链滴

实践：路径遍历（三） 迷宫探秘

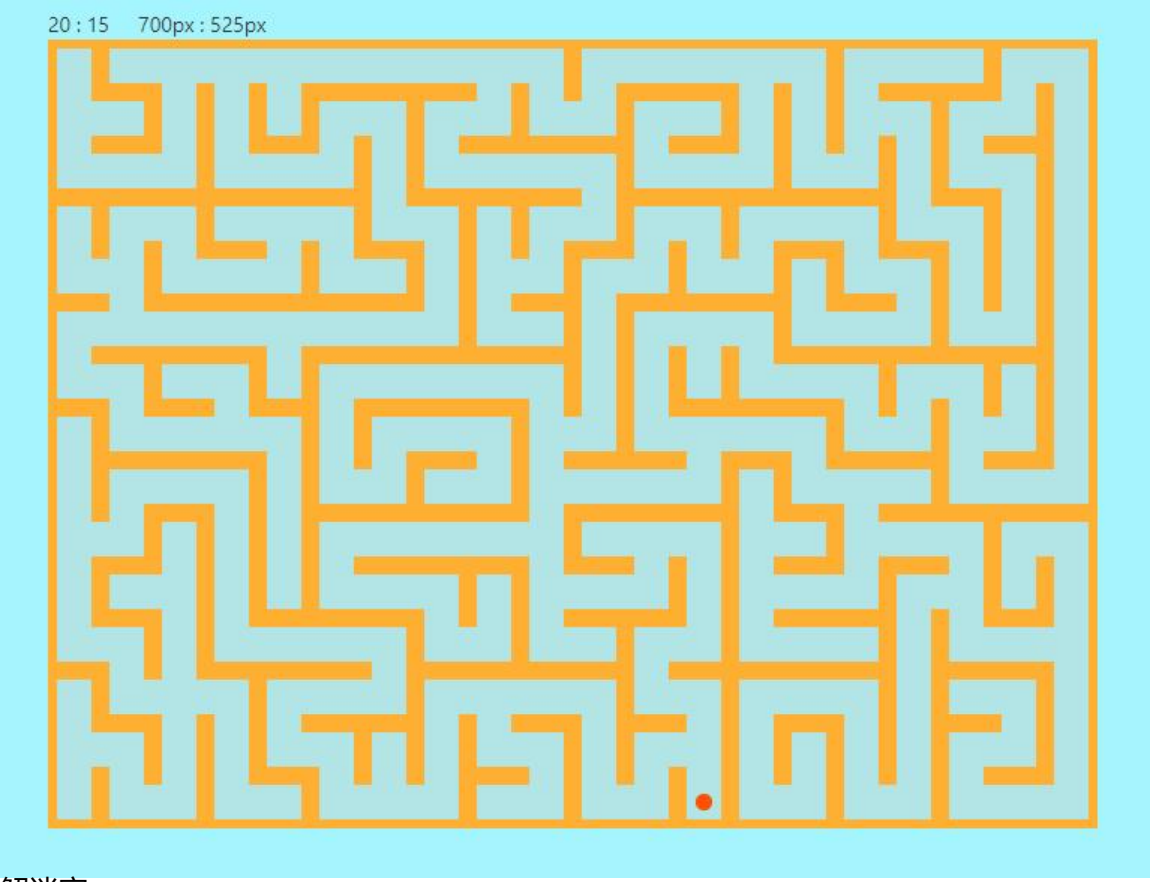
作者：hudk

原文链接：https://ld246.com/article/1590573605540

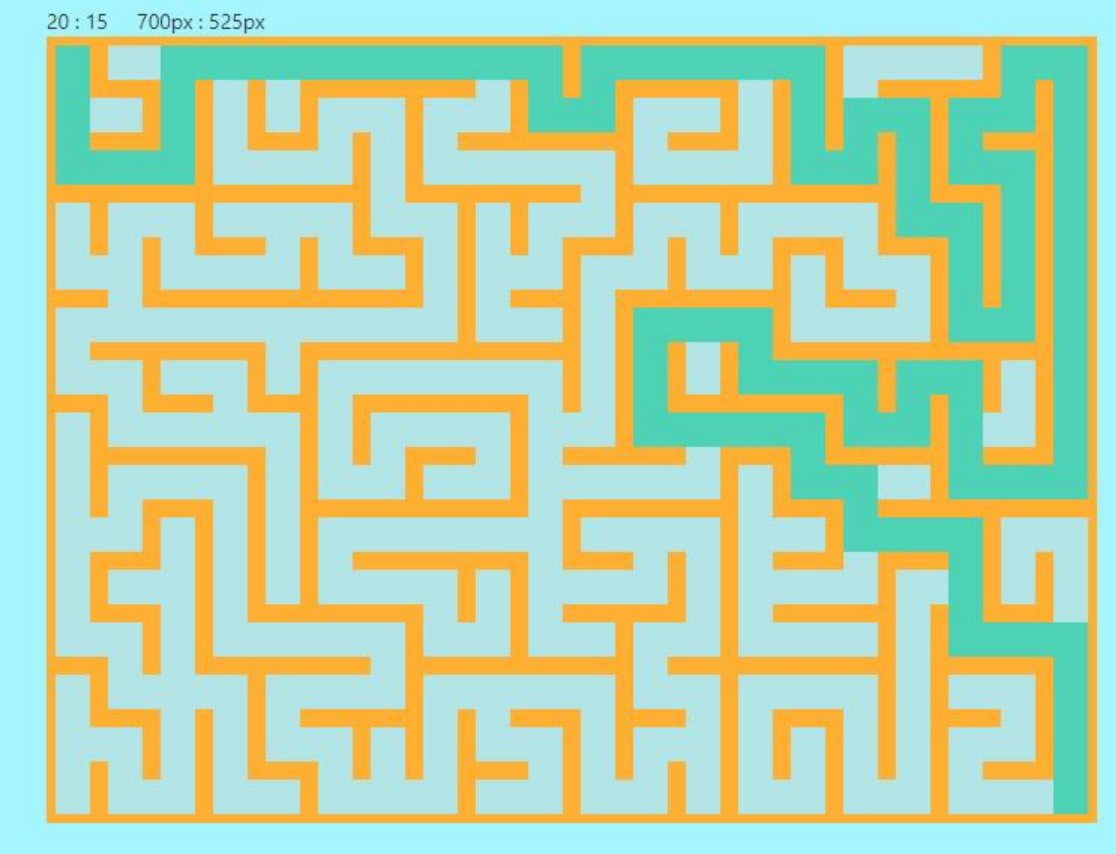
来源网站：链滴

迷宫程序 链接，

迷宫生成：



解迷宫：



原文链接：实践：路径遍历（三） 迷宫探秘

迷宫生成算法的可视化演示，使用React来实现。

迷宫的生成，涉及在图中找出生成树相关算法。

解迷宫，就是在生成的树上，进行深度遍历，寻找链接指定两个点的唯一可达路线。

以下为具体的部分关键实现代码。

```
import React, { Component } from 'react';
import './maze.css';
import Grid from './Grid';

class AppDemo extends Component {
    constructor(proprs) {
        super(proprs);
        var column = 20;//默认列数
        var row = 15;//默认行数
        var pixel = 35;//默认单位方格像素数
        var staticMatrix = this.initStaticMatrix(column, row);//静态二维数组，第一维度是位置编号
第二维度是当下位置的邻居编号
        var dynamicMatrix = this.initDynamicMatrix(column, row);//动态二维数组，第一维度是位
编号，第二维度是当下位置已连接的方向信息
        var elements = this.initElements(column, row);
        var arr = this.initArr(column, row);
        var start = Math.floor(Math.random() * column * row + 1);//随机初始位置
        this.state = {
            staticMatrix: staticMatrix,
            dynamicMatrix: dynamicMatrix,
            now: start,//当下位置*
            begin: start,//开始位置
            end: start,//最后一个位置
            column: column,//列数
            row: row,//行数
            timeID: 0,//定时器ID
            speed: 100,//速度（毫秒）
            speedtype: "中",//用于界面速度显示
            pixel: pixel,//方格像素数
            step: false,//是否参与过单步
            time: 0,//用时显示
            begintime: 0,//起始时间
            stoptime: 0,//完成时间
            reverse: [0, 3, 4, 1, 2], //用于快速反向
            elements: elements,//边界元素集合
            over: column * row,
            maxwidth: 900,//最大宽度像素数
            maxheight: 600,//最大高度像素数

            arr: arr,//路径元素编号信息
            historyPath: [],//路径栈
            findtime: 0,//用时显示
            findbegintime: 0,//起始时间
            findstoptime: 0,//完成时间
            findtimeID: 0,//定时器ID
            findover: false,//寻路完毕
            findStep: false,//是否参与过单步
```

```
            hide: false //是否隐藏界面中的标记
        }
    }

    /**
     * 每一步的处理算法，调用一次，迷宫界面就会向前走一步，方向是随机的
     */
    handle() {
        var elements = this.state.elements;
        var r = this.around(this.state.now, this.state.dynamicMatrix);//返回未涉足邻居编号集合
        if (r.length < 1) {//若未涉足邻居个数小于1（即从当下位置出发已经无路可走），则随机切换
边界集合中的其他位置
            if (this.state.over <= 1) {
                this.stop();
                //this.info("迷宫已经生成完毕。");
                return;
            }
            var randomNow = this.randomElement(elements);
            this.setState({
                time: (new Date().getTime() - this.state.begintime) / 1000,
                now: randomNow
            });
        } else {
            //方向随机1表示向上，2表示向右，3表示向下，4表示向左
            var direction = this.randomDirection(r);//随机取一个方向
            var next = this.state.staticMatrix[this.state.now][direction];//下一个位置编号
            if (this.around(next, this.state.dynamicMatrix).length >= 2) {//如果下一个位置的未涉足
居个数大于等于2，则标记为边界。
                elements[next] = 1;
            }
            var dynamicMatrix = this.state.dynamicMatrix;//动态生成的“树”
            this.connect(this.state.now, direction, next, dynamicMatrix);//连接now和next
            this.setState({
                time: (new Date().getTime() - this.state.begintime) / 1000,
                now: next,
                elements: this.freshElements(elements, next, dynamicMatrix),
