UNIVERSITY OF MALAYA

Software Maintenance and Evolution

WIF3005

Individual Final Alternative Assignment

By
Mohammad Mahdinur Rahman

1. Game Loop Component

Source File: common.js **Component Name**: Game.run **Original Purpose**: Racing game animation and game state management

The Game.run component provides a fundamental game engine loop that handles, Frame timing with requestAnimationFrame, Asset loading coordination, input handling, State updates, Render scheduling, Performance monitoring

Code

```javascript
var Game = {
  run: function(options) {
    Game.loadImages(options.images, function(images) {
      options.ready(images);
      Game.setKeyListener(options.keys);

      var canvas = options.canvas,
          update = options.update,
          render = options.render,
          step   = options.step,
          now    = null,
          last   = Util.timestamp(),
          dt     = 0,
          gdt    = 0;

      function frame() {
        now = Util.timestamp();
        dt  = Math.min(1, (now - last) / 1000);
        gdt = gdt + dt;

        while (gdt > step) {
          gdt = gdt - step;
          update(step);
        }

        render();
        stats.update();
        last = now;
        requestAnimationFrame(frame, canvas);
      }
      frame();
    });
  }
};

```

Explanation

1. **Time Delta Calculation**:
   - Tracks time between frames using Util.timestamp()
   - Calculates delta time (dt) for smooth animations
   - Ensures consistent game speed regardless of frame rate
2. **Update Loop**:
   - Accumulates time in gdt (game delta time)

- Runs update logic in fixed time steps
- Prevents spiral of death in slow frame rates

3. **Render Scheduling**:
   - Calls render function after updates
   - Uses requestAnimationFrame for optimal performance
   - Maintains synchronization with browser refresh rate

2. Render Component

Source File: common.js **Component Name**: Render.sprite **Original Purpose**: Racing game sprite and scene rendering

The Render component handles sprite and graphical element drawing with features for: Sprite positioning and scaling, Visual effects (fog, depth), Screen space calculations, Sprite clipping and culling

Code

```
var Render = {
  sprite: function(ctx, width, height, resolution, roadWidth,
                   sprites, sprite, scale, destX, destY, offsetX, offsetY, clipY) {
    var destW = (sprite.w * scale * width/2) * (SPRITES.SCALE * roadWidth);
    var destH = (sprite.h * scale * width/2) * (SPRITES.SCALE * roadWidth);

    destX = destX + (destW * (offsetX || 0));
    destY = destY + (destH * (offsetY || 0));

    var clipH = clipY ? Math.max(0, destY+destH-clipY) : 0;
    if (clipH < destH)
      ctx.drawImage(sprites, sprite.x, sprite.y,
                   sprite.w, sprite.h - (sprite.h*clipH/destH),
                   destX, destY, destW, destH - clipH);
  }
};

```

Explanation

1. **Size Calculations**:
   - Computes sprite dimensions based on scale and road width
   - Maintains aspect ratio during scaling
   - Handles resolution independence

2. **Positioning**:
   - Applies offset adjustments for precise placement
   - Supports relative positioning with offsetX/Y

  - Handles sprite anchoring

3. **Clipping**:
   - Implements vertical clipping with clipY parameter
   - Prevents drawing outside visible area
   - Optimizes rendering performance

Practical Reuse Example:

