任务67：结束流程

1、就是让y轴变成负数，这样就能达到从天而降:

```
4 个引用
public void Draw()
{
    if (pos.y < 0) return;//屏幕外就不绘制
    Console.SetCursorPosition(pos.x, pos.y);
    switