场景说明

汇众标志场景(第一个场景)

一小段轻音乐后自动进入游戏说明场景

此处简单的介绍了人物主要技能的出招按键，技能的作用和拳皇中的八神技能大致相同，按下空格键后进入格斗场景。

此场景可以两个玩家 PK，二者都是八神角色，技能一样，下面对各玩家的按键以及角色技能的使用方法和功效做出详细介绍：

玩家 1 按键 上下左右 WSAD　　重拳 U　轻拳 I　重腿 J　轻腿 K

玩家 2 按键 上下左右 ↑↓←→　重拳 4　轻拳 5　重腿 1　轻腿 2

基本操作

1.左右控制人物前进与后退，当对方处于攻击状态时，按后退键可以格挡

2.下键下蹲，上键跳跃

3.连按两次后退，人物向后小跳一段

4.连按两次前进，人物向前奔跑

小技能

1.暗勾手

按键：后下前 重拳

向前方发出蓝色火花贴地前行碰到对方后对其造成伤害，属远程攻击，可上下格挡

2.鬼烧

按键：下前　重拳

向上跳起甩出蓝色火焰攻击对方，近距离攻击，可打倒对方，对方可上下格挡

3.葵花三式
按键：下后　重拳 第一式和第二式将结束时重复按键即可连招
向前突击，攻击对方，近距离攻击，前两式可上下格挡，第三式只能上格挡

4.屑风
按键：下后前 重拳
命中后将对方从前方拉到后方，对方状态要一小段时间恢复，此技能主要用于衔接其它技能，比如暗勾手，神技，葵花三式或者其它普通攻击，需要与对方贴身才可命中，对方站立和下蹲状态时无法格挡，贴身必中。

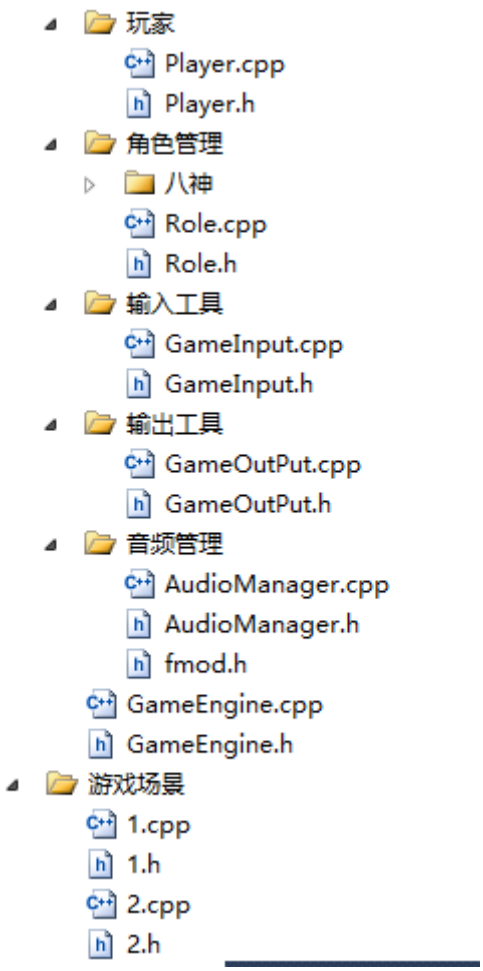
绝招
1.八稚女
按键：下前后 重拳
技能发出后贴地快速前行，命中后对对方持续攻击，最后一击时将对方抛出，可以超远距离攻击，对方可以格挡

2.暗削
按键：下前下前 重拳
此技能不可单独释放，只能在八稚女最后一击时按键连招，发出后将对方向上抛至空中在释放水柱打击对方。

3.八酒杯
按键：下前下前 重拳
此技能可瞬发，也可点击重拳时按住不放，等待合适时机再松开发出，会释放一个快速前行的冰柱，打重对方后将其冻在当前位置一段时间，不能移动，在此期间对方将没有触碰体积，可以穿越，之后可释放其它技能攻击对方。
4.神技
按键：下前后 重腿
此技能是八神终极技能，可攻击距离最远，技能触发后快速向前跳跃，途中碰到对方即将其按至地面，向前滑行一段时间后将对方举起，一段时间后将对方抛出。

