NORTHERN ARIZONA UNIVERSITY
The W. A. Franke College of Business

Using *z-Tree* for a Non-Interactive Accounting Experiment *

Working Paper Series—10-05 | April 2010

Tom Downen
Assistant Professor of Accounting
Northern Arizona University
P. O. Box 15066
Flagstaff, AZ 86011
(928) 523-8522
Tom.Downen@nau.edu

* This paper is derived from the final chapter of my dissertation, completed at Texas Tech University. I appreciate greatly the patience and guidance provided by my dissertation committee members: Steve Buchheit (chair), Ralph Viator, Penelope Bagley, and Peter Westfall. This paper has also benefited from many helpful comments provided by Beau Barnes, Derek Dalton, Dawn Fischer, Francesca Flores, Nancy Harp, Becky Hyde, Gregg Murray, Susan Murray, Marc Ortegren, and Ming Zhou. Additionally, I thank Zafar Miller at Texas Tech University and Brandon Jones and Damien Plunkett at Northern Arizona University for providing technical assistance. Theresa Stacy-Ryan at Northern Arizona University should also be recognized for her extensive graphic design assistance. I also thank an anonymous reviewer and participants at the 2010 Conference of the Southwest Region of the American Accounting Association for helpful feedback.

Using *z-Tree* for a Non-Interactive Accounting Experiment

I. INTRODUCTION

Behavioral researchers in accounting have long faced challenges in how to best present experimental materials to subjects and to collect subject responses.[1] As technology has advanced in recent decades, more researchers are shifting away from pen-and-paper instruments in favor of computerized experiments. There are a number of potential advantages of using computerized experiments, including: (1) reduced data entry effort for the researcher, (2) less risk of data entry errors, (3) greater flexibility in the layout and presentation of materials, (4) more efficient interaction between subjects, and (5) more timely performance feedback. Although the study described in this paper (from Downen, 2010, hereafter referred to as "my study") does not involve interaction between subjects, it does benefit in the other noted ways.[2]

Although *z-Tree* (Zurich Toolbox for Readymade Economic Experiments) was primarily designed for use in economic experiments, it has also been used for behavioral research in a variety of other disciplines, including accounting (Bowlin, 2008; Bowlin et al., 2009; Christ, 2008; Hales and Williamson, 2009; Newman, 2009), political science (Dickson et al., 2008), finance (Haruvy and Noussair, 2006; Dittrich et al., 2005), and marketing (Engelbrecht-Wiggans et al., 2007). One primary advantage of using *z-Tree* is set forth early in Fischbacher's tutorial, with his remark that "an experimenter with a certain amount of experience can program a public goods game in less than an hour and a double auction in less than a day" (Fischbacher 2002, p 4). Although my study is neither a public goods game nor a double auction, *z-Tree* proved to be extremely easy to use for even a relatively inexperienced designer. From its inception, my experiment was functional within a few days and complete and ready to be administered within about a week. Just entering the data associated with my study may have taken longer than a week, if it had been administered using pen-and-paper methods, and would have introduced much greater risk of error.

This paper provides a step-by-step description of the process used to program a non-interactive accounting experiment using *z-Tree*. It provides some basic coding examples and various tips and suggestions. The primary focus of this paper, and of *z-Tree* in general, is developing quick and easy-to-use experiments. Other software options, such as linked workbooks in Microsoft Excel or custom web-based programming, likely offer greater aesthetics and some improvements in functionality, but also usually require considerably more time in development and maintenance.[3]

The next section of this paper describes general processes for getting started with using *z-Tree*. The third section describes some basic program components, and the fourth section provides custom program elements. The fifth section describes unique and specific *z-Tree* coding examples and the sixth section concludes.

---

[1] Although it is not acceptable protocol anymore to refer to research participants as subjects, that label is used throughout the English version of *z-Tree* and its supporting materials. Therefore, for consistency purposes within this paper, I too will use the term subjects rather than participants.

[2] Though such functionality was not used in my study, *z-Tree* does have the capability for interacting subjects, including the development of small economic markets.

[3] My experiment uses version 2.1.4 of *z-Tree*. A more recently released version of the software, version 3.3.6, claims to include increased aesthetical options and graphical reporting. However, the supporting documents (tutorials and reference manuals) for the newer version are very limited and my initial attempts to use the newer version of the software resulted in what seemed to be frequent bugs or errors.

II. GETTING STARTED

The *z-Tree* software is free to download from the University of Zurich. All that is required is that users sign a license agreement with terms of use that include providing the appropriate reference in any papers generated from a *z-Tree* experiment. The specific reference required is as follows: "The experiment was programmed and conducted with the software *z-Tree* (Fischbacher, 2007)". The license agreement can be downloaded at http://www.iew.uzh.ch/ztree/howtoget.php. Additional *z-Tree* related materials are available from the *z-Tree* home website at http://www.iew.uzh.ch/ztree/index.php.

Once the license agreement form has been completed and signed, it must be mailed to Zurich for approval. When it arrives in Zurich and has been approved, the *z-Tree* administrators send the requester an e-mail message with a user ID and password for downloading the software. Using standard mail, it took approximately two weeks for my license agreement to get from the central United States to Zurich and to be processed. The *z-Tree* administrator also signs the license agreement and returns a copy of it to the requester by mail.

When the download instructions have been received by e-mail, the process to download the software is very straightforward. *z-Tree* is a client / server system, with *z-Tree* being loaded on the server and *z-Leaf* being loaded on the client computers. For testing purposes, as described in more detail later, it may be useful to download both *z-Tree* and *z-Leaf* onto the primary researcher's computer during program development. *z-Tree* also now has the capability of online administration of experiments, with a stable IP address on the server side. However, the process described here used a laboratory setting, with a traditional client / server framework.[4]

III. BASIC PROGRAM SETUP

z-Tree was developed in Switzerland, where a dialect of German is predominant. *z-Tree* can be converted to a variety of languages, but German is the default. To use *z-Tree* in English, the following command must be added to the target field in the properties option for the *z-Tree* shortcut: C:\zTree\zTree.exe **/language English**. If *z-Tree* is not directed to use English, then any server / error messages that subjects see will be in German and are likely to be very confusing. The same syntax also should be added to the run command for the *z-Leaf* program on the client computers.

There are various level descriptors in *z-Tree*, including sessions, treatments, periods, and stages. An experimental study often involves multiple sessions, though each session is usually identical except for the actual subjects involved. My study involved four sessions, with essentially no differences between them except for the subjects participating. The lab being used for my experiment had limited capacity, which required multiple sessions in order to include all planned subjects.[5]

A treatment typically involves multiple periods. For a repeated measures study such as mine, similar periods are grouped together into treatments. Within a treatment, there are not typically any changes to the manipulations for a specific subject. Instead, a treatment simply involves repeating the

[4] When using a computer lab to administer a *z-Tree* experiment, be aware that administrator authority is often required for downloading executable programs. Please be sure to allow sufficient time and planning to get *z-Tree* loaded onto the lab server and *z-Leaf* loaded onto all of the subject computers in the lab. Also, depending upon the setup of the lab computers, it may be necessary to add the following command to the run procedure for *z-Leaf* on the client computers: **/server IPADDRESS** (where IPADDRESS is replaced by the actual IP address for the server). This allows the client computers to recognize the server and to connect with *z-Tree* on the server. Please also be aware that certain firewall and security blocks may need to be softened to allow for proper connections between clients and the server.

[5] Also, as described in more detail later, there can be time advantages associated with administering a *z-Tree* experiment using several smaller groups of subjects. *z-Tree* games require that all subjects in a session proceed at the same pace. Therefore, if the group of subjects is overly large, then one or two slow responding subjects can unnecessarily use up a lot of time for many other subjects.

same general task over multiple periods. When a manipulation change for a specific subject is needed, a new treatment is usually established. In my study, there were two treatments, each of which corresponded to a different module of the study. My two modules differed from one another with regard to cost structure (one within-subjects manipulation). Between-subjects manipulations are best handled within the treatment, unless subjects from different conditions will not be completing the experiment at the same time.

A period can be divided into stages, with the most common stages being a decision stage and a reporting stage. However, *z-Tree* allows multiple decision stages and multiple reporting stages within the same period, depending upon the structure of the game. My experiment included, for each period, an introduction stage, a decision stage, and a profit display stage. Input variables can be included in any or all of these stages, though they are most common for the decision stage. Table 1 provides a brief summary of these levels and their application in my study.

Table 1. Levels Used in *z-Tree* and Sample Applications

z-Tree Level	***Description of Application for My Study***
Treatment	A single game module (out of two), where multiple repeated-measures periods are completed and various within-subject manipulations are shifted.
Period	A single decision round, including several decision variables, after which feedback is provided.
Stage	A component of a period, examples of which are the introduction stage, the decision stage, and the profit display stage.

The first step in designing a *z-Tree* experiment is the same as with any other programming project: PLANNING! A little extra time spent planning the design upfront will save a lot of time in making revisions later. In addition to thinking about the treatments and stages that will best accommodate the research needs, it is worthwhile to give thought to the most appropriate presentation of information on the screens and any peripheral information you might like to collect during the game.

Figure 1 provides a screen display of a new *z-Tree* treatment. It shows the primary screens (the ACTIVE SCREEN and the WAITINGSCREEN) and the primary tables (GLOBALS, SUBJECTS, SUMMARY, CONTRACTS, and SESSION) used in *z-Tree*. Esarey (2005) provides a good summary of the structure and content of each of the *z-Tree* tables:

- GLOBALS table – holds variables that will be the same for all subjects but may differ between periods
- SUBJECTS table – holds variables that may be different between subjects and between periods
- SUMMARY table – like the subjects table, but used to display running statistics of an experiment for the experimenter on the screen
- CONTRACTS table – holds buy and/or sell offers in an auction
- SESSION table – holds variables that may differ across subjects but persist across treatments

As these descriptions suggest, most of the data used in the program, for displaying to subjects and for collecting responses, is stored in the SUBJECTS table.

