CHAPTER 6

Creating the RTS Game World

Real-time strategy (RTS) games combine fast-paced tactical combat, resource management, and economy building within a defined game world.

A typical RTS game consists of a map of a world with different units, buildings, and terrain, as well as an interface to control and manipulate these elements. The player uses the interface to handle tasks such as gathering resources, constructing buildings, and creating an army, and then manages the army to achieve a set of goals defined for each level.

Although these games have an extensive history, the RTS genre was largely popularized by the games released by Westwood Studios and Blizzard Entertainment in the 1990s. Westwood's *Dune II* and *Command & Conquer* series are considered classics that helped define the genre. With its engaging story line and addictive multiplayer mode, Blizzard's *StarCraft* went on to elevate RTS gaming to an e-sport with professional competitive tournaments held around the world.

HTML5 now makes it possible to bring this genre to the browser in a way that wasn't possible earlier. In fact, one of my better-known game programming–related achievements a few years ago was single-handedly re-creating the original *Command & Conquer* entirely in HTML5. While generating a lot of buzz on the Web, this project proved beyond a doubt that HTML5 was now ready for the next generation of games.

Over the next few chapters, we will use what you learned in previous chapters and build upon it to create our own RTS game. We will define a game world with buildings, units, and an overarching story line to create an engaging single-player campaign. We will then use HTML5 WebSockets to add real-time multiplayer support to our game.

Most of the artwork for this game has been provided by Daniel Cook (`www.lostgarden.com`), who originally designed this art for an unreleased RTS title called *Hard Vacuum*. We will be reusing the artwork that he has graciously shared but will create our own game concept. Our game, *Last Colony*, will be about a small band of survivors on a planet that has just been attacked. We will explore the story and gameplay in more detail over the next few chapters.

While developing this game, we will keep the code as generic and customizable as possible so that you can later reuse this code to build your own ideas. If you would like to follow along with the book, you can find all the necessary starting assets, including the images and the audio, inside the `assets` folder of this chapter's code.

So, let's get started.

Basic HTML Layout

Like the previous game we developed, *Froot Wars*, our RTS game will consist of several layers. The following are the first few layers that we will define:

- *Splash screen and main menu*: Shown when the game loads and allows the player to select campaign or multiplayer mode
- *Loading screen*: Shown whenever the game is loading assets

A. R. Shankar, *Pro HTML5 Games*, DOI 10.1007/978-1-4842-2910-1_6

- *Mission screen*: Shown before a mission starts, with instructions for the mission
- *Game interface screen*: The main game screen that includes the map area and a dashboard for controlling the game

We will define more screens as needed in later chapters. We will be organizing all of the artwork inside an images folder. Unlike the previous game, we will break the JavaScript code into several files (such as `buildings.js`, `vehicles.js`, `levels.js`, and `common.js`) inside the `js` folder so as to make the code easier to maintain.

Creating the Splash Screen and Main Menu

We will start by creating an HTML file and adding the markup for our containers, as shown in Listing 6-1.

Listing 6-1. Basic HTML Skeleton with Layers Added (index.html)

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">

        <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1,
        minimum-scale=1, width=device-width">

        <title>Last Colony</title>

        <script src="js/common.js" type="text/javascript"></script>
        <script src="js/game.js" type="text/javascript"></script>
        <script src="js/mouse.js" type="text/javascript"></script>
        <script src="js/singleplayer.js" type="text/javascript"></script>
        <script src="js/levels.js" type="text/javascript"></script>

        <link rel="stylesheet" href="styles.css" type="text/css">
    </head>
    <body>
        <div id="wrapper">
            <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="loadingscreen" class="gamelayer">
                    <div id="loadingmessage"></div>
                </div>
            </div>
        </div>
    </body>
</html>
```

The code first refers to the external JavaScript and CSS files we will be using. We will be creating and implementing all these JavaScript files over the course of this game. Within a main `wrapper div`, we also define a `gamecontainer div` that contains our first two game layers: `gamestartscreen` and `loadingscreen`.

The next thing we will do is define the initial style for the game container inside `styles.css`, as shown in Listing 6-2.

Listing 6-2. Initial Style Sheet (styles.css) for Game Container and Layer

```
body {
    background: #090009;

    /* Disable scroll bars */
    overflow: hidden;

    /* Disable long touch hold select on mobile browsers */
    -webkit-touch-callout: none !important;
}

#wrapper {
    position: absolute;

    /* Wrapper covers entire window height and width */
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;

    /* Prevent the ugly blue highlighting from accidental selection of text */
    user-select: none;
}

#gamecontainer {
    /* Start with a default width that we can change later */
    width: 640px;
    height: 480px;

    /* Use a wider splash screen and center it within the container */
    background: url("images/screens/splashscreen.png");
    background-position: center;
    background-repeat: no-repeat;

    /* Center the game container relative to outer wrapper */
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
    transform-origin: center center;
}
```

```
.gamelayer {
    width: 100%;
    height: 100%;
    position: absolute;
    display: none;
}
```

In this code, we first start with setting the body background color and disabling the scrollbar and long press context menus for mobile devices.

Next, we center the game container within the `wrapper div` and assign a background splash screen. We use a wide splash screen image so our game can dynamically adjust to different aspect ratios later, just like we did in our previous game, Froot Wars. For now, however, we assign the container an initial size of 640px by 480px.

Finally, we set the `gamelayer` class to position all the game layers on top of each other, assign them the same dimensions as the container, and hide them by default.

When we load `index.html` in the browser, we should now see our new splash screen, as shown in Figure 6-1.

Figure 6-1. *The initial game splash screen*

Now that the splash screen is in place, we can implement the main menu screen and the game loading screen.

We will start by setting up the asset loader using the exact same code as we did in our previous game. We will place this code inside a separate file called `common.js`, as shown in Listing 6-3.