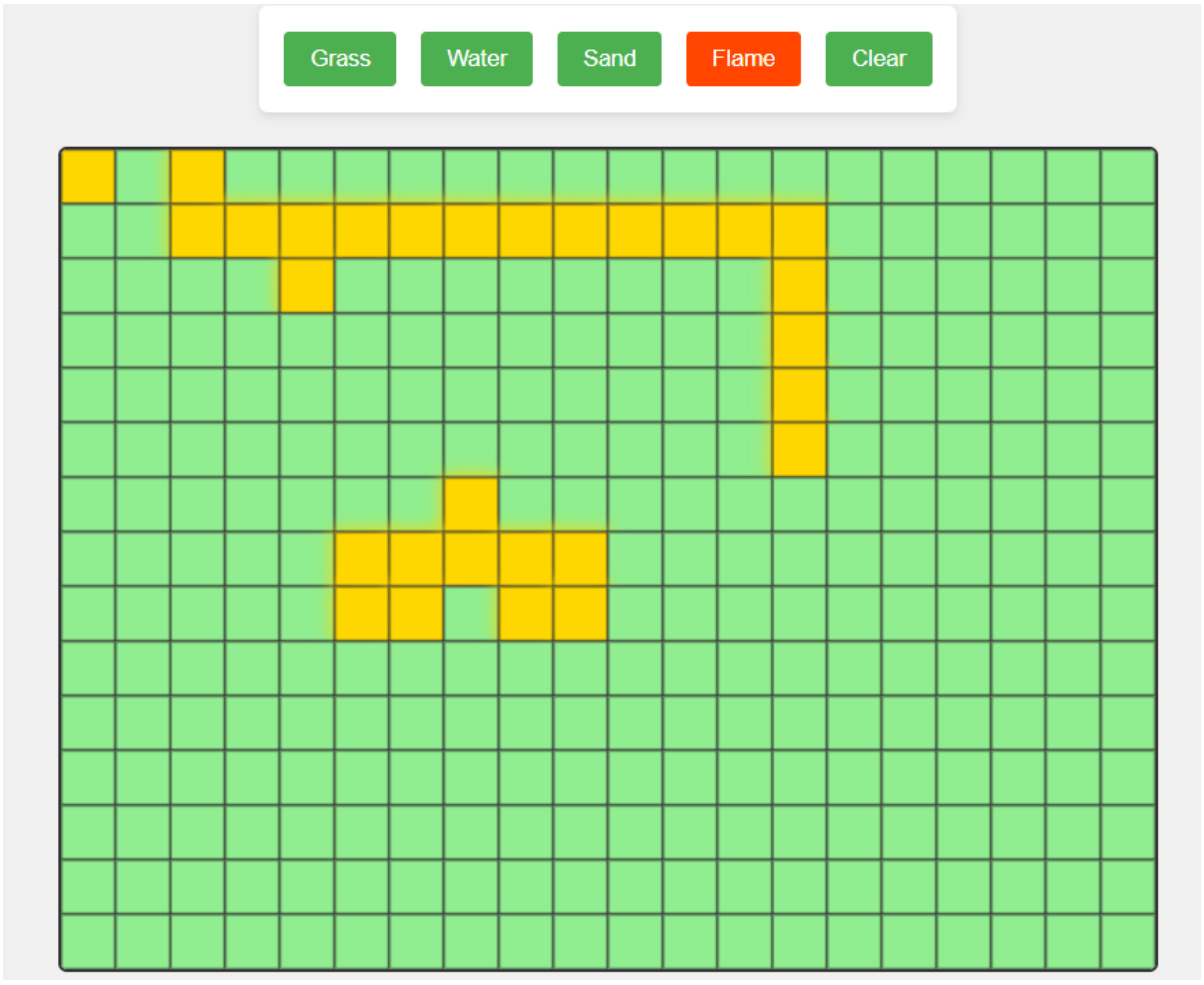
We can use the component to create a simple tile map editor where the tiles will glow if they are flames, for others it u can add to different parts of the map:

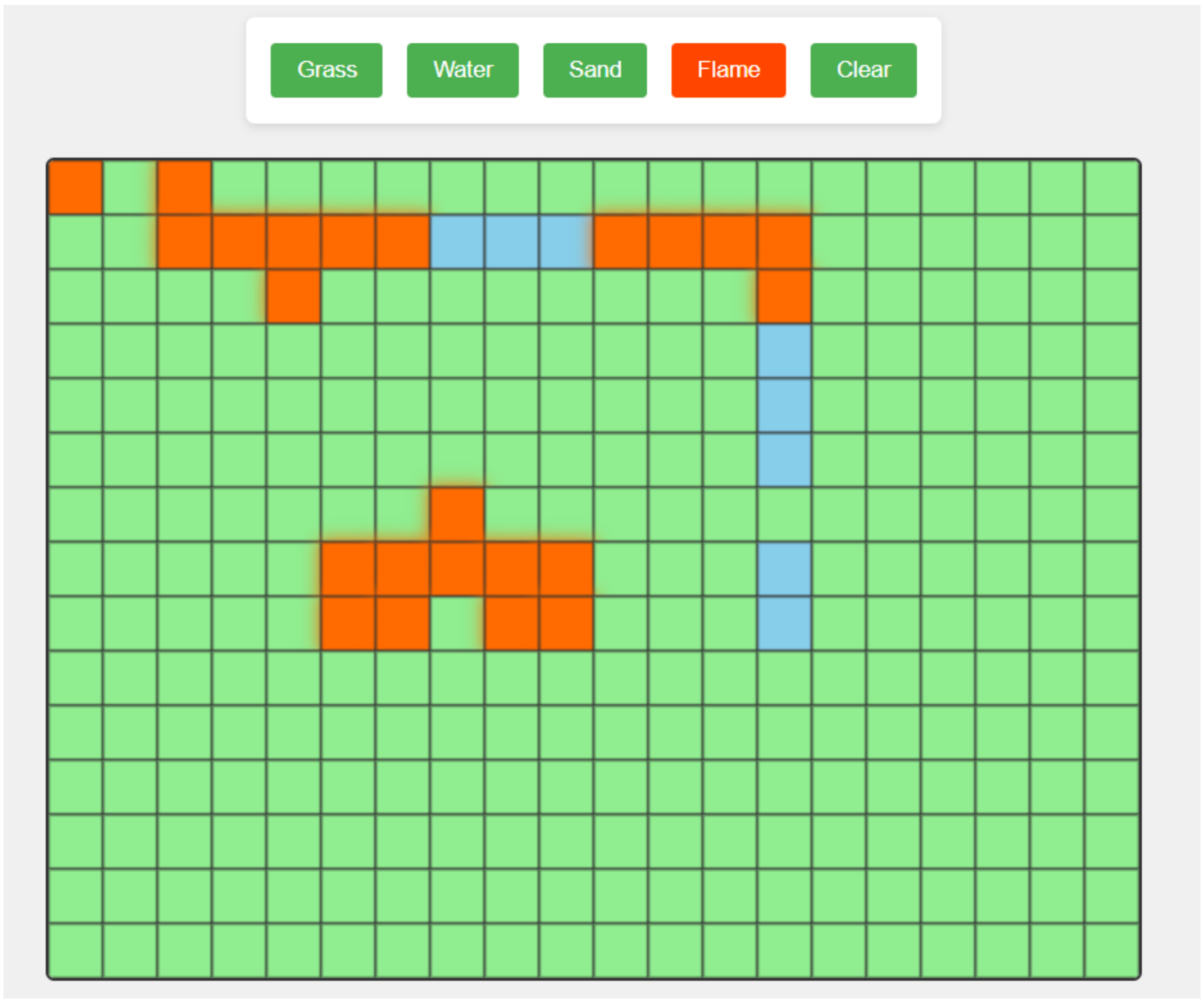
1. **Game Loop Reuse**:
   - Manages animation frames for flame effects
   - Handles continuous updates for color changes
   - Maintains consistent animation timing
2. **Render System Reuse**:
   - Draws tiles using modified sprite system
   - Handles tile positioning and sizing
   - Implements special effects (glow for flames)