Double-clicking on the BACKGROUND option allows the researcher to establish several important parameters, as shown in Figure 2. Those include identifying the number of subjects, the number of groups (for interactive games), the number of practice periods, the number of paying periods, the exchange rate (from experimental currency units to Swiss francs or other appropriate currency), the lump-sum payment, and the show-up fee. Depending upon the nature of the study, some of these parameters may not be relevant or deserving of consideration.

One available option included in the BACKGROUND is to establish or modify BANKRUPTCY RULES. If the game being programmed includes the prospect of losing experimental currency in any particular period, then it may be wise to establish appropriate BANKRUPTCY RULES. Otherwise, for example, if a subject loses money in the very first period, then he or she could end up with a negative

Figure 1. *z-Tree* Screen Display for a New Treatment

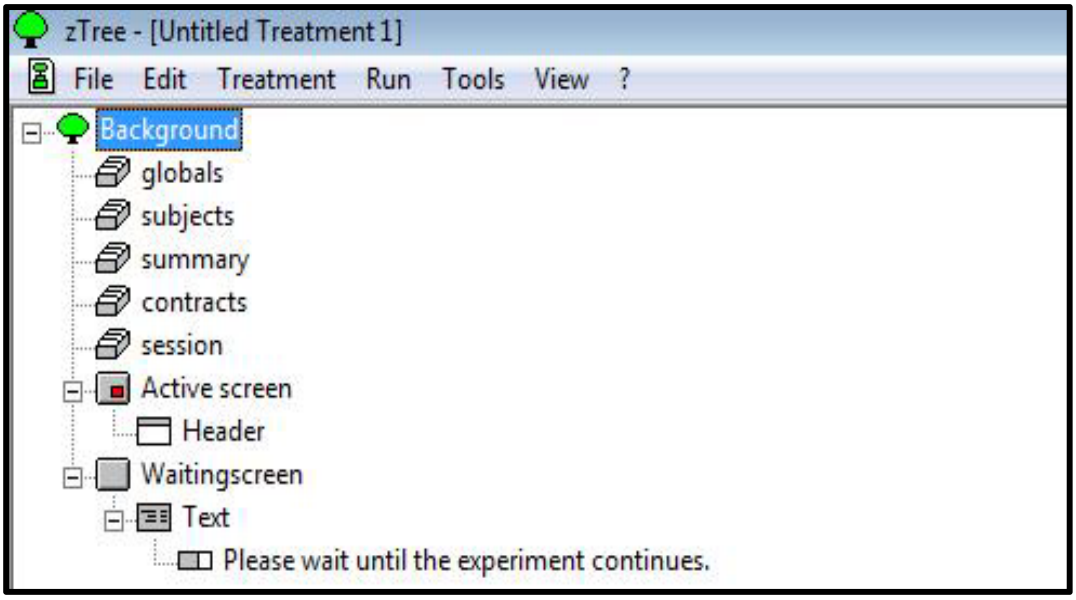
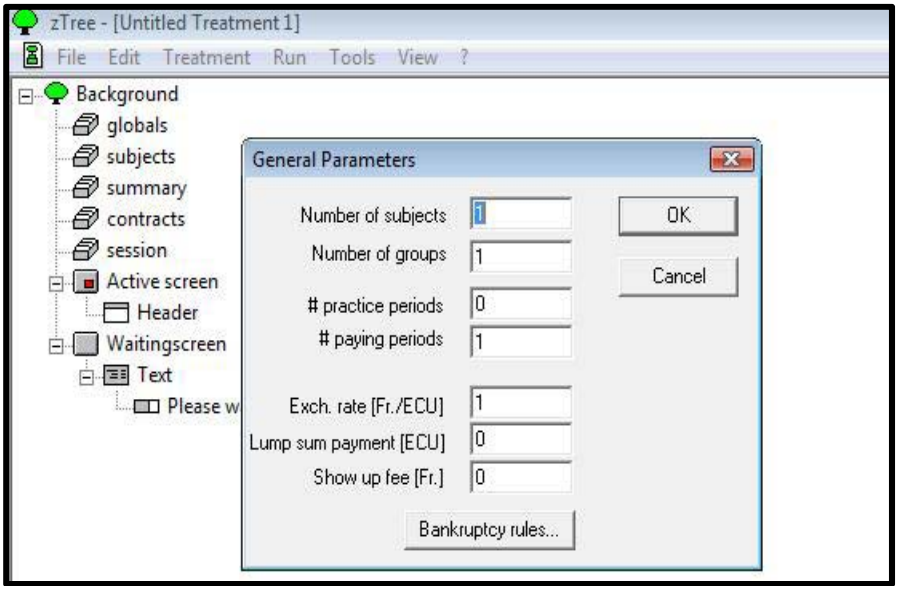


Figure 2. *z-Tree* Screen Display for Background Parameters



balance and *z-Tree* will not know how to proceed. Use of the BANKRUPTCY RULES allows the researcher to indicate whether a subject with a negative balance will be allowed to continue in the experiment, perhaps by applying part of the lump-sum payment or the show-up fee to cover the deficit.

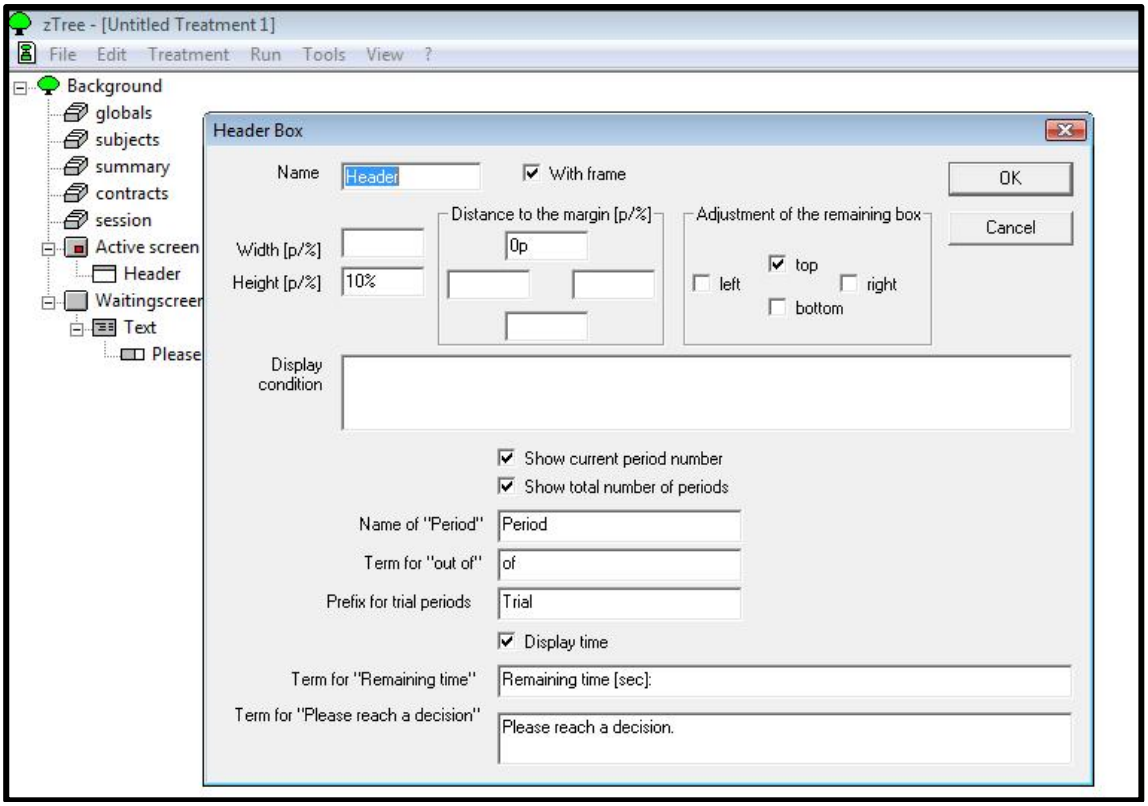
Programs, discussed in more detail later, can also be added to the BACKGROUND. The advantage of using programs to define variables and assign values in the BACKGROUND is that they remain defined as such throughout all stages of a particular period. If variables are instead defined in one of the stages, they cannot generally be used (without repeating the definition program) in other stages.

The HEADER option allows the user to define the layout of the top part of the screen that appears for subjects throughout most of the game. Figure 3 provides a screen display of the HEADER options. The options that are likely most relevant are the name used for periods (in my study, I referred to periods as towns) and the message to be displayed to subjects when their time has expired and they have not yet provided a response. There are also options for screen / box placements and configurations that may be helpful for presenting information in the most organized manner.

Variables in a *z-Tree* treatment can only contain numeric values. Therefore, decision variables that involve responses such as "Yes" or "No" or other categories must be treated like dummy variables, where numeric values are assigned to each categorical choice. For collection of long-answer data such as would be included in surveys, *z-Tree* has a questionnaire function that is outside of a treatment. The questionnaire follows immediately after the completion of a treatment.

