Implementing Map Panning

The first thing we will do is set up mouse input by creating a mouse object inside mouse.js (see Listing 6-18).

Listing 6-18. Setting Up the mouse Object

```
var mouse = {
    init: function() {
        // Listen for mouse events on the game foreground canvas
        let canvas = document.getElementById("gameforegroundcanvas");

        canvas.addEventListener("mousemove", mouse.mousemovehandler, false);

        canvas.addEventListener("mouseenter", mouse.mouseenterhandler, false);
        canvas.addEventListener("mouseout", mouse.mouseouthandler, false);

        mouse.canvas = canvas;
    },

    // x,y coordinates of mouse relative to top-left corner of canvas
    x: 0,
    y: 0,

    // x,y coordinates of mouse relative to top-left corner of game map
    gameX: 0,
    gameY: 0,

    // game grid x,y coordinates of mouse
    gridX: 0,
    gridY: 0,

    calculateGameCoordinates: function() {
        mouse.gameX = mouse.x + game.offsetX ;
        mouse.gameY = mouse.y + game.offsetY;

        mouse.gridX = Math.floor((mouse.gameX) / game.gridSize);
        mouse.gridY = Math.floor((mouse.gameY) / game.gridSize);
    },
```

```
    setCoordinates: function(clientX, clientY) {
        let offset = mouse.canvas.getBoundingClientRect();

        mouse.x = (clientX - offset.left) / game.scale;
        mouse.y = (clientY - offset.top) / game.scale;

        mouse.calculateGameCoordinates();
    },

    // Is the mouse inside the canvas region
    insideCanvas: false,

    mousemovehandler: function(ev) {
        mouse.insideCanvas = true;
        mouse.setCoordinates(ev.clientX, ev.clientY);
    },

    mouseenterhandler: function() {
        mouse.insideCanvas = true;
    },

    mouseouthandler: function() {
        mouse.insideCanvas = false;
    },
};
```

We start by defining an `init()` method that assigns event listeners on the foreground canvas for a few mouse events: `mousemove`, `mouseenter`, and `mouseout`. We also save a reference to the canvas in `mouse.canvas`.

Next, we define variables to store the mouse coordinates relative to the canvas (`x,y`), relative to the map (`gameX,gameY`), and in terms of the map grid (`gridX,gridY`). We also define several variables to store the mouse state (`buttonPressed`, `dragSelect`, and `insideCanvas`). We also define a method called `calculateGameCoordinates()` that converts the mouse x and y coordinates to game coordinates.

Next, we define a helper method called `setCoordinates()`, which is called whenever the mouse is moved. We use the mouse event `clientX` and `clientY` properties, which are coordinates relative to the top of the window, and convert them to be relative to the game canvas, while adjusting for the game scale. We then call `calculateGameCoordinates()` so that the game-related coordinates are updated as well.

Finally, we define the actual event handler methods: `mousemovehandler`, `mouseenterhandler`, and `mouseouthandler`.

Whenever the mouse is moved, we set the `insideCanvas` flag to `true` since we know the mouse is somewhere inside the canvas. We then call `setCoordinates()` to update the stored mouse coordinates.

When the mouse enters the canvas, we set the `insideCanvas` flag to true, and when the mouse leaves the canvas, we set this flag to `false`. This way we can always know whether the mouse is inside or outside our game canvas area using `mouse.insideCanvas`, and can handle it as needed.

Now that we have set up our `mouse` object, we will modify our `game` object inside `game.js` to use the mouse. The first thing we need to do is call the `mouse.init()` method from inside the `game.init()` method. The updated `game.init()` method will look like Listing 6-19.