提示：部分技能对应有游戏截图，游戏中技能的使用技巧玩家可自己探索。格斗场景属于 Test 模式，释放技能无需耗气，没有设置时间和血量限制，玩家可以尽情的打斗。

框架说明



Main.cpp:

游戏入口点，加载游戏各个场景，启动游戏循环。

GameEngine:

应用程序类的定义与声明,提供了应用程序初始化、运行、销毁等方法, 本类包含了框架所需要各种工具类，输入、输出、场景切换、动作、技能等

Scene.cpp 和 Scene.h

场景类，定义了场景的框架，场景的进入初始化，绘制，逻辑，退出均为虚函数，游戏中各个场景类皆继承自此父类，引擎利用多态切换运行不同游戏场景。

Skill.h 和 Skill.cpp

技能类，此类定义了角色技能的框架，每个技能都包括，技能触发判定，触发后持续控制，技能命中判定，命中后持续控制四个虚成员函数，在场景里面只需要通过技能基类指针调用这些虚成员函数就可以实现实际运行时根据不同的技能执行不同的判定，此类属于本游戏核心设计。角色的各种技能在角色初始化时加载到引擎，引擎里面用基类指针 map 映射保存各个技能地址，游戏的逻辑循环中首先根据玩家按键输入从技能出招表中匹配到某个技能的 ID，在从引擎中获取到这个技能的地址，它的类型为基类指针类型，接下来调用技能触发判定函数时会根据该技能的触发判定逻辑来判断技能是否满足触发条件，若满足，将玩家的当前技能赋值为此技能，之后的游戏循环中检测到当前技能指针有值则调用触发后持续控制函数来完成该技能触发后的各种特效和动作调整，在游戏逻辑循环最后的矩形碰撞中如果检测到了玩家 1 的攻击矩形和玩家 2 的防御矩形碰撞则执行玩家 1 当前技能的命中判定函数来判断技能是否命中，若命中则会将对方的当前被攻击技能赋值为此技能，之后的逻辑循环中如检测到玩家的被攻击技能有值则会调用该技能的命中后持续控制来实现对被攻方的动作调整

TeXiao.h 和 TeXiao.cpp

特效类，此类定义了游戏中各种特效的框架，游戏中的各种特效都继承此类，每种特效有不同的构造函数，但在游戏循环中都通过 Run 虚函数来执行不同特效，本项目中设计了五种类型的特效，人物闪烁，加减帧率（主要用于放慢镜头），图片序列，人物残影和屏幕震动。

Player.cpp 和 Player.h

玩家类，用于管理玩家的按键输入，人物朝向，当前动作，当前帧，当前动作等信息。

Role.cpp 和 Role.h

角色类，主要用于初始化加载不同角色的技能，动作，声音等数据

GameInput.cpp 和 GameInput.h

输入工具，接收当前的键盘和鼠标输入。

GameOutPut.h 和 GameOutPut.cpp

输出工具，用于绘制游戏画面中的各种图片、图形、和文字。

AudioManager.cpp 和 AudioManager.h

音效音乐管理类的定义与声明,提供了应用程序所有的音效音乐方法, 提供对音乐音效的加载、释放和使用等功能。

SceneChange.cpp 和 SceneChange.h

场景切换类，此类定义了场景切换的框架，各具体切换方案类皆继承自此类，此处场景切换指场景切换动画。此类中动画运行函数是虚函数，可方便引擎通过基类指针调用不同的场景切换方案类对象执行不同的动画播放方法。

MakeMove.cpp 和 MakeMove.h:

动作编辑类，此类在本游戏中也属于核心类，用于定义人物角色的各种动作信息，包括人物锚点，攻击矩形，防御矩形，维持帧，动作帧长度等信息，这些信息对于游戏渲染中的人物动作绘制以及游戏逻辑运算中的体人物碰撞，攻击判定，动作切换等至关重要。

核心代码

```
//技能触发条件判定（是匹配到按键后的判定，距离，双方状态等）
virtual bool TriggerJudge(CPlayer * actPlayer, CPlayer *defPlayer)=0;
//触发后控制
virtual void TriggerControl(CPlayer * actPlayer, CPlayer *defPlayer);
//技能命中条件判定（是矩形碰撞后的双方状态判断）
virtual bool HitJudge(CPlayer * actPlayer, CPlayer *defPlayer);
//技能命中后对被攻击者的持续控制
virtual void HitControl(CPlayer * actPlayer, CPlayer *defPlayer);
```

以上是技能相关的各种判定

```
//将对方的每个防御矩形与自身的攻击矩形 子弹矩形 防御矩形做碰撞
for(int u=0;u<CGameEngine::GetGE()->GetMoves()->GetDefLength(m_Player[(e+1)%2]->m_CurMove,m_Player[(e+1)%2]->m_CurF);u++)
{
  //对方防御矩形
  MyRect mydefR=CGameEngine::GetGE()->GetMoves()->GetDefRect(m_Player[(e+1)%2]->m_CurMove,m_Player[(e+1)%2]->m_CurF)[u];
  RECT def={
    m_Player[(e+1)%2]->m_x+(m_Player[(e+1)%2]->m_Face==_FACE_RIGHT ? mydefR._x1 : -mydefR._x2),
    m_Player[(e+1)%2]->m_y+mydefR._y1,
    m_Player[(e+1)%2]->m_x+(m_Player[(e+1)%2]->m_Face==_FACE_RIGHT ? mydefR._x2 : -mydefR._x1),
    m_Player[(e+1)%2]->m_y+mydefR._y2,
  };
  RECT c;


//与自身攻击矩形做碰撞
for(int v=0;v<CGameEngine::GetGE()->GetMoves()->GetActLength(m_Player[e]->m_CurMove,m_Player[e]->m_CurF);v++)
{
  MyRect myactR=CGameEngine::GetGE()->GetMoves()->GetActRect(m_Player[e]->m_CurMove,m_Player[e]->m_CurF)[v];
  RECT act={
    m_Player[e]->m_x+(m_Player[e]->m_Face==_FACE_RIGHT ? myactR._x1 : -myactR._x2),
    m_Player[e]->m_y+myactR._y1,
    m_Player[e]->m_x+(m_Player[e]->m_Face==_FACE_RIGHT ? myactR._x2 : -myactR._x1),
    m_Player[e]->m_y+myactR._y2,
  };
  if(IntersectRect(&c, &act, &def))
  {
    m_Player[e]->m_CurS->HitJudge(m_Player[e],m_Player[(e+1)%2]);
    break;        //与每个子弹的攻击矩形做碰撞
  }               for(std::list<Bullet>::iterator b=m_Player[e]->m_Bullet.begin();b!=m_Player[e]->m_Bullet.end();)
}                 {
                    bool tag=false;
                    for(int i=0;i<CGameEngine::GetGE()->GetMoves()->GetActLength(b->_CurM,b->_CurF);i++)
                    {
                      MyRect bactR=CGameEngine::GetGE()->GetMoves()->GetActRect(b->_CurM,b->_CurF)[i];
                      RECT bact={
                        b->_x+(b->_mirror==_MIRROR_NONE?bactR._x1:-bactR._x2),
                        b->_y+bactR._y1,
                        b->_x+(b->_mirror==_MIRROR_NONE?bactR._x2:-bactR._x1),
                        b->_y+ bactR._y2};
                      //与对方角色碰撞
                      if(IntersectRect(&c, &bact, &def))
                      {
                        b->_Sk->HitJudge(m_Player[e],m_Player[(e+1)%2]);
                        if(b->_type==_ZD_LAST)
                        {
```

```
            b->_y+bactR_y2);
        //与对方角色碰撞
        if(IntersectRect(&c, &bact, &def))
        {
            b->_Sk->HitJudge(m_Player[e],m_Player[(e+1)%2]);
            if(b->_type==_ZD_LAST)
            {
                b=m_Player[e]->m_Bullet.erase(b);
                tag=true;
            }
            break;
        }
        //子弹与墙壁碰撞 即子弹出界
        if((bact.left>virX+640||bact.right<virX)&&b->_type==_ZD_LAST)
        {
            b=m_Player[e]->m_Bullet.erase(b);
            tag=true;
            break;
        }
    }
    if(!tag)
        b++;

}
int mSite=CGameEngine::GetGE()->GetMoves()->GetSite(m_Player[e]->m_CurMove,m_Player[e]->m_CurF);
int hSite=CGameEngine::GetGE()->GetMoves()->GetSite(m_Player[(e+1)%2]->m_CurMove,m_Player[(e+1)%2]->m_CurF);
//与自身防御矩形做碰撞
for(int v=0;v<CGameEngine::GetGE()->GetMoves()->GetDefLength(m_Player[e]->m_CurMove,m_Player[e]->m_CurF);v++)
{
    MyRect hdefR=CGameEngine::GetGE()->GetMoves()->GetDefRect(m_Player[e]->m_CurMove,m_Player[e]->m_CurF)[v];
    RECT hdef={
        m_Player[e]->m_x+(m_Player[e]->m_Face==_FACE_RIGHT ? hdefR._x1 : -hdefR._x2),
        m_Player[e]->m_y+hdefR._y1,
        m_Player[e]->m_x+(m_Player[e]->m_Face==_FACE_RIGHT ? hdefR._x2 : -hdefR._x1),
        m_Player[e]->m_y+hdefR._y2,
    };
    //检测玩家与墙壁的碰撞 左右边界
    if(hdef.left<virX)
    {
        if(def.right-hdef.left<640)
        {
```

```
            m_Player[(e+1)%2]->m_x-=def.right-virX-640;
        }
        //检测两个玩家的碰撞
        if(IntersectRect(&c, &hdef, &def)&&m_Player[e]->m_HurtS==NULL&&m_Player[(e+1)%2]->m_HurtS==NULL)
        {
            if(m_Player[e]->m_x<m_Player[(e+1)%2]->m_x)
                m_Player[e]->m_x=m_Player[e]->m_x-c.right+c.left;
            else
                m_Player[e]->m_x=m_Player[e]->m_x+c.right-c.left;
        }

    }//防御矩形碰撞完毕

}//全部碰撞结束
```

