目录

一.系统功能说明

本游戏是一款简单的双人对战的台球游戏。它包含了一款台球游戏所具有的基本功能，如台球击打、积分变换、积分系统等，也包含了其他攻击功能，如视角变换、犯规系统、力度控制、双人对战等。它采用鼠标操作与键盘操作相结合，有变换视角、声效等，较好的贴近了现实中的台球游戏。

1.基本功能

（1）游戏的环境：游戏由台球桌面，灯光，以及状态板构成。初始情况下，白球放置在中线近侧，十五个彩球放在中线较远侧，几个球带与实际台球桌分布相符。状态板上有包括当前击球玩家，两位玩家的比分，以及击球力度等。

（2）游戏的得分与扣分：玩家将一个彩球击入球袋中后，玩家加一分；如果使白球落入袋中，则玩家扣一分，游戏恢复到上一个状态。

（3）游戏的启动与退出：玩家通过按下空格键（SPACE）可以重新开始一盘游戏，按下 ESC 键可以退出程序。

2.高级功能

（1）视角变换：通过上下左右键可以改变游戏中的视角，可以让玩家获得不同的击球视角，更准确的判断击球位置和击球方向，比二维游戏更加贴近真实情况。

（2）力度控制：玩家在选择好击球角度后，通过按下鼠标的长短控制击球的力度，相应的力度大小实时显示在状态栏中。

（3）双人对战：两名玩家轮流击球，如果一名玩家得分，则可再打一杆，否则交换球权。

二．使用说明

1.安装说明

（1）编译出可执行文件

OpenGL 环境的安装：

方法一：

找到目录..\VC\Tools\MSVC\14.10.25017\include\gl(没有的话就建立一个)。 将 glut.h 放到里面。找到目录..\VC\Tools\MSVC\14.10.25017\lib\x86

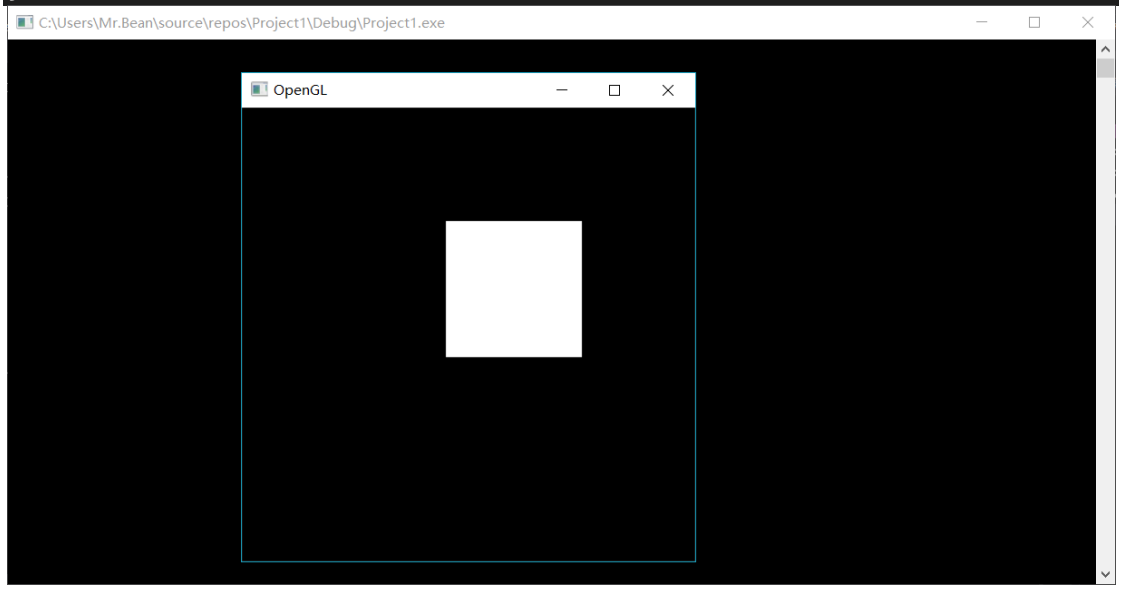
将 glut32.lib 放到里面。最后把 glut32.dll 放到 C:\Windows\SysWOW64(64 位系统）。

