Asymmetric Interface for Seaweed System

Paige Phillips

December 9, 2009

1 Introduction

Seaweed is a web application in which economists can build simple games to post to the web in order to obtain research data. However, this system was lacking the ability to create asymmetric games. For my Undergraduate Advanced Project, I have implemented an extension to the Seaweed interface to allow designers to create this class of games (and, of course, modified the game engine so that these games can be played).

2 Background

2.1 Game Theory

So far, Seaweed has primarily been used to gather data for game theory research. Originally, Seaweed only supported symmetric games - that is, games in which both players are presented with the same choices for the same potential payoff. To illustrate, the Prisoner’s Dilemma is a classic example of a symmetric game. In the Prisoner’s Dilemma two prisoners who are being held separately by the police (regarding a crime in which they both participated) are given the opportunity to confess [4]. They are told that:

- If only one prisoner confesses, the confessor will go free as a reward for cooperation and the other will face the entire 10-year sentence (or you could say that the confessor receives 5 points and the other receives 0).
• If both confess, they both serve five years (both get 3 points).

• If both deny, they both serve one year (each gets 1 point).

In this game, each player is given the same choice: to confess or deny. Each player has the same potential payoffs for their choice: no sentence, a five-year sentence, or a one-year sentence.

The symmetry is especially obvious when you look at the potential choices and payoffs in the form of a payoff matrix. For example, the payoff matrix for Prisoner’s Dilemma might look like this:

<table>
<thead>
<tr>
<th></th>
<th>Deny</th>
<th>Confess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deny</td>
<td>(3, 3)</td>
<td>(0, 5)</td>
</tr>
<tr>
<td>Confess</td>
<td>(5, 0)</td>
<td>(1, 1)</td>
</tr>
</tbody>
</table>

Note that both the choices (along the top and left side) and payoffs (within each cell) are symmetric.

Asymmetric games, on the other hand, present different players with different choices; in asymmetric games, a player’s strategy depends on which role he is assigned. For example, in the game of Bully, one player is be afraid of mutual defection but interested in defecting otherwise, and the other always prefers to defect [3]. In this case, the players’ strategy clearly depends on which role they play.

However, Bully is uninteresting from the standpoint of Seaweed’s user interface design. Although it is technically asymmetric, as you can see from the payoff matrix below, it is not visually asymmetric from the viewpoint of the player; the payoffs in the payoff matrix may be asymmetric, but the choices are the same.
The sort of games that are interesting are games in which the choices, rather than the payoffs, are asymmetric. Seaweed has supported arbitrary payoff matrices for some time - but not arbitrary choices.

As an example of this kind of interesting game, consider the ultimatum game, where one player proposes a division of resources (perhaps they have the options to give nothing, half, or everything), and the other can accept (both players are rewarded according to their agreement) or deny (neither player gets anything)[2]. In this case, the choices each player is offered are different depending on which role the player is assigned - this kind of game requires a significantly different user interface to build.

2.2 Seaweed

The Seaweed interface presents a challenging problem, because we cannot assume that the users have any experience with programming. However, we have successfully provided designers with the tools to build games controlled by logic and responsive to events; this involves a combination of good design and the judicious use of metaphors they are already familiar with (for example, making the interface resemble Microsoft PowerPoint)[1].

Because of these challenges, Seaweed is itself interesting from a user interface standpoint. As Chilton puts it, “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.” [1] Designers need not implement event handlers, nor concern themselves with synchronization; Seaweed’s game engine takes care of ensuring that players are synchronized when necessary (that is, when what one player sees depends on the results of one of the other player’s actions).
3 Design

The main alteration to the existing UI (see Fig. 1) was the addition of a mechanism to the screen preview sidebar on the left to allow for role selection (i.e., which player role would see the screen).

![Original Seaweed interface](image)

Figure 1: Original Seaweed interface

The first design decision to be made at this point was how to trigger the switch into asymmetric mode. The original paper-prototyped design simply had a “Role” button in the top menu bar, alongside Screen, Payoff, Input, etc. However, user testing revealed this to be difficult to find. The better design turned out to be placing the mode-switch button adjacent to the area it affected (the screen preview/organization sidebar) to make it more discoverable; the revised design also changed the button text to “Add a role” (or, after switching to asymmetric mode, “Go to single-role view”). See Fig. 2 for details.

The asymmetric interface simply adds a role chooser to each screen in the sidebar (see Fig. 3). For consistency’s sake, when initially switching to asymmetric mode, the interface
Figure 2: Designer’s default blank game
defaults to selecting both roles for each screen. This keeps the game in an equivalent state on role switch. However, existing role information is maintained on subsequent switches - if the user switches to symmetric mode and back, their original role choices will be preserved through the switch.

Figure 4 shows a completed asymmetric game (a variant on the Trust Game).

The asymmetric game interface also allows users to rename the roles, as shown in Figure 5.

Various other features of the Seaweed interface also had to be updated for consistency. For example, the “me” and “opponent” terminology in the original Data Builder interface (see fig. 6) doesn’t necessarily make sense in an asymmetric game. Among other things, it could cause the designer to build incorrect data expressions; in asymmetric games, there isn’t necessarily a “me” or “opponent” for every screen, if only one role ever sees it. The system could determine “me” to be the role of the screen being edited (and this clearly has
Figure 3: Designer immediately after clicking “Add a Role”

Figure 4: A completed asymmetric game
flaws for screens seen by both roles) and then only show valid choices in the drop-down box, but that would still require the user to recall which role corresponds with “me”/“opponent”. Out of all the possible options, replacing “me” and “opponent” with a list of roles that are valid for that variable seems to be the clearest, but it has drawbacks as well - it doesn’t necessarily work for screens seen by both roles. This is an issue that will take further study to solve; in the meantime, the system is using the role-only solution (see fig. 7).

