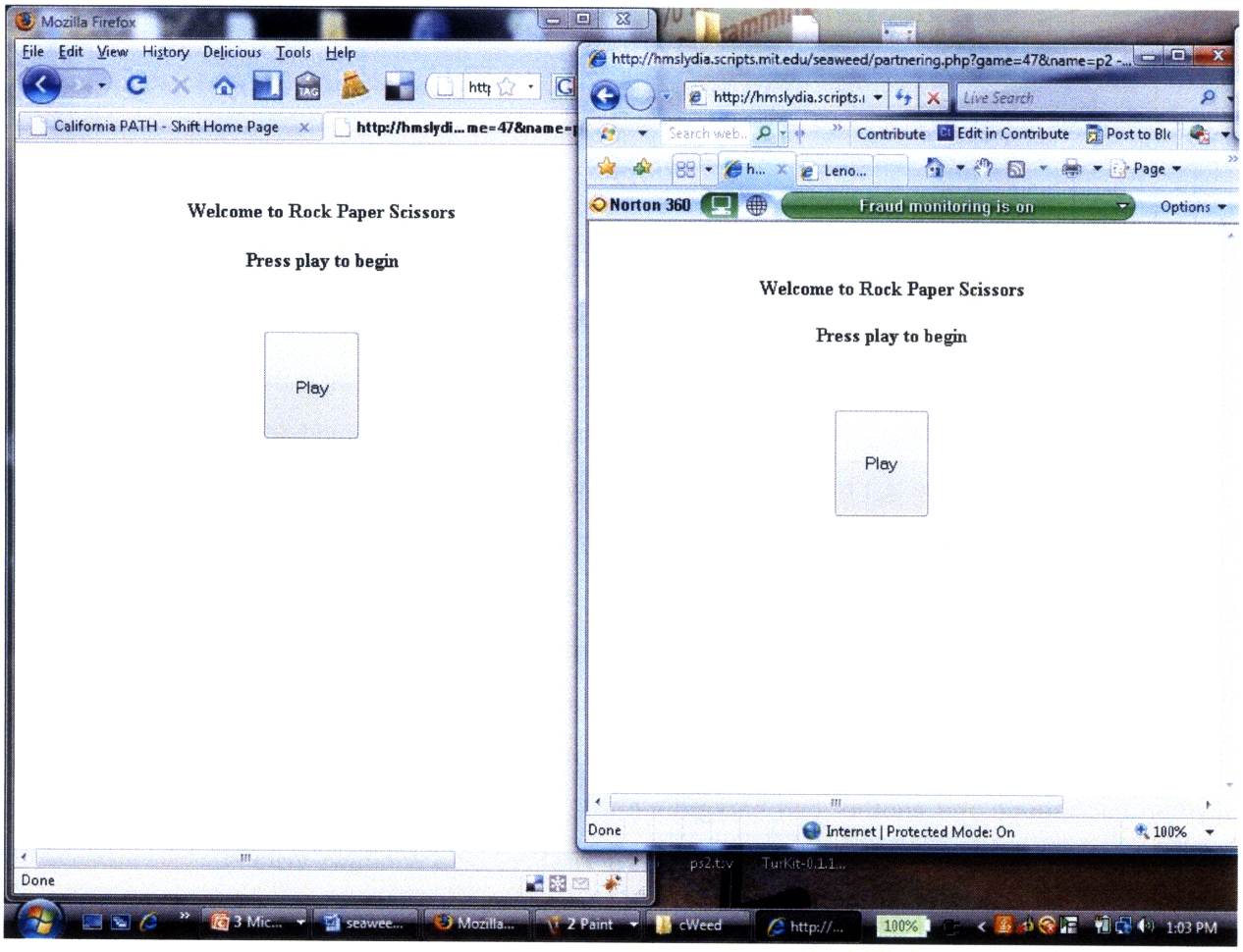

Figure 25. One person playing both players in separate browsers

Task #2: *Explain how "Rock Paper Scissors" was created in the designer*

After playing the game, the designer was taken to the design interface of Seaweed that created the "Rock, Paper, Scissors" game. The designers were asked to describe what elements of the interface were responsible for creating the game they had just played. Designers were allowed to explore any aspect of the interface, including making changes and playing the game with the changes.

