南京邮电大学通达学院

算法与数据结构设计报告

（2017/2018学年 第二学期）

|  |  |
| --- | --- |
| 题 目： | **2048游戏** |

|  |  |
| --- | --- |
| **专 业** | **软件工程** |
| **学 号** | **\*\*\*** |
| **姓 名** | **\*\*\*** |
| **指 导 教 师** | **陈兴国** |
| **指 导 单 位** | **计算机学院、软件学院** |
| **日 期** | **2018-06-21** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **评价准则** | | **优秀** | **良好** | **中等** | **合格** | **不合格** |
| 能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，进行合理的分析与设计。 | |  |  |  |  |  |
| 能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行设计、建模和预测。 | |  |  |  |  |  |
| 能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行系统设计、实现与验证。 | |  |  |  |  |  |
| 掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试。 | |  |  |  |  |  |
| 具备一定的算法与数据结构设计与实现能力，能够完成课题要求的各项任务和指标。 | |  |  |  |  |  |
| 具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便。 | |  |  |  |  |  |
| 能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握。 | |  |  |  |  |  |
| 具备一定的表达能力与文字处理能力，能够熟练使用文字处理软件完成课题报告一份。报告完整记录了课题的工作及总结，内容详实，格式规范。 | |  |  |  |  |  |
| 具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能。 | |  |  |  |  |  |
| 在上机过程中遵守机房规章制度，出勤与平时表现规范。 | |  |  |  |  |  |
| 课题完成过程及结果能够反映其具备一定的解决工程问题的能力。 | |  |  |  |  |  |
| **本次算法与数据结构设计**  **能力达成评价** | **□ 优秀 □ 良好 □ 中等**  **□ 及格 □ 不及格** | | | | | |
| **备注：** | | | | | | |
| **指导教师： 年 月 日** | | | | | | |

**2048游戏**

**一、课题内容和要求**

实现2048游戏。主要功能为游戏界面显示、上下左右键的响应以及当前得分统计。通过该课题全面熟悉数组、字符串等的使用，掌握程序设计的基本方法及友好界面的设计。

**二、课题需求分析**

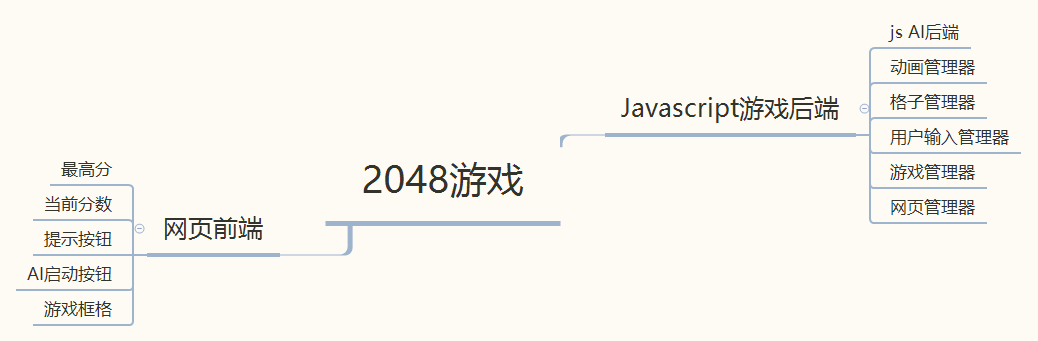


图1 功能框架图

本课题目标系统“2048游戏”的功能框架图如图1所示。

（1）游戏界面显示：数字、上下左右移动、合并等特效以及得分。

（2）动作选择：上下左右键对应于上下左右移动的合并功能。

（3）得分统计功能：判定能否删除行、并统计得分总数等。

（4）用户数据管理：排行榜功能。

（5）自动玩游戏的程序：90%的情况下能达到2048赢得游戏。

**三、课题相关数据结构及算法设计**

1 主要数据结构

initiateModel();

while(!game\_over)

{

for each 3 possible move:

evaluateResult()

execute move with best score

if no move is available, execute forbidden move and undo, recalculateModel()

}

evaluateResult() {

calculatesBestCurrentModel()

calculates distance to chosen model

stores result

}

calculateBestCurrentModel() {

(according to the current highest tile acheived and their distribution)

