基于WebGL与WebSocket技术实现的立体五子棋

**一．准备工作**

**1.1 HTML5简介**

HTML5是HTML下一个主要的修订版本，现在仍处于发展阶段。目标是取代1999年所制定的HTML 4.01和XHTML 1.0 标准，以期能在因特网应用迅速发展的时候，使网络标准达到符合当代的网络需求。广义论及HTML5时，实际指的是包括HTML、CSS和JavaScript在内的一套技术组合。它希望能够减少浏览器对于需要插件的丰富性网络应用服务（plug-in-based rich internet application，RIA)，如Adobe Flash、Microsoft Silverlight，与Oracle JavaFX的需求，并且提供更多能有效增强网络应用的标准集。[[1]](#footnote-1)

HTML5提供以下一些新特性

Web Socket 定义了一套API, 允许网页能够使用Web Socket协议来和远程主机进行双工通信。

Web Storage 定义了一套API, 能够在Web客户端以key-value的形式对数据进行持久化存储。

Web SQL 定义了一套API, 能够将数据存储在数据库，并使用类似SQL的方式进行查询。

Web Workers 定义了一套API，能够允许脚本运行于后台，进行类似于线程化的操作。

WebGl 定义了一套API, 能够允许在网页中使用类似于Open GL，实际上是一套基于OpenGL ES 2.0的3d图形API。这些API是通过HTML 5的canvas标签来使用的。[[2]](#footnote-2)

本次我主要使用了HTML5中的WebGL和WebSocket技术制作了一个立体的五子棋小游戏。主要使用的技术是WebGL，在所有版本中出现，实现三维场景的绘制； WebSocket实现通信，只使用于联机版本中。由于使用了HTML5中的技术，无法采用IE正常打开使用，建议采用Firefox、Chrome（推荐）等浏览器打开。

**1.2 Three.js简介**

WebGL的原生接口非常复杂，直接使用比较不便，所以我这里使用了three.js，这是一款相当易用的WebGL第三方库。three.js的具体信息可详见杨侃同学的介绍。three.js的源文件在mrdoob-three.js-daf110f.7z压缩包中，除了库的源代码文件外,还有examples文件夹下的诸多示例和docs文件下的index.html说明文档,可以帮助对three.js的学习使用。本作中只使用了其中Three.js和TrackballControls.js两个文件，并已将其复制到网页所在目录下。Three.js是库中实现三维绘制和显示最重要且最基础的部分，TrackballControls.js提供了trackball方式的视角旋转。

**1.3 Node.js简介**

Javascript是一种脚本语言，不需编译。广泛用于客户端[Web](http://baike.baidu.com/view/3912.htm)开发，本作的客户端以网页形式呈现，可以直接由浏览器运行，但要用javascript实现服务端程序，就需要一个解释器。Node.js就是一个服务器端javaScript解释器。（.js只是该软件的名字，以表现这是javascript的解释器，非扩展名）

服务器端并不一定要通过javascript实现，可以其他使用各种语言。但是由于其他语言暂时还没有实现针对WebSocket的API，大多只支持Socket，虽然可以通过Socket自己编写WebSocket的API，但这样需要了解WebSocket通信协议，并自己编写对通信数据进行处理的函数，工作量比较大，较为不便。

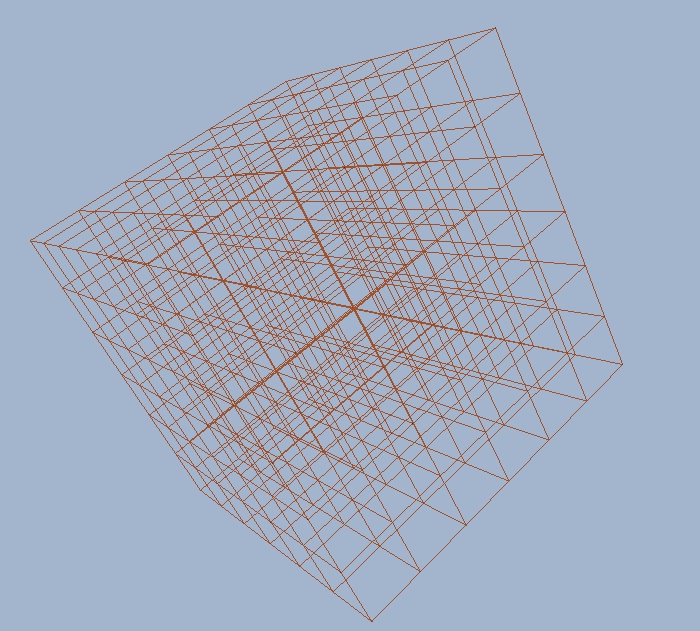
Node.js要使用WebSocket的API，还需要下载第三方开发的WebSocket库，在联机版中node\_modules文件夹下。Node.exe即Node.js的主程序。