方法二（使用 VS 的插件）：打开 vs 创建一个 C++项目（Win32 控制台-空白项目），然后点击 项目—管理 Nuget 程序包，然后点击浏览—在搜索栏输入 NupenGL，然后安装（如果有两个就安装两个）。

运行下面的代码，得到相应的结果则证明已配置 OpenGL 环境成功。接着打开项目工程文件，生成解决方案，运行即可。

```
#include <GL/glut.h>
void Show()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-0.1f, -0.1f, 0.5f, 0.5f);
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(400, 400);
    glutCreateWindow("OpenGL");
    glutDisplayFunc(Show);
    glutMainLoop();
    return 0;
}
```



（2）直接运行：直接打开\src\Project4\x64\Debug\Project4.exe

2.使用手册

（1）双人模式：玩家 1 先击打球，每个球 1 分；

（2）犯规：白球进洞或者没有碰到彩色球，-1 分；

（3）长按鼠标可以实现力度控制，最大 100%；

（4）移动上下左右键可以变换视角，实现精确击打。

三．程序结构

1.opengl 有大量的函数，生涩难懂，因此本任务多是照着教程一步步写下来的

和我们以前写的第三次图形库很像，先初始化然后注册一系列的回调函数

```
int main(int argc, char* argv[])
{
    // initialise GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);//显示模式
    glutInitWindowSize(1280, 720);//窗口大小
    glutCreateWindow("Billiard Ball game"); //窗口名字

    // register callbacks 注册一些列回调函数
    glutKeyboardFunc(keyboard);//键盘功能
    glutSpecialFunc(special);//响应键盘方向控制键
    glutMouseFunc(mouse);//鼠标功能
    glutPassiveMotionFunc(passive);//相应鼠标没有被按下去时，移到鼠标的情形
    glutDisplayFunc(display);//显示当前窗口
    glutTimerFunc(10, timer, 1);
    quadricObject = gluNewQuadric();
    glutMainLoop();//程序进入事件处理循环。该函数必须是main主函数的最后一条语句。
    return 0;
}
```

2.初始化完毕后开始绘制整个场景 我们对 opengl 的使用并不熟练，因此使用了网上的一些教程上的代码（截图是部分代码）

```
// draw scene
void display(void)//显示场景
{
    // update the game to the current time
    int currentTime = glutGet(GLUT_ELAPSED_TIME);//计算程序每秒的帧数
    table.update(currentTime);

    // clear window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);// 清除颜色缓冲区和深度缓冲区
    glEnable(GL_DEPTH_TEST);//启用之后,绘制时检查，当前像素前面是否有别的像素，如果别的像素挡道了它，那它就不会绘制，也就是说，OpenGL就只绘制最前面的一层。

                            // window sizes
    int width = glutGet(GLUT_WINDOW_WIDTH);//得到窗口参数属性
    int height = glutGet(GLUT_WINDOW_HEIGHT);

    // set perspective projection
    glMatrixMode(GL_PROJECTION); //将当前矩阵指定为投影矩阵
    glLoadIdentity(); //然后把矩阵设为单位矩阵
    double fieldOfView = 50.0;  //越小看到的范围越小
    double aspectRatio = (double)width / (double)height;//实际窗口的纵横比
    double nearPlane = 0.1;//近处的裁面
    double farPlane = 100.0;//远处的裁面
    gluPerspective(fieldOfView, aspectRatio, nearPlane, farPlane);

    // set camera
    glMatrixMode(GL_MODELVIEW);//模型视景矩阵
    glLoadIdentity();
    glTranslated(0, 0, -8); //函数参数(x, y, z)
    glRotated(cameraAngle2, 1, 0, 0);//根据相机角度调整画面位置 //使图象沿X轴旋转
    glRotated(cameraAngle1, 0, 1, 0);//使图象沿Y轴旋转
```