After exploring the interface, I explained the aspects of Seaweed that the designer didn't discover. Regardless of what the designers discovered by themselves, I was sure to explain three things:

1. Screens create variables; the buttons on each screen are the possible values of that variable.
2. Clicking on a button automatically advances the player to the next screen.
3. There is syntax for accessing values in the database. I showed them the examples found in the Rock, Paper, Scissors game.

Task #3: *Fix the bug wherein a tie results in the screen displaying* "You lose"

If the designer noticed the bug while playing as both players, they were reminded of that. if they didn't, the bug was explained to them, and demonstrated by playing the game.

There are two places this bug could originate: either the payoff matrix is wrong, or the text "You lose" was displayed under the wrong conditions. In this case, the second is true. The conditional text on the "You win" text object prints "You win" if the payoff matrix results in "win", and "You lose" otherwise. There is no condition for a tie. Logic errors in conditionals are common, and we wanted to test whether Seaweed designers would be able to find the bug and see the flaw in the logic – a potentially difficult task for non-programmers.

Task #4: *Fix the bug wherein the last screen of the last round asks if the player wants to* "Play again?"

If the designer noticed the bug while playing as both players, they were reminded of it. If they didn't, the bug was explained to them, and demonstrated by playing the game.

The intended way to fix this bug was to put a condition on the text of the button to say something other than "Play again?" when the round was equal to the total number of rounds. This was similar to task #3 in that it tested Seaweed's interface for logic at a level that would be accessible to our designers. This task required them to take an object that didn't have a condition and add a condition to it.

Task #5: *Augment the "Rock Paper Scissors" Game to allow for a fourth option*

The fourth option could be called anything and could have any logic attached to it. The intended way to fix the bug is to first add a fourth button to the page, then edit the payoff matrix, which will automatically have a new row and new column added to the matrix for the new variable. This task is the type of change a designer might make to an existing game to test for a different behavior. The task was suggested by an experienced economic game designer who was not one of the test subjects. It is more subtle because it requires two changes: adding a new button on the screen for that option, and augmenting the payoff matrix to say what happens in the new cases just created.

5.1.2. Results of Design Interface Evaluation

All five users were able to complete all the tasks. A common observation was that the user was trying to do the task in a way that the current design of Seaweed did not support, but a future version clearly should support. Only in that case did I intervene and suggest trying something different, fearing that they might get stuck on that task.

Task#1: *Play both players' roles in "Rock Paper Scissors"*

The only real question in this task was whether the designers would notice the two bugs they would be fixing for tasks 3 and 4.

Finding the "You lose" bug is an interesting exercise because it only shows up when two players make the same choice. The designers played four rounds and three of the nine possible combinations ended in a tie, so the odds of encountering the bug were on their side if they were paying attention to the feedback from the game play. All five designers noticed that a tie incorrectly resulted in the "You lose" message, indicating that systematic trial of a game is a debugging approach that Seaweed's designers can make effective use of.

None of the designers noticed the inconsistency of the "Play Again?" button. This is more of a user interface bug than a logical bug. The fact that designers didn't notice it could indicate that they are attending to logical bugs more than user interface bugs even though user interface bugs are arguably just as important.

Task #2: *Explain how "Rock Paper Scissors" was created in the designer*

A number of the elements of the interface were easily identified by all five designers. The analogy to PowerPoint was clear. The representation of screens as the basic unit of design was recognized. The designers found the buttons to create new text or button objects and used them. Designers edited text on text objects and button objects, moved and resized them easily. Designers edited the existing payoff matrix and had a firm understanding of how to enter outcomes for both players for every scenario. They noticed the repetition of the screens in the round section of the screen section. Four of the five designers discovered how conditionals were assigned to text and how to edit, add and delete conditions.