}

2 主要算法流程

1. 格子规则：通过上下左右控制移动来合并类似的块，以得到“更大”的块。每次移动后，一个新的块出现在随机空位上，值为2或4。当所有的格子都被填满并且没有可以合并的移动时，游戏结束。当格子还没被填满时合并得到了2048块就赢得了游戏。
2. 格子生成算法：

// 预分配项目

Grid.prototype.indexes = [];

for (var x=0; x<4; x++) {

Grid.prototype.indexes.push([]);

for (var y=0; y<4; y++) {

Grid.prototype.indexes[x].push( {x:x, y:y} );

}

}

// 创建特定大小的格子

Grid.prototype.build = function () {

for (var x = 0; x < this.size; x++) {

var row = this.cells[x] = [];

for (var y = 0; y < this.size; y++) {

row.push(null);

}

}

};

// 寻找第一个可用的随机位置

Grid.prototype.randomAvailableCell = function () {

var cells = this.availableCells();

if (cells.length) {

return cells[Math.floor(Math.random() \* cells.length)];

}

};

// 在随机位置放置一个方块

Grid.prototype.addRandomTile = function () {

if (this.cellsAvailable()) {

var value = Math.random() < 0.9 ? 2 : 4;

//var value = Math.random() < 0.9 ? 256 : 512;

var tile = new Tile(this.randomAvailableCell(), value);

this.insertTile(tile);

}

};

// 向特定方向移动一个块

// 如果移动成功就返回True

Grid.prototype.move = function (direction) {

// 0: up, 1: right, 2:down, 3: left

var self = this;

var cell, tile;

var vector = this.getVector(direction);

var traversals = this.buildTraversals(vector);

var moved = false;

var score = 0;

var won = false;

this.prepareTiles();

traversals.x.forEach(function (x) {

traversals.y.forEach(function (y) {

cell = self.indexes[x][y];

tile = self.cellContent(cell);

if (tile) {

var positions = self.findFarthestPosition(cell, vector);

var next = self.cellContent(positions.next);

// 是否只有一个可以合成

if (next && next.value === tile.value && !next.mergedFrom) {

var merged = new Tile(positions.next, tile.value \* 2);

merged.mergedFrom = [tile, next];

self.insertTile(merged);

self.removeTile(tile);

// 两个块合并

tile.updatePosition(positions.next);

// 更新分数

score += merged.value;

// 看是不是得到了2048

if (merged.value === 2048) {

won = true;

}

} else {

self.moveTile(tile, positions.farthest);

}

if (!self.positionsEqual(cell, tile)) {

self.playerTurn = false;

moved = true; // 块从原始位置移动了!

}

}

});

});

return {moved: moved, score: score, won: won};

};