**二．功能演示**

**2.1 基本功能及规则**

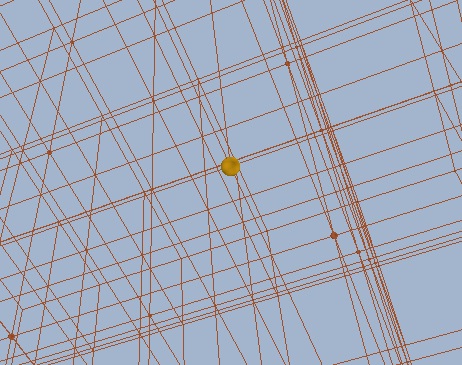
本作是立体棋盘中的五子棋，具体规则由普通五子棋简单推广可得。

初始棋盘如下：

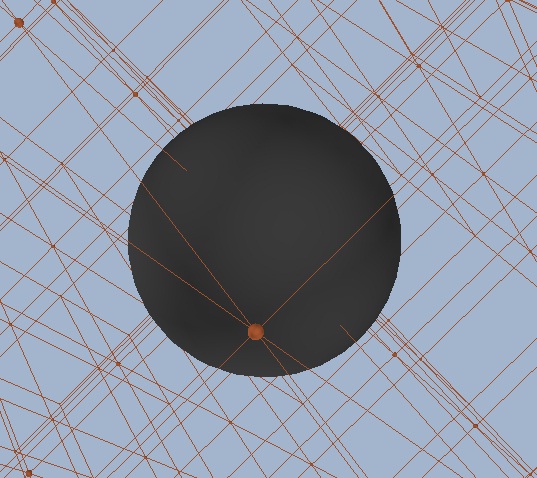


视角的调整方式为鼠标左键拖动旋转，鼠标滚轮调整远近距离。

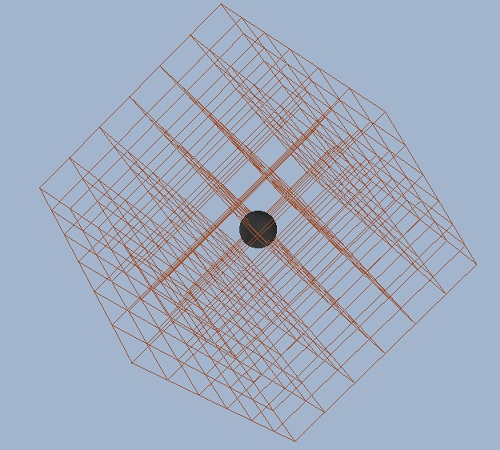
棋盘的每个交叉点上有一个小球，鼠标移上去颜色变亮：



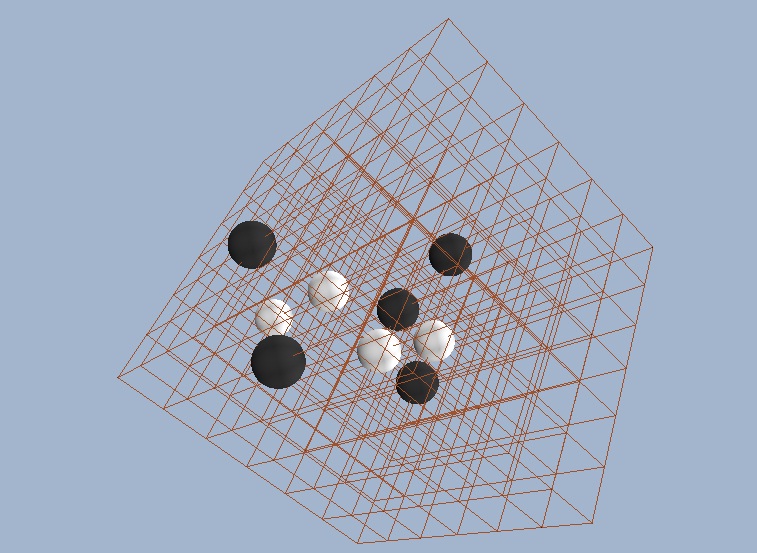
点击小球可以在该节点上落子：



距离拉远后的效果：



胡乱落下几子后的效果:



**2.2 几个不同的版本**

2.1.1 雏形版本

雏形版本为单机版下的version 0.5,version 0.6,version 0.65。为第一次使用javascript编程的热身之作，同时是后续版本的基础。

version0.5仅提供黑子和白子轮流出现功能。0.5取半成品之意，实为第一个保存备份的。

version0.6加入了胜负判定。

version0.65判断获胜方是黑方还是白方，并通过刷新页面达到清空棋盘的目的，相比version0.6增加代码极少，认为不足以称为0.7版。

2.1.2 联机版本

实现联机版本最主要是添加了服务端程序，和客户端服务端之间的通信。服务器端在各个客户端之间传递数据，同时进行棋盘储存与胜负的判定。

启动服务端，双击Run Server.bat即可，Run Server.bat实质是Node ChessServer.js的命令，即用Node运行ChessServer.js，也可以直接在cmd中于该目录下打入Node ChessServer.js运行。

打开服务器端后可以打开客户端网页连接到服务端程序，本作默认连接到本机地址127.0.0.1,实际联机中需要修改客户端网页中的IP地址为服务器的IP。本作品中为方便处理，选择第一个进入的人为黑棋，第二个为白棋，轮流落子。之后进入的作为旁观者，只能观看前两个人下棋。如果前两个棋手中有人退出，则游戏重新开始，从旁观者中最早进入房间的继续作为棋手。

2.1.3 人机版

该版本实现了一个AI与玩家下棋，回归为单机版本，具体介绍见下。

**三．设计过程及代码实现分析**

**3.1 三维场景的绘制**

3.1.1 开启渲染器

var renderer=new THREE.WebGLRenderer();//生成渲染器对象

renderer.setSize(window.innerWidth, window.innerHeight);//指定渲染器大小

document.body.appendChild(renderer.domElement);//追加渲染器至页面

3.1.2 设置摄像机

var camera=new THREE.PerspectiveCamera(45,window.innerWidth/window.innerHeight,1,3000);//生成摄像机对象

camera.position.z=1200;//设置摄像机初始坐标

3.1.3视角旋转的实现

var controls= new THREE.TrackballControls(camera);//生成控制器对象

controls.noPan=true;//禁用右键水平推动功能

controls.minDistance=10;

controls.maxDistance=1500;//设置视点与中心的最大最小距离

3.1.4 设置场景及光源

var scene=new THREE.Scene();//生成场景

light=new THREE.DirectionalLight(0xffffff,0.8,0);

light.position.set(200,200,200);

scene.add(light);

light=new THREE.DirectionalLight(0xffffff,0.8,0);

light.position.set(-500,200,200);

scene.add(light);

light=new THREE.DirectionalLight(0xffffff,0.8,0);

light.position.set(200,-500,200);

scene.add(light);

light=new THREE.DirectionalLight(0xffffff,0.8,0);

light.position.set(200,200,-500);

scene.add(light);

//加入从四个方向飞别射来的平行光线

3.1.5 绘制棋盘

var material=new THREE.LineBasicMaterial({color:0xA0522D,opacity:1});

for (var i=-3;i<=3;i++) {

for (var j=-3;j<=3;j++) {

geometry=new THREE.Geometry();

geometry.vertices.push(new THREE.Vector3(100\*i,100\*j,-300));

geometry.vertices.push(new THREE.Vector3(100\*i,100\*j,300));

line=new THREE.Line(geometry,material);

scene.add(line);

}

}

for (var i=-3;i<=3;i++) {

for (var k=-3;k<=3;k++) {

geometry=new THREE.Geometry();

geometry.vertices.push(new THREE.Vector3(100\*i,-300,100\*k));

geometry.vertices.push(new THREE.Vector3(100\*i,300,100\*k));

line=new THREE.Line(geometry,material);

scene.add(line);

}

}

for (var j=-3;j<=3;j++) {

for (var k=-3;k<=3;k++) {

geometry=new THREE.Geometry();

geometry.vertices.push(new THREE.Vector3(-300,100\*j,100\*k));

geometry.vertices.push(new THREE.Vector3(300,100\*j,100\*k));

line=new THREE.Line(geometry,material);

scene.add(line);