                dynamicMatrix: dynamicMatrix,
                over: this.state.over - 1
            });
        }
    }

    findPath() {
        if (this.state.findover === false && this.state.over <= 1) {

            var historyPath = this.state.historyPath;//历史路径
            var dynamicMatrix = this.state.dynamicMatrix;
            var arr = this.state.arr;//路径编号
            var staticMatrix = this.state.staticMatrix;
            if (historyPath.length === 0) {
                historyPath.push(dynamicMatrix[1].concat());
                //arr[historyPath[historyPath.length - 1][0]][] = 1;
            }
            var nowRow = historyPath[historyPath.length - 1][0];//获取当下的位置编号
```

```
        var b = false;
        for (var i = 1; i <= 4; i++) {
            var p = historyPath[historyPath.length - 1][i];
            if (p === 1) {
                var next = staticMatrix[nowRow][i];
                historyPath.push(dynamicMatrix[next].concat());
                historyPath[historyPath.length - 1][this.state.reverse[i]] = 0;
                arr[next][this.state.reverse[i]] = 1;
                arr[nowRow][i] = 1;
                this.setState({
                    findtime: (new Date().getTime() - this.state.findbegintime) / 1000,
                    historyPath: historyPath,
                    arr: arr
                });
                if (next === this.state.column * this.state.row) {
                    this.findStop();
                    this.setState({
                        findover: true
                    });
                    //this.info("迷宫路径已找到（暂时只支持从左上到右下）。");
                    return;
                }
                b = true;
                break;
            }
        }
        if (!b) {//如果无路可走
            //判断当下路径（未退步之前）是否包含于历史记录。
            var nown = historyPath[historyPath.length - 1][0];
            var last = historyPath[historyPath.length - 2][0];
            arr[nown] = [nown,0,0,0,0];
            historyPath.pop();
            for (var j = 1; j <= 4; j++) {
                var p2 = historyPath[historyPath.length - 1][j];
                if (p2 === 1) {
                    arr[last][j] = 0;
                    historyPath[historyPath.length - 1][j] = 0;
                    this.setState({
                        findtime: (new Date().getTime() - this.state.findbegintime) / 1000,
                        historyPath: historyPath,
                        arr: arr
                    });
                    break;
                }
            }
        }
    } else {
        this.findStop();
    }
}

/**
 * 启动解迷宫程序
 */
```

```
findStart() {
    if (this.state.over <= 1) {//若迷宫已经生成完毕，才可以解迷宫
        var findtimeID = this.state.findtimeID;
        if (findtimeID === 0) {//若 当下定时器不为0，则启动新定时器（防止重复启动多个定时器）
            findtimeID = setInterval(
                () => this.findPath(),
                this.state.speed
            );
            this.setState({
                findbegintime: new Date().getTime() - this.state.findtime * 1000,
                findtimeID: findtimeID,
                now: 0
            });
        }

    }else{
        this.info("需要先生成迷宫，才能解迷宫的！笨蛋");
    }

}
findStop() {
    var findtimeID = this.state.findtimeID;
    if (findtimeID !== 0) {
        clearInterval(findtimeID);
        this.setState({