1. AI实现规则：我们既然知道每两个数字相同的格子可以合成一块，那在移动中就要让AI把数字相同的格子尽可能的靠近，减少孤立格子数，同时把数字越大的越放在角落，有利于新格子加入后的合成。（如图二所示）

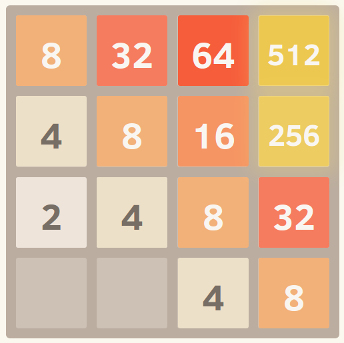


图2 理想摆放位置

除了数字相同的格子要靠近，大小相近的格子也应该相邻，方便于较小的格子合成后马上与旁边的格子合成。

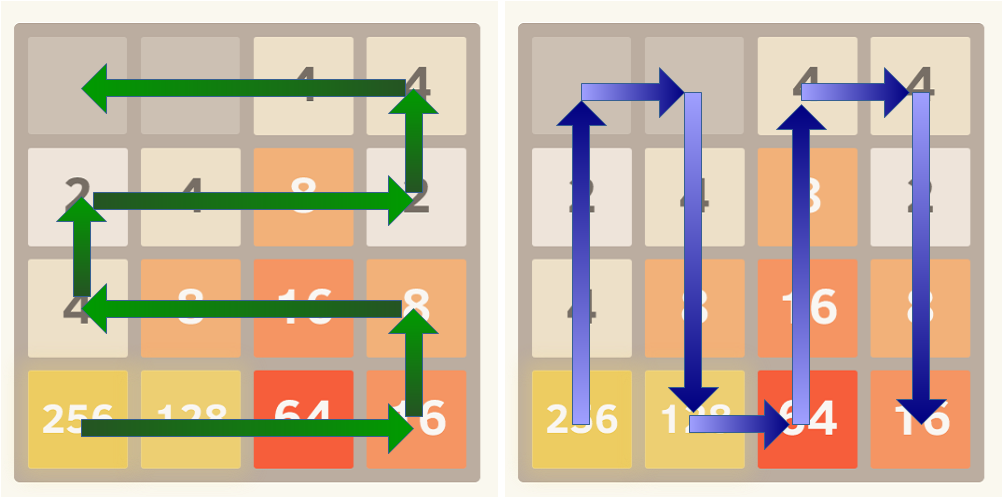


图3 最优路径图

如图三，有两种最优路径，就要求路径的开头和结尾分别对应最大的数字和最小的数字，这样一旦末尾有两个相同的数字就能一路合并到路径的开头。

1. 用户信息管理：既然是通过浏览器展示游戏，那浏览器有一个特性就可以用上：Cookie。Cookie就是网页可以存储在用户浏览器本地的字符串，通过Cookie，可以在用户关闭网页后依然留存用户信息，下次打开网页时由网页读取并展示给用户。存取Cookie可以用以下函数实现：

HTMLActuator.prototype.getCookie = function (c\_name) {

if (document.cookie.length > 0) { //判断Cookie字符串长度是否大于零

c\_start = document.cookie.indexOf(c\_name + "=")

if (c\_start != -1) {

c\_start = c\_start + c\_name.length + 1

c\_end = document.cookie.indexOf(";", c\_start)

if (c\_end == -1) c\_end = document.cookie.length

return unescape(document.cookie.substring(c\_start, c\_end))

}

}

return ""

}

HTMLActuator.prototype.setCookie = function (c\_name,value) {

var expiredays = 365

var exdate = new Date()

exdate.setDate(exdate.getDate() + expiredays)

document.cookie = c\_name + "=" + escape(value) +

((expiredays == null) ? "" : ";expires=" + exdate.toGMTString())

}

HTMLActuator.prototype.clearCookie = function (c\_name) { //清除现有Cookie

var expiredays = -1

var exdate = new Date()

exdate.setDate(exdate.getDate() + expiredays)

document.cookie = c\_name + "=" + ((expiredays == null) ? "" : ";expires=" + exdate.toGMTString())

}



图4 Cookie调用

当网页加载时从Cookie里调出用户的信息，把信息加载到网页里。（如图四）



图5 Cookie为空

当网页读取Cookie发现为空时显示欢迎信息，如图五所示。

游戏启动时运行以下代码：

var high = this.actuator.getCookie("high"); //读取最高分数据

if (high != null && high != "" && high != "undefined") {

this.highest = high;

}

else {

this.highest = 0;

}

this.actuator.setRankTable();

this.actuator.loadScore(this.highest);

var username = this.actuator.getCookie('username');

if (username != null && username != "") {

this.actuator.setWelcome("欢迎回来，" + username + "!"); //设置欢迎语

document.getElementsByClassName("inputname")[0].innerHTML = "";

}

else {

this.actuator.setWelcome("欢迎你，陌生人");

}

// Update the actuator

this.actuate();

};

1. 排行榜的实现：



图6 排行榜

当把鼠标移到最高分区域时会显示出Cookie中存储的历史分数，如图六所示。

通过Javascript的函数可以实现修改或增加HTML中的内容，排行榜本身是一个表格，每一行表示一个分数，两列分别是排名和分数，通过每次加载页面时的运行函数动态显示分数，代码如下：

HTMLActuator.prototype.setRankTable = function (){

var \_score = this.getCookie("score");

this.\_score = \_score.split(',');

if(\_score == ""){

this.\_score.pop();

return;

}

this.clearContainer(this.table);

var tr,td;

tr = this.table.insertRow(this.table.rows.length);

td = tr.insertCell(tr.cells.length);

td.innerHTML = "排名";

td = tr.insertCell(tr.cells.length);

td.innerHTML = "分数";

for(var i=0;i<this.\_score.length;i++){

//循环插入元素

tr = this.table.insertRow(this.table.rows.length);

for(var j=0;j<2;j++){

td = tr.insertCell(tr.cells.length);

j==0?td.innerHTML = i+1:td.innerHTML = this.\_score[i]; //j表示列，i表示行

}

}

}

**四、源程序代码**

function AI(grid) {

this.grid = grid;