}

}

//棋盘由三个方向各49条线段组成

var group=new THREE.Object3D();

for (var i=-3;i<=3;i++) {

for (var j=-3;j<=3;j++) {

for (var k=-3;k<=3;k++) {

sphere = new THREE.Mesh(new THREE.SphereGeometry(1,16,16), new THREE.MeshLambertMaterial({color:0xA0522D}));

sphere.position.set(100\*i,100\*j,100\*k);

group.add(sphere);

}

}

}

scene.add(group);

//绘制每个节点上的小球，并将小球统一储存于group对象中

**3.2 鼠标位置的判断**

判断鼠标指针是否指在小球上是一个比较复杂的问题，因为我们看到屏幕上显示的图像其实是一个三维模型的投影，不是一般意义上的二维图像。

3.2.1 鼠标事件的监听

document.addEventListener('mousemove',onDocumentMouseMove,false);//鼠标移动的监听

document.addEventListener('mousedown',onDocumentMouseDown,false);//鼠标点击的监听

3.2.2 鼠标位置的判定

var vector = new THREE.Vector3( (event.clientX / window.innerWidth) \* 2 - 1, - (event.clientY / window.innerHeight) \* 2 + 1 , 0.5 );//获取鼠标的二维位置

projector.unprojectVector(vector,camera);//将鼠标的位置转化为三维位置

var ray=new THREE.Ray(camera.position,vector.subSelf(camera.position).normalize());//创建从摄像机到小球方向的射线

var intersects=ray.intersectObjects(group.children);//把group中的对象即小球中与射线相交的存入intersects数组

if (intersects.length>0) {//如果数组不为空，即存在与射线相交的小球

if (INTERSECTED!=intersects[0].object) //数组中第一个即离摄像机最近的小球就是鼠标指针所指小球

{

//若鼠标在小球上的动作

}

}

上述代码的原理，通过光线的直线传播容易理解。

**3.3 WebSocket通信的实现**

3.3.1 服务端实现

wsServer.on('request',function(request) {

var client={

connection:request.accept(null,request.origin),//定义连接

index:clients.length

}//为了便于判断客户端的身份，将连接和一个作为编号的整数定义在一个结构中，以下可通过cilent.index获取客户端编号

//省略客户端建立连接的处理

client.connection.on('message',function(message){

//省略客户端发送信息的处理

});

client.connection.on('close',function(connection){

//省略客户端断开连接的处理

});

});

我研究学习的例子中没有考虑对客户端身份辨识问题，以下为该情况下的代码

wsServer.on('request',function(request) {

var connection=request.accept(null,request.origin);

//省略客户端建立连接的处理

connection.on('message',function(message){

//省略客户端发送信息的处理

});

connection.on('close',function(connection){

//省略客户端断开连接的处理

});

});

发送数据语句为client.connection.sendUTF()。

JSON.stringify({type,data})函数则可将一般数据结构转化为JSON字符串用于网络传输。

接受数据则是在connection.on('message',function(message)中调用message变量。

3.3.2 客户端实现

var connection=new WebSocket('ws://127.0.0.1:8080');//建立连接，修改IP地址于此处

connection.onopen=function(){

//连接建立的动作

};

connection.onerror=function(error){

//连接失败的动作

};

connection.onmessage=function(message){

var json=JSON.parse(message.data);//将接受到的JSON字符串转化为原来的数据

//对接受数据的处理

}

}

相互传输的数据有三种ctrl,step,board，棋手使用客户端向服务端发送step数据，表示落子位置，服务端将其转发至所有客户端，同时在服务端进行判断，对于不同情况如游戏结束，棋手逃离，通过ctrl类型数据告知客户端，同时通过board向刚进入的观众客户端传输目前的棋盘局势。

**3.4 AI的实现**

关于AI的实现，代码较长，具体实现细节较多，不便具体分析，简单讲讲主要思想。

该AI采用搜索算法，基本思想为遍历现局面下以后所有落子的可能性，并取最优策略。搜索到分出胜负的局面是不可能的，状态数过大，所需等待事件无法接受，所以采取限制搜索深度，只搜索以后的有限步数，并通过估价函数对某一局面评估出一个分数，作为比较的标准，在估价函数设计合理的情况下，分数最高的下法为最好或较好落子选择。