            findtimeID: 0
        });
    }

}
findStep() {
    this.findStop();
    this.findPath();
    this.setState({
        findStep: true
    });
}
hide(){
    this.setState({
        hide: !this.state.hide
    });
}
render() {
    return (
        <div className="maze">
            <div className="aaa">
                <Grid
                    pixel={this.state.pixel}
                    width={this.state.column}
                    height={this.state.row}
                    dynamicMatrix={this.state.dynamicMatrix}
                    now={this.state.now}
                    elements={this.state.elements}
                    begin={this.state.begin}
```

```
                    end={this.state.end}
                    arr={this.state.arr}
                    hide={this.state.hide}
                />
            </div>
            <div className="bbb">
                <div className="control navbar navbar-default">
                    <span className="title">生成迷宫：（用时{this.state.step ? "单步参与后不再计
" : (this.state.time + "s")}）</span><br />
                    <span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.start()}>开始</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.stop()}>暂停</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.start()}>继续</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.step()}>单步</button><span >       &nb
p;    </span>
                </div>
                <div className="control navbar navbar-default">
                    <span className="title">解迷宫：（用时{this.state.findStep ? "单步参与后不再
时" : (this.state.findtime + "s")}）<span >   </span>{this.state.findover?("
到，路径长度为："+this.state.historyPath.length)+"":"迷宫入口为左上角，出口为右下角。"}</spa
><br />
                    <span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.findStart()}>开始</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.findStop()}>暂停</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.findStart()}>继续</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.findStep()}>单步</button><span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.resetFind()}>清除</button>
                </div>
                <div className="control  navbar navbar-default">
                    <span className="title">速度：{this.state.speedtype}</span><br />
                    <span >   </span>
                    <button className="btn btn-primary navbar-btn" type="button" onClick={()
> this.speed(1000)}>极慢</button><span >   </span>
                    <button className="btn btn-primary navbar-btn" type="button" onClick={()
> this.speed(500)}>慢</button><span >   </span>
                    <button className="btn btn-primary navbar-btn" type="button" onClick={()
> this.speed(100)}>中</button><span >   </span>
                    <button className="btn btn-primary navbar-btn" type="button" onClick={()
> this.speed(50)}>快</button><span >   </span>
                    <button className="btn btn-primary navbar-btn" type="button" onClick={()
> this.speed(10)}>极□□□</button>
                </div>
                <div className="control  navbar navbar-default">
                    <span className="title">列数：{this.state.column}</span><br />
                    <span >   </span>
```

```
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addwidth(1)}>列+1</button><span >   </
