CONTENTS

1. Altering the Status Line

The first part of this guide not only alters the standard status line, but is an illustration of how TADS can send a simple message to a browser to call a JavaScript function.

To begin with, when making changes to the game's interface, it's very useful to create a folder named 'webuires' in the game's resource files. If you want to make changes to any of the various files which the game uses to define the look and behaviour of its webpage (e.g. main.html, main.css, main.js), you can simply replace the game file with one of the same name in webuires, and the game will override its own version with your new version. You can then add files to webuires (the normal way, by creating and editing files in that folder), and they will appear in the TADS 3 project.

In workbench:

- Using Windows, create a folder named 'webuires' in the same folder as your game (must be this exact name, all lower case)
- Make sure the game's 'project' window is open (click 'view' in the top menu then 'project' if necessary)
- Select 'Resource Files'
- Click on 'Project' then 'Add folder…'
- Browse to the 'webuires' folder, and click OK (it's also possible to just drag and drop 'webuires' into 'Resource Files' in the project window)

Modifying the status line is a good place to start in changing the overall look of a TADS 3 game. The main game window is divided into iFrames, one for the status bar and one for the command window in the basic game. The files which relate to the status bar are: status.t within the TADS adventure library, and statwin.htm, statwin.css and statwin.js in webuires. It's also worth noting, if you're trying to change things like the background colour of the status bar, that changes in statwin.css are often overridden by settings in another file, defaultPrefs.js, which you may also like to copy into the webuires folder if you are thinking of making changes.

As some of the most common questions for new IF players are along the lines of 'what is the aim?', or 'what am I supposed to be doing?' we can show this explicitly. So, we set up a class 'aim' and then create three instances for the demo game. We'll do this in a separate file for all the status line modifications, so create a file statusLineMod.t (or whatever name you like), add it to the project and put all the status line changes there.

```
#charset "us-ascii"
#include "adv3.h"

class Aim : Thing
text = ''
;
visitHallway: Aim
   text = 'Visit the hallway. '
;
```

```
visitKitchen: Aim
   text = 'Visit the kitchen. ' ;

eatBread: Aim
   text = 'Eat the bread. '
```

The 'goals' in the built-in hint system correspond to this, but are hard to access programmatically, so we set up a simpler system of 'aims' which can accessed directly.

Next, we can add a line in the 'enteringRoom' method of the rooms that changes the aim when the player enters. This is just for the demo game – you could use any trigger you want for your own game (e.g. a particular item coming into the player's inventory or a first meeting a specific NPC).

```
modify entryway
   enteringRoom(traveler)
   {
      statusLine.currentAim = visitHallway.text;
   }
;

modify hallway
   enteringRoom(traveler)
   {
      statusLine.currentAim = visitKitchen.text;
   }
;

modify kitchen
   enteringRoom(traveler)
   {
      statusLine.currentAim = eatBread.text;
   }
;

modify kitchen
   enteringRoom(traveler)
   {
      statusLine.currentAim = eatBread.text;
   }
;
```

The next thing is to modify the status line itself so that the 'aim' actually shows up. We're also going to stop displaying the room name in the status bar, since this is also shown in the command window, and we'll be setting up an auto-map later anyway. To do this, we need to modify showStatusLeft(), which displays the left portion of the status line. We also need to set up the 'startingAim' and 'currentAim' variables.

```
modify statusLine

startingAim = visitHallway

currentAim = startingAim.text

showStatusLeft()

   {

   "Aim: <<statusLine.currentAim>>";

   }

;
```

The status line should now update every time the aim changes (i.e. every time we enter a new room). However, it might be a good idea to draw the player's attention to this (so they notice the change), so we are going to make the status line flash every time it updates. The real point of this is to show how easily TADS can interact with JavaScript to produce changes in the display, using the .sendWinEvent method of the WebWindow class (of which statuslineBanner is a member).