Unfortunately, due to an oversight in paper prototyping, the necessary Data Builder updates were overlooked until a rather late date. Because I was only testing a small section of the Seaweed interface, I chose to simply hand the user pre-made “screens” rather than requiring them to create the entire game from scratch in the prototype. Their task was, then, to organize the screens correctly and to assign them the correct role, so that each player in a pair would see the appropriate interface. Unfortunately, testing simply on screen organization didn’t catch many of the peripheral consistency issues.

This design was actually one of two that were originally prototyped; the other alternative,
which the author initially believed was more intuitive, was a more graph-structured interface (see Figure 8). Unlike the other design, it explicitly showed the user all available information on timing and synchronicity in something much closer to a timeline view. However, despite extra visual cues such as color coding, user testing revealed it to be needlessly complicated, and ultimately less clear than the simpler design. It also had the drawback of making it much easier for users to put the game they were working on into a nonsensical state. (Of course, validation would catch this, but it is a poor alternative to a user interface that does not imply to the user that these nonsense states are valid.)
4 Implementation

4.1 Designer

4.1.1 Model

In order to maintain state about role names, I created a Role model. As Seaweed is only planning to support two-player games for the foreseeable future, I simply gave each Game two Role models. Each Screen, then, has a boolean corresponding to each of the game’s Roles, to indicate whether or not that screen is intended to be seen by said Role. (See Fig. 10; a diagram of the old version of Designer is included in 9 for comparison.)

As well as two Role objects, Game also acquired a mode flag to indicate whether or not
the interface is in asymmetric mode. I chose this because it meant that information was never lost on mode switch (the other possibility would be treating a symmetric game as a special case of an asymmetric game, and so mode switching would involve modifying the screens' role markers).

4.1.2 View

I simply added an extension to ScreenThumbnail to add a RoleChooser object. RoleChoosers allow for changing of role names as well as role selection; to accomplish this, each RoleChooser has its nameA and nameB property bound to the name property of the Game’s Role objects. Likewise, the “selected” property of the RoleChooser checkboxes is directly bound to each Screen’s role selection boolean.
Figure 9: Diagram of original Designer structure

Figure 10: Diagram of new Designer structure
4.2 Backend support

When saving and loading games, it was obviously important to support saving and loading of Role information. To accomplish this, I modified the database schema, and the game saving and loading code.

In the database, I added two tables: the role table, and the roleScreen table. The role table simply stores the Seaweed-generated ID of each Role object, the ID of the game it belongs to, and its name. roleScreen stores only a mapping between a role ID and a game ID.

On game load, the roleScreen and role tables are queried; the newly created Game’s Role objects are initialized to values supplied from the role table, and an iteration over roleScreen sets all of the game’s Screens’ role flags appropriately.

Game saving involves approximately the same procedure, in reverse. However, the saved game data is used by the GameEngine as well as Designer. When executing games, the GameEngine checks the controlFlow table to determine which screen to retrieve next. I decided that the most straightforward way to make this work was to simply build separate controlFlows for each role at the same time as the symmetric controlFlow (necessary for laying out screens in Designer) was being built; the items the asymmetric versions contain are a subset of the asymmetric controlFlow, and the resulting entries in the DB’s controlFlow table are tagged accordingly with either a role or NULL.

4.3 GameEngine

The GameEngine required very little modification. When playing an asymmetric game, the user will be assigned a role on partnering - either role A or role B, depending on whether or not the player is the first or second member of the two-player partnership (only size-2 partnering groups are currently supported for asymmetric games).

Then, when asking for the next screen of an asymmetric game, the GameEngine sends
the player's role to the database as well, and the controlFlow entry for the next screen corresponding to that role is retrieved.

See figures 11, 12, 13 for an illustration of an asymmetric game in action. This is the same variant on the Trust Game shown in figure 4.
Figure 11: Playing an asymmetric game
Figure 12: Playing an asymmetric game

5 Conclusion

5.1 Issues

The problems of validation and deadlock are issues that have not been addressed in this interface. For example, although the Data Builder now disallows invalid player selections, user-entered expression syntax isn’t checked for such inconsistencies. Another problem could arise when a screen is waiting on data that will never be logged, or not be logged until after that screen has been displayed. Although lack of validation is currently a weakness of Seaweed as a whole, the extra constraints that asymmetric games require make validation an even more pressing concern.

I was also unable to find any really interesting or complex asymmetric games in the
literature. I thought that the users might want to build games with players in independent loops, or with complex flow control between roles, but all of the model asymmetric games I found were far less complex. I’m afraid that the demands of more complex games may reveal flaws in my design that I was unaware of; on the other hand, perhaps all of the games in the literature are simple because games that are scientifically interesting just don’t have to be complex (in which case, it’s a lesson on making assumptions about what users want!)

5.2 Future Work

Whenever Seaweed supports N-player games, an asymmetric interface that can build games with N roles would be an obvious extension of this project. As mentioned previously, tools (such as role reversal) for manipulating role assignments while playing may also be worth-
while. Extending this interface to support asymmetric games on graphs could also be an interesting addition to develop.

References