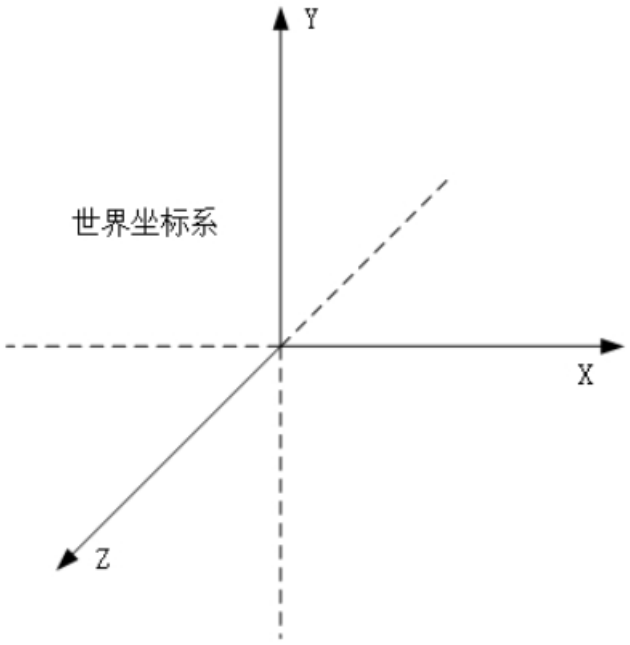
3. 绘制台球桌和各种物件

流程如下：

保存当前坐标系 glPushMatrix();

平移（glTranslated），变换坐标系（glRotated），调用绘制函数(drawXXX)

恢复坐标系 glPopMatrix();




```
// draw floor
glColor3d(0.5, 0.5, 0.4); // grey colour
setShininess(0.0); // not shiny
glPushMatrix();
glTranslated(0, -2, 0);//向下平移两个单位
drawCuboid(20, 0.1, 20);//宽度，高度，长度
glPopMatrix();

// draw walls
glColor3d(0.4, 0.5, 0.4); // light green colour
glPushMatrix();
glTranslated(-10, 0, 0); drawCuboid(0.1, 20, 20);
//函数功能：沿X轴正方向平移x个单位(x是有符号数)
//沿Y轴正方向平移y个单位(y是有符号数)
//沿Z轴正方向平移z个单位(z是有符号数)
glTranslated(+20, 0, 0); drawCuboid(0.1, 20, 20);
glColor3d(0.3, 0.4, 0.3); // dark green colour
glTranslated(-10, 0, -10); drawCuboid(20, 20, 0.1);
glTranslated(0, 0, +20); drawCuboid(20, 20, 0.1);
glPopMatrix();
```

（drawCuboid 设置参数 1,1,1 的效果）

4. 交换球权

如果击球之后符合换球权条件，那么我们将调用函数交换球权。

由于函数是一直处在循环里的，刚开始我没能理解所以一直实现不了，后来想起来 CAD 开发中用到的状态变量，发现确实很好用。

```
if (!moving())
{//所有球静止了
    if (curNumber == visibleNumber() && isPlay && isChange) {//犯规情况下，球个数不变；没打进，球个数不变这些情况都要交换球权
        switchPlayer();//交换球权
        renew(); //update the original position
    }
    else
    {
        if (balls[0].getVisible())
        {
            curNumber = visibleNumber();//更新当前数量的球
            renew();
        }
        else
        {
            restore();
            switchPlayer();
        }
    }
    isChange = false;//防止重复更新
}
else
{
    isChange = true;
}
}
```

5. 记分牌绘制

绘制物体是一个调整参数的过程

只要按照相对的位置，逐步调整参数的大小（x,y,z）最终得到我们满意的答案

6. 字体绘制

Opengl 没有内置函数，要自己写，我从网上获得资料后加到了程序中，并改成自己喜欢的样式

```
void glDrawString(unsigned char *str)
{
    GLYPHMETRICSFLOAT pgmf[1];

    HDC hDC = wglGetCurrentDC();
    HFONT hFont;
    LOGFONT lf;
    memset(&lf, 0, sizeof(LOGFONT));
    lf.lfHeight = 1;
    lf.lfWidth = 0;
    lf.lfEscapement = 0;
    lf.lfOrientation = 0;
    lf.lfWeight = FW_NORMAL;
    lf.lfItalic = FALSE;
    lf.lfUnderline = FALSE;
    lf.lfStrikeOut = FALSE;
    lf.lfCharSet = GB2312_CHARSET;
    lf.lfOutPrecision = OUT_TT_PRECIS;
    lf.lfClipPrecision = CLIP_DEFAULT_PRECIS;
    lf.lfQuality = PROOF_QUALITY;
    lf.lfPitchAndFamily = VARIABLE_PITCH | TMPF_TRUETYPE | FF_MODERN;
    lstrcpy(lf.lfFaceName, "宋体");
    hFont = CreateFontIndirect(&lf);
    //设置当前字体
    SelectObject(wglGetCurrentDC(), hFont);
    DWORD dwChar;
    int ListNum;
```

