JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 6, July-August 2007

EDUCATOR'S CORNER

Converting a C#/.NET Desktop Application to an ASP.NET 2005 Web Application

Richard Wiener, Editor-in-Chief, JOT, Chair, Department of Computer Science, University of Colorado at Colorado Springs

1 INTRODUCTION

In the May/June, 2007 issue of JOT I presented a paper entitled, "A New Software Development Project Using an Old Game" (Vol 6, No 4). As indicated in that paper, "As many of us who teach more advanced programming courses know, finding software development projects that are engaging to students and that provide sufficient richness to allow for interesting designs and implementations while being solvable within the time-constraints of a semester is challenging. Some games provide a basis for satisfying these requirements since they do not require prior domain expertise. The rules can be specified precisely in contrast to many real-world problems where only fuzzy and changing specifications are the norm."

As is evident if you read the details of the game specification given in the paper cited above, only one human player competes against three computer players. That makes for a useful desktop game but does not allow for the dynamics of two or more human players in direct competition. This fact provides motivation for converting the game from a desktop GUI application to a web-based application. That would enable up to four human players to compete and computer players (residing on the server) filling in the gap if fewer than four human players cannot be found.

The principal challenge in designing the web-based game is how to inform the human players residing on separate client machines what the state of the game is as human or computer players complete their moves. Common web programming protocol does not easily support direct computer-to-computer communication but only communication from one client computer to the server and directly back to the client computer.

The solution to this problem is quite simple in principle. A timer on each client computer must periodically solicit game state from the server. If this is done using

Cite this column as follows: Richard Wiener: "Converting a C#/.NET Desktop Application to an ASP.NET 2005 Web Application, vol. 7, no. 6, July - August 2007, pp 35-39 http://www.jot.fm/issues/issue_2007_07/column4

traditional web programming facilities, each client will experience a page load (page refresh) each instant the timer obtains state information from the server and then updates the GUI on the client. Such page refreshs would promote an intolerable blinking of information.

The solution that was used involves the use of some of the facilities of the Ajax Extensions now available in ASP.NET 2005. The Ajax UpdatePanel allows GUI controls embedded in an *UpdatePanel* to be updated without the entire GUI incurring a page refresh.

2 THE DESIGN FORM

A screenshot of the ASP.NET GUI that uses such *UpdatePanel* components is shown below.