pan>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addwidth(-1)}>列-1</button>
                    <span >         </sp
n>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addwidth(5)}>列+5</button><span >   </
pan>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addwidth(-5)}>列-5</button><br />
                </div>
                <div className="control  navbar navbar-default">
                    <span className="title">行数：{this.state.row}</span><br />
                    <span >   </span>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addheight(1)}>行+1</button><span >   </
pan>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addheight(-1)}>行-1</button>
                    <span >         </sp
n>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addheight(5)}>行+5</button><span >   </
pan>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.addheight(-5)}>行-5</button><br />
                </div>
                <div className="control  navbar navbar-default">
                    <span className="title">大小：{this.state.column * this.state.pixel}px :
nbsp;{this.state.row * this.state.pixel}px</span><br />
                    <span >   </span>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.big(1)}>放大+1</button><span >   </spa
>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.big(-1)}>缩小-1</button>
                    <span >         </sp
n>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.big(5)}>放大+5</button><span >   </spa
>
                    <button className="btn btn-primary navbar-btn" style={{ width: "80px" }} typ
="button" onClick={() => this.big(-5)}>缩小-5</button><br />
                </div>
                <div className="control navbar navbar-default">
                    <span className="title">其他：</span><br />
                    <span >   </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.hide()}>{this.state.hide?"显示标记":"隐藏标记"}</button><span >   
nbsp;         </span>
                    <button type="button" className="btn btn-primary navbar-btn" onClick={()
> this.resetAll()}>全部重置</button>
```

```
                </div>
            </div>

        </div >

    )
}

/**
 * 连接now和next两个位置
 * @param {*} now
 * @param {*} direction
 * @param {*} next
 * @param {*} dynamicMatrix
 */
connect(now, direction, next, dynamicMatrix) {
    dynamicMatrix[now][direction] = 1;//连接下一个位置
    dynamicMatrix[next][this.state.reverse[direction]] = 1;//连接下一个位置
}
/**
 * 从边界集合中随机选择一个返回
 * @param {} elements
 */
randomElement(elements) {
    var u = this.state.column * this.state.row
    var n = Math.floor(Math.random() * u);
    var i = 0;
    while (elements[n] !== 1) {
        i++;
        n++;
        if (n > u) {
            n = 1;
        }
        if (i > u) {
            n = 0;
            break;
        }
    }
    return n;
}
/**
 * 初始化StaticMatrix
 * @param {*} width
 * @param {*} height
 */
initStaticMatrix(width, height) {
    var matrix = [[0, 1, 2, 3, 4]];
    for (var numb = 1; numb <= width * height; numb++) {
        var up = numb > width ? numb - width : 0;
        var right = (numb % width) !== 0 ? numb + 1 : 0;
        var down = numb <= width * (height - 1) ? numb + width : 0;
        var left = ((numb - 1) % width) !== 0 ? numb - 1 : 0;
        var arr = [numb];
        arr.push(up);
```

```
        arr.push(right);
        arr.push(down);