Various items can be added to the background and/or the screens of a treatment. Those items include programs, text strings, data input screens, variable values, and buttons, among others. Programs and data input screens are discussed in more detail on the pages that follow. Text strings are informational items presented to subjects, with or without variable values included. Buttons can be used to complete a stage, with a label such as "OK". To provide for a more aesthetic presentation of information in my study, with the older version of *z-Tree*, I used a variety of rich text format (RTF) options in my item coding.

Figure 3. *z-Tree* Screen Display for Header Options

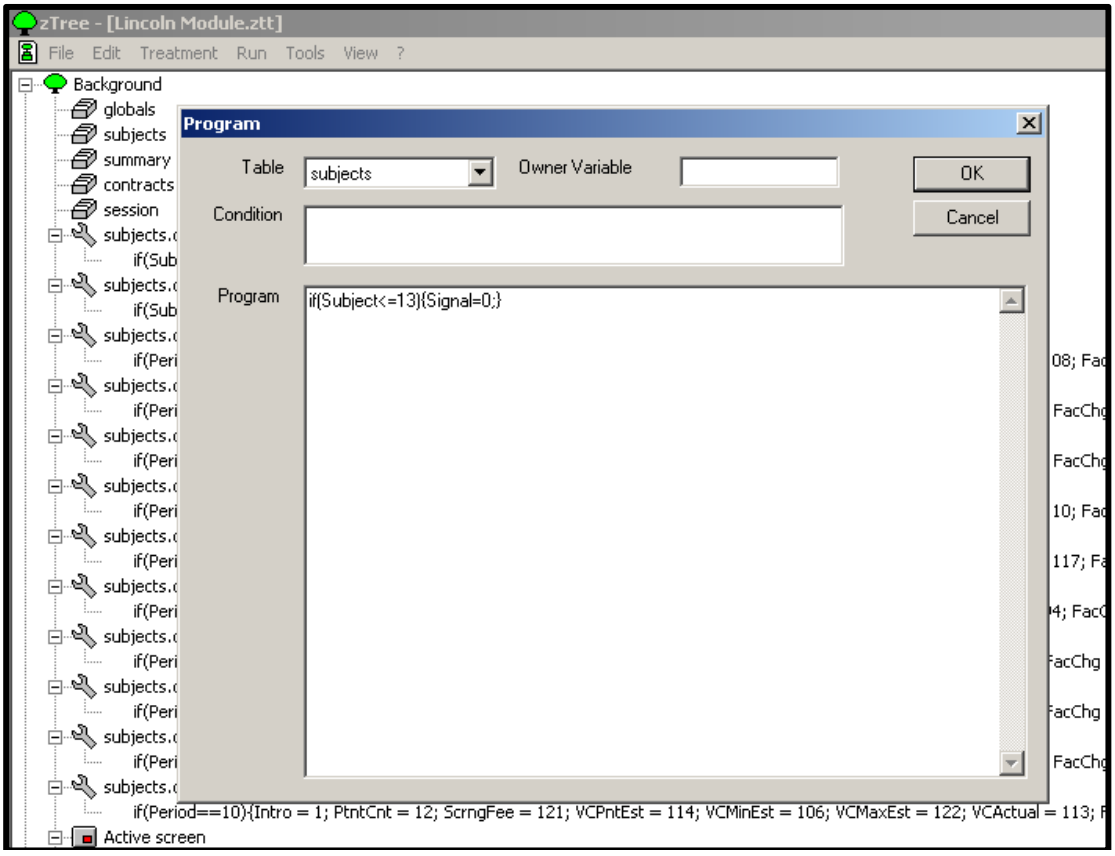


IV. CUSTOM PROGRAM DESIGN

The custom elements described in this section pertain to my specific experimental design. To demonstrate the game design, I will focus on the Lincoln module of my experiment. However, my experiment in its entirety included an example module (treatment), a Great Falls module (treatment), a Great Falls survey, a Lincoln module (treatment), and a Lincoln survey. The order of the two primary modules was randomly varied between experimental sessions, to remove any potential order effects; otherwise, the sessions were identical.

My study included one between-subjects manipulation (referred to as the signal manipulation), relating to whether a subject was presented with a point estimate for variable costs per patient or a confidence interval estimate. Within each session, I wanted approximately half of the subjects to fall into each of these conditions. Therefore, I included a program at the start of my treatment that assigned the signal variable a value of 0 (point estimate) for the first 13 of 26 subjects in the lab and a value of 1 (confidence interval estimate) for the last 13 subjects.[6] For the one session that I conducted that involved fewer than the lab capacity of 26 subjects, I simply changed the values in this program to <=10 and >=11 (to split 19 subjects roughly equally between conditions).[7] Figure 4 provides a screen display of the program used to assign the signal condition, for the point estimate subjects.

Figure 4. *z-Tree* Screen Display for the Program for Assigning Signal Condition



[6] The Subjects variable is pre-defined in *z-Tree*, and is assigned values based on the order in which client computers execute the *z-Leaf* program. An experimenter who wishes to control the values for the Subject variable, as a matter of controlling the condition assignments, should either execute *z-Leaf* on the client computers himself or herself or should direct subjects as to when to execute *z-Leaf*.

[7] Since the layout of the computers in my lab included four rows, I decided to switch the signal condition assignments between sessions. In the second session, the subjects in the front of the room (the first 13 subjects) received the confidence interval estimates and the subjects in the back of the room received the point estimates.

For each period of my game, numerous variables took on different values. For example, the values assigned for patient count, screening fee, variable costs point estimate, variable costs minimum estimate, variable costs maximum estimate, and actual variable costs all differed from one period to the next. Therefore, I used programs within the BACKGROUND to assign these values. Each program used conditions, to ensure that the values were only assigned in the appropriate period. Figure 5 provides a screen display of the variable assignment program included for one of the periods.

A variable used to control the introduction screen was also included in the Background programs. Since I wanted subjects to be able to choose whether or not to play each module, I included a prompt in the introduction screen for them to indicate their choice. However, I did not want subjects to see this prompt for every period. So, the introduction variable was set to a different value in the first period, compared to the other nine periods of the treatment, to control which introduction screen appeared to subjects. In the first period, the introduction screen provided general information about the module in addition to the prompt for subjects to choose whether or not to play the module (see Figure 6). In subsequent periods, the introduction screen simply restated for subjects their current balance and stalled the progress of the game briefly for them to make any notes that they wanted.

There is no apparent programming mechanism that would allow subjects to skip all of the screens if they chose not to play a module. Therefore, the introduction screen included a note indicating that subjects would still be required to complete each screen even if they were not playing the module, but that none of their decisions would be recorded and they would simply retain their starting balance. Including this requirement served a couple of purposes, other than being easy to code. First, it discouraged subjects from opting out of modules just to be lazy and sit in the lab for the duration of the session. Second, by forcing subjects to complete the screens, it hopefully gave them a sense of what profit possibilities existed and therefore discouraged them from opting out of future modules.

The information that I presented to subjects in the decision stage was dependent upon their signal condition. Therefore, I created a separate program clause for each condition. Both clauses include reporting of the town under review, the patient count, the screening fee (revenue per patient), and the facility charge. In the signal condition, the subjects were provided with a confidence interval estimate of variable costs per patient. In the no-signal condition, a point estimate was provided instead. Figure 7 provides a screen display of the coding used for the decision stage in the signal condition, and Figure 8 presents the screen as it appeared for subjects.

Figure 5. *z-Tree* Screen Display for the Program for Assigning Variable Values for a Period

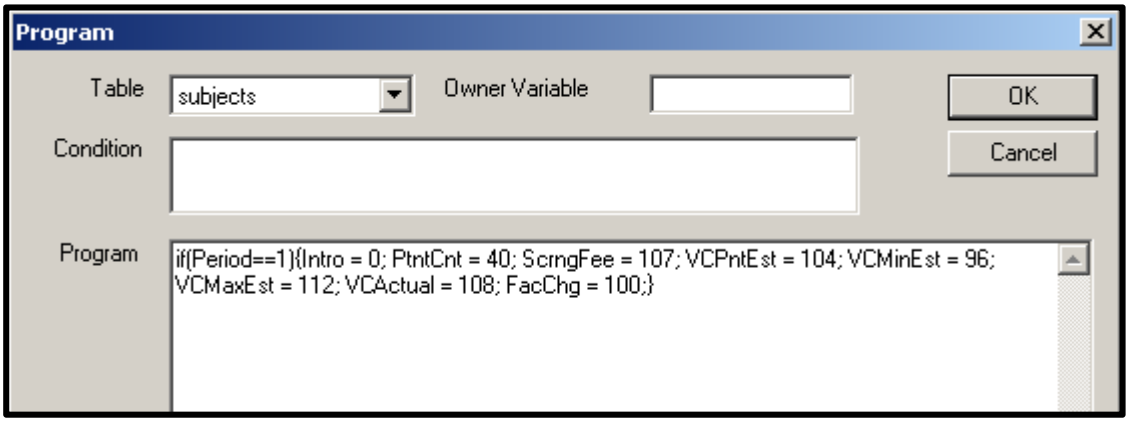


Figure 6. *z-Leaf* Introduction Screen for the Lincoln Module

Figure 7. *z-Tree* Screen Display for the Confidence Interval Decision Stage Coding

