**Javascript俄罗斯方块**

html代码和css代码：

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <style>

        body div {

            float: left;

        }

        /\* 游戏进行界面的容器 \*/

        .container {

            position: relative;

            width: 200px;

            height: 360px;

            background-color: #000;

            margin-top: 18px;

        }

        /\* 下落时的方块 \*/

        .activity\_model {

            position: absolute;

            width: 20px;

            height: 20px;

            background-color: tomato;

            border: .3px solid #fff;

            box-sizing: border-box;

        }

        /\* 着地时的方块 \*/

        .fixed\_model {

            position: absolute;

            width: 20px;

            height: 20px;

            background-color: yellow;

            border: .3px solid #fff;

            box-sizing: border-box;

        }

        /\* 左侧游戏菜单界面 \*/

        /\* 右侧游戏说明界面 \*/

        .lalala {

            width: 200px;

            height: 360px;

            flex: 1;

            margin-top: 18px;

            margin-left: 480px;

            border: 1px solid #000;

            border-right: transparent;

        }

        /\* 开始游戏 \*/

        .bt {

            display: block;

            width: 100px;

            height: 40px;

            color: white;

            line-height: 40px;

            text-align: center;

            background-color: rgb(93, 64, 224);

            margin: 50px auto;

            cursor: pointer;

        }

        /\* 暂停游戏 \*/

        .bt1 {

            display: block;

            width: 100px;

            height: 40px;

            color: white;

            line-height: 40px;

            text-align: center;

            background-color: rgb(93, 64, 224);

            margin: 70px auto;

cursor: pointer;

        }

        h5 {

            text-align: center;

        }

        /\* 右侧游戏说明界面 \*/

        .lalala1 {

            width: 220px;

            height: 360px;

            flex: 1;

            margin-top: 18px;

            border: 1px solid #000;

            border-left: transparent;

        }

        .lalala1 ul {

            margin-left: -30px;

        }

        li {

            width: 200px;

            height: 20px;

            line-height: 20px;

            list-style: none;

            margin: 20px 0;

        }

    </style>

</head>

<body onload="init()">

    <div class="lalala">

        <h5>俄罗斯方块</h5>

        <button id="bt" class="bt">重新开始</button>

        <button id="bt1" class="bt1">暂停游戏</button>

        <h5>得分：<span id="score">0</span> </h5>

    </div>

    <div class="container" id="container">

    </div>

    <div class="lalala1">

        <h5>游戏说明</h5>

        <div class="detail">

            <ul>

                <li>下左右分别控制方块的移动</li>

                <li>上控制方块的旋转</li>

                <li>空格可以暂停游戏/继续游戏</li>

                <li>点击重新开始可以重置方块</li>

            </ul>

        </div>

    </div>

</body>

<script src="lodash.min.js"></script>

<script src="els.js"></script>

</html>

**javascript代码**：

// 步长

var step = 20;

// 总计行数

var row\_count = 18;

// 总计列数

var col\_count = 10;

//18行 10列

//创建每个模型的数据源

var MODELS = [

        //L型

        {

            0: {

                row: 0,

                col: 0

            },

            1: {

                row: 1,

                col: 0

            },

            2: {

                row: 2,

                col: 0

            },

            3: {

                row: 2,

                col: 1

            }

        },

        //倒T型

        {

            0: {

                row: 1,

                col: 0

            },

            1: {

                row: 1,

                col: 1

            },

            2: {

                row: 1,

                col: 2

            },

            3: {

                row: 0,

                col: 1

            }

        },

        //田字型

        {

            0: {

                row: 0,

                col: 0

            },

            1: {

                row: 1,

                col: 0

            },

            2: {

                row: 0,

                col: 1

            },

            3: {

                row: 1,

                col: 1

            }

        },

        //长条型

        {

            0: {

                row: 0,

                col: 0

            },

            1: {

                row: 0,

                col: 1

            },

            2: {

                row: 0,

                col: 2

            },

            3: {

                row: 0,

                col: 3

            }

        },

        //z字型

        {

            0: {

                row: 2,

                col: 0

            },

            1: {

                row: 2,

                col: 1

            },

            2: {

                row: 1,

                col: 1

            },

            3: {

                row: 1,

                col: 2

            }

        }

    ]

    //当前使用的模型

var currentModel = {};

var currentX = 0;

var currentY = 0;