7. 速度显示，分数显示，当前玩家显示

这些内容其实是一个东西，因此放到一块儿讲，为了封装良好，写成一个个独立的函数。然后调整参数显示到计分板上，任务就完成了。

```
void Table::paintScore() {
    glPushMatrix();
    selectFont(24, ANSI_CHARSET, "Comic Sans MS");
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.5f, 0.5f, 0.5f);
    glRasterPos3f(-4.5f, 0.4f, 2.0f);// 高度
    glTranslated(-3, 0, 0);
    char str1[100] = "P1:";
    char temp1[1000];
    sprintf_s(temp1, "%d", scores[1]);
    strcat_s(str1, temp1);
    //glTranslated(0,0, 0);
    drawString(str1);
    glPopMatrix();

    glPushMatrix();
    selectFont(24, ANSI_CHARSET, "Comic Sans MS");
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.5f, 0.5f, 0.5f);
    glRasterPos3f(-4.5f, 0.4f, -1.5f);// 高度
    glTranslated(-3, 0, 0);
    char str2[1000] = "P2:";
    char temp[100];

    sprintf_s(temp, "%d", scores[0]);
    strcat_s(str2, temp);
    //glTranslated(0,0, 0);
    drawString(str2);
    glPopMatrix();
}
```

8. 模拟球的碰撞

由于最多只有 16 个球，遍历所有球的开销是可以接受的。找到两个球，判断距离，相撞的更新速度。找到球和洞，判断距离，进洞的把他变成不可见。遇到白球进洞，放回中间，游戏恢复到上一个状态。

```
void Table::update(int currentTime)
{
    double timeStep = 0.001; // update interval in seconds
    int i, j;
    // update the time until it catches up with current time
    while (time < currentTime)
    {
        // collision detection for each ball
        for (i = 0; i < 16; i++)   //遍历所有球
        {
            // collide ball with holes and cushions
            balls[i].collideHoles();
            balls[i].collideCushions();
            // collide ball with other balls
            for (j = 0; j < i; j++)
            {
                balls[i].collideBall(balls[j]);
            }
        }

        // update ball speeds and positions
        for (i = 0; i < 16; i++)
        {
            balls[i].updateSpeed(timeStep);
            balls[i].updatePosition(timeStep);
        }

        // if cue ball is hidden, place it at the centre
        if (!moving() && !balls[0].getVisible())
        {
            balls[0].setPosition(0, 0);
```

四．系统设计难点及其解决方法

1.碰撞中的物理问题

想要解决台球的碰撞问题，不得不涉及到一个问题，即非对心碰撞。

我们考虑理想环境下台球的碰撞问题

1）**正碰，一球静止 一球运动**：很简单，交换速度即可；

2）**正碰，两球都在运动**:可计算得两球碰撞后速度；

如图所示，在光滑水平面上，质量为 m1、m2 的两球发生对心弹性碰撞，碰撞前速度分别为 v1 和 v2 设碰撞后速度变为 v1 ′ 和 v2 ′ 在弹性碰撞过程中，分别根据动量守恒定律、机械能守恒定律得：

$$m1v1+ m2v2= m1v1' + m2v2' \qquad ①$$

$$\frac{1}{2}m1v12 + \frac{1}{2}m2v22 = \frac{1}{2}m1v1' 2+ \frac{1}{2}m2v2' 2 \qquad ②$$

