本次课题：制作HTML5在线五子棋对弈游戏。

预计开发周期：还没想好看心情，先预计7天完成。

备注：最近做学校课题有关数据挖掘的，有时候搞得没得头绪，做个小游戏和大家分享一下，并将小程序开源到Github，欢迎下载。

接上期。传送门：

<http://liuxinyumo.cn/index.php?s=/Home/Index/content/id/6.html>

项目仓库地址：<https://github.com/liuxinyumocn/LittleGame>

现在已经实现了辅助器跟随鼠标移动、给出落点等基本功能，接下来需要独立建设五子棋逻辑模块，注意，在设计逻辑模块时需脱离视图部分，将游戏逻辑独立出来，至于将来从视图上所展现是需要在两者中间增加接口（MVC）。

现在我们脱离Canvas坐标单独研究游戏逻辑（GameCore），该部分我们目前采用Javascript语言完成，而实际上为保证游戏公平性，在未来服务器端也需要有相同的判断逻辑，这里是为了满足制作非联网五子棋的小伙伴。

五子棋游戏逻辑非常简单，建立15\*15大小二维数组用于代表每一个落子点，默认元素值为0，1为黑子，2为白子。Javascript中并没有直接定义二维数组方式，因此我们采用分步法创建一个默认值为0的15\*15二维数组。

var GameCore = function(){ //游戏逻辑核心

this.Data = null;

this.Init();

}

GameCore.fn = GameCore.prototype = {

Init:function(){ // 两层循环初始化15\*15长度的二维数组

this.Data = new Array();

for(var i = 0;i<15;i++){

this.Data[i] = new Array();

for(var n=0;n<15;n++){

this.Data[i][n] = 0;

}

}

}

}

之后是2个基本功能，落子以及胜负判断，我一并给出完整代码，其中注释很清晰：

var GameCore = function(){ //游戏逻辑核心

this.Data = null;

this.Init();

this.CurrentPlayer = 1; //执黑先手

this.Winner = 0;

this.Steps = new Array(); //步骤记录 用于悔棋

this.StepsNum = 0;

}

GameCore.fn = GameCore.prototype = {

Init:function(){ // 两层循环初始化15\*15长度的二维数组

this.Data = new Array();

for(var i = 0;i<15;i++){

this.Data[i] = new Array();

for(var n=0;n<15;n++){

this.Data[i][n] = 0;

}

}

},

Action:function(x,y){ //落子动作 玩家由系统自动判断 提供的落点是棋盘编号

//判断合法性

if(Winner != 0)

return false; //游戏已经结束

if(x<1||x>15||y<1||y>15)

return false;

if(this.Data[x-1][y-1] != 0)

return false; //已经有落子

this.Data[x-1][y-1] = this.CurrentPlayer;

this.AddSteps(x,y);

this.CurrentPlayer = this.CurrentPlayer == 1 ? 2:1;//交换选手

return true;

},

Check:function(){ //判断胜负

//return true 胜者为 this.winner false 无胜者

//扫描是否存在5子连要从4个方向判断 右上（或左下） 右（或左） 右下（或左上） 下（或上）

for(var y=0;y<15;y++){

for(var x=0;x<15;x++){

if(this.Data[x][y] == 0)

continue; //未落子不扫描

var count = 0; //连子数目

//右上扫描

for(var i=1;i<=5;i++){

if(x+i > 14 || y-i<0)

break;

if(this.Data[x+i][y-i] == this.Data[x][y]){

count++;

}else{

break;

}

}

if(count >= 5){

this.Winner = this.Data[x][y];

return true;

}

//右扫描

for(var i=1;i<=5;i++){

if(x+i > 14)

break;

if(this.Data[x+i][y] == this.Data[x][y]){

count++;

}else{

break;

}

}

if(count >= 5){

this.Winner = this.Data[x][y];

return true;

}

//右下扫描

for(var i=1;i<=5;i++){

if(x+i > 14 || y+i > 14)

break;

if(this.Data[x+i][y+i] == this.Data[x][y]){

count++;

}else{

break;

}

}

if(count >= 5){

this.Winner = this.Data[x][y];

return true;

}

//下扫描

for(var i=1;i<=5;i++){

if(y+i > 14)

break;

if(this.Data[x][y+i] == this.Data[x][y]){

count++;

}else{

break;

}

}

if(count >= 5){

this.Winner = this.Data[x][y];

return true;

}

}

}

return false;

},

AddSteps:function(x,y){

var p = this.CurrentPlayer;

this.Steps[this.StepsNum++] = {x:x,y:y,p:p};

}

}

接下来是视图控制器，用于将游戏核心逻辑部分，与游戏界面（视图）部分联系起来(GameControl)。

视图控制器并不是要控制视图中的所有元素，而是要控制和管理活跃元素，诸如棋子，记分板，玩家等等，并负责在游戏开局以及另开一局时的资源初始化。那么有了视图控制器之后，我们在Gobang的类内所产生的临时测试棋子的部分则应注释掉，完全交由GameControl负责。下面开始定义GameControl：

var GameControl = function(MainPage){

this.MainPage = MainPage;

this.GameCore = null;

this.Pieces = new Array();

this.PiecesNum = 0;

this.WinnerPNG = this.MainPage.AddElement("WinnerPNG");

this.WinnerPNG.Left(330);

this.WinnerPNG.Top(170);

}

GameControl.fn = GameControl.prototype = {

Start:function(){ //开始游戏 或者 重新一局

this.GameCore = new GameCore(); //创建游戏核心

this.Show(); //映射核心 该函数会被反复使用 有可能用于重连载入

this.WinnerPNG.Visible(false);