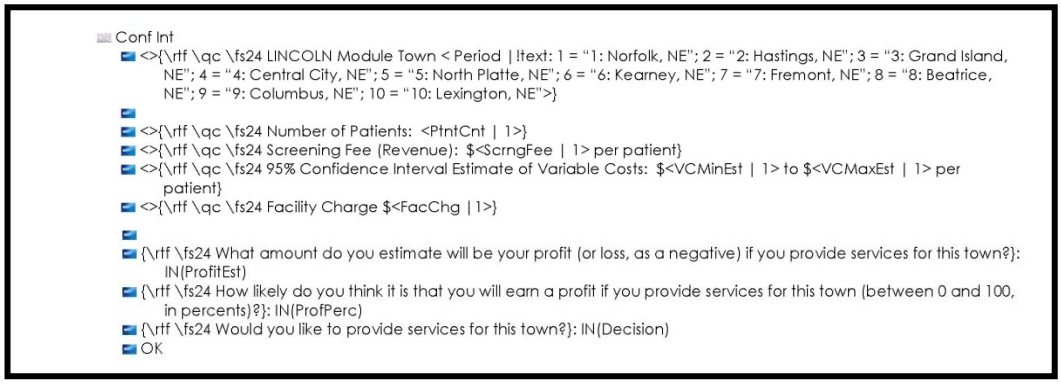
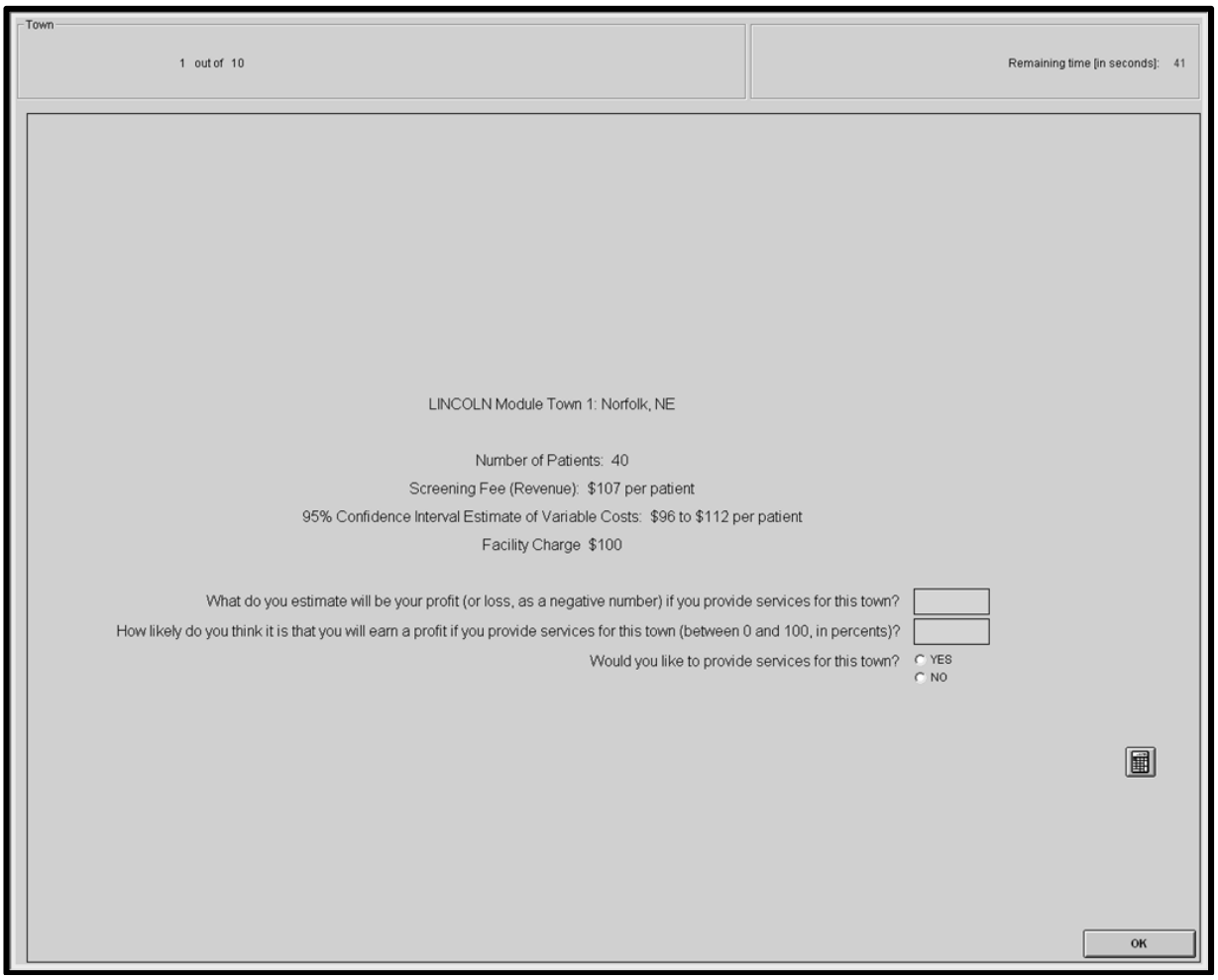


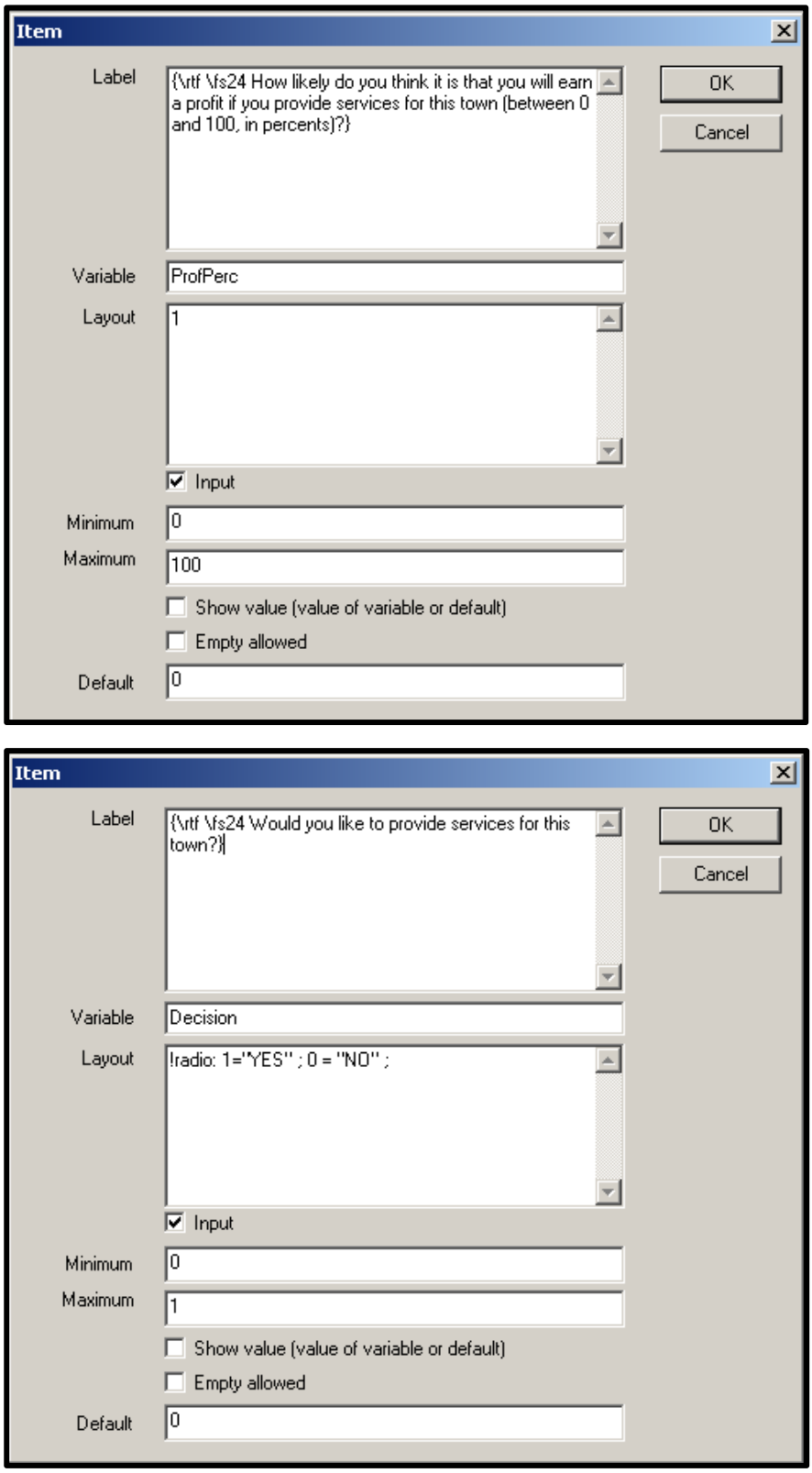
Figure 8. *z-Leaf* Decision Stage Screen with a Confidence Interval



The decision stage screen included three input variables. The first input variable asked subjects to estimate what the profit or loss would be for the period if they chose to provide services. The second input variable asked subjects to estimate the likelihood of earning a profit if they provided services for the period. Answers to these two questions did not impact performance evaluation of the subjects, but instead were intended to get subjects thinking about their decisions and to help me to identify possible inconsistencies or incompetencies.[8] The third input variable was the critical decision in the game – whether or not to provide services. Each of these input variables had rules associated with it. For example, only integer values between 0 and 100 were allowed for the question about the likelihood of earning a profit. And only one of the two radio buttons (for "Yes" or "No") could have been selected for the question about whether to provide services. Figure 9 provides the code screen displays for these two input variable items.

[8] For example, not considering risk preferences, I generally expected subjects who predicted a profit to also choose to provide services. And, if they predicted a profit, then the likelihood that they provided for a profit occurring should have been greater than 50%. To the extent that subjects were routinely inconsistent in their answers to these questions and their decisions to provide services, then it suggested that the subjects did not really understand the game and maybe should have been eliminated from certain analyses.

