Loading the Mission Briefing Screen

The first thing we will do is add the HTML code for the mission briefing screen into the `gamecontainer div` within our HTML file. The `gamecontainer div` will now look like Listing 6-7.

Listing 6-7. Adding the Mission Briefing Screen (index.html)

```
<div id="gamecontainer">
    <div id="gamestartscreen" class="gamelayer">
        <span class="game-title">LAST<br>COLONY</span>
        <span class="game-option" onclick = "singleplayer.start();">Campaign</span>
        <span class="game-option" onclick = "multiplayer.start();">Multiplayer</span>
    </div>

    <div id="missionbriefingscreen" class="gamelayer">
        <img src="images/screens/interface-left.png" class="left-panel" draggable="false">
        <img src="images/screens/interface-right-briefing.png" class="right-panel"
        draggable="false">
        <input type="button" id="entermission" onclick = "singleplayer.play();">
        <input type="button" id="exitmission" onclick = "singleplayer.exit();">
        <div id="missionbriefing"></div>
    </div>
```

```
    <div id="loadingscreen" class="gamelayer">
        <div id="loadingmessage"></div>
    </div>
</div>
```

The `missionscreen` `div` contains two buttons; they are for entering the mission screen and exiting the mission screen. It also contains a `missionbriefing` `div` that we will use to display the briefing message. Additionally it contains two images for the left and right side of the interface background area. We set the `draggable` attribute of these images to `false` to prevent them from being dragged around if the player accidentally clicks one of them.

Now that we have the HTML markup in place, we need to add the CSS styles for the mission screen into `styles.css`, as shown in Listing 6-8.

Listing 6-8. CSS Style for Mission Screen

```
/* Mission Briefing Screen */

#missionbriefingscreen {
    background: url("images/screens/interface-middle.png");
}

input[type="button"] {
    border-width: 0;
    outline: none;

    background-color: transparent;
    background-repeat: no-repeat;
    background-image: url("images/buttons.png");

    cursor: pointer;
}

.left-panel {
    left: 0;
    top: 0;
    position: absolute;
}

.right-panel {
    right: 0;
    top: 0;
    position: absolute;
}

#entermission {
    height: 52px;
    width: 188px;

    position: absolute;
    top: 82px;
    left: 3px;

    background-position: -4px -4px;
}
```

```
#entermission:disabled, #entermission:active {
    background-position: -196px -4px;
}

#exitmission {
    height: 52px;
    width: 72px;

    position: absolute;
    top: 82px;
    right: 164px;

    background-position: -4px -64px;
}

#exitmission:disabled, #exitmission:active {
    background-position: -84px -64px;
}

#missionbriefing {

    position: absolute;

    top: 170px;
    left: 40px;
    right: 220px;
    height: 270px;

    text-align: justify;

    color: rgb(130, 150, 162);
    text-shadow: -1px 1px black;

    font-size: 16px;
    font-family: "Courier New", Courier, monospace;
}
```

We define a new background for the mission briefing screen that fits into the center, behind the left and right images defined in the HTML. This center background automatically repeats itself to fit all available space. This way, the briefing screen can automatically adjust for different aspect ratios by keeping the left and right side of the interface the same size and expanding the center area as needed to adjust for different aspect ratios.

We then position the button and `div` elements to fit on top of the background. We keep different images for the enabled and disabled states of the buttons but store all of these sprites in a single sprite-sheet image file (`buttons.png`). Note that we specify left and right positions so that the buttons and briefing area automatically position and scale appropriately when the game container width changes.

Now that the mission briefing layer is in place, we will implement the `singleplayer` object inside `singleplayer.js`, as shown in Listing 6-9.

Listing 6-9. Implementing the Basic singleplayer Object (singleplayer.js)

```
var singleplayer = {

    // Begin single-player campaign
    start: function() {
        // Hide the starting menu screen
        game.hideScreens();

        // Begin with the first level
        singleplayer.currentLevel = 0;

        // Start initializing the level
        singleplayer.initLevel();
    },

    currentLevel: 0,
    initLevel: function() {
        game.type = "singleplayer";
        game.team = "blue";

        // Don't allow player to enter mission until all assets for the level are loaded
        var enterMissionButton = document.getElementById("entermission");

        enterMissionButton.disabled = true;

        // Load all the items for the level
        var level = levels.singleplayer[singleplayer.currentLevel];

        game.loadLevelData(level);

        // Enable the Enter Mission button once all assets are loaded
        loader.onload = function() {
            enterMissionButton.disabled = false;
        };

        // Update the mission briefing text and show briefing screen
        this.showMissionBriefing(level.briefing);
    },

    showMissionBriefing: function(briefing) {
        var missionBriefingText = document.getElementById("missionbriefing");

        // Replace \n in briefing text with two <br> to create next paragraph
        missionBriefingText.innerHTML = briefing.replace(/\n/g, "<br><br>");

        // Display the mission briefing screen
        game.showScreen("missionbriefingscreen");
    },
```

```
    exit: function() {
        // Display the main game menu
        game.hideScreens();
        game.showScreen("gamestartscreen");
    },

};
```

We define a `singleplayer` object with four methods: `start()`, `initLevel()`, `showMissionBriefing()`, and `exit()`.

The `start()` method first hides all game layers and sets `singleplayer.currentLevel` to 0, which refers to the first level in the `maps.singleplayer` array that we defined earlier. Finally, it calls the `singleplayer.initLevel()` method that we will call every time we want to load a level.