//定义16宫格的初始坐标

//定时器

var myInterval = null;

var fixedBlocks = {}

    //记录块元素位置

    //key=行——列 Value=块元素

    //初始化

function init() {

    createModel();

    onKeyDown();

}

//根据数据源创建对应块元素

function createModel() {

    if (isGameOver() == false) {

        alert('游戏结束')

        return

    }

    // 利用0-5的随机数向下取整随机产生不同的模型

    currentModel = MODELS[Math.floor(Math.random() \* 5)];

    //重新初始化 16宫格位置

    currentX = 0;

    currentY = 0;

    //遍历对象，生成对应数量块元素

    for (var key in currentModel) {

        var divEle = document.createElement('div');

        divEle.className = 'activity\_model';

        document.getElementById('container').appendChild(divEle);

    }

    locationBlocks();

    // 调用自动下落

    autoDown();

}

//根据数据源来定位块元素的位置

function locationBlocks() {

    //判断块元素越界行为

    checkBound();

    //拿到所有块元素

    //找到每个块元素对应的数据

    // 根据数据来指定位置

    var eles = document.getElementsByClassName('activity\_model');

    for (var i = 0; i < eles.length; i++) {

        //单个块元素

        var eleson = eles[i];

        //找到每个块元素对应行列

        var blockModel = currentModel[i];

        //块元素由两个因素确定 1.16宫格位置 2. 块元素再16宫格的位置

        eleson.style.top = (currentY + blockModel.row) \* step + 'px';

        eleson.style.left = (currentX + blockModel.col) \* step + 'px';

    }

}

//监听用户键盘事件

function onKeyDown() {

    document.onkeydown = function(e) {

        console.log(e.keyCode);

        //37 38 39 40 左上右下

        switch (e.keyCode) {

            case 37:

                move(-1, 0);

                console.log('左');

                break;

            case 38:

                console.log('上');

                rotate();

                break;

            case 39:

                move(1, 0);

                console.log('右');

                break;

            case 40:

                move(0, 1);

                console.log('下');

                break;

            case 32: // 暂停

                !pause ? suspend() : autoDown();

                break;

        }

    }

}

//移动

function move(x, y) {

    // var kuai = document.querySelectorAll('.activity\_model');

    // for (var i = 0; i < kuai.length; i++) {

    //     kuai[i].style.top = parseInt(kuai.style.top || 0) + y \* step + 'px';

    //     kuai[i].style.left = parseInt(kuai.style.left || 0) + x \* step + 'px';

    // }

    if (isMet(currentX + x, currentY + y, currentModel)) {

        if (y !== 0) {

            fixedBottomModel();

        }

        return;

    }

    currentX += x;

    currentY += y;

    //根据16宫格重新定位块元素

    locationBlocks();

}

//旋转模型

function rotate() {

    //遍历数据源

    //旋转后的行 = 旋转前的列

    // 旋转后的列=3-旋转前的行

    //克隆以下 currentModel

    var cloneCurrentModel = \_.cloneDeep(currentModel);

    for (var key in cloneCurrentModel) {

        var blockModel = cloneCurrentModel[key];

        var temp = blockModel.row;

        blockModel.row = blockModel.col;

        blockModel.col = 3 - temp;

    }

    if (isMet(currentX, currentY, cloneCurrentModel)) {

        return;

    }

    currentModel = cloneCurrentModel;

    locationBlocks();

}

//控制模型只能在容器内部操作

function checkBound() {

    //定义模型可以活动的边界

    var leftBound = 0,

        rightBound = col\_count,

        bottomBound = row\_count;

    //当块元素超出边界后，让16宫格后退一步

    for (var key in currentModel) {

        var blockModel = currentModel[key];

        //左侧边界

        if ((blockModel.col + currentX) < leftBound) {

            currentX++;

        }

        if ((blockModel.col + currentX) >= rightBound) {

            currentX--;

        }

        if ((blockModel.row + currentY) >= bottomBound) {

            currentY--;

            fixedBottomModel();

        }

    }

}

//把模型固定在底部

function fixedBottomModel() {

    //改变模型(块元素)样式

    var eles = document.getElementsByClassName('activity\_model');

    for (var i = eles.length - 1; i >= 0; i--) {

        var ele = eles[i]

        ele.className = 'fixed\_model';

        //让模型不可移动

        var blockModel = currentModel[i];