It is not surprising that these elements were easily discovered because they are analogous to systems the designers are familiar with. The layout of payoff matrices is similar to how they are drawn in game theory texts, and the direct manipulation interface of designing screens is analogous to PowerPoint. PowerPoint allows users to create a textbox by clicking and dragging to the desired size. Seaweed creates text and buttons with a single click and many users tried to drag click, and ended up getting a text box, but unfortunately not with the expected position and size. Implementing this behavior more consistently with PowerPoint would correct this problem.

A surprising finding was that none of the five designers made any mention of how any of the events were being handled. My question to the designers was "Explain how this interface made the 'Rock Paper Scissors' game." In their answers, none of them talked about how partnering was done, what made the screens advance, what actions logged data or what triggered the evaluation of a payoff matrix. The fact that nobody mentioned events supports the assertion made in early in the design process that events are too low-level to require designers to specify.

A handful of features were requested by all five users, none of them great surprises. Undo was a popular request, particularly after mistakenly adding a condition to text or accidentally deleting a text or button object. Copy and paste of screens, text objects, and button objects was also requested.

Task #3: *Fix the bug wherein a tie results in the screen displaying* "You lose"

The designers had all discovered the conditional text option and understood the panel it displayed. All the designers were uncomfortable with the syntax they were asked to use to retrieve data from the database, but did understand the underlying principle of making text conditioned on past data in terms of the round, variable and player they wanted data on.

Two of the five users didn't immediately see how to add a new condition with the "Add a condition" button, but did find it on their own after a few moments. One of the five users reserved the "otherwise" condition to display an error. In other words, he added two new conditions, one for "lose" and one for "tie". The text to be displayed in the otherwise case was "Error." This style of programming is useful for debugging, and it was interesting to see a non-programmer use it.

One of the designers actually completed this task without being prompted, starting by modifying the payoff matrix to have 1, 0 and -1 instead of "win," "lose" and "tie." He had programming experience and was accustomed to using indicator variables encoded in low integers rather than strings. He was slightly confused by the order that each conditional was specified. He wanted the condition to come first, then the rich text area, consistent with how most programming languages order an if statement.

Three of the five testers made the change and immediately said something to the effect of "Now I want to test this" presumably because they realize that testing is an essential part of completing a programming task. Those who didn't ask to test their changes were prompted to test it anyway. The designers were successful in finding small logic errors they had introduced and were satisfied that the logic was correct after it tested successfully. It's unclear how comfortable Seaweed designer in general are with testing their changes early and often. Knowing when to test may be something they need help with.

Task #4: *Fix the bug wherein the last screen of the last round asks if the player wants to* "Play again?"
In general, designers found this difficult. Realizing that they had to add a condition to the test was non-obvious. Designers had different strategies for completing this task. Notably, one designer wanted to copy the slides in the round and repeat them in the post-round section and make changes to them there. This is certainly a valid solution. Unfortunately, copy and paste of screens isn't supported, so instead of remaking the screens, I encouraged the designer to look for an easier solution.

One problem was that the designers felt like the right design choice would be to remove the button entirely on the last round. In Seaweed, it is text that is conditional, not the objects, so that design choice isn't possible. Expanding the interface to allow for objects, such as buttons, to be conditional is a necessity. However, if designers had done that in this case, they would have been left with no button on the screen to advance to the next screen. Forgetting to put a button on every screen is a potential problem, but it will almost certainly be realized when testing.

The preferred answer is to set a condition when the round is equal to the number of total rounds (Figure 26). Three of the five designers were reluctant to use the actual number of rounds (which happened to be 4) in the equality test; they wanted to be able to use a variable. Using a variable is the more correct thing to do, and I was surprised that non-programmers would realize that immediately and request it.