},

Show:function(){

//该函数用于将场景完全恢复后映射GameCore中所存数据

//棋盘数据

this.HidePiece();

for(var x=0;x<15;x++){

for(var y=0;y<15;y++){

if(this.GameCore.Data[x][y] != 0){

this.SetPositon(this.GetPiece(this.GameCore.Data[x][y]),x+1,y+1);

}

}

}

},

HidePiece:function(){

for(var i = 0;i<this.PiecesNum;i++){

this.Pieces[i].Visible(false);

}

},

SetPositon:function(piece,x,y){ //设置位置并显示

var p = PiecePostionKit.BoardPostion(x,y);

piece.Left(p.x);

piece.Top(p.y);

piece.Visible(true);

},

GetPiece:function(Color){ //获取一个空闲棋子 并设置颜色

for(var i=0;i<this.PiecesNum;i++){

if(this.Pieces[i].Visible() == false){

if(Color == 1){

this.Pieces[i].Action("SetBlack()");

}else{

this.Pieces[i].Action("SetWhite()");

}

return this.Pieces[i];

}

}

var p = this.MainPage.AddElement("Piece");

this.Pieces[this.PiecesNum++] = p;

if(Color == 2)

p.Action("SetWhite()");

return p;

},

Down:function(x,y){

var ps = PiecePostionKit.ScreenPostion(x,y);

if(ps.x != -1){//说明有落点

this.GameCore.Action(ps.x,ps.y);

this.Show();

var re = this.GameCore.Check();

if(re){

this.WinnerPNG.Action("SetColor()",this.GameCore.Winner);

this.MainPage.SetPosition(this.WinnerPNG,-1);

this.WinnerPNG.Visible(true);

}

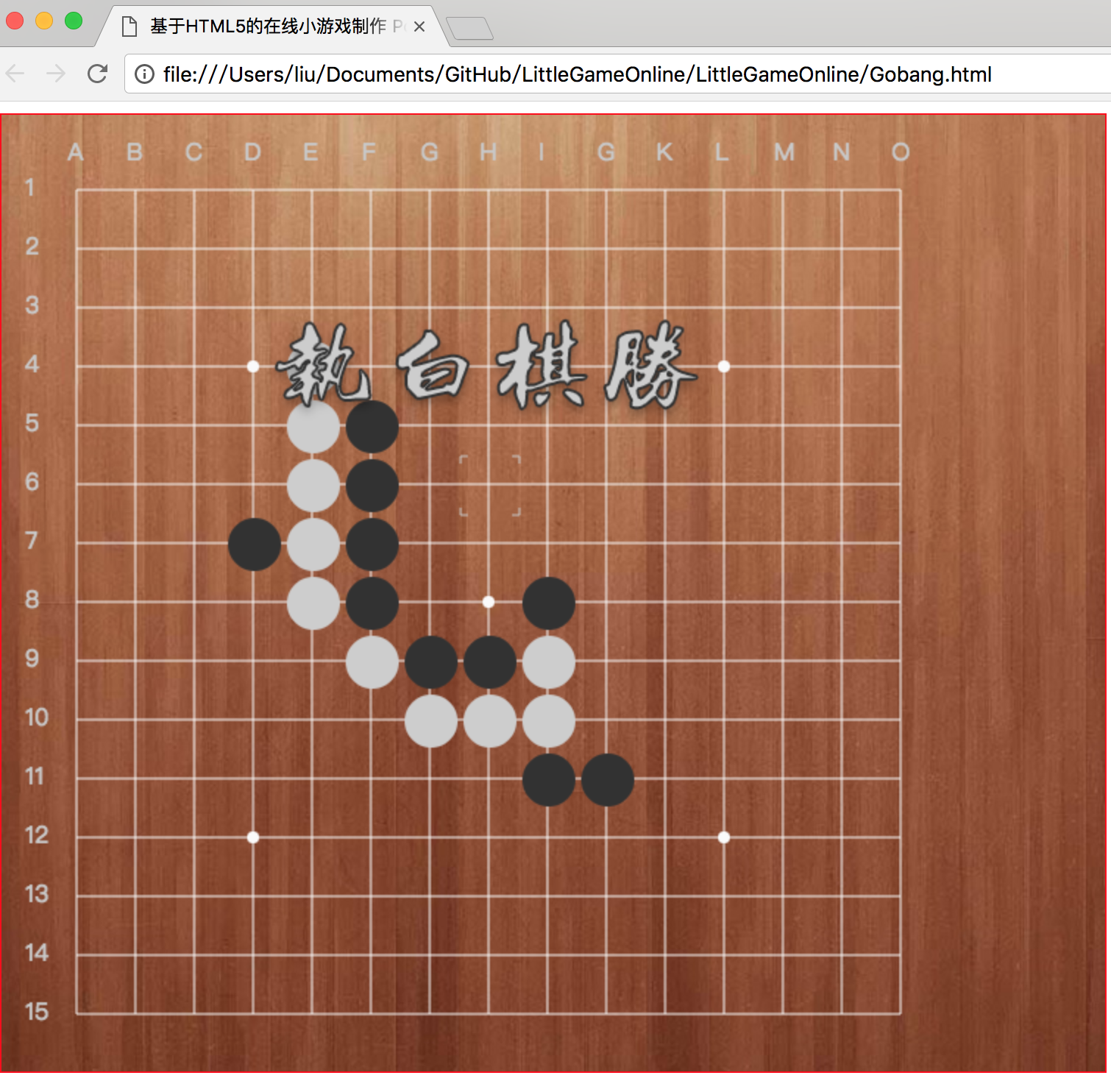
}

}

}

再此期间我定义了几个新的场景元素，获胜提示等。直接参阅新增部分即可，不再陈述。

看下效果：



还算完美，

本篇完毕。