由①、②解得：

$v1' = \frac{(m1-m2)v1+2m2v2}{m1+m2}$ ；

$v2' = \frac{(m2-m1)v2+2m1v1}{m1+m2}$ ；

3）**斜碰，一动一静**；

根据动量守恒定律有：$m1\vec{v}1 + m2\vec{v}2 = m1\vec{v}10$

得： $\vec{v}1 + \vec{v}2 = \vec{v}10$ ；

两边平方得：$v12 + v22 + 2\vec{v}1 \cdot \vec{v}2 = v102$ ①

又根据能量守恒定律有：$\frac{1}{2}m1v12 + \frac{1}{2}m2v22 = \frac{1}{2}m1v102$

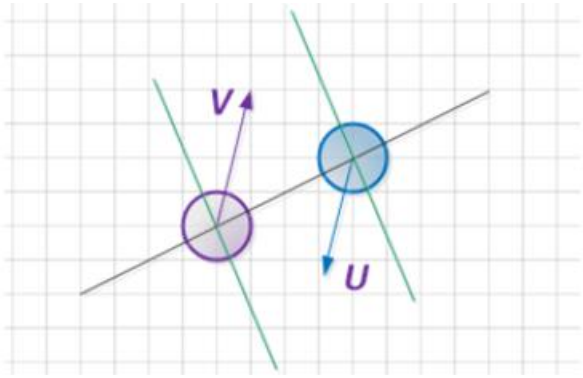
得： $v12 + v22 = v102$ ②

①-②得： $2\vec{v}1 \cdot \vec{v}2 = 0$

即 $\vec{v}1 \perp \vec{v}2$

可见，碰撞后两球成直角离开。

4）**斜碰，两个都在动**；



动量是向量，其在正交的两个方向上，互相守恒，我们选取两球圆心的直线为 x 轴，垂直于圆心直线的为 y 轴，如上图所述，

X 轴上满足动量守恒：$m1vx + m2ux = m1vx' + m2ux'$

并假设两球碰撞是完全弹性碰撞，得：

$vx' = \frac{(m1-m2)vx+2m2ux}{m1+m2}$ ；

$ux' = \frac{(m2-m1)ux+2m1vx}{m1+m2}$ ；

假设两球质量相等，则：$vx' = ux$ ， $ux' = vx$ ，即交换速度

Y 轴上两球分速度不变，vy ′ = vy ， uy ′ = uy ；

碰撞的调试出过很多问题，纸笔反复验算好多次，确实比较难写，也参考了网上一些源代码。

2.击球力度的控制：

可通过计时器回调函数实现，使用鼠标回调函数确定某个全局变量的值，再根据此此值和上次计时器函数使用时的此值确定鼠标的状态并加以操作，最终可实现：①鼠标左键一直处于按下状态时，白球初始速度不断增加②鼠标左键由按下转为松开状态时，运行击球函数③储存鼠标当前状态，为下次计时器回调提供依据。

相关代码：

```
'''
void timer(int timerid)
{
    if (timers == 1 && value == 1)
    {
        speed += 0.3;
    }
    else if (timers == 2 && value == 1)
    {
        if (!table.moving()) table.shoot();
        speed = 1;
    }

    if (timers == 1) value = 1;
    else if (timers == 2) value = 2;

    glutTimerFunc(10, timer, 1);//10 ms
}

// handle mouse clicks
void mouse(int button, int state, int x, int y)//鼠标监听事件
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) //如果鼠标左键单击并且按下,开始计时
    {
        timers = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
    {
        timers = 2;
        shootsound();
    }
}
```

3.WINDOWS 系统 API 的使用

在程序 opengl 的整体框架中引入了 windows 的 api，实现从系统获得时间并且接入了系统媒体接口。导入的头文件有 mmsystem.h 和 windows.h。

```
#include <stdio.h>
#include <windows.h>
#include <mmsystem.h>
 #pragma comment(lib, "winmm.lib")
```

由于水平有限这里只能简单的调用来播放声音

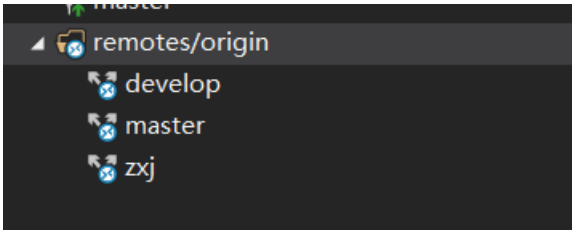
```
void shootsound() {
    PlaySound(TEXT("6530.wav"), NULL, SND_FILENAME | SND_ASYNC);
}
```