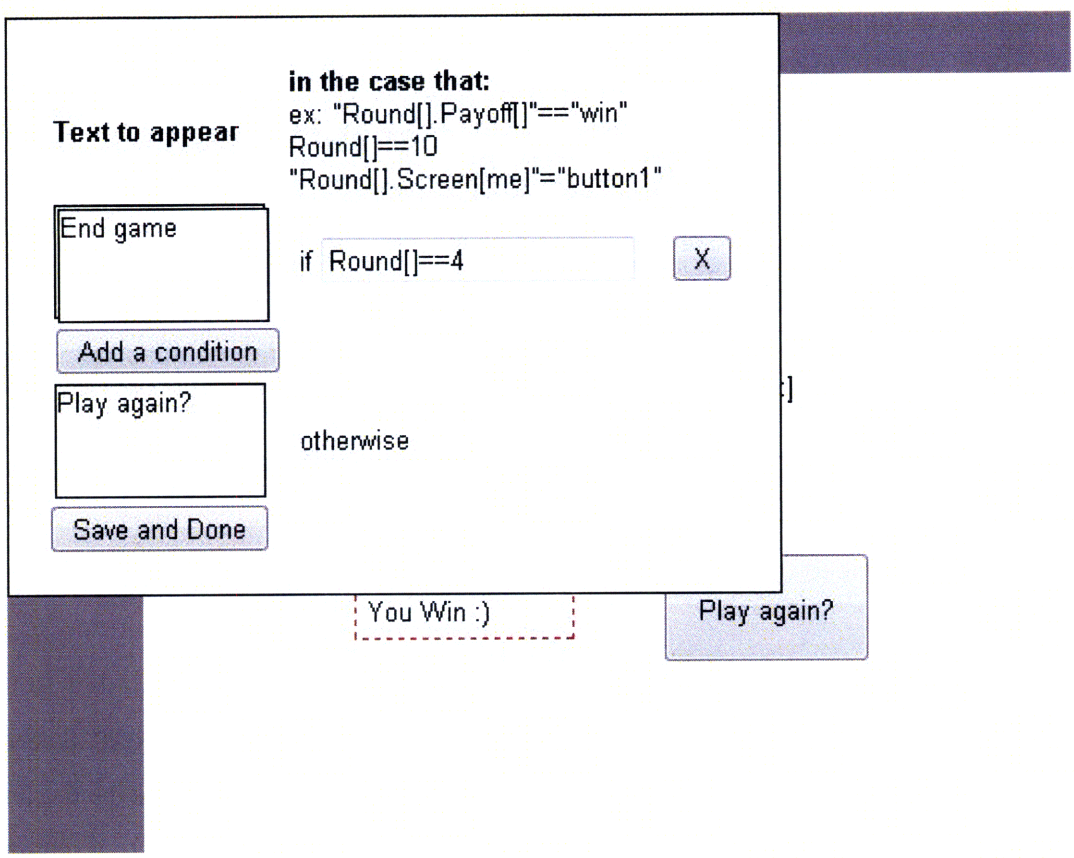

Figure 26 Intended patch for the "Play Again?" bug

Task #5: *Augment the "Rock Paper Scissors" Game to allow for a fourth option*

Two designers completed this task easily. They created a new button, edited the payoff matrix, and then tested their game. The three remaining designers tried editing the payoff matrix first, which doesn't work because the only the currently-specified variable values appear on the axis. They then realized that they needed to add a button before editing the payoff matrix and were able to complete the task successfully, but it raises an interesting design point. Many economists consider the payoff matrix to be the primary definition of the game. The payoff matrix editing interface should allow for adding a variable, which will result in adding a button automatically to the corresponding screen.

Other Observations:

The final design iteration eliminated the variables panel, on the theory that it wasn't providing useful information to designers. Since knowing variable names is still necessary for writing expressions, we listed the names of the variables' values under the screens and next to the payoff matrices. In the evaluation, none of the designers used that information. When asked at the end of the task what the lists were, they had to think before guessing they were the values of the variables. The variable information may be more valuable in a larger or more complicated game, but the results of the evaluation definitely support not having a complex variables panel.

5.2 Game Play Evaluation on MTurk

The ability to put games online and get a large population of subjects from around the world to play them immediately could have a huge impact on how fast economists can get results from their games Moreover, it could drastically improve the the reliability of the results because economists can run much larger studies with subjects who aren't all recruited on a university campus. We want to know if MTurk is a platform where Seaweed can make partnerships and get lots of games run quickly.

Games are a very unusual task for MTurk. The prototypical example of an MTurk task is to be paid 2 cents for looking at an image and writing a few labels for the image. This takes about 10 seconds and can be done at any time by a single person. In contrast, Seaweed games take about a minute and games are preferably done in pairs, although they can be played against the computer if there is nobody else to play with. In order to increase the likelihood of getting pairs, we offer workers 20 cents, which is higher than a typical MTurk task, but the task is longer than normal and high-paying HITs tend to be taken as soon as they are posted, enabling Seaweed to make more partners. To our knowledge, these are the first HITs on MTurk where workers are acting in real-time coordination.

To test the speed and scale of MTurk, we posted 100 HITs asking workers to play five rounds of "Rock Paper Scissors" for 20 cents. After accepting the HIT, players wait up to 20 seconds for a partner. If a partner is found they begin playing against each other, and if not, they start playing against the computer. At no point are the workers told whether they are playing with a partner or a computer. There is a 10 second timeout on every screen. If a player exceeds that timeout without clicking a button, they forfeit the game and get logged out. If their partner exceeds it, the game is forfeited to them and they will get paid for the HIT.

5.2.1 Evaluation of Game Play on MTurk

We posted 100 HITs on MTurk at 1:30pm on Thursday May 7, 2009. From those HITs, 65 partnerships were formed by 31 different workers. Almost all the HITs were formed within 4 minutes of being posted.

Of the 65 partnerships, 34 partnerships finished all five rounds of the game: 22 of those partnerships were with human partners and 12 of those partnerships were with the computer. This demonstrates that human and computer partnerships can be made successfully on MTurk. 14 of the total partnerships were a computer because there was no one available to partner with. The following is a breakdown of the partnering results:

- 65 Total Partnerships:
  - 51 human-human partnerships:
    - 22 completed the game
    - 17 started the game but didn't complete it
    - 12 had at least one partner who never played
  - 14 Human-computer partnerships:
    - 12 completed the game
    - 2 didn't complete the game
  - 34 Completed games:
    - 22 human-human partnerships
    - 12 human-computer partnerships

There were 51 human-human partnerships, which put 102 people in a partnership with another human. Had every worker found a partner and completed the game, there would have been 50 partnerships and 100 attempts at partnerships. Some of the partnerships didn't complete the game and thus attempted but did not complete the HIT, and it was returned for somebody else to take. As it was, there were 116 attempts are partnerships – 14 were with a computer and 102 were with humans.

Although Seaweed made partnerships successfully, it is clear from the data that not all partners successfully completed the HIT. Some workers start but don't finish a game and some workers never play at all. There are several reasons why workers may start but not complete a game. They could take too long making a decision and get logged out. They could decide they don't want to play and leave the game. They could start the game then quit because they don't want to wait for their partner. Their internet connection suddenly gets very slow. They could have walked away from their computer entirely.

The partnerships where one partner never plays could mean one of two things. The first is that a worker could put himself in the queue to find a partner, but then leave before he is told he has a partner. The second is that there could be an error rendering the game for that player. There were 7 instances of workers who were in partnerships but who never played (6 in human-human partnerships and 1 in a human-computer partnership). Two of them were most likely due to the workers' environment – one was playing on an iPod and the other was playing over a slow AOL connection. Seaweed games haven't been tested for any mobile browsers and probably don't work; and Seaweed games are dependent on timing which may be problematic over slow connections.

In "Rock Paper Scissors" on MTurk, we forced workers to wait up to 15 seconds for a partnership. This number was a conservative estimate of how long we would have to wait to find partnerships. Many workers would have had to wait longer than 15 seconds (after 15 seconds they were automatically partners), but in looking at the distribution of waiting time (Figure 27) we see that 46 of the 51 partnerships happened in under 6 seconds. Half of the partnerships (26 of 51) happened in less than 2 seconds.