Figure 9. *z-Tree* Screen Displays for Two Different Input Variable Items



If subjects tried to respond to one of these input variables in a manner not allowed, such as to enter a percentage for the first question that was not between 0 and 100, they would receive an error message from the program (hopefully in English!) reminding them of the value constraints for the variable.

Notice also that I added a calculator button to the decision screen. Many subjects brought calculators with them to the lab, but having this option available to them on the screen allowed for equal computational power (at least mechanically) across the entire subject pool.

Before the profit display screen could function appropriately, several computations were needed. Therefore, additional programs were added. A number of these programs were designed to compute revenues, expenses, and profit for the current period, depending upon the decision the subject made; the other programs computed the revenues, expenses, and profit that would have resulted if the alternate decision had been made. These programs could not have been included in the background section, with the other variable assignment programs, because they required the values gathered from the decision stage screen (in particular, whether or not services were to be provided). The profit display screen first reminded subjects of what decision they had made and then summarized for them the resulting outcome from the decision. Then, they were also told what the results would have been if they had made the alternate decision. Figures 10 and 11 provide example profit display screens, for a subject in the signal condition who either chose to provide services or chose not to provide services, respectively.

Figure 10. *z-Leaf* Profit Reporting Screen with a Confidence Interval and Providing Services

Figure 11. *z-Leaf* Profit Reporting Screen with a Confidence Interval and Not Providing Services

Keeping the experiment on pace can be challenging, particularly for a large lab. With 26 subjects participating simultaneously, it was very common for one or two of the subjects to delay all of the others. Including time limits on certain decision screens can be a helpful way to maintain a reasonable pace. Although I did not have any screens that exited and used a default value for slow decision makers, that is an option that *z-Tree* provides and which may be appropriate in some studies.

To most easily test a program being designed, multiple shortcuts for *z-Leaf* can be established on the desktop of the primary experimenter computer. Each shortcut just needs to have a different name assigned. The name can be assigned using the shortcut properties, with the following coding in the target field: C:\zTree\zLeaf.exe **/name Plyr1** /language English /size 640x480 /position 10,10. For my experiment, it was meaningful to simultaneously preview the two different between-subjects conditions. Therefore, I created two *z-Leaf* shortcuts with the names Plyr 1 and Plyr 2. I also used a shrunken version of the *z-Leaf* screen, appearing toward the top left of my monitor, with the size command and the position command as shown above. The Plyr 2 target included the position command of /position 620,280 to correspond to the bottom right part of the monitor.

Since I had only one between-subjects manipulation, it was effective for me to test my program using only two *z-Leaf* shortcuts. To test each condition, I modified the program to assign a condition value of zero for the first subject and one for the second subject. Then, with *z-Tree* active on the desktop, I simply opened each of the two shortcuts for *z-Leaf*. The first *z-Leaf* shortcut opened is identified as the first subject and the second *z-Leaf* shortcut opened is identified as the second subject, regardless of the names (Plyr 1 or Plyr 2, for example) used in the shortcut properties. Using ALT-TAB, a researcher can then jump between the program in *z-Tree* and each *z-Leaf* screen.

To begin a *z-Tree* program, once all applicable *z-Leaf* clients are connected, select RUN from the top menu and then START TREATMENT. If there are no issues, such as having the wrong number of clients connected (remember, the number of clients is specified in the BACKGROUND), then starting the treatment should automatically update the client monitors to show the first *z-Leaf* screen for the program started rather than just the default *z-Leaf* screen.

At the end of each module of my experiment, I included a short questionnaire. The *z-Tree* programming for a questionnaire is similar to that used for a treatment, with a number of simplifications. The *z-Tree* programming for the Lincoln questionnaire is provided in Figure 12, with the specific coding for one likert-scale response variable shown in Figure 13. A questionnaire can only be run after a treatment has been run. To begin a *z-Tree* questionnaire, simply select Run from the top menu and then Start Questionnaire when the questionnaire code is active on the desktop. *z-Tree* will expect the same number of subjects that were used for the immediately preceding treatment. Figure 14 provides the questionnaire screen as it appeared for subjects.

z-Tree stores the data collected in several different files with differing structures, using the same file location as the *z-Tree* executable file. Table 2 provides some sample file names and descriptions related to a recent administering of my *z-Tree* program, for the two primary files that get used by most researchers. *z-Tree* also creates similar payment (.pay), address (.adr), and game safe (.gsf) files, which could become important in certain experimental situations. Having the variable data from the treatments available in Microsoft Excel format makes for easy reorganizing and analysis, a process which can otherwise be very time consuming after an experiment has been administered (such as by pen-and-paper methods).

Table 2. Sample Primary *z-Tree* File Names and Descriptions

File Name [a]	***Description***
090611HB.xls	Stores the values from all variables used in a treatment or a series of treatments, in Microsoft Excel format. Subjects are represented in rows whereas variables are represented in columns.
090611JS.sbj	Stores the values from all variables used in a questionnaire or a series of questionnaires, in Notepad format. Subjects are represented in columns whereas variables are represented in rows.

[a] The first four characters of each file name represent the date (YYMMDD) when the file was created, in this case 06/11/2009. The next two characters in each name represent the time when the file was created, using an alphabetic scale with the first character corresponding to the hour and the second character corresponding to the minute. That scale assigns hour values between A and X, for hours 00 through 23, and minute values between 0 and T, in two minute increments (i.e., 0 = 00 minutes, 1 = 02 minutes, …, 9 = 18 minutes, A = 20 minutes, …, T = 58 minutes).

Figure 12. *z-Tree* Screen Display for the Questionnaire Code for the Lincoln Module

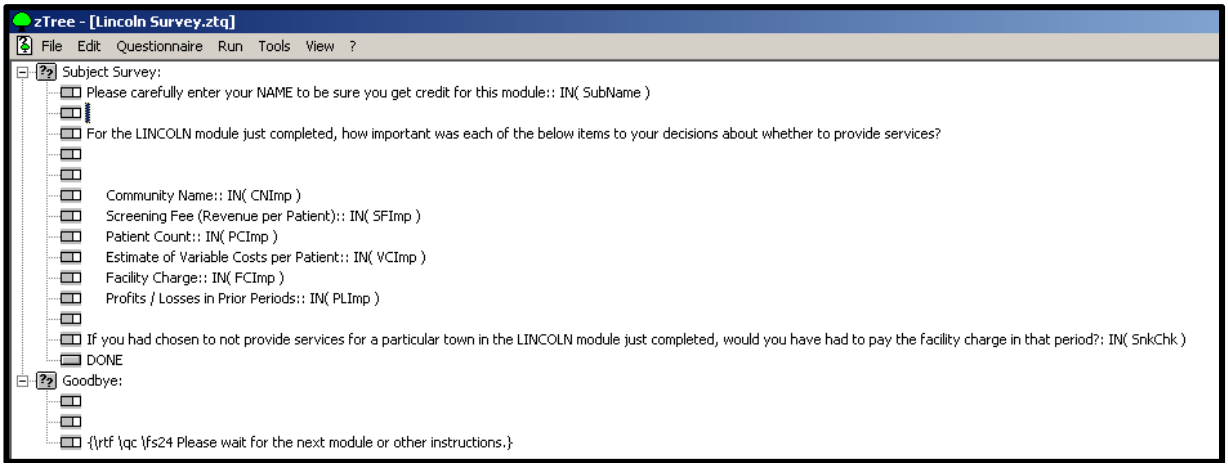


Figure 13. *z-Tree* Screen Display for a Likert-Scale Questionnaire Response Variable

