Creating Our First Level

There are many viable approaches to defining maps or levels for our game.

One approach is to store all the information about the map terrain as metadata and then assemble all the necessary images for the terrain on the browser at runtime to draw the map. This approach, while slightly cumbersome, allows the use of sprite sheets for the map terrain, reducing the size of the map.

Another approach, which is slightly simpler, is to store the basic map as a large image with the terrain drawn out using our own level-designing tool. We then need to store only the location of the map image along with metadata such as game entities and mission objectives. This is the approach that we will be using for our game.

Map images can be designed very quickly by using general-purpose tile map–editing software such as Tiled (`www.mapeditor.org`). Tiled is an excellent free tool that is available for several operating systems including Windows, Mac, and Linux. Once you start the application, you can load the sprite sheet for the terrain as a tile set and then use it to draw the map as if you were using a painting application (see Figure 6-3).

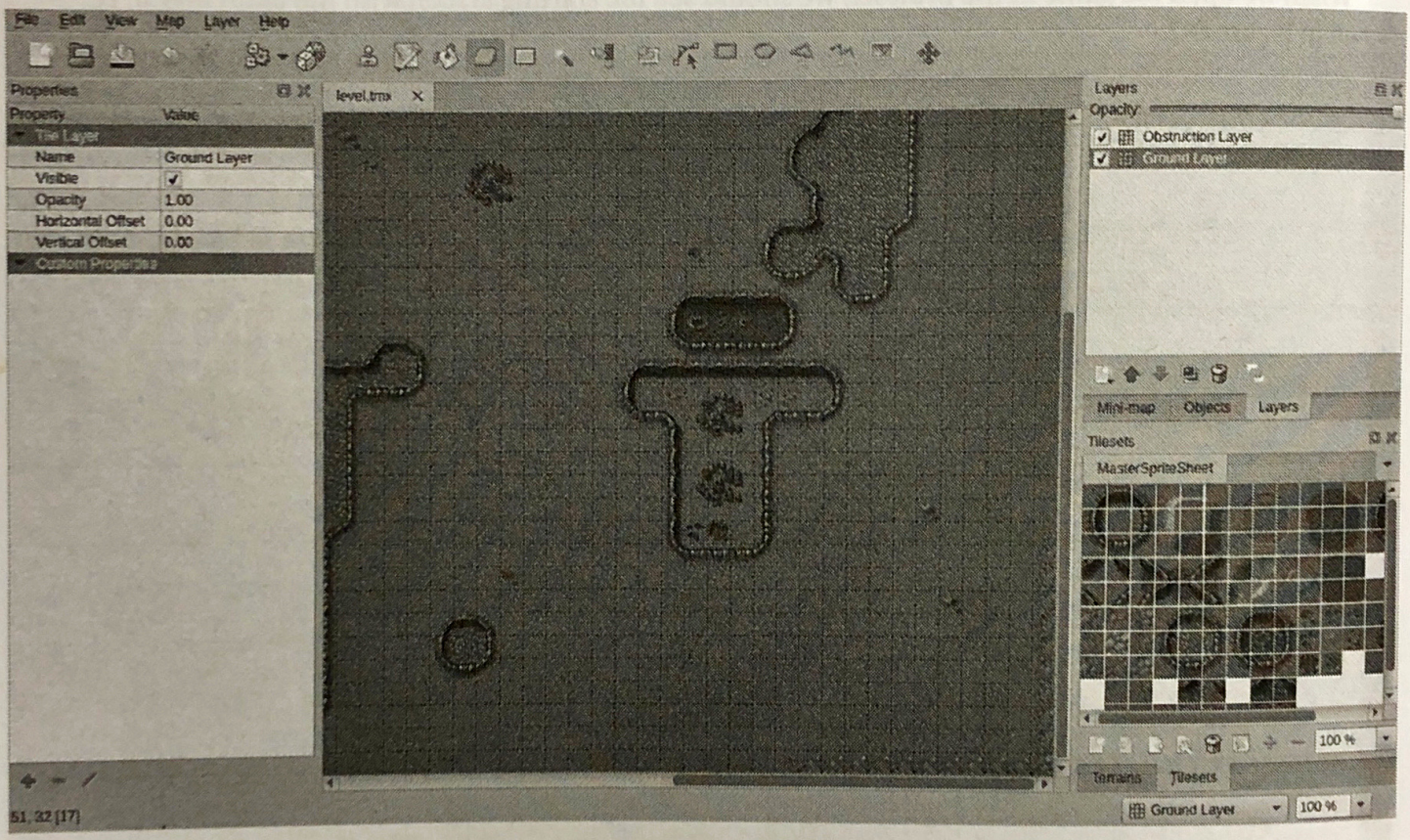

Figure 6-3. *Drawing a map using Tiled*

Note that we can use Tiled's layer feature to design the level in two layers. All the game terrain and obstructions are stored in a separate Obstruction layer. When the level JSON file is generated, we can use the metadata to identify areas of the map that are impassable or obstructed.

Once you draw the map, you can export it to several different file formats such as PNG images or JSON metadata.

You won't need to use this tool to follow along with the book since the maps we need for our game have already been generated. However, if you are considering developing your own game, I strongly recommend exploring Tiled's features.

All the files that you need, the exported level images, the JSON metadata, the master sprite sheet, and the Tiled project file are inside the `level` folder of this chapter's code. The exported images include a debug version of the map with all the grid lines between tiles drawn in.

The `level` folder also contains a `convert-levels.js` file, which is a Node.js script that takes the exported `level.json` file and creates a `level-obstructed-terrain.json` file for use within our game.

■ **Note** The Tiled editor's JSON format contains references to the sprite sheet and offsets for all the tiles it uses. This means you can also use the JSON files to create maps that are assembled at runtime (instead of the preassembled ones we are creating).

Once we have our first map image designed, we will need to create the basic metadata describing the level. We will do this inside `levels.js`, as shown in Listing 6-6.

Listing 6-6. Defining the Basic Level Metadata (levels.js)

```
/* Details of the maps used by the levels */
var maps = {
    "plains": {
        "mapImage": "plains-debug.png",