        arr.push(left);
        matrix.push(arr);
    }
    return matrix;
}
/**
 * 初始化DynamicMatrix
 * @param {*} width
 * @param {*} height
 */
initDynamicMatrix(width, height) {
    var dynamicMatrix = [[0, 1, 2, 3, 4]];
    for (var numb = 1; numb <= width * height; numb++) {
        var up = 0;
        var right = 0;
        var down = 0;
        var left = 0;
        var arr = [numb];
        arr.push(up);
        arr.push(right);
        arr.push(down);
        arr.push(left);
        dynamicMatrix.push(arr);
    }
    return dynamicMatrix;
}
initArr(width, height) {
    var dynamicMatrix = [[0, 1, 2, 3, 4]];
    for (var numb = 1; numb <= width * height; numb++) {
        var up = 0;
        var right = 0;
        var down = 0;
        var left = 0;
        var arr = [numb];
        arr.push(up);
        arr.push(right);
        arr.push(down);
        arr.push(left);
        dynamicMatrix.push(arr);
    }
    return dynamicMatrix;
}
/**
 * 初始化Elements边界集合
 * @param {*} column
 * @param {*} row
 */
initElements(column, row) {
    var arr = new Array(column * row + 1);
    for (var i = 0; i < arr.length; i++) {
        arr[i] = 0;
    }
```

```
        return arr;
    }
    /**
     * 返回未涉足邻居编号集合
     * 参数new,是当下位置的编号
     */
    around(now, dynamicMatrix) {
        var r = [];
        for (var j = 1; j <= 4; j++) {//每个位置周边共四个“邻居”
            var runod = this.state.staticMatrix[now][j];//其中一个邻居
            //判断“邻居”是否已经被涉足，若没有，则将此邻居方向编号加入将要返回的集合中
            if (runod !== 0 && (dynamicMatrix[runod][1] === 0
                && dynamicMatrix[runod][2] === 0
                && dynamicMatrix[runod][3] === 0
                && dynamicMatrix[runod][4] === 0)) {
                r.push(j);
            }
        }
        return r;//返回未涉足邻居方向编号集合
    }
    /**
     * 从邻居中随机选出一个作为下一个位置
     * @param {*} r
     */
    randomDirection(r) {
        if (r.length === 1) {
            return r[r.length - 1];
        }
        return r[Math.floor(Math.random() * r.length)];
    }

    /**
     * 刷新边界集合，（每前进一步，整个盘面唯有一个位置有所改变，所以只扫面更新此位置四周即
)
     * @param {*} elements
     * @param {*} next
     * @param {*} dynamicMatrix
     */
    freshElements(elements, next, dynamicMatrix) {
        for (var n = 1; n <= 4; n++) {
            var runod = this.state.staticMatrix[next][n];//其中一个邻居
            //判断“邻居”是否已经被涉足，若没有，则将此邻居编号加入将要返回的集合中
            if (elements[runod] !== 0 && this.around(runod, dynamicMatrix).length === 0) {//如
elements[n]周围没有未涉足区域，则从边缘集合中去掉
                elements[runod] = 0;
            }
        }
        return elements;
    }
    /**
     * 添加列数，n为添加的列数
     * @param {*} n
     */
    addwidth(n) {
```

```
    this.stop();
    var column = this.state.column;
    column = column + n;
    if (column > 0 && column * this.state.pixel <= this.state.maxwidth) {
        var staticMatrix = this.initStaticMatrix(column, this.state.row);
        var dynamicMatrix = this.initDynamicMatrix(column, this.state.row);
        var elements = this.initElements(column, this.state.row);
        var arr = this.initArr(column, this.state.row);
        var start = Math.floor(Math.random() * column * this.state.row + 1);
        this.setState({
            staticMatrix: staticMatrix,
            dynamicMatrix: dynamicMatrix,
            begin: start,
            now: start,
            column: column,