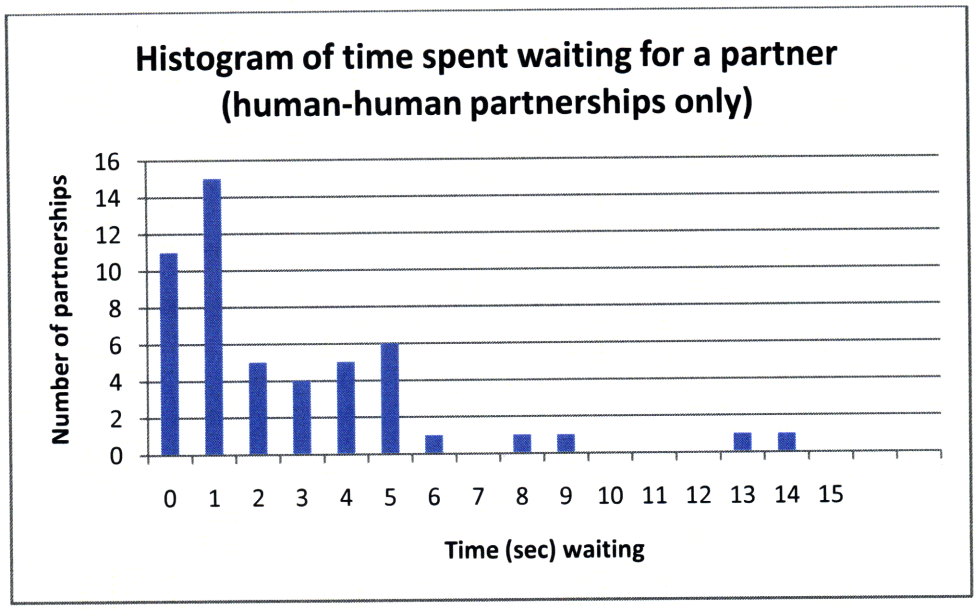

Figure 27 Most human-human partnerships made in under 6 seconds

One might expect that all the fastest partnerships were among the first partnerships. Data on waiting time as a function of how old this HIT is does not complete support that theory (Figure 28). The partnerships are made fairly uniformly across the first 4 minutes of the HIT being posted. After the first 4 minutes, no more human-human partnerships were formed. However, 10 more human-computer partnerships were formed over the next 25 minutes, mostly by one worker in the last 8 minutes (8 HITs each taking 1 minute) (Figure 28).