Listing 6-19. Calling mouse.init() from Inside game.init() (game.js)

```
init: function() {
    // Initialize objects
    loader.init();
    mouse.init();

    // Initialize and store contexts for both the canvases
    game.initCanvases();

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

Next we will define a handlePanning() method inside the game object, as shown in Listing 6-20.

Listing 6-20. Defining the handlePanning() Method (game.js)

```
// Distance from edge of canvas at which panning starts
panningThreshold: 80,
// The maximum distance to pan in a single drawing loop
maximumPanDistance: 10,

handlePanning: function() {

    // Do not pan if mouse leaves the canvas
    if (!mouse.insideCanvas) {
        return;
    }

    if (mouse.x <= game.panningThreshold) {
        // Mouse is at the left edge of the game area. Pan to the left.
        let panDistance = game.offsetX;

        if (panDistance > 0) {
            game.offsetX -= Math.min(panDistance, game.maximumPanDistance);
            game.refreshBackground = true;
        }
    } else if (mouse.x >= game.canvasWidth - game.panningThreshold) {
        // Mouse is at the right edge of the game area. Pan to the right.
        let panDistance = game.currentMapImage.width - game.canvasWidth - game.offsetX;

        if (panDistance > 0) {
            game.offsetX += Math.min(panDistance, game.maximumPanDistance);
            game.refreshBackground = true;
        }
    }
```

```
    if (mouse.y <= game.panningThreshold) {
        // Mouse is at the top edge of the game area. Pan upwards.
        let panDistance = game.offsetY;

        if (panDistance > 0) {
            game.offsetY -= Math.min(panDistance, game.maximumPanDistance);
            game.refreshBackground = true;
        }
    } else if (mouse.y >= game.canvasHeight - game.panningThreshold) {
        // Mouse is at the bottom edge of the game area. Pan downwards.
        let panDistance = game.currentMapImage.height - game.offsetY - game.canvasHeight;

        if (panDistance > 0) {
            game.offsetY += Math.min(panDistance, game.maximumPanDistance);
            game.refreshBackground = true;
        }
    }

    if (game.refreshBackground) {
        // Update mouse game coordinates based on new game offsetX and offsetY
        mouse.calculateGameCoordinates();
    }
},
```

We start by defining two new variables, `panningThreshold` and `maximumPanDistance`, that store how close to the canvas edge the mouse cursor needs to be for panning to occur and how fast the panning can be.

The `handlePanning()` method itself checks to see whether the mouse is inside the canvas, near any of the edges of the canvas, and that there is still some map left to pan. If all the conditions are met, we adjust the offsets in the appropriate direction by the panning distance, and set the `refreshBackground` flag to `true`. This will let the `drawBackground()` method know that the background needs to be redrawn.

Finally, if the map was panned, we also refresh the mouse game coordinates since they will change any time the map pans.

The last change that we will make to the game object is calling the `handlePanning()` method from inside `game.drawingLoop()`. The final `drawingLoop()` method will look like Listing 6-21.

Listing 6-21. Calling game.handlePanning()(game.js)

```
drawingLoop: function() {
    // Pan the map if the cursor is near the edge of the canvas
    game.handlePanning();

    // Draw the background whenever necessary
    game.drawBackground();

    // Call the drawing loop for the next frame using request animation frame
    if (game.running) {
        requestAnimationFrame(game.drawingLoop);
    }
},
```

At this point, if we run the game, we should be able to pan around the map by moving the mouse near the edges of the canvas so that we can explore the entire map, as shown in Figure 6-6.

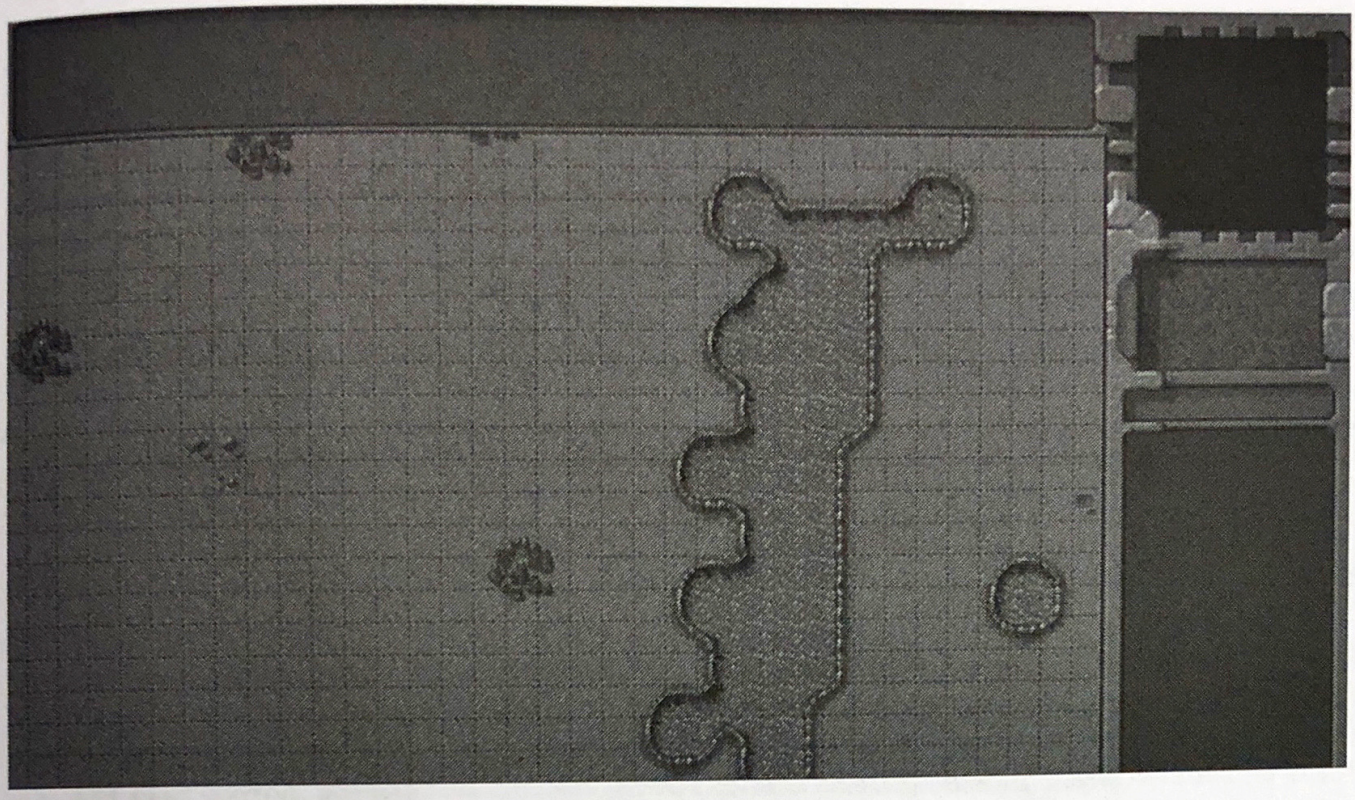

Figure 6-6. *Panning around the map*

Summary

In this chapter, we set out to develop the basic framework for our RTS game.

Just like in Chapter 2, we implemented a splash screen and a starting menu. We also used ideas from Chapter 5 to make the game responsive and automatically adjust to different window sizes and aspect ratios.

We looked at creating a map using the Tiled editor and exporting it as an image. We then created our first level by combining the map image with some basic level metadata.

We implemented a `singleplayer` object that loads map data and displays a mission briefing screen. We then created the game interface screen and set up the animation and drawing loops for the game so we could load and see the initial map on the canvas. Finally, we captured and used mouse events to let the user pan around the level.

While we have a lot of the essential elements of our game world in place, we are still missing the actual entities to interact with, such as buildings and vehicles.

In the next chapter, we will start adding these different entities to our level. We will draw them on the screen using sprite sheets and animation states. We will then set up a framework for selecting these entities so we can interact with them.