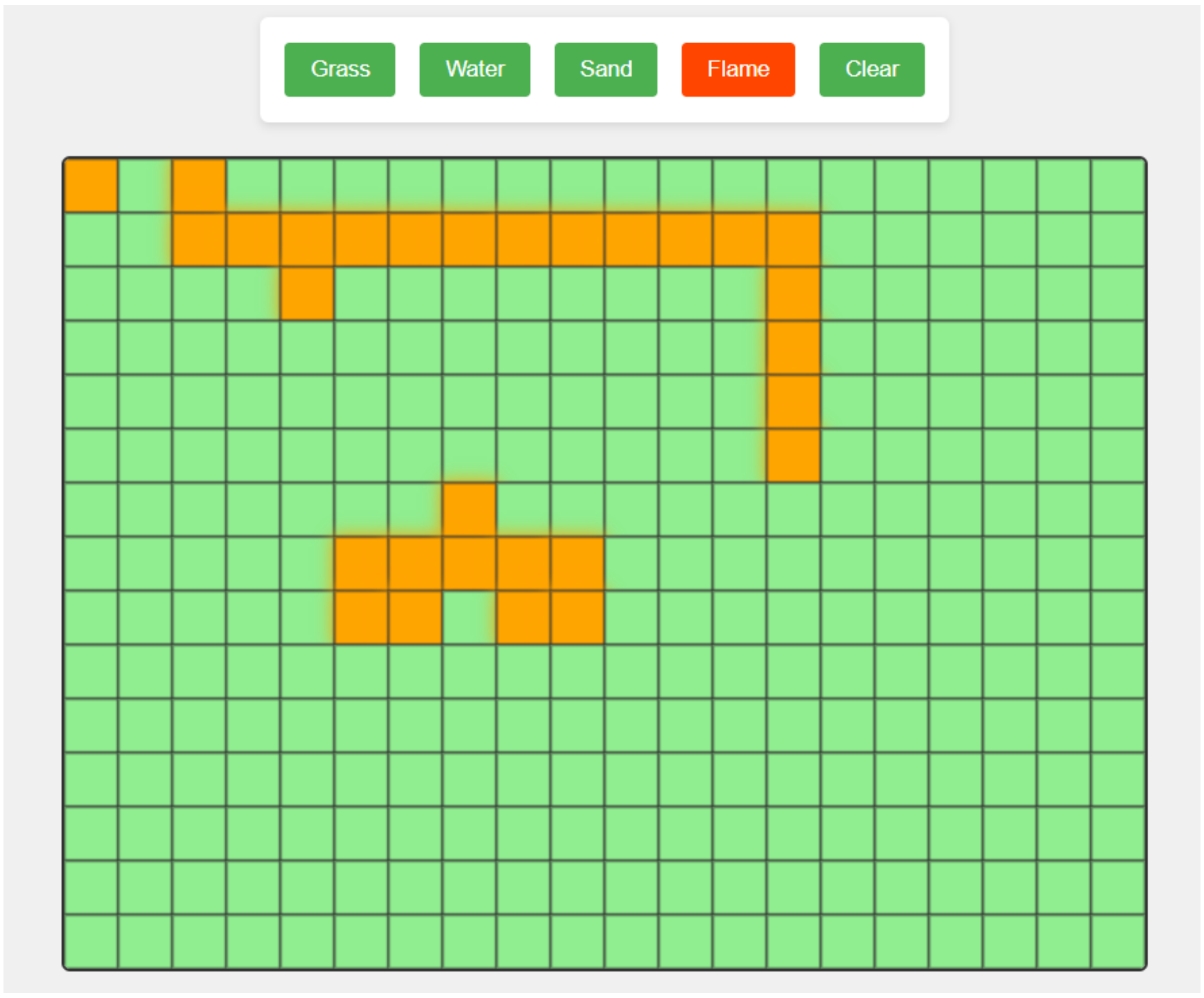
We can use this to implement a simple tile editor, where the rendering step is used to draw the flames color after every time stamp, and also use the canvas component to do the rendering

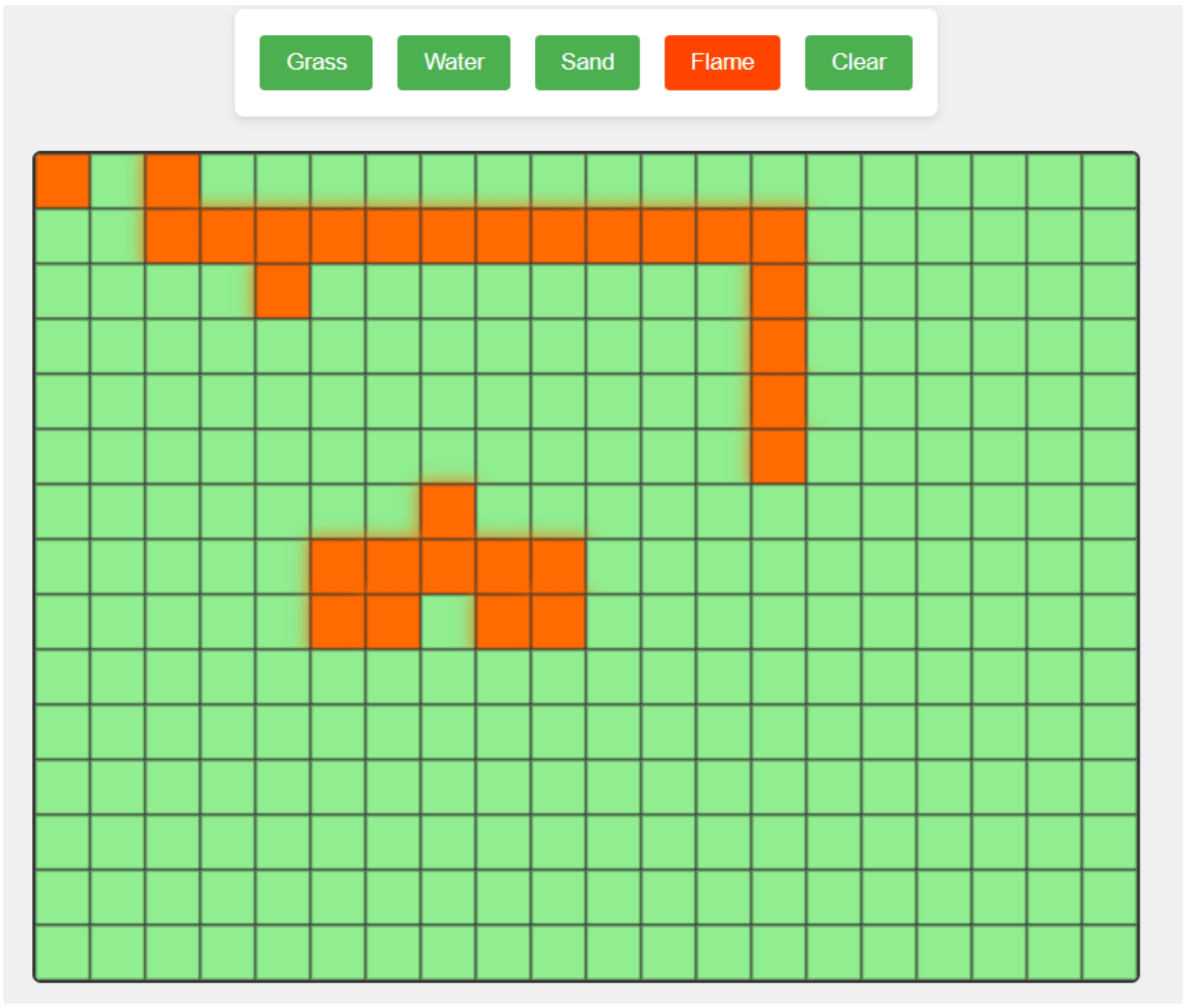
```
<!DOCTYPE html>
<html>
<head>
    <title>Tile Map Editor</title>
    <style>
        body {
            display: flex;
            flex-direction: column;
            align-items: center;
            background-color: #f0f0f0;
            font-family: Arial, sans-serif;
        }
        #controls {
            margin: 20px;
            padding: 10px;
            background-color: white;
            border-radius: 5px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }
        #gameCanvas {
            border: 2px solid #333;
            border-radius: 5px;
            background-color: white;
        }
        button {
            margin: 5px;
            padding: 8px 15px;
            background-color: #4CAF50;
            color: white;
            border: none;
            border-radius: 3px;
            cursor: pointer;
        }
        button:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <div id="controls">
        <button onclick="mapEditor.setTile('GRASS')">Grass</button>
        <button onclick="mapEditor.setTile('WATER')">Water</button>
        <button onclick="mapEditor.setTile('SAND')">Sand</button>
        <button onclick="mapEditor.setTile('FLAME')" style="background-color:
#FF4500">Flame</button>
        <button onclick="mapEditor.clear()">Clear</button>
    </div>
    <canvas id="gameCanvas" width="640" height="480"></canvas>

    <script>
        // Utility functions (reused from original game)
        const Util = {
            timestamp: function() { return new Date().getTime(); },
            toInt: function(obj, def) {
                if (obj !== null) {
                    const x = parseInt(obj, 10);
                    if (!isNaN(x)) return x;
                }
                return Util.toInt(def, 0);
            }
        };

        // Simplified Game loop component (reused from original game)
        const Game = {
            run: function(options) {
                const canvas = options.canvas;
                const update = options.update;
                const render = options.render;
                const step = options.step;

                let now = null;
                let last = Util.timestamp();
                let dt = 0;
                let gdt = 0;

                function frame() {
                    now = Util.timestamp();
                    dt = Math.min(1, (now - last) / 1000);
                    gdt = gdt + dt;

                    while (gdt > step) {
                        gdt = gdt - step;
                        update(step);
                    }

                    render();
                    last = now;
                    requestAnimationFrame(frame);
                }

                // Start the game loop
                frame();
            }
        };

        // Simplified Render component (reused from original game)
        const Render = {
            sprite: function(ctx, tileSize, sprite, x, y) {
                ctx.fillStyle = sprite.color;
                ctx.fillRect(x * tileSize, y * tileSize, tileSize, tileSize);
                ctx.strokeStyle = '#333';
                ctx.strokeRect(x * tileSize, y * tileSize, tileSize, tileSize);
            }
        };

        // Map Editor implementation
        class MapEditor {
            constructor(canvas) {
                this.canvas = canvas;
                this.ctx = canvas.getContext('2d');
                this.tileSize = 32;
                this.mapWidth = Math.floor(canvas.width / this.tileSize);
                this.mapHeight = Math.floor(canvas.height / this.tileSize);
                this.currentTile = 'GRASS';
                this.flameColors = [
                    '#FF4500', // red-orange
                    '#FF6B00', // bright orange
                    '#FF8C00', // dark orange