First, we need to add the following code to the end of the statwin.css file in our version of the webuires folder. It creates a 'blinking' class, which, when added to an element through JavaScript, will cause the text to blink. You'll need to copy statwin.css to the webuires folder you created in order to make changes, if you're using Workbench TADS.

```
.blinking{

   animation:blinkingText 0.8s infinite;

}

@keyframes blinkingText{

   0%{    color: #000;    }

   49%{   color: transparent; }

   50%{   color: transparent; }

   99%{   color:transparent;  }
```

Now we can add the lines in statusLineMod.t that will cause the text to blink when the aim changes.

```
modify entryway
   enteringRoom(traveler)
   {
           statusLine.currentAim = visitHallway.text;
           statuslineBanner.sendWinEvent('<startBlink></startBlink>');
           statuslineBanner.sendWinEvent('<stopBlink></stopBlink>');
   }
;

modify hallway
   enteringRoom(traveler)
   {
           statusLine.currentAim = visitKitchen.text;
           statuslineBanner.sendWinEvent('<startBlink></startBlink>');
           statuslineBanner.sendWinEvent('<stopBlink></stopBlink>');

   }
;

modify kitchen
   enteringRoom(traveler)
   {
           statusLine.currentAim = eatBread.text;
           statuslineBanner.sendWinEvent('<startBlink></startBlink>');
           statuslineBanner.sendWinEvent('<stopBlink></stopBlink>');

   }
;
```

We also need to add a few lines of JavaScript to the file 'statwin.js' in webuires, so that the browser knows to listen for the <startBlink> and <stopBlink> xml messages (changes are highlighted). The very handy 'xmlHasChild(parent, name)' function is defined in util.js. The 'name' attribute corresponds to the xml message text (e.g. 'startBlink') sent from TADS.

```
function onGameEvent(req, resp)

{

   if (xmlHasChild(resp, "text"))

      setStatusText(xmlChildText(resp, "text"));

   if (xmlHasChild(resp, "appendText"))

      appendStatusText(xmlChildText(resp, "appendText"));

   if (xmlHasChild(resp, "resize"))
```

```
        parent.calcLayout();
    if (xmlHasChild(resp, "setsize"))
        parent.setWinLayout(this, xmlChildText(resp, "setsize"));
    if (xmlHasChild(resp, "startBlink"))
        startBlink();
    if (xmlHasChild(resp, "stopBlink"))
        setTimeout(stopBlink, 5000);
}

function startBlink()
{
    var element = document.getElementById("statusline");
    element.setAttribute("class", "blinking");
}

function stopBlink()
{
    var element = document.getElementById("statusline");
    element.removeAttribute("class", "blinking");
}
```

Not only should this make the status line blink, but we've just sent our first xml messages (the '<startBlink>' and '<stopBlink>') from TADS to the browser! So now you can call pretty much any function you can write in JavaScript. There are an almost infinite amount of JS tutorials and books on the web.

KEY POINT: The code that sends the message to JavaScript in the browser is the 'sendWinEvent' method of statuslineBanner. For further details, look this up in the library reference manual. The message is in the format webWindowName.sendWinEvent('<xmlMessage></xmlMessage>'). The Javascript that picks up the message is:

```
if (xmlHasChild(resp, "xmlMessage"))
        javaScriptFunction();
```

2. Adding a Menu

Next, we're going to build a menu so that the player has buttons for some frequently used commands. This illustrates how easy it is to call TADS commands from the browser.

In terms of window formatting, the easiest thing is just to add a new iFrame for the menu. Firstly, we need to create an object, 'MenuBar' in a game (.t) file. Go ahead and create a new file (I've called mine 'modifications.t). Then add the menubar object:

```
transient menuBar: WebWindow

   vpath = '/menubar.htm'

   src = 'webuires/menubar.htm'

;
```

The menu is a 'transient' object, which simply means it will persist if the player does a restore, undo or restart operation (see the object definitions chapter of the system manual for more on 'transient').

'WebWindow' is the class of object that controls and remembers the state of a window in the browser user interface. From the library reference manual:

'The HTML page is the expression of the window in the browser, and the WebWindow object is the expression of the same information in the game program. The two are different facets of the same conceptual UI object. The reason we need the two separate expressions is that the server controls everything, but the client has to do the actual display work, and the two parts of the program speak different languages - the server is TADS, and the client is HTML.'

'vpath' defines the virtual path to a resource (in this case, the HTML page for the menu bar). The game's default setting is to look for all WebResources (of which WebWindow is a subclass) in the webuires folder. 'src' defines the actual source location.