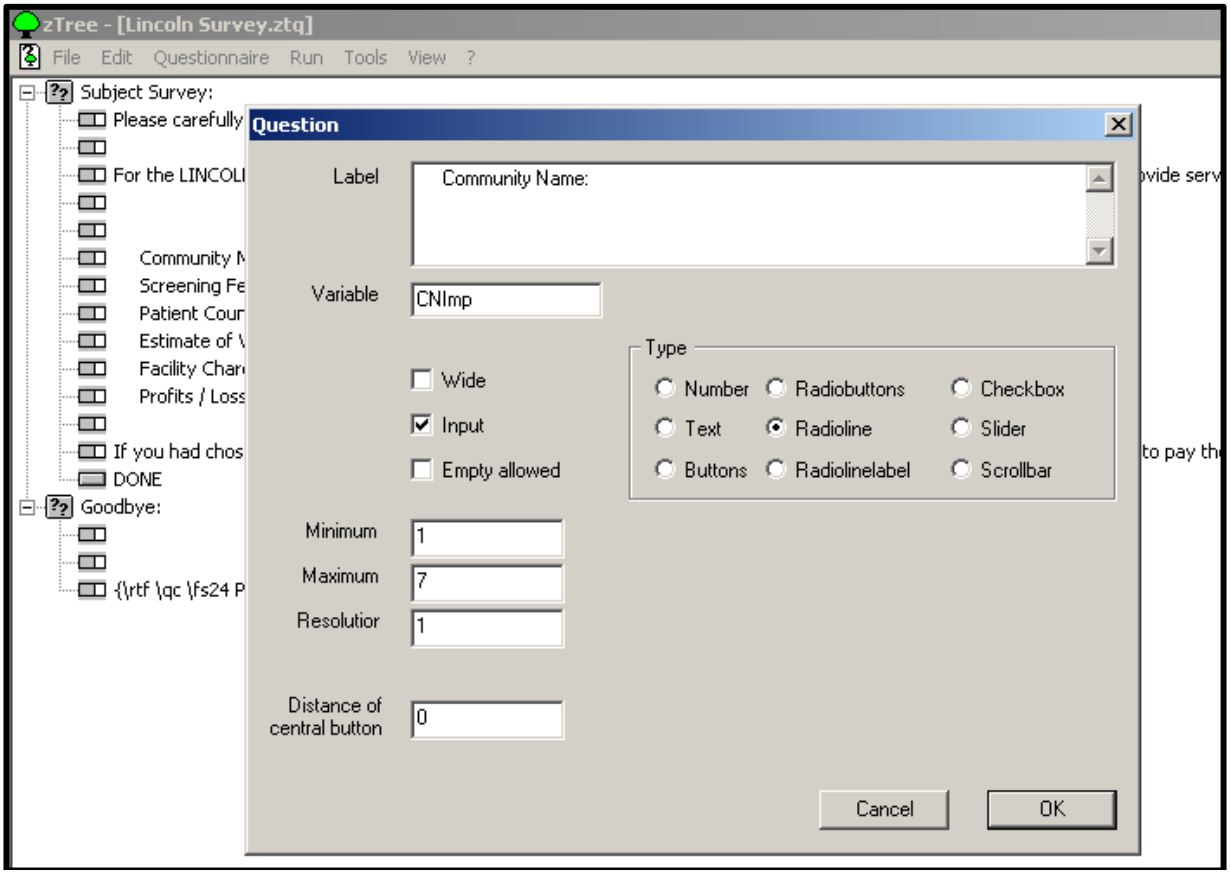
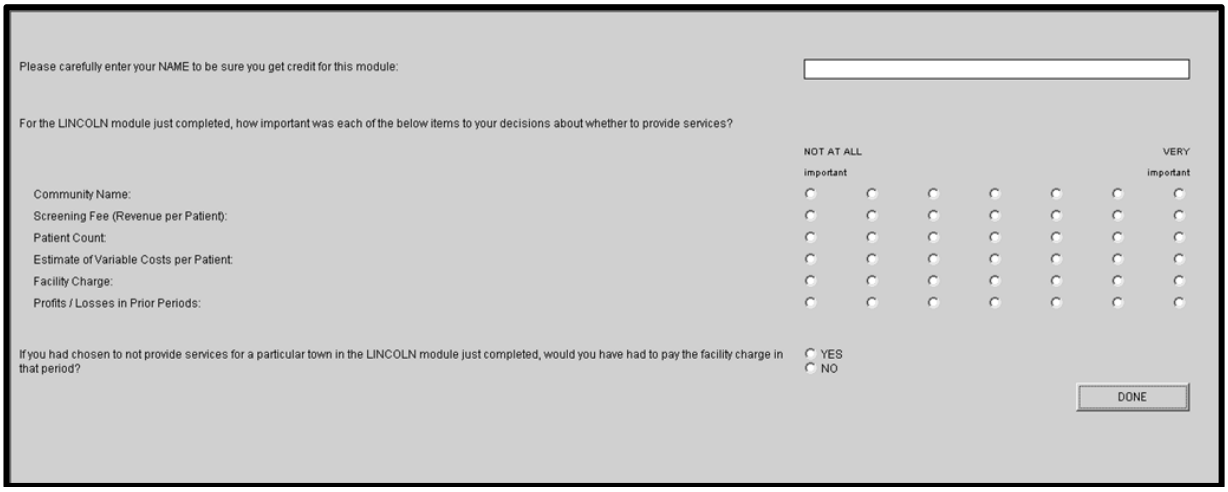


Figure 14. *z-Leaf* Questionnaire Screen for the Lincoln Module



V. SELECTED CODE DESCRIPTIONS

The entire block of code used for the Lincoln module of my study is provided in Appendix A. Several aspects of my *z-Tree* coding may seem unusual and therefore likely warrant further discussion.

"*`If-Then-Else`*" *Clauses*. "`If-Then-Else`" clauses are a standard element in almost any computer programming language, although the exact syntax varies from one language to the next. For *z-Tree*, two signs of comparison are always needed in the `If` component of the clause. For example, when I assign subjects to different conditions for the between-subjects manipulation, I specify one condition value for subjects <= 13 and the other condition value for subjects >=14. Logically, it would make sense to simply use <14 and >13 for the two conditions, but *z-Tree* does not seem to respond to such syntax. Also, when specifying an equality condition in the `If` statements, two equal signs are required (i.e., Period == 1). Note that such dual sign syntax is not required in the `Then` component of the clause. Also, for "`If-Then-Else`" clauses, the `If` statements are enclosed in normal parentheses and the `Then` statements are enclosed in squiggly brackets; each `Then` statement is concluded with a semicolon. And the word "then" is not included.

Rich Text Formatting. Rich text formatting (RTF) is not a component of *z-Tree*, per se, but it does allow a researcher to design more aesthetically pleasing screens using the older version of *z-Tree*. There are numerous RTF parameters that can be specified, and a quick search online for RTF commands will provide additional examples. However, the RTF commands that I found most useful and that are specified often in my *z-Tree* code are summarized in Table 3. Each statement that involves RTF commands must begin with an open squiggly bracket and the expression `\rtf`. The statement must end with a closing squiggly bracket, and each RTF command should be followed by a space.

Table 3. Common Rich Text Formatting (RTF) Commands Used

RTF Command	*Purpose*
\b	Initiates bolding of text
\b0	Concludes bolding of text
\fs24	Initiates a change in the font size, in this case to a font size of 24
\i	Initiates italicizing of text
\i0	Concludes italicizing of text
\qc	Initiates center alignment of text [a]
\ql	Initiates left alignment of text [a]
\ul	Initiates underlining of text

[a] When rich text formatting is not specified, *z-Tree* uses text centering in most cases and in other cases places text / variable fields in columns.

Time Limits and Automatic Exits. For many of the stages included in a *z-Tree* treatment, the researcher can specify how much time is permitted for subject responses and whether or not to automatically exit at the end of that time allotment.[9] When a time limit is specified, but an automatic exit is not, then the time remaining for that stage will count down in the upper right portion of the subject screen and,

[9] In the *z-Tree* code (see Appendix A), the time allowed and the exit specification are indicated at the end of the description for the screen. For example, at the end of the description for the INTRODUCTION SCREEN, there is a "`=|=(5)N`" notation. That indicates that the screen has a time limit of five seconds and does not automatically exit.

when time has expired, a warning message will flash encouraging the subject to complete the necessary actions for the stage. If an automatic exit is specified, then default values for any input variables in that stage must also be specified so that *z-Tree* will know how to treat non-responders. For decision variables critical to the research purpose, it is probably better to avoid using automatic exits and associated default values. Pilot tests of an experiment should help to identify appropriate time limits, considering that some subjects will act more quickly than others. (The time limit should generally be long enough to accommodate the slowest responding subjects, particularly if an automatic exit is specified.)

Variable Values. When a variable value is included in a *z-Tree* statement, the syntax includes adjacent carrot brackets at the start of the line, individual carrot brackets before and after the variable name, and a specification for the decimal format to be used for the variable value. For example, in one of the lines in my decision stage screen, I include the following syntax: `<>{\rtf \qc \fs24 Number of Patients: <PtntCnt | 1>}`. In this case, the assigned value for the variable `PtntCnt` (as defined in the BACKGROUND) would be shown after the label, and it would be rounded to the nearest integer. If instead the specification for the decimal format was 0.1 (to replace 1 in the syntax above), then the value for the variable would be rounded to the nearest tenth of an integer.
If it is more appropriate to present a text string rather than the variable value, but to base the text string on the variable value, then slightly different syntax is required. An abridged example from my code block is as follows: `<>{\rtf \qc \fs24 LINCOLN Module Town < Period | !text: 1 = "1: Norfolk, NE"; 2 = "2: Hastings, NE">}`. Using this code, the text that will be shown is "LINCOLN Module Town 1: Norfolk, NE" if it is the first period and "LINCOLN Module Town 2: Hastings, NE" if it is the second period. Other similar text strings can be added using the `!text:` syntax.

Sequential Variable Definition Programs. More than one variable definition statement can be included in the same program block. However, if the value of a variable being defined is based on the value of one or more other variables, then the other variable(s) must have been defined in a prior program block. For example, I use one program block in the PROFIT DISPLAY stage to define the values for the variables `Revs` and `VarCosts` (depending upon the decision made by the subject). Those variable values are based on other variables (`PtntCnt`, `ScrngFee`, and `VCActual`) for which values were assigned in the BACKGROUND. Then, I want to compute the value for the `Profit` variable. Since the value for the `Profit` variable is based on the values for the `Revs` and `VarCosts` variables (as well as the value for the `FacChg` variable, which was defined in the BACKGROUND), a separate and subsequent program block must be used.

VI. CONCLUSION

Clearly, *z-Tree* is a viable tool that can be used for a variety of behavioral accounting research studies. The study described here is non-interactive, meaning that subjects are not sending information to one another and the performance of one subject is not based at all on the performance of another subject. (In other words, the program could be administered for a single subject at a time if it were efficient and effective to do so.) In about the time that it would take to design and print paper forms for subjects to complete a pen-and-paper experimental design, a custom program can be written in *z-Tree*. The additional benefits of reduced risk of data entry error and more timely subject feedback can also then be attained. Other computerized methods for conducting experiments, such as web programming and linked spreadsheets, can offer some additional flexibility, but they also usually require more time and effort for development and maintenance. For researchers interested in quick and easy computerization, *z-Tree* should definitely be considered.

REFERENCES

Bowlin, K. "Can Strategic Reasoning Prompts Improve Auditors' Sensitivity to Fraud Risks?" Working Paper, University of Texas, 2008.