                    '#FFA500', // orange
                    '#FFD700'  // gold
                ];
                this.flameColorIndex = 0;
                this.flameTimer = 0;

                // Initialize empty map
                this.map = Array(this.mapHeight).fill().map(() =>
                    Array(this.mapWidth).fill('GRASS')
                );

                // Sprite definitions
                this.sprites = {
                    GRASS: { color: '#90EE90' },
                    WATER: { color: '#87CEEB' },
                    SAND: { color: '#F4A460' },
                    FLAME: { color: this.flameColors[0] }
                };

                // Set up mouse event handlers
                this.canvas.addEventListener('mousedown', this.handleMouse.bind(this));
                this.canvas.addEventListener('mousemove', this.handleMouse.bind(this));

                // Start game loop
                Game.run({
                    canvas: this.canvas,
                    update: this.update.bind(this),
                    render: this.render.bind(this),
                    step: 1/60
                });
            }

            handleMouse(event) {
                if (event.buttons !== 1) return; // Only handle left mouse button

                const rect = this.canvas.getBoundingClientRect();
                const x = Math.floor((event.clientX - rect.left) / this.tileSize);
                const y = Math.floor((event.clientY - rect.top) / this.tileSize);

                if (x >= 0 && x < this.mapWidth && y >= 0 && y < this.mapHeight) {
                    this.map[y][x] = this.currentTile;
                }
            }

            setTile(type) {
                this.currentTile = type;
            }

            clear() {
                this.map = Array(this.mapHeight).fill().map(() =>
                    Array(this.mapWidth).fill('GRASS')
                );
            }

            update(step) {
                // Update flame animation
                this.flameTimer += step;
                if (this.flameTimer >= 0.1) { // Change color every 0.1 seconds
                    this.flameTimer = 0;
                    this.flameColorIndex = (this.flameColorIndex + 1) % this.flameColors.length;
                    this.sprites.FLAME.color = this.flameColors[this.flameColorIndex];
                }
            }

            render() {
                this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);

                // Render all tiles
                for (let y = 0; y < this.mapHeight; y++) {
                    for (let x = 0; x < this.mapWidth; x++) {
                        const tileType = this.map[y][x];
                        const sprite = {...this.sprites[tileType]};

                        // Add glow effect for flame tiles
                        if (tileType === 'FLAME') {
                            this.ctx.save();
                            this.ctx.shadowColor = sprite.color;
                            this.ctx.shadowBlur = 10;
                        }

                        Render.sprite(
                            this.ctx,
                            this.tileSize,
                            sprite,
                            x,
                            y
                        );

                        if (tileType === 'FLAME') {
                            this.ctx.restore();
                        }
                    }
                }
            }
        }

        // Initialize the map editor
        const canvas = document.getElementById('gameCanvas');
        const mapEditor = new MapEditor(canvas);
    </script>
</body>
</html>

```

Implementation Screen Shot:




Grass
Water
Sand
Flame
Clear


Grass
Water
Sand
Flame
Clear


Grass
Water
Sand
Flame
Clear