The next step is to modify initDisplay(). This function is responsible for initializing the display and creating the layout. The .createframe method creates an iFrame within the main game window. The first parameter (e.g. 'statuslineBanner', 'commandWin' below) refers to 'the WebWindow object that will be displayed within the iFrame. The method automatically loads the HTML resource from the WebWindow into the new iFrame (see webui.t). The next parameter (e.g. 'statusline', 'command') is the name of the window, which allows it to be referred to in order to position relatively to it (e.g. 'statusline.bottom'). The third parameter gives, in order, the left, top, right and bottom edges of the iFrame. Changes to the original version of initDisplay() are highlighted.

```
modify initDisplay()

{

   /* set up the command window and status line */

   webMainWin.createFrame(statuslineBanner, 'statusline',

                      '0, 0, 100%, 10%');

   statuslineBanner.init();

   statusLine.statusDispMode = StatusModeBrowser;

   webMainWin.createFrame(commandWin, 'command',

                      '0, statusline.bottom, 100%, 80%');

   webMainWin.createFrame(menuBar, 'menubar', '0, 90%, 100%, 10%');

}

;
```

Of course, this won't do anything unless we actually create the html page referred to in menuBar, so create 'menuBar.htm' in the webuires folder, and copy the following html code into it:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN">

<html>

<link rel="stylesheet" type="text/css" href="/webuires/menubar.css">

<script type="text/javascript" src="/webuires/util.js"></script>

<script type="text/javascript" src="/webuires/menubar.js"></script>


<div class="nav-bar">

   <div class="nav-bar">

   <div id="button-wrapper">

   <ul>

      <li>

         <a title="Quit the game"
href="http://seriousgames.atwebpages.com/Index.htm"><button
type="button">Quit</button></a>

</li>
```

```
<li>

          <a title="Restore Saved Game" href="Restore" onclick="javascript:return
gamehref(event,'Restore', 'main.command', this);">

          <button type="button">Restore</button></a>

       </li>

       &nbsp; &nbsp; &nbsp; &nbsp;

       <li>

          <a title="Go North" href="north" onclick="javascript:return gamehref(event,'north',
'main.command', this);">

          <button type="button">North</button></a>

       </li>

       <li>

          <a title="Go South" href="south" onclick="javascript:return gamehref(event,'south',
'main.command', this);">

          <button type="button">South</button></a>

       </li>

       <li>

          <a title="Go East" href="east" onclick="javascript:return gamehref(event,'east',
'main.command', this);">

          <button type="button">East</button></a>

       </li>

       <li>

          <a title="Go West" href="west" onclick="javascript:return gamehref(event,'west',
'main.command', this);">

          <button type="button">West</button></a>

       </li>

       &nbsp; &nbsp; &nbsp; &nbsp;

       <li>

          <a title="Repeat the description of the room or area you are in" href="look"

          onclick="javascript:return gamehref(event,'look', 'main.command', this);">

          <button type="button">Look around</button></a>

       </li>
```

```
    </ul>
</div>
</div>
</body>
</html>
```

For this to work properly, we also need to add the css file linked to at the beginning of menuBar.htm, so here it is (again, it’s easiest to just put it in the webuires folder along with everything else):

```
.nav-bar {
    flex: 1;
    padding: 1rem;
    height: 100%;
    width: 100%;
    position: fixed;
    float: right;
    top: 0;
    right: 0;
    background-color: #3d4342;
}

#button-wrapper {
    display: flex;
    align-items: center;
    height: 100%;
}
```

```
ul {
        list-style-type: none;
        text-align: left;
 }

a {
    color: white;
    font-family: sans-serif;
    font-size:24;
    text-decoration: none;
}

li a:hover {
    background-color: #555;
    color: #6af268;
}

li { display: inline; }

button {
    background-color: #6af268;
    border: none;
    color: black;
    padding: 5px 10px;
    text-align: center;
    font-size: 12px;
    border-radius: 8px;
    box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2), 0 6px 20px 0 rgba(0,0,0,0.19);
}
```

```
button:hover {

   background-color: green;

}
```

KEY POINT: The important thing to know is that the html ‘onclick’ event calls JavaScript to return the gamehref() function (defined in the util.js file). The function accepts the names of TADS commands (in quotes) as a parameter, which allows any game command to be called very easily from a button on the webpage.

3. Creating an Auto-Map

One of the key difficulties that puts off players new to IF from delving deeper into the genre is the difficulty of keeping track of one's location in the game. Most people simply can't process the NSEW directions into a coherent mental map of the game. Whilst a pencil-and-paper map *could* be drawn, this is simply too retro (and inconvenient) for the vast majority of people, so we're going to create an auto-map screen to show the player where they're supposed to be in the game. For an excellent example of how well a simple map can enhance a text-based adventure, check out Emily Short's 'Counterfeit Monkey' on IFDB.

NOTE: We're using CSS grid, so this won't display properly at all on the version of Internet Explorer that Workbench uses for local testing. Either run the file locally on a Mac, or upload and run through the TADS servers.

First, we need to give each room its own x- and y- co-ordinates. This is as simple as just adding an appropriately-named variable to each of the room objects:

```
entryway: Room 'Entryway'
…
   xcoord = 1
   ycoord = 1
…
   ;
   hallway: Room 'Hallway'
…
      xcoord = 1
      ycoord = 2
…
   ;
   kitchen: Room 'Kitchen'
…
      xcoord = 1
      ycoord = 3
…
   ;
```

Next, we need to create the html (mapwin.htm) that displays the map. It’s based on a simple tiling/x-y co-ordinate system. The central point of the map display moves with the player, so that their location is always displayed in the centre of the window. Of course, another (possibly simpler) way to do this would be to move the location indicator around a static map. The advantage of the method here is that it could be expanded to deal with a map of potentially any size. There are only three rooms in the demo game, but we are going to surround them with (unreachable) border tiles, making this a 3 by 5 tile map.

```
<!-- <!DOCTYPE HTML PUBLIC "-//W3C//Ddiv HTML 4.01 Strict//EN"> -->

<!DOCTYPE HTML>

<html>

<head>

<script type="text/javascript" src="/webuires/util.js"></script>

<script type="text/javascript" src="/webuires/mapWin.js"></script>


<link rel="stylesheet" type="text/css" href="/webuires/mapWin.css">

<body onload="javascript:mapWinInit();">


</head>

<body>

  <div class="wrapper">

    <div id="1,1"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="1,2"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="1,3"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="2,1"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="2,2"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="2,3"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="3,1"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="3,2"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="3,3"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="4,1"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="4,2"><img class='rsz' src='webuires/tiles/hidden.png'></div>

    <div id="4,3"><img class='rsz' src='webuires/tiles/hidden.png'></div>
    </div>

</body>

</html>
```

Next we need to create the css file (mapWin.css) which controls the grid layout of the map:

```
html, body, .wrapper {

    height: 100%;

}


body {

    background-color: black;

    font-size: 0px;

    font-family: Verdana, sans-serif;

    overflow: hidden;

}


img {

    width: 100%;

    height: 100%;

    object-fit: fill;

}


.wrapper {

    display: grid;

    grid-template-columns: 1fr 1fr 1fr;

    grid-template-rows: 25% 25% 25% 25%;

}


.rsz {
    display: flex;
```

Finally, of course, we need to write the JavaScript to do the actual work of calculating the display based on the player’s location. The code needs to be pasted in a file named mapWin.js in the webuires folder.

```
function mapWinInit()
    {
        utilInit();
        getInitState();
    }

var mapImages = new Array("entryway.png", "grass.png", "hallway.png", "hidden.png",
"kitchen.png")

var imageTable = new Array
    imageTable[0]=[1, 1, 1]
    imageTable[1]=[1, 0, 1]
    imageTable[2]=[1, 2, 1]
    imageTable[3]=[1, 4, 1]
    imageTable[4]=[1, 1, 1]
    imageTable[5]=[1, 1, 1]

var revealTable = new Array
    revealTable[0]=[1, 1, 1]
    revealTable[1]=[1, 1, 1]
    revealTable[2]=[1, 0, 1]
    revealTable[3]=[1, 0, 1]
    revealTable[4]=[1, 1, 1]
    revealTable[5]=[1, 1, 1]
```

```
function onGameEvent(req, resp) {
        if (xmlHasChild(resp, "map")) {
         var x=parseInt(xmlChildText(resp, "xcoord"));
         var y=parseInt(xmlChildText(resp, "ycoord"));
// use the x and y co-ordinates to adjust the reveal table if necessary
         revealTable[y][x] = 1;
// the lowest row of the -now- 4x5 grid
         tileIndex = imageTable[y-1][x+1]
         tile = mapImages[tileIndex]
         imgSrc='webuires/Tiles/' + tile;
           if (revealTable[y-1][x+1] == 1)
             {document.getElementById("4,3").innerHTML='<img src="' + imgSrc + '"/>';}
           else
             {document.getElementById("4,3").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
         tileIndex = imageTable[y-1][x-1]
         tile = mapImages[tileIndex]
         imgSrc='webuires/Tiles/' + tile;
           if (revealTable[y-1][x-1] == 1)
             {document.getElementById("4,1").innerHTML='<img src="' + imgSrc + '"/>';}
           else
             {document.getElementById("4,1").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
         tileIndex = imageTable[y-1][x]
         tile = mapImages[tileIndex]
         imgSrc='webuires/Tiles/' + tile;
           if (revealTable[y-1][x] == 1)
             {document.getElementById("4,2").innerHTML='<img src="' + imgSrc + '"/>';}
           else
             {document.getElementById("4,2").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
```