}

// static evaluation function

AI.prototype.eval = function() {

var emptyCells = this.grid.availableCells().length;

var smoothWeight = 0.1,

//monoWeight = 0.0,

//islandWeight = 0.0,

mono2Weight = 1.0,

emptyWeight = 2.7,

maxWeight = 1.0;

return this.grid.smoothness() \* smoothWeight

//+ this.grid.monotonicity() \* monoWeight

//- this.grid.islands() \* islandWeight

+ this.grid.monotonicity2() \* mono2Weight

+ Math.log(emptyCells) \* emptyWeight

+ this.grid.maxValue() \* maxWeight;

};

// alpha-beta depth first search

AI.prototype.search = function(depth, alpha, beta, positions, cutoffs) {

var bestScore;

var bestMove = -1;

var result;

// the maxing player

if (this.grid.playerTurn) {

bestScore = alpha;

for (var direction in [0, 1, 2, 3]) {

var newGrid = this.grid.clone();

if (newGrid.move(direction).moved) {

positions++;

if (newGrid.isWin()) {

return { move: direction, score: 10000, positions: positions, cutoffs: cutoffs };

}

var newAI = new AI(newGrid);

if (depth == 0) {

result = { move: direction, score: newAI.eval() };

} else {

result = newAI.search(depth-1, bestScore, beta, positions, cutoffs);

if (result.score > 9900) { // win

result.score--; // to slightly penalize higher depth from win

}

positions = result.positions;

cutoffs = result.cutoffs;

}

if (result.score > bestScore) {

bestScore = result.score;

bestMove = direction;

}

if (bestScore > beta) {

cutoffs++

return { move: bestMove, score: beta, positions: positions, cutoffs: cutoffs };

}

}

}

}

else { // computer's turn, we'll do heavy pruning to keep the branching factor low

bestScore = beta;

// try a 2 and 4 in each cell and measure how annoying it is

// with metrics from eval

var candidates = [];

var cells = this.grid.availableCells();

var scores = { 2: [], 4: [] };

for (var value in scores) {

for (var i in cells) {

scores[value].push(null);

var cell = cells[i];

var tile = new Tile(cell, parseInt(value, 10));

this.grid.insertTile(tile);

scores[value][i] = -this.grid.smoothness() + this.grid.islands();

this.grid.removeTile(cell);

}

}

// now just pick out the most annoying moves

var maxScore = Math.max(Math.max.apply(null, scores[2]), Math.max.apply(null, scores[4]));

for (var value in scores) { // 2 and 4

for (var i=0; i<scores[value].length; i++) {

if (scores[value][i] == maxScore) {

candidates.push( { position: cells[i], value: parseInt(value, 10) } );

}

}

}

// search on each candidate

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

var position = candidates[i].position;

var value = candidates[i].value;

var newGrid = this.grid.clone();

var tile = new Tile(position, value);

newGrid.insertTile(tile);

newGrid.playerTurn = true;

positions++;

newAI = new AI(newGrid);

result = newAI.search(depth, alpha, bestScore, positions, cutoffs);

positions = result.positions;

cutoffs = result.cutoffs;

if (result.score < bestScore) {

bestScore = result.score;

}

if (bestScore < alpha) {

cutoffs++;

return { move: null, score: alpha, positions: positions, cutoffs: cutoffs };

}

}

}

return { move: bestMove, score: bestScore, positions: positions, cutoffs: cutoffs };

}

// performs a search and returns the best move

AI.prototype.getBest = function() {

return this.iterativeDeep();

}

// performs iterative deepening over the alpha-beta search

AI.prototype.iterativeDeep = function() {

var start = (new Date()).getTime();

var depth = 0;

var best;

do {

var newBest = this.search(depth, -10000, 10000, 0 ,0);

if (newBest.move == -1) {

break;

} else {

best = newBest;

}

depth++;

} while ( (new Date()).getTime() - start < minSearchTime);

return best

}

AI.prototype.translate = function(move) {

return {

0: 'up',

1: 'right',

2: 'down',

3: 'left'