以上是人物相关的各种碰撞检测，子弹指的时人物甩出去的技能，比如暗勾手

```
if(m_Player->m_CurMove!=m_Move)
    return false;
PreData pd={m_Player->m_CurMove,m_Player->m_CurF,m_Player->m_x,m_Player->m_y,m_Player->m_Face};
m_PreF[m_Last]=pd;
m_Last=(m_Last+1)%m_Max;
if(m_Size<m_Max)
    m_Size++;
//第一个残影的下标
int startpos=(m_Last-m_Size+m_Max)%m_Max;
for(int i=0;i<m_Size;i++)
{
    int n=(startpos+i)%m_Max;
    int dx=m_PreF[n].x-vir_x- CGameEngine::GetGE()->GetMoves()->GetMoveX(m_PreF[n].m, m_PreF[n].f);
    int dy=m_PreF[n].y  - CGameEngine::GetGE()->GetMoves()->GetMoveY(m_PreF[n].m, m_PreF[n].f);
    int sw=CGameEngine::GetGE()->GetMoves()->GetPX2(m_PreF[n].m, m_PreF[n].f);
    int mirror=_MIRROR_NONE;
    float alpha=i*1.0/m_Size;
    if(alpha<0)
        alpha=0;
```

以上是用循环队列实现人物残影特效的部分代码，其它特效比如震屏、人物闪光的实现不一一列出。