```
// the lower row of the -now- 4x5 grid
            tileIndex = imageTable[y][x+1]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
               if (revealTable[y][x+1] == 1)
                  {document.getElementById("3,3").innerHTML='<img src="' + imgSrc + '"/>';}
               else
                  {document.getElementById("3,3").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
            tileIndex = imageTable[y][x-1]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
               if (revealTable[y][x-1] == 1)
                  {document.getElementById("3,1").innerHTML='<img src="' + imgSrc + '"/>';}
               else
                  {document.getElementById("3,1").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
            tileIndex = imageTable[y][x]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
               if (revealTable[y][x] == 1)
                  {document.getElementById("3,2").innerHTML='<img src="' + imgSrc + '"/>';}
               else
                  {document.getElementById("3,2").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
// the centre row of the 4x5 grid
            tileIndex = imageTable[y+1][x+1]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
               if (revealTable[y+1][x+1] == 1)
                  {document.getElementById("2,3").innerHTML='<img src="' + imgSrc + '"/>';}
```

```
                else
                    {document.getElementById("2,3").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
            tileIndex = imageTable[y+1][x-1]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
                if (revealTable[y+1][x-1] == 1)
                    {document.getElementById("2,1").innerHTML='<img src="' + imgSrc + '"/>';}
                else
                    {document.getElementById("2,1").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
            tileIndex = imageTable[y+1][x]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
                if (revealTable[y+1][x] == 1)
                    {document.getElementById("2,2").innerHTML='<img src="' + imgSrc + '"/>';}
                else
                    {document.getElementById("2,2").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
// the upper row of the 4x5 grid
            tileIndex = imageTable[y+2][x+1]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
                if (revealTable[y+2][x+1] == 1)
                    {document.getElementById("1,3").innerHTML='<img src="' + imgSrc + '"/>';}
                else
                    {document.getElementById("1,3").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
            tileIndex = imageTable[y+2][x-1]
            tile = mapImages[tileIndex]
            imgSrc='webuires/Tiles/' + tile;
                if (revealTable[y+2][x-1] == 1)
```

```
                    {document.getElementById("1,1").innerHTML='<img src="' + imgSrc + '"/>';}
                else
                    {document.getElementById("1,1").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
              tileIndex = imageTable[y+2][x]
              tile = mapImages[tileIndex]
              imgSrc='webuires/Tiles/' + tile;
                if (revealTable[y+2][x] == 1)
                    {document.getElementById("1,2").innerHTML='<img src="' + imgSrc + '"/>';}
                else
                    {document.getElementById("1,2").innerHTML='<img
src="webuires/Tiles/hidden.png"/>';}
    }
}


function onGameState(req, resp)
    {}
```

Now we need to set up the screen layout in TADS so that we have a subwindow for the map. To do this, we need to create the map window object (mapWin), in a similar way to how we created an object for the menu, and further modify the initDisplay() function.

Notice that we've now added in the highlighted line that creates a frame for the map window, and also altered the width of the command window to fit the map in.

```
transient mapWin: WebCommandWin, WebBannerWin
  vpath = '/mapWin.htm'
  src = 'mapRes/mapWin.htm'


;
modify initDisplay()


{
    /* set up the command window and status line */
    webMainWin.createFrame(statuslineBanner, 'statusline',
                    '0, 0, 100%, 10%');

    statuslineBanner.init();

    statusLine.statusDispMode = StatusModeBrowser;

    webMainWin.createFrame(commandWin, 'command','60%, statusline.bottom, 40%, 80%');

    webMainWin.createFrame(mapWin, 'map','0, statusline.bottom, 60%, 80%');

    webMainWin.createFrame(menuBar, 'menubar', '0, 90%, 100%, 10%');

    menuBar.init();