        /* Terrain Data - Auto Generated By level/convert-levels.js */
        "mapGridWidth": 60,
        "mapGridHeight": 40,
        "mapObstructedTerrain": [[0, 0], [1, 0], [2, 0], [26, 0], [27, 0], /* Extremely huge
        array snipped for brevity */ [58, 39], [59, 39]],
    }
};

/* The actual levels played in the game */
var levels = {
    "singleplayer": [
        {
            "name": "Introduction",
            "briefing": "In this level you will learn how to pan across the map.\n\nDon't
            worry! We will be implementing more features soon.",

            /* Map Details */
            "mapName": "plains",
            "startX": 4,
            "startY": 4,
        }
    ],

    "multiplayer": [

    ]
};
```

We first define a `maps` object that contains details of the one map we have generated, named "plains." This includes the map image, and some terrain data that has been generated by `convert-levels.js`—the width and height of the map, as well as an extremely huge `mapObstructedTerrain` array, which contains the x and y coordinates of every grid square in the map that is impassable or obstructed.

I have snipped the array in Listing 6-6 because there is absolutely no point in showing you the entire array with several hundred numbers in it. You will find the complete `mapObstructedTerrain` array inside the `level-obstructed-terrain.json` file, as well as the finished game code. When you make your own maps using Tiled, you can use the `convert-level.js` script to generate the data for them.

The map image is broken down into a grid of squares 20 pixels wide by 20 pixels high (based on the size of the tiles we are using). For now, we are using a "debug" version of the map that has the grid drawn on top of the map. This will make it easier for us to position elements inside the level while we are building the game.

Next, we create a `levels` object that will contain all the levels within our game, with arrays for single-player and multiplayer.

The `singleplayer` array currently contains details for only one level. This array will eventually contain all our single-player campaign levels in chronological order. When the single-player campaign is started, the `singleplayer` object will load the first level in this array and then proceed down the list as the player completes each level.

The details that we store for the level include the level name and a mission briefing that we will display before we start the level.

We then refer to the plains map that we have defined earlier. By using this system of separating maps and levels, we can have multiple levels share the same map as needed, depending on the game's story line.

The starting map coordinates (`startX` and `startY`) let us decide where to position the screen on the map when we start the level using the grid coordinates.

Now that we have a simple map defined, we will set up the `singleplayer` object to display the mission briefing screen.