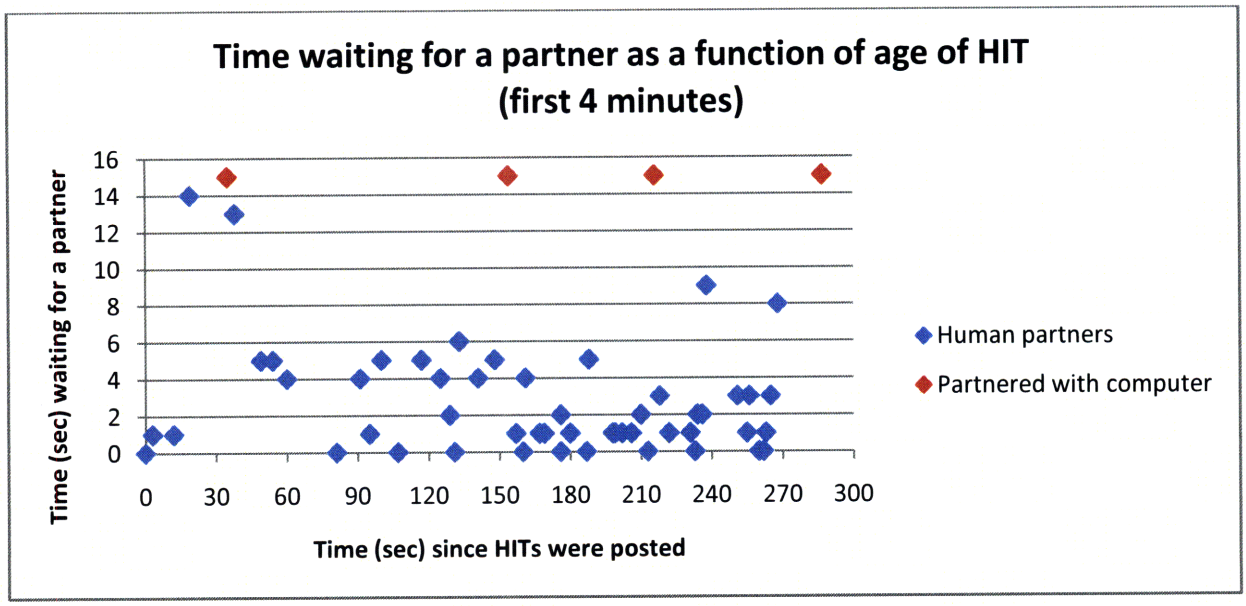

Figure 28 Most of the partnerships happened within 4 minutes of the HITs being posted

Eventually, we would like to know how to post HITs to MTurk in order to maximize the number of human-human partnerships. Several factors might affect the number of people who decide to take

them quickly. The time of day the HIT is posted may affect the available workers. The reward offered to play the game may affect the how many people decide to accept the HIT, and whether the HIT is listed in the 10 most recent HITs may affect how many people see the HIT. The fact that the waiting time for partnerships didn't decrease with time but rather dropped off when there were only 10 HITs left indicates that the number of HITs remaining may affect whether people decide to take the HITs quickly. Workers have a preference for work that they can do repeatedly and a large number of HITs usually means the workers can do a number of them in a row.

6. Conclusion

Seaweed is a domain-specific end user visual programming platform for experimental economists to design and deploy economic games. With Seaweed, experimental economists can quickly create new games and make variations on old ones without having to hire a programmer. They can also pay more subjects to play these games in very short amounts of time because Seaweed deploys the games on the internet using Amazon's Mechanical Turk instead of depending on subjects coming into a computer lab.

The design and implementation challenge in Seaweed is to provide an end user programming environment that creates games responsive to events and controlled by logic without the designer understanding programming concepts such as events and synchronization or being burdened by specifying low-level programming detail. Seaweed achieves this by providing high-level visual representations for variables, control flow, and logic, and by automating behaviors for event handling, synchronization, and function evaluation. Currently, Seaweed can design and deploy two-player, symmetric, games with discrete choices. Using the same general design techniques, Seaweed will be extended to cover more classes of games in future work.

We paid 100 workers on MTurk to play five rounds of "Rock Paper Scissors" with a partner for a total of $20.00. If a partner couldn't be found they would play with a computer. Seaweed made 65 partnerships, of which 22 human-human partnerships completed the game and 12 human-computer partnerships completed the game demonstrating that partners can be found and that a large portion of partnerships last until the HIT is completed. 55 of the 65 games including all of the human-human partnerships were completed within 4 minutes of posting the HITs, demonstrating the high game through-put of which MTurk is capable.

Seaweed makes the following contributions:

1. The design and implementation of a domain-specific end user visual programming system for economists to design and deploy economic games on the web.
2. The design of visual representations of key programming concepts (e.g. variables as screens and buttons, control flow as an augmented PowerPoint slide chooser, and logic as payoff matrices and conditional text).

3. Methods for simplifying distributed game design by automating event handling, synchronization and evaluation of logic.
4. An experiment demonstrating the efficiency in time and money of running partner-based economic games on Amazon's Mechanical Turk.
5. To our knowledge, Seaweed's evaluation demonstrates the first instance of real-time cooperative tasks completed on Amazon's Mechanical Turk.

7. Future Work

Seaweed can be taken in possible directions. The most important direction is to extend the design interface and game engine to accommodate multi-player games, asymmetric games and to provide designs with a variety of input widgets to put in their games such as textboxes, radio button, and sliders. Additionally, to make Seaweed even more powerful, it needs an API for programmers to make truly novel games – games that interface with Facebook.com, games with complex interfaces like Scrabble, and games that make partnerships along the edges of a graph rather than randomly.