Bowlin, K. O., J. Hales, and S. J. Kachelmeier. "Experimental Evidence of How Prior Experience as an Auditor Influences Managers' Strategic Reporting Decisions." *Review of Accounting Studies* 14, no. 1 (2009): 63-87.

Christ, M.H. "Intending to Control: An Experimental Investigation of the Interactions among Intentions, Reciprocity and Control" Working Paper, University of Texas, 2008.

Dickson, E. S., C. Hafer, and D. Landa. "Cognition and Strategy: A Deliberation Experiment." *The Journal of Politics* 70 (2008): 974-989.

Dittrich, D., W. Guth, and B. Maciejovsky. "Overconfidence in Investment Decisions: An Experimental Approach." *European Journal of Finance* 11, no. 6 (2005): 471-491.

Downen, T. "Signaling Accounting Uncertainty: Potential Decision Implications." Working Paper, Northern Arizona University, 2010.

Engelbrecht-Wiggans, R., E. Haruvy, and E. Katok. "A Comparison of Buyer-Determined and Price-Based." *Marketing Science* 26, no. 5 (2007): 629-641.

Esarey, J. "zTree Workshop: Fundamentals of zTree." *Florida State University - Department of Political Science.* 2005. http://userwww.service.emory.edu/~jesarey/ztree.ppt (accessed 06 11, 2009).

Fischbacher, U. *z-Tree Tutorial 2.1.* Zurich: University of Zurich, 2002.

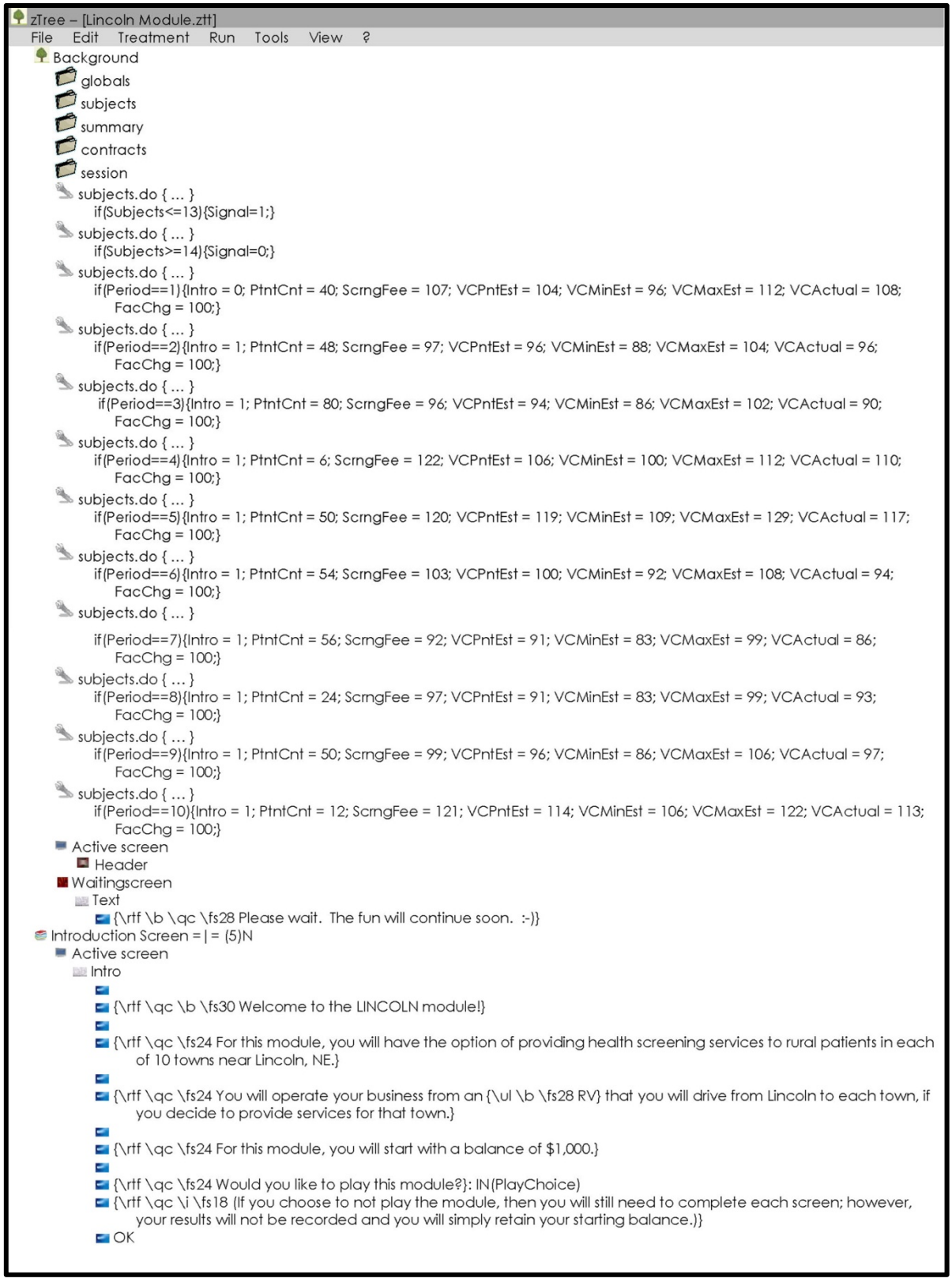
Fischbacher, U. "z-Tree: Zurich Toolbox for Ready-Made Economic Experiments." *Experimental Economics* 10, no. 2 (2007): 171 - 178.

Hales, J. and M. G. Williamson. "Implicit Employment Contracts: The Limits of Management Reputation for Promoting Firm Productivity" *Journal of Accounting Research* (forthcoming) (2009).

Haruvy, E., and C. N. Noussair. "The Effect of Short Selling on Bubbles and Crashes in Experimental Spot Asset Markets." *The Journal of Finance* 61, no. 3 (2006): 1119-1157.

Newman, A. H. "The Behavioral Effect of Cost Targets on Managerial Cost Reporting Honesty." Dissertation, Georgia State University, 2009.

APPENDIX A. *z-TREE* CODE FOR THE LINCOLN MODULE



```
zTree – [Lincoln Module.ztt]
File  Edit  Treatment  Run  Tools  View  ?
Background
  globals
  subjects
  summary
  contracts
  session
  subjects.do { ... }
    if(Subjects<=13){Signal=1;}
  subjects.do { ... }
    if(Subjects>=14){Signal=0;}
  subjects.do { ... }
    if(Period==1){Intro = 0; PtntCnt = 40; ScrngFee = 107; VCPntEst = 104; VCMinEst = 96; VCMaxEst = 112; VCActual = 108;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==2){Intro = 1; PtntCnt = 48; ScrngFee = 97; VCPntEst = 96; VCMinEst = 88; VCMaxEst = 104; VCActual = 96;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==3){Intro = 1; PtntCnt = 80; ScrngFee = 96; VCPntEst = 94; VCMinEst = 86; VCMaxEst = 102; VCActual = 90;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==4){Intro = 1; PtntCnt = 6; ScrngFee = 122; VCPntEst = 106; VCMinEst = 100; VCMaxEst = 112; VCActual = 110;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==5){Intro = 1; PtntCnt = 50; ScrngFee = 120; VCPntEst = 119; VCMinEst = 109; VCMaxEst = 129; VCActual = 117;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==6){Intro = 1; PtntCnt = 54; ScrngFee = 103; VCPntEst = 100; VCMinEst = 92; VCMaxEst = 108; VCActual = 94;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==7){Intro = 1; PtntCnt = 56; ScrngFee = 92; VCPntEst = 91; VCMinEst = 83; VCMaxEst = 99; VCActual = 86;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==8){Intro = 1; PtntCnt = 24; ScrngFee = 97; VCPntEst = 91; VCMinEst = 83; VCMaxEst = 99; VCActual = 93;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==9){Intro = 1; PtntCnt = 50; ScrngFee = 99; VCPntEst = 96; VCMinEst = 86; VCMaxEst = 106; VCActual = 97;
      FacChg = 100;}
  subjects.do { ... }
    if(Period==10){Intro = 1; PtntCnt = 12; ScrngFee = 121; VCPntEst = 114; VCMinEst = 106; VCMaxEst = 122; VCActual = 113;
      FacChg = 100;}
  Active screen
    Header
  Waitingscreen
    Text
      {\rtf \b \qc \fs28 Please wait.  The fun will continue soon.  :-)}
Introduction Screen =|= (5)N
  Active screen
    Intro
      {\rtf \qc \b \fs30 Welcome to the LINCOLN module!}
      {\rtf \qc \fs24 For this module, you will have the option of providing health screening services to rural patients in each
        of 10 towns near Lincoln, NE.}
      {\rtf \qc \fs24 You will operate your business from an {\ul \b \fs28 RV} that you will drive from Lincoln to each town, if
        you decide to provide services for that town.}
      {\rtf \qc \fs24 For this module, you will start with a balance of $1,000.}
      {\rtf \qc \fs24 Would you like to play this module?}: IN(PlayChoice)
      {\rtf \qc \i \fs18 (If you choose to not play the module, then you will still need to complete each screen; however,
        your results will not be recorded and you will simply retain your starting balance.)}
      OK
```