            time: 0,
            elements: elements,
            over: column * this.state.row,

            arr: arr,
            historyPath: [],
            findtime: 0,
            findover: false
        });
    } else {
        var maxColumn = Math.floor(this.state.maxwidth / this.state.pixel);
        var staticMatrix1 = this.initStaticMatrix(maxColumn, this.state.row);
        var dynamicMatrix1 = this.initDynamicMatrix(maxColumn, this.state.row);
        var elements1 = this.initElements(maxColumn, this.state.row);
        var arr1 = this.initArr(maxColumn, this.state.row);
        var newstart = Math.floor(Math.random() * maxColumn * this.state.row + 1);
        this.setState({
            staticMatrix: staticMatrix1,
            dynamicMatrix: dynamicMatrix1,
            begin: newstart,
            now: newstart,
            column: maxColumn,
            time: 0,
            elements: elements1,
            over: maxColumn * this.state.row,
            arr: arr1,
            historyPath: [],
            findtime: 0,
            findover: false
        });
    }
}
/**
 * 添加行数，n为添加的行数
 * @param {*} n
 */
addheight(n) {
    this.stop();
    var row = this.state.row;
```

```
        row = row + n;
        if (row > 0 && row * this.state.pixel <= this.state.maxheight) {
            var staticMatrix = this.initStaticMatrix(this.state.column, row);
            var dynamicMatrix = this.initDynamicMatrix(this.state.column, row);
            var elements = this.initElements(this.state.column, row);
            var arr = this.initArr(this.state.column, row);
            var start = Math.floor(Math.random() * this.state.column * row + 1);
            this.setState({
                staticMatrix: staticMatrix,
                dynamicMatrix: dynamicMatrix,
                begin: start,
                now: start,
                row: row,
                time: 0,
                elements: elements,
                over: this.state.column * row,
                arr: arr,
                historyPath: [],
                findtime: 0,
                findover: false
            });
        } else {
            var maxRow = Math.floor(this.state.maxheight / this.state.pixel);
            var staticMatrix1 = this.initStaticMatrix(this.state.column, maxRow);
            var dynamicMatrix1 = this.initDynamicMatrix(this.state.column, maxRow);
            var elements1 = this.initElements(this.state.column, maxRow);
            var arr1 = this.initArr(this.state.column, maxRow);
            var newstart = Math.floor(Math.random() * this.state.column * maxRow + 1);
            this.setState({
                staticMatrix: staticMatrix1,
                dynamicMatrix: dynamicMatrix1,
                begin: newstart,
                now: newstart,
                time: 0,
                row: maxRow,
                elements: elements1,
                over: this.state.column * maxRow,
                arr: arr1,
                historyPath: [],
                findtime: 0,
                findover: false
            });
        }
    }
    /**
     * 缩放界面大小，n为每个单位方格改变的像素数
     * @param {*} n
     */
    big(n) {
        var pixel = this.state.pixel;
        pixel = pixel + n;
        if (pixel >= 5 && ((pixel * this.state.column <= this.state.maxwidth) && (pixel * this.state.