        //把该块元素放入变量中

        fixedBlocks[(currentY + blockModel.row) + '\_' + (currentX + blockModel.col)] = ele;

    }

    //判断是否铺满

    isRemoveLine();

    //创建新的模型

    createModel();

}

//判断模型碰撞

//x y表示16宫格【将要】移动的位置  model表示当前模型数据源【将要】完成的变化

function isMet(x, y, model) {

    //碰撞就是一个固定位置已经存在一个固定的块元素时，那么活动中的模型不可以再占用这个位置

    //判断模型将要移动的位置是否已经存在被固定的块元素

    //如果存在返回true 否则返回false

    for (var k in model) {

        var blockModel = model[k];

        if (fixedBlocks[(y + blockModel.row) + '\_' + (x + blockModel.col)]) {

            return true;

        }

    }

    return false;

}

//判断一行是否被铺满

function isRemoveLine() {

    //在一行中 每一列都存在块元素 那么该行就需要被清理了

    //遍历所有行中的所有列

    // 遍历所有行

    for (var i = 0; i < row\_count; i++) {

        //标记符 true表示已经铺满了

        var flag = true;

        for (var j = 0; j < col\_count; j++) {

            if (!fixedBlocks[i + '\_' + j]) {

                flag = false;

                break;

            }

        }

        if (flag) {

            console.log('底部已经铺满');

            removeLine(i);

        }

    }

}

var score = 0;

var bang = document.getElementById('score');

//清理被铺满的一行

function removeLine(line) {

    //删除此行所有块元素

    //删除此行所有块元素的数据源

    //遍历行中所有列

    for (var i = 0; i < col\_count; i++) {

        var fa = document.querySelector('.container')

        fa.removeChild(fixedBlocks[line + '\_' + i]);

        fixedBlocks[line + '\_' + i] = null;

    }

    downLine(line);

    score += 100;

    bang.innerHTML = score;

}

//让被清理之上的下落

function downLine(line) {

    //让被清理之上的下落所在行数加一

    //让块元素在容器中的位置下落

    //清理掉之前的块元素

    //遍历清理行之上的所有行

    for (var i = line - 1; i >= 0; i--) {

        //遍历改行所有列

        for (var j = 0; j < col\_count; j++) {

            if (!fixedBlocks[i + '\_' + j]) {

                continue;

            } else {

                //让被清理之上的下落所在行数加一

                //让块元素在容器中的位置下落

                //清理掉之前的块元素

                fixedBlocks[(i + 1) + '\_' + j] = fixedBlocks[i + '\_' + j];

                fixedBlocks[(i + 1) + '\_' + j].style.top = (i + 1) \* step + 'px';

                fixedBlocks[i + '\_' + j] = null;

            }

        }

    }

}

var myInterval = '';

//让模型自动下落

function autoDown() {

    pause = false;

    if (myInterval != null) {

        clearInterval(myInterval);

    }

    myInterval = setInterval(function() {

        move(0, 1);

        console.log('下');

    }, 400)

}

var reset = true;

//判断游戏结束

function isGameOver() {

    // if (myInterval) {

    //     clearInterval(myInterval);

    // }

    // 当第0行存在块元素时

    for (var i = 0; i < col\_count; i++) {

        if (fixedBlocks['0\_' + i]) {

            clearInterval(myInterval);

            // reset = false;

            return false

        }

        // return true;

    }

    // return false;

}

// 开始游戏按钮

var start = document.getElementById('bt');

// start.addEventListener('click', init());

start.onclick = function() {

    // for (var i = dd.length - 1; i >= 0; i--) {

    //     qingkong.removeChild(dd[i]);

    // }

    document.location.reload();

    // console.log(reset);

}

//暂停函数

var pause = false;

function onKeyFail() {

    document.onkeydown = function(e) {

        console.log(e.keyCode);

        switch (e.keyCode) {

            case 37:

                break;

            case 38:

                break;

            case 39:

                break;

            case 40:

                break;

            case 32:

                !pause ? suspend() : beginagain();

                break;

        }

    }

}

function suspend() {

    clearInterval(myInterval);

    onKeyFail();

    pause = true;

    bt1.innerHTML = '继续游戏';

    return;

}

var bt1 = document.getElementById('bt1');

function beginagain() {

    onKeyDown();

    autoDown();

    bt1.innerHTML = '暂停游戏';

}