The `initLevel()` method first sets the `game.type` and `game.team` variables to `singleplayer` and `blue`, respectively. We will use these values later once the game starts running. It then temporarily disables the Enter Mission button on the screen and starts loading the level assets. Once the assets are loaded, the Enter Mission button is enabled so that the player can click it and enter the game. Finally, it calls the `showMissionBriefing()` method, which puts the level briefing inside the `missionbriefing div` and displays the `missionbriefingscreen div`.

The `exit()` method hides all the game layers and takes us back to the main menu.

■ **Note** We replace carriage returns with `<br>` tags so that they show up in the HTML. This way, we can easily break out the mission briefing into multiple paragraphs if we want.

Next we will define the `loadLevelData()` method inside the `game` object as shown in Listing 6-10.

Listing 6-10. Loading the Level (game.js)

```
loadLevelData: function(level) {
    game.currentLevel = level;
    game.currentMap = maps[level.mapName];

    // Load all the assets for the level starting with the map image
    game.currentMapImage = loader.loadImage("images/maps/" + maps[level.mapName].mapImage);
},
```

For now, we just store the level and map objects and load the current level's map image. This method will eventually load all the assets for a given level.

When we load the game in the browser and click the Campaign option, we should see the mission briefing screen for the first level, as shown in Figure 6-4.

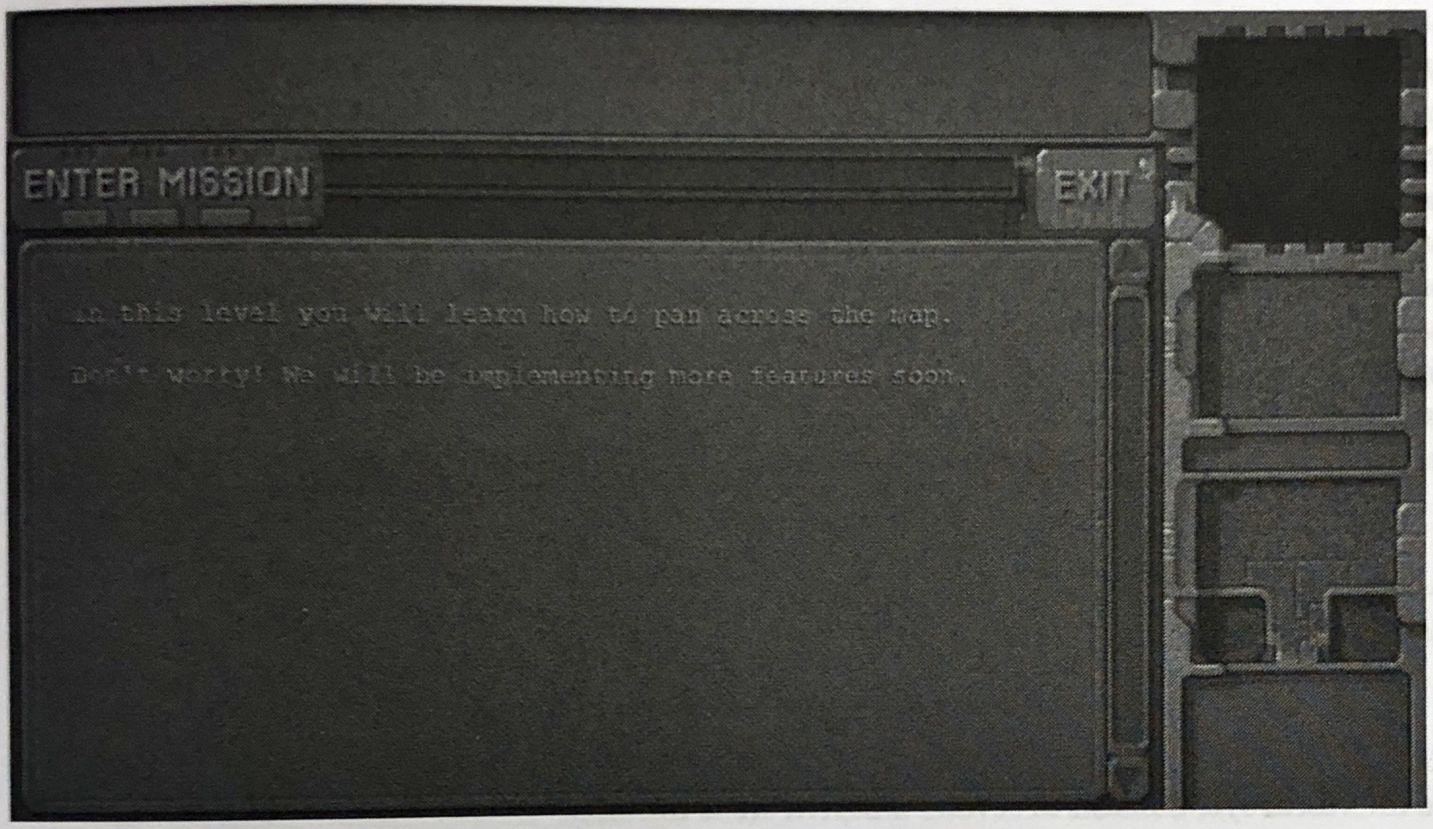

Figure 6-4. *The mission briefing screen for our first level*

The advantage of displaying the briefing screen while loading the assets in the background is that players can spend their time reading the mission briefing while waiting for all the assets to load. You will notice that the screen automatically adjusts to different aspect ratios and screen sizes by expanding the center region while keeping the left and right sides the same.

Clicking the Exit button should take us back to the main menu. Once the level data has loaded completely, the enter mission button will get enabled. We still can't enter the mission until we implement the actual game interface and the game animation and drawing loops, which is what we will be doing next.