ow <= this.state.maxheight))) {
            this.setState({
```

```
                pixel: pixel
            });
            return;
        }
        if(pixel < 5 ){
            this.info("已经最小，再小你就看不见了，你以为你是显微镜呀！");
            this.setState({
                pixel: 5
            });
        }else {
            this.info("不能再大了，笨蛋");
            this.setState({
                pixel: Math.floor((pixel * this.state.column / this.state.maxwidth > pixel * this.state.r
w / this.state.maxheight) ? (this.state.maxwidth / this.state.column) : (this.state.maxheight / thi
.state.row))
            });
        }
    }
    /**
     * 重置应用，使其除行列和大小外，其他因素设置成初始状态
     */
    resetAll() {
        this.stop();
        var staticMatrix = this.initStaticMatrix(this.state.column, this.state.row);
        var dynamicMatrix = this.initDynamicMatrix(this.state.column, this.state.row);
        var elements = this.initElements(this.state.column, this.state.row);
        var arr = this.initArr(this.state.column, this.state.row);
        var start = Math.floor(Math.random() * this.state.column * this.state.row + 1);
        this.setState({
            staticMatrix: staticMatrix,
            dynamicMatrix: dynamicMatrix,
            begin: start,
            now: start,
            time: 0,
            elements: elements,
            over: this.state.column * this.state.row,
            arr: arr,//路径元素编号信息
            historyPath: [],//路径栈
            findtime: 0,//用时显示
            findbegintime: 0,//起始时间
            findstoptime: 0,//完成时间
            findtimeID: 0,//定时器ID
            findover: false,//寻路完毕
            findStep: false,//是否参与过单步
            hide: false //是否隐藏界面中的标记
        });
    }
    /**
     * 重置应用，使其除行列和大小外，其他因素设置成初始状态
     */
    resetFind() {
        this.findStop();
        var arr = this.initArr(this.state.column, this.state.row);
        this.setState({
```

```
        arr: arr,//路径元素编号信息
        historyPath: [],//路径栈
        findtime: 0,//用时显示
        findbegintime: 0,//起始时间
        findstoptime: 0,//完成时间
        findtimeID: 0,//定时器ID
        findover: false,//寻路完毕
        findStep: false,//是否参与过单步
        hide: false //是否隐藏界面中的标记
    });
}

/**
 * 单步
 */
step() {
    this.stop();
    this.handle();
    this.setState({
        step: true
    });
}
/**
 * 开始
 */
start() {
    var timeID = this.state.timeID;
    if (timeID === 0) {
        timeID = setInterval(
            () => this.handle(),
            this.state.speed
        );
        this.setState({
            begintime: new Date().getTime() - this.state.time * 1000,
            timeID: timeID
        });
    }

}
/**
 * 暂停
 */
stop() {
    var timeID = this.state.timeID;
    if (timeID !== 0) {
        clearInterval(timeID);
        this.setState({
            timeID: 0
        });
    }

}
/**
 * 改变速度，n为新的时间间隔，单位ms，每隔n ms时间，定时器调用一次相关方法
```

```
     * @param {*} n
     */
    speed(n) {
        var timeID = this.state.timeID;
        if (timeID !== 0) {
            clearInterval(timeID);
            timeID = 0;
            timeID = setInterval(
                () => this.handle(),
                n
            );
        }
        var findtimeID = this.state.findtimeID;
        if (findtimeID !== 0) {
            clearInterval(findtimeID);
            findtimeID = 0;
            findtimeID = setInterval(
                () => this.findPath(),
                n
            );
        }
        var speedtype = "";
        if (n === 10) {
            speedtype = "极快";
        }
        if (n === 50) {
            speedtype = "快";
        }
        if (n === 100) {
            speedtype = "中";
        }
        if (n === 500) {
            speedtype = "慢";
        }
        if (n === 1000) {
            speedtype = "极慢";
        }
        this.setState({
            timeID: timeID,
            findtimeID: findtimeID,
            speed: n,
            speedtype: speedtype
        });
    }

    info(info){
        alert(info);
    }
}

export default AppDemo;
```