决定搜索算法性能的因素，估价函数设计的合理性和搜索的深度都非常重要。我采用的估价函数是分别给两子，三子，四子，五子以上给定一个分数，如果一头或两头被堵，该分数会按一定规则减小，对自己和对手都按照这种规则累加分数，自己的分数减去对手的分数作为该局面的分值，具体细节可见代码。搜索的深度不能过大，会造成计算时间过长。同时为了决策更优，又需要尽可能加大搜索深度，此时可进行搜索剪枝，即搜索时抛弃一些不合法或明显较差额下法，可大大减少搜索宽度，减少搜索时间。一个剪枝是只搜索已有棋子位置附近的位置，因为五子棋中突然在较远处落子往往是不合理的着法。同时采用alphabeta搜索，alphabeta是一种基础搜索方法，是在最大最小搜索中加入最优化剪枝的搜索算法。有关资料再网上可以方便地找到，而且表述相当清晰易懂，故此处不作赘述。

**四．尚待优化的问题**

本作品主要目的在于学习，以设想的基本实现为目标，同时时间精力有限，造成作品中还有很多缺陷，代码优化不够，风格较差，不易维护，而且很多实现比较粗糙。对于其中大部分问题，我已经考虑到了解决方案，不过暂时没有实现。由于一些可优化问题的措施存在相关性，以下的叙述的问题中，分类比较混乱，望读者见谅。

**4.1 美观问题**

本作品只使用了最基础的图形，没有特意进行美化。对于背景，棋盘，棋子，皆可通过一些装饰美化。同时可提供一个界面并通过它实现更多功能，如落子剩余时间。

**4.2 多用户问题**

本作品目前只能同时运行一局游戏，可以参照一般游戏大厅的设计，提供多对用户同时进行游戏的设计，同时可加入用户注册登录系统，记录用户积分等，直接在代码中修改服务器地址也显得不方便，可将其置于登录界面中。

**4.3 安全性问题**

本作品中假设所有人都正常使用客户端，服务端不对接受数据的合法性进行判断，如果在实际应用中安全性不够，特别对于javascript这个脚本语言。如果用户修改客户端代码，可进行原来不能做到的非法操作。这是由于是否能落子是在客户端的towalk变量决定的，而服务器接受到落子坐标和颜色就会视为合法并执行。同时，本作品未在实际网络上测试过，只是在本机上测试，推测如果出现网络拥堵数据丢失的情况，可能造成两方都无法继续落子的bug。而要解决这些问题，只需要取消towalk变量，并在服务器端判断数据来源和合法性，这只要一些简单的修改。对于一些网络可能导致的其他问题，也可通过检验收到的数据并在收到非法数据时进行回复等措施解决。

**4.4 AI性能问题**

本作品中AI的搜索深度不得不取得很浅，只有两层。测试过四层，就需要等待很长时间。分析可能有以下原因：

1.估价函数的计算效率过低。

2.搜索顺序不当，使alphabeta搜索的剪枝效果不够。

3.Javascript本身的运行效率不高。

不过经过本人测试，AI的表现还是可接受的。

**4.5 游戏性问题**

可以加入独创的规则，比如三人即以上在一个棋盘中进行游戏，比如加大棋盘。这里要提的是之所以采取7\*7\*7的棋盘是由于当初试过更大的棋盘，感觉视觉上比较杂乱，同时更大的棋盘和游戏人数增加对AI也是一个新的挑战。同时顺道一提关于立体五子棋的创意，其实确实是我自己想出来的，不过后来网上一查，发现已经有人捷足先登了，制作出了现实中的立体五子棋，同时虚拟的立体五子棋也看到有人用flash制作了一个3\*3\*3的，实质为bingo立体版的东西。

另外五子棋毕竟是一种比较简单，游戏性本身不强的游戏，通过HTML5技术可以实现许多更具趣味性和实用性的应用。

**参考资料**

杨侃 《A Brief Introduction to WebGL》

刘江沙 《websocket在实时web中的应用》

Bruce Moreland 《对弈程序基本技术》

1. 引用自维基百科 [↑](#footnote-ref-1)
2. 引用自http://www.cnblogs.com/jyli/archive/2010/07/25/1784902.html 2010-07-25 李嘉昱 [↑](#footnote-ref-2)