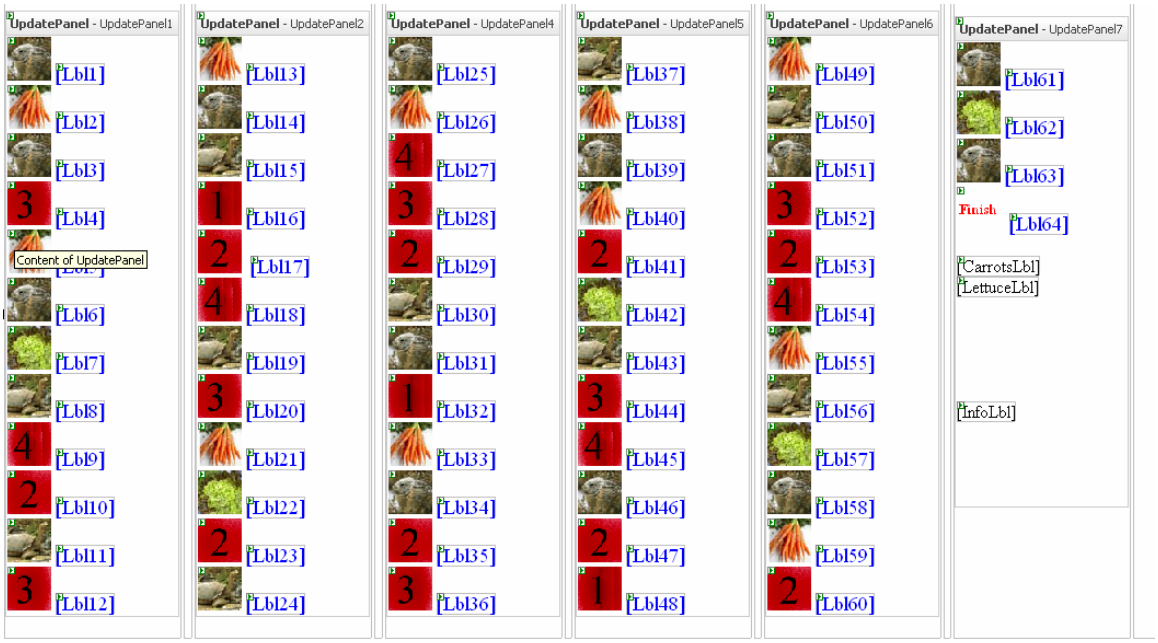

Figure 1 – Screenshot of Design Form

A screenshot of the actual web-based game is shown in Figure 2.

Figure 2 – Screenshot of Web-Based Game

The layout of the game had to be modified compared to the look of the desktop version (see the May/June 2007 paper for the screenshot of the desktop application) in order to allow the image button controls to be placed in a collection of UpdatePanel Ajax components.

An Ajax Timer control was set to 1.5 seconds.

3 SOME ASP.NET PROGRAMMING DETAILS

The timer event handler invokes the local *Status* method every 1.5 seconds. Some of the details of the *Status* method, which shows the communication between the client and the server, are presented below.

```
public partial class _Default : System.Web.UI.Page {

    // Fields
    private GameModel model;
    private ImageButton[] buttons;
    private Label[] labels;
```

// Many other details skipped here

```
protected void Status() {
   if (!gameOver) {
        model = (GameModel)Application["Model"];
   if (model != null && Session["PlayerNumber"] != null) {
        for (int i = 1; i <= 64; i++) {
             labels[i].Text = "";
        }
        if (model != null) {
             int[] positions = model.Positions;
             for (int i = 1; i <= 4; i++) {
                  if (positions[i] > 0) {
                  labels[positions[i]].Text = "" + i;
              }
        }
        if (model.GameOver) {
             InfoLbl.Text = "Game over. Winner is " +
                            model.Winner;
             model.Positions[model.Winner] = 64;
        }
        int playerNumber = (int)Session["PlayerNumber"];
        if (playerNumber == model.Turn &&
                  model.NoPlaceToMoveFor(playerNumber)) {
             model.IncrementTurn();
             InfoLbl.Text = "Skipped turn …";
             if (model.Turn >= model.FirstComputerPlayer &&
                        !model.GameOver) {
                  playerNumber = model.Turn;
                  model.MakeComputerMove(playerNumber);
                  model.IncrementTurn();
                  if (model.Turn >= model.FirstComputerPlayer
                            && !model.GameOver) {
                       playerNumber = model.Turn;
             model.MakeComputerMove(playerNumber);
             model.IncrementTurn();
        }
        // Remaining details not shown
```

The *Application* variable is used to obtain the current game model object each time the status is updated for each client.

4 CONCLUSIONS

Working in conjunction with the *GameModel* object are the same dozen or so model support classes that were developed in conjunction with the desktop application. The core business logic architecture was therefore preserved. The approximate time to complete the migration from desktop application to web-application was two days compared to the roughly two months of effort expended in designing and implementing the original desktop application. The *UpdatePanel* and *Timer* components worked as advertised. When a human player clicks one of the 64 image control buttons, there is no page refresh as would typically occur when pushing a button in a web form. Allowing for the possible 1.5 second time-lag after a human player executes a move, all the other players are updated in a timely manner. Since players are allowed only to move in sequence (when the "YOUR TURN" label is made visible), there is seemless communication among the human players as the game evolves.

The ASP.NET 2005 web application framework provided useful support and leverage in making the transition from a desktop to web-based application smooth and efficient.

About the author

Richard Wiener is Chair of Computer Science at the University of Colorado at Colorado Springs. He is also the Editor-in-Chief of JOT and former Editor-in-Chief of the Journal of Object Oriented Programming. In addition to University work, Dr. Wiener has authored or co-authored 22 books and works actively as a consultant and software contractor whenever the possibility arises. His latest book, published by Thomson, Course Technology in April 2006, is entitled *Modern Software Development Using C#/.NET*.