Porting the design interface into Adobe Flex would offer performance benefits. The design interface would work in any web browser and games could be played in Flash, eliminating the need for clients to constantly pull data from the server to find out the status of their partner. If mobile phones continue to move toward supporting Flash, Seaweed games will then also work on mobile browsers.

Finding the best way to make partnerships on MTurk is still an open problem. One could experiment with the amount of time to wait for a partner, the reward for the hit, the time of day the HIT is put on MTurk and the number of HITs in the group. One could also try a more structured approach to creating HITs by potentially announcing a time for workers to come back and find a partner.

Lastly, Seaweed could be extended to make interfaces for fields other than experimental economics. Researchers in games as they pertain to law, negotiation, and marketing have already expressed interest in Seaweed. Some of Seaweeds techniques for end-user programming could probably be applied to fields other than economics – the consistency with PowerPoint and Excel for example would transfer to most fields. However, Seaweed also has payoff matrices – a visual representation of logic drawn from economics – which we only evaluated with economists. Lawyers, negotiators and marketers may need different visual representations drawn from their field.

8. Bibliography

[1] L. von Ahn and L. Dabbish, "Designing Games with a Purpose". CACM Vol 51, No.8, August 2008, pp. 58-61.

[2] L. von Ahn and L. Dabbish, "Labeling images with a computer game". Proc. of CHI 2004, pp. 319-326.

[3] L. von Ahn , R. Liu , M. Blum, "Peekaboom: a game for locating objects in Images", Proceedings of the SIGCHI conference on Human Factors in computing systems, April 22-27, 2006, Montréal, Québec, Canada

[4] Antoniotti, M. and Göllü, A. 1997. "SHIFT and SMART-AHS: a language for hybrid system engineering modeling and simulation." In *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (Dsl), 1997* (Santa Barbara, California, October 15 - 17, 1997). USENIX Association, Berkeley, CA, 14-14.

[5] H. Bernhard, U. Fischbacher and E. Fehr; "Parochial altruism in humans." Nature 442, 24 August 2006, 912-915.

[6] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. "Automation and Customization of Rendered Web Pages." *UIST,* 2005, pp. 163-172.

[7] Cypher, Allen. "Bringing Programming to End Users". In Cypher A, ed. *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge MA, 1993, pp. 1-11.

[8] A. Cypher and D.C. Smith. "KidSim: End User Programming of Simulations." CHI 1995, pp. 27-34.

[9] A. Dreber, D.G. Rand, D. Fudenberg, and M.A. Nowak. "Winners Don't Punish," *Nature*, 452, pp.348-351.

[10] U. Fischbacher, "z-Tree: A Toolbox for Readymade Economic Experiments-Experimenter's Manual," 1999. University of Zurich Working Paper No. 21.

[11] U. Fischbacher, *z-Tree: Zurich Toolbox for Ready-made Economic experiments, Experimental Economics*, 10(2), 2007, pp.171-178.

[12] K.J. Haley, and D.M.T. Fessler, "Nobody's watching? Subtle cues affect generosity in an anonymous economic game", Evolution and Human Behavior, Vol. 26, No. 3., May 2005, pp. 245-256.

[13] K. Kahn, "ToonTalk – An Animated Programming Environment forChildren." Journal of Visual Languages and Computing, 1996. 7(2): pp.197-217.

[14] A. Monroy-Hernandez and M. Resnick. "Empowering Kids to Create and Share Programmable Media." Interactions March-April 2008, pp. 51-53

[15] J. Pane, *A Programming System for Children that is Designed for Usability*, PhD dissertation, Carnegie Mellon University, May 2002, 258 p.

[16] A. Repenning and T. Sumner. "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages." IEEE Computer, Vol. 28, pp. 17-25,1995.

[17] C. Scaffidi, M. Shaw, and B. Myers. "Estimating the Numbers of End Users and End User Programmers." VL/HCC, 2005, pp.207-214.

[18] D.P. Tracer, "Market Integration, Reciprocity, and Fairness in Rural Papua New Guinea: Results from a Two-Village Ultimatum Game Experiment."*Foundations of Human Sociality,* 2004, pp. 232-259.