  }
```

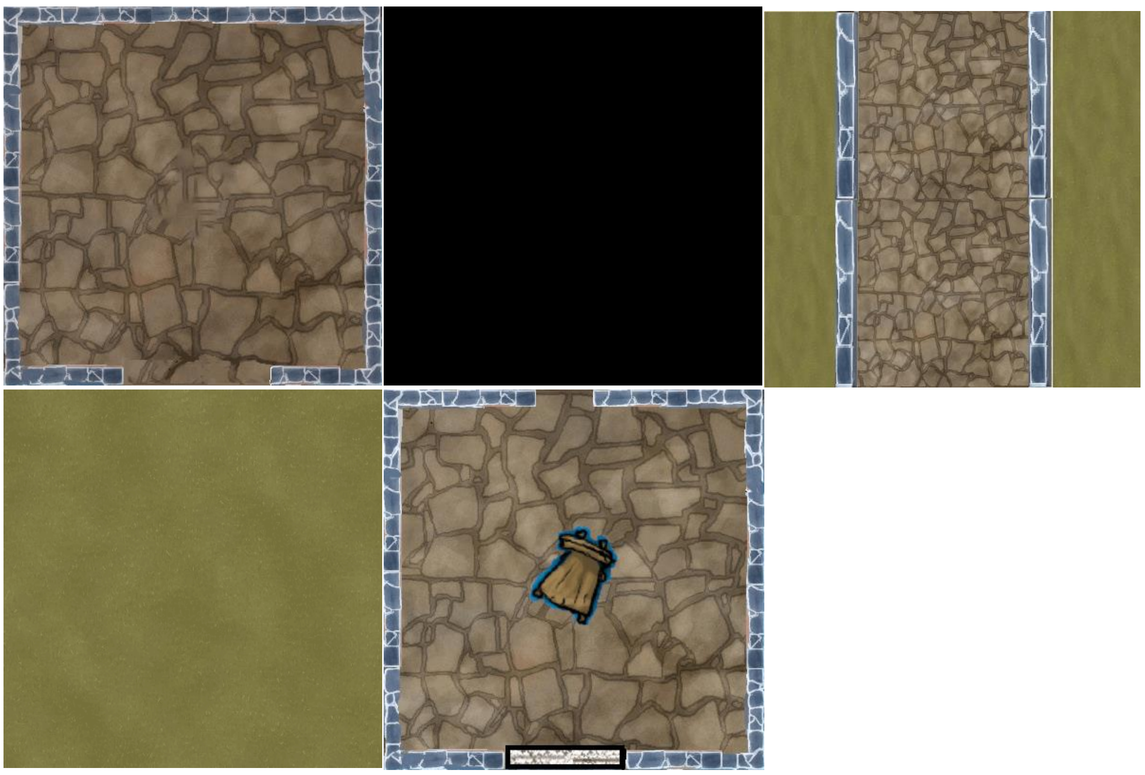
Also, we need to modify the 'travelVia' action so that it sends a message to the browser with the updated map co-ordinates every time the player moves from one room to another:

```
modify TravelViaAction

   execAction()

   {

           inherited;
          mapWin.sendWinEvent('<map><ycoord><<me.location.ycoord>></ycoord><xcoord><<
          me.location.xcoord>></xcoord></map>');

   }
```

Notice that useful 'sendWinEvent' method for communicating with the server again. Finally, we need to add a few lines of code to the showIntro() methond so that the map displays immediately before the player enters the first room:

```
showIntro()

   {

      "Welcome to the TADS 3 Starter Game!\b";

      "<.inputline>";

      "Press ENTER or <<aHref('', 'CLICK HERE ')>>to continue.";

      "<./inputline>";

      do

      {
            inputManager.getInputLine(nil, nil);

        }

        while (nil);

        cls();


mapWin.sendWinEvent('<map><ycoord><<me.location.ycoord>></ycoord><xcoord><<me.locati
on.xcoord>></xcoord></map>');

    }
```

Of course, it's also necessary to create the image tiles themselves and put them in webuires/Tiles. They need to be named exactly as in the .js file ("entryway.png", "grass.png", "hallway.png", "hidden.png", "kitchen.png")



KEY POINT: Hopefully, this guide has been somewhat useful in illustrating the ease and power with which TADS WebUI can interface with Html, CSS and Javascript in the browser. The game provided is only meant as a demonstration, and there's no need to restrict what you do to simply altering the interface – the whole power of the web is at your disposal.