}[move];

}

**五、测试数据及其结果分析**

游戏中AI移动算法的步数和得分之间可以用下面这个图线表示

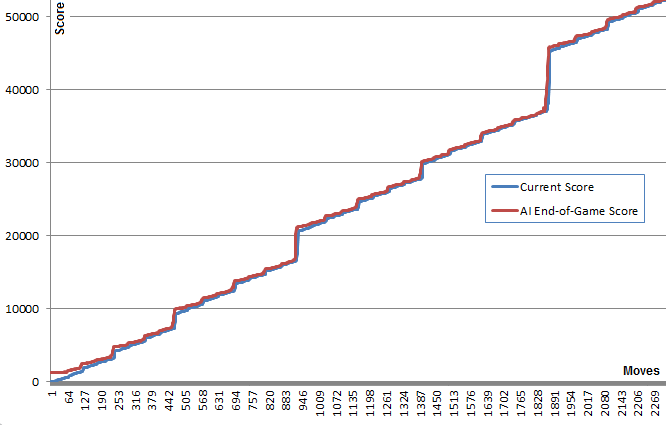


图7 移动步数-分数曲线

从图7中可以看出，分数有一些增加很快的节点，图中的算法解除了2048的限制让AI可以在达到2048后继续移动，可以猜测分数从3.7万增加到4.5万的一段就是4096的合成，中间2万左右是第一个2048块的合成，以此类推，1万分附近是第一个1024块的合成。

为了让移动美观，我还在每一次移动后预留了200毫秒的时间用来让方块从起始位置移动到终点位置，此外AI思考的最短时间为100毫秒，这保证了90%的游戏通过率，经过计时，默认参数下AI通过游戏的时间为6分钟。而如果不考虑美观，把预留时间置零，通过时间就减少到了4分钟，牺牲美观但大大提高了效率。更进一步，把最短思考时间调为10，同样不预留移动时间，这次仅仅用了30秒就通关，可以说是很快了，但与此同时牺牲的是算法对游戏的通过率，有更大的几率AI无法通过游戏。



图8 AI通过了游戏

项目完成后我把网页上传到了服务器里，域名是<http://2048.mcyo.pw>，通过域名访问游戏就可以在任何地方任何设备玩游戏了。

**六、课题完成过程中遇到的问题及解决方法**

问题1：Cookie只能存取字符串，并不能存储数组或者json格式数据，怎么存储排行榜分数成了一个问题

解决方法：通过arry函数创建数组，以字符串的形式存储到Cookie中，读取时先把读取到的字符串用.split(“,”)就可以把字符串重新分成数组

**七、总结**

经过了几天的努力终于把2048这个游戏完成了，从最开始的什么都不懂，只有题目要求的我，开始网上搜寻资料，一点一点的拼凑项目需要的知识，最终选定了HTML这个我有一点基础的语言，加上Javascript，还有让网页美观的CSS，我的项目框架就这样定下了。

2048是一个很好玩的游戏，我从高中就听说过这个游戏，自己和周围的同学都在手机上玩过，是一个很好的益智游戏，玩的时候需要我去思考，选择出上下左右中最优的移动，然后让方块合并。自己玩是玩的很开心，也经常输掉游戏，却从没想过有没有一个最优解的存在从而每一局都能赢而且在很短的时间内。

直到遇到了这个项目，我知道了通过电脑的计算，是存在一个最优解的，只要通过遍历每一步的所有结果，计算能得到的最高分，向能得到最高分的那个方向移动，如此循环，就能得到2048块。

算法定下来了框架也定下来了接下来就是具体的游戏界面设计和写每一个函数了。Javascript语言，是一个和C语言很类似的语言，函数的定义、变量的定义在简单的学习了以后都很容易上手，接下在就是在用的过程中发现某一个函数的功能不理解，随后上网搜索，在搜索中继续学习，迭代自己的能力，到一个项目完成的时候，其实也就是验证自己学有所成的时候。最好的学习就是先从借鉴别人的作品开始，看大牛的代码来理解他们的思想，就像是在和他们交流。通过这样的“交流”，能学到很多网上搜索不到的知识，有时候就是短短几行代码，实现了一个我想了几天也没想出来的功能，然后恍然大悟：原来可以这样。这就是学习的过程。