APPENDIX A. *z-TREE* CODE FOR THE LINCOLN MODULE (CONT'D)

```
    Wait
      <>{\rtf \qc \b \fs24 Once again, your current LINCOLN module balance is $<TotalProfit | 1>.}
      {\rtf \qc \fs24 Please make any notes you would like and click OK when you are ready for the next period.}
      OK
  Waitingscreen
Decision Stage =|= (45)N
  Active screen
    Pnt Est
      <>{\rtf \qc \fs24 LINCOLN Module Town < Period |!text: 1 = "1: Norfolk, NE"; 2 = "2: Hastings, NE"; 3 = "3: Grand Island,
          NE"; 4 = "4: Central City, NE"; 5 = "5: North Platte, NE"; 6 = "6: Kearney, NE"; 7 = "7: Fremont, NE"; 8 = "8: Beatrice,
          NE"; 9 = "9: Columbus, NE"; 10 = "10: Lexington, NE">}
      <>{\rtf \qc \fs24 Number of Patients:  <PtntCnt | 1>}
      <>{\rtf \qc \fs24 Screening Fee (Revenue):  $<ScrngFee | 1> per patient}
      <>{\rtf \qc \fs24 Estimate of Variable Costs:  $<VCPntEst | 1> per patient}
      <>{\rtf \qc \fs24 Facility Charge $<FacChg |1>}
      {\rtf \fs24 What do you estimate will be your profit (or loss, as a negative number) if you provide services for this town?}:
          IN(ProfitEst)
      {\rtf \fs24 How likely do you think it is that you will earn a profit if you provide services for this town (between 0 and 100,
          in percents)?}: IN(ProfPerc)
      {\rtf \fs24 Would you like to provide services for this town?}: IN(Decision)
      OK
    Conf Int
      <>{\rtf \qc \fs24 LINCOLN Module Town < Period |!text: 1 = "1: Norfolk, NE"; 2 = "2: Hastings, NE"; 3 = "3: Grand Island,
          NE"; 4 = "4: Central City, NE"; 5 = "5: North Platte, NE"; 6 = "6: Kearney, NE"; 7 = "7: Fremont, NE"; 8 = "8: Beatrice,
          NE"; 9 = "9: Columbus, NE"; 10 = "10: Lexington, NE">}
      <>{\rtf \qc \fs24 Number of Patients:  <PtntCnt | 1>}
      <>{\rtf \qc \fs24 Screening Fee (Revenue):  $<ScrngFee | 1> per patient}
      <>{\rtf \qc \fs24 95% Confidence Interval Estimate of Variable Costs:  $<VCMinEst | 1> to $<VCMaxEst | 1> per
          patient}
      <>{\rtf \qc \fs24 Facility Charge $<FacChg |1>}
      {\rtf \fs24 What do you estimate will be your profit (or loss, as a negative number) if you provide services for this town?}:
          IN(ProfitEst)
      {\rtf \fs24 How likely do you think it is that you will earn a profit if you provide services for this town (between 0 and 100,
          in percents)?}: IN(ProfPerc)
      {\rtf \fs24 Would you like to provide services for this town?}: IN(Decision)
      OK
    Calculator
  Waitingscreen
Profit Display =|= (25)A
  subjects(Decision==1).do { ... }
    Revs = PtntCnt * ScrngFee;
    VarCosts = PtntCnt * VCActual;
  subjects(Decision==0).do { ... }
    Revs = 0;
    VarCosts = 0;
    OthRevs = PtntCnt * ScrngFee;
    OthVarCosts = PtntCnt * VCActual;
  subjects(Decision==1).do { ... }
    Profit = Revs – VarCosts - FacChg;
    OthRevs = 0; OthVarCosts = 0; OthProfit = 0;
  subjects(Decision==0).do { ... }
    Profit = 0;
    OthProfit = OthRevs – OthVarCosts - FacChg;
  subjects.do { ... }
    Balance = TotalProfit + Profit;
  Active screen
    Results YES Pnt Est
      <>{\rtf \qc \fs24 Results for < Period |!text: 1 = "Norfolk, NE"; 2 = "Hastings, NE"; 3 = "Grand Island, NE"; 4 = "Central
          City, NE"; 5 = "North Platte, NE"; 6 = "Kearney, NE"; 7 = "Fremont, NE"; 8 = "Beatrice, NE"; 9 = "Columbus, NE"; 10 =
          "Lexington, NE">}
      {\rtf \qc \b \fs24 \i For this town, you chose to PROVIDE SERVICES. }
```

APPENDIX A. *z-TREE* CODE FOR THE LINCOLN MODULE (CONT'D)

```
<>{\rtf \qc \b \fs24 As a reminder, the VC estimate was $<VCPntEst | 1> per patient.}
<>{\rtf \qc \b \fs24 Randomly Determined \i ACTUAL \i0 Variable Costs were \fs30 $<VCActual | 1>\fs24 per patient.}

<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab \tab \tab Revenues $<Revs | 1> \b0 ($<ScrngFee | 1>
    per patient X <PtntCnt | 1> patients)}
<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab      less Variable Costs $<VarCosts | 1> \b0 ($<VCActual
    | 1> per patient X <PtntCnt | 1> patients)}
<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab     less Facility Charge $ <FacChg | 1> \b0 (flat amount)}

<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab \tab \tab     PROFIT $ <Profit | 1>}

<>{\rtf \qc \b \fs24 Your New LINCOLN Module Balance is $<Balance | 1>.}

<>{\rtf \qc \fs24 If you had chosen to NOT provide services, your reported profit would have been $<OthProfit | 1>.}
Results YES Conf Int
<>{\rtf \qc \fs24 Results for < Period | !text: 1 = "Norfolk, NE"; 2 = "Hastings, NE"; 3 = "Grand Island, NE"; 4 = "Central
    City, NE"; 5 = "North Platte, NE"; 6 = "Kearney, NE"; 7 = "Fremont, NE"; 8 = "Beatrice, NE"; 9 = "Columbus, NE"; 10 =
    "Lexington, NE">}

{\rtf \qc \b \fs24 \i For this town, you chose to PROVIDE SERVICES. }

<>{\rtf \qc \b \fs24 As a reminder, the VC estimate was between $<VCMinEst | 1> and $<VCMaxEst | 1> per patient.}
<>{\rtf \qc \b \fs24 Randomly Determined \i ACTUAL \i0 Variable Costs were \fs30 $<VCActual | 1>\fs24 per patient.}

<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab \tab \tab Revenues $<Revs | 1> \b0 ($<ScrngFee | 1>
    per patient X <PtntCnt | 1> patients)}
<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab      less Variable Costs $<VarCosts | 1> \b0 ($<VCActual
    | 1> per patient X <PtntCnt | 1> patients)}
<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab     less Facility Charge $ <FacChg | 1> \b0 (flat amount)}

<>{\rtf \ql \b \fs24 \tab \tab \tab \tab \tab \tab \tab \tab \tab \tab     PROFIT $ <Profit | 1>}

<>{\rtf \qc \b \fs24 Your New LINCOLN Module Balance is $<Balance | 1>.}

<>{\rtf \qc \fs24 If you had chosen to NOT provide services, your reported profit would have been $<OthProfit | 1>.}
Results NO Pnt Est
<>{\rtf \qc \fs24 Results for < Period | !text: 1 = "Norfolk, NE"; 2 = "Hastings, NE"; 3 = "Grand Island, NE"; 4 = "Central
    City, NE"; 5 = "North Platte, NE"; 6 = "Kearney, NE"; 7 = "Fremont, NE"; 8 = "Beatrice, NE"; 9 = "Columbus, NE"; 10 =
    "Lexington, NE">}

{\rtf \qc \b \fs24 \i For this town, you chose to NOT PROVIDE SERVICES. }

<>{\rtf \qc \b \fs24 PROFIT $<Profit | 1>}
<>{\rtf \qc \b \fs24 Your New LINCOLN Module Balance is $<Balance | 1>.}

{\rtf \qc \fs24 If you had chosen to provide services, you would have had the following results: }

<>{\rtf \qc \fs24 As a reminder, the VC estimate was $<VCPntEst | 1> per patient.}
<>{\rtf \qc \fs24 Randomly Determined \i ACTUAL \i0 Variable Costs were \fs30 $<VCActual | 1>\fs24 per patient.}

<>{\rtf \ql \fs24 \b \tab \tab \tab \tab \tab \tab \tab \tab \tab \b0    Revenues $<OthRevs | 1> ($< ScrngFee
    | 1> per patient X <PtntCnt | 1> patients)}
<>{\rtf \ql \fs24 \b \tab \tab \tab \tab \tab \tab \tab      \b0 less Variable Costs $<OthVarCosts | 1> ($<VCActual |
    1> per patient X <PtntCnt | 1> patients)}
<>{\rtf \ql \fs24 \b \tab \tab \tab \tab \tab \tab \tab      \b0 less Facility Charge $ <FacChg | 1> (flat amount)}

<>{\rtf \ql \fs24 \b \tab \tab \tab \tab \tab \tab \tab \tab \tab \b0    PROFIT $ <OthProfit | 1>}
Results NO Conf Int
<>{\rtf \qc \fs24 Results for < Period | !text: 1 = "Norfolk, NE"; 2 = "Hastings, NE"; 3 = "Grand Island, NE"; 4 = "Central
    City, NE"; 5 = "North Platte, NE"; 6 = "Kearney, NE"; 7 = "Fremont, NE"; 8 = "Beatrice, NE"; 9 = "Columbus, NE"; 10 =
    "Lexington, NE">}

{\rtf \qc \b \fs24 \i For this town, you chose to NOT PROVIDE SERVICES. }

<>{\rtf \qc \b \fs24 PROFIT $<Profit | 1>}
<>{\rtf \qc \b \fs24 Your New LINCOLN Module Balance is $<Balance | 1>.}
```

APPENDIX A. *z-TREE* CODE FOR THE LINCOLN MODULE (CONT'D)