更深入的还有播放不同格式的音频，实现暂停、循环，以及模拟双声道等

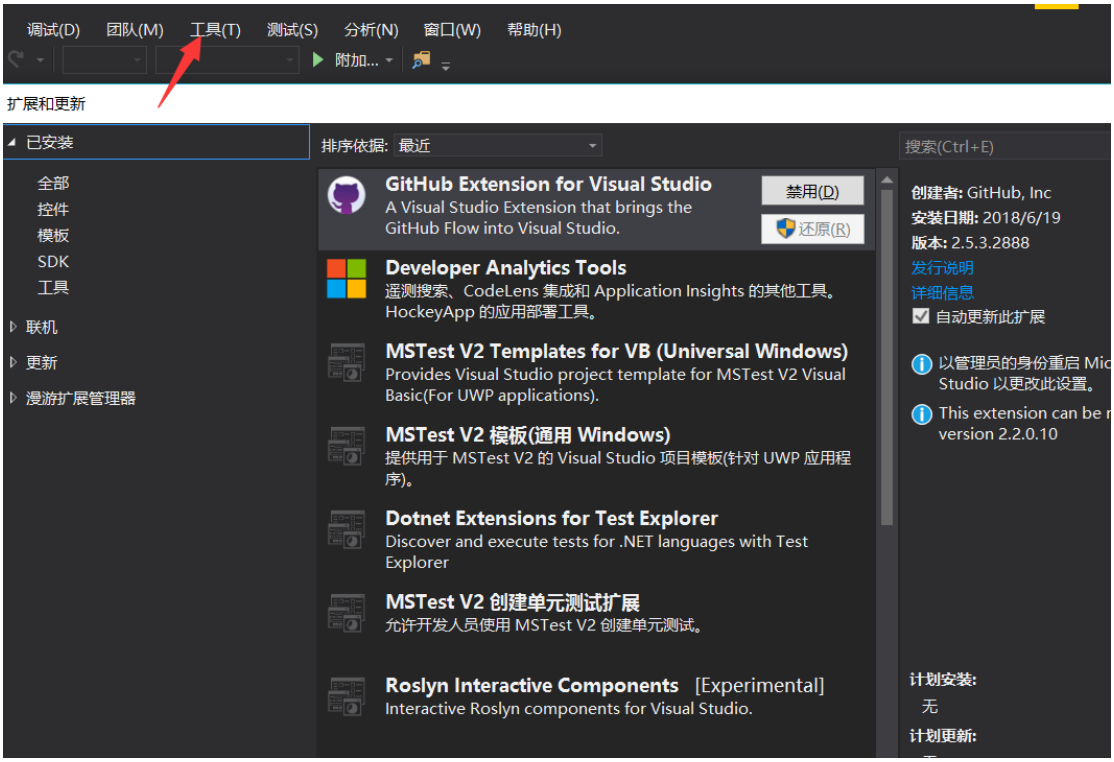
同属于 windows.h 下还利用了一个函数 GetLocalTime(&time),当交换球权的时候，利用此函数把系统时间赋给结 SYSTEMTIME 结构体，调用其中的 wSecond 参数;同样的操作保持另一个结构体与系统时间同步,其中差值即为经过的时间，计算后显示在计分板上即可实现倒计时。

4.Github 多人开发：VS 环境

在网页版 github 里创建好仓库然后邀请组员加入，一个 master 分支一个 develop 分支（开发环境下最新版本），开发成功后合并到 master 分支



使用方法：



安装完后启用即可，然后设置一下密码，可视化界面操作很方便

更改某个文件后提交至 develop 分支，测试完成后合并到 master 分支

五．本程序亮点和小组成就

1.3D 模式下的台球，击球趣味性更强，控制更加方便灵活。

2.更加真实的碰撞效果，游戏体验更佳。

3.小组分工严密，每个人完成自己的部分，把函数都封装起来，方便相互调用。

六．不足之处

1.台球碰撞不够真实：本程序考虑的台球碰撞都是基于两个质点间的碰撞，没有考虑台球体积造成的影响，因此与真实的台球碰撞相比仍有一定差距。

2.击球点位置：在球杆击打主球时，可以有不同的击球点。最常见的击球点有 3 个，即高点、中心点和低点。此外还有 6 个比较常用的击球点，分别是左上点和右上点、左 侧点和右侧点、左下点和右下点。除了击中中心点，击中其他点时白球都会发生一定的旋转，因此会影响白球的运动及其与其他球碰撞时两球的状态，这一考虑过于复杂，因此我们的程序未能考虑击球点的变化

3.还有一些功能没有实现，包括前面的开始界面、使用说明等界面以及已进球的显，台球轨迹预测等。

七．人员分工说明

小组成员分工明确，在组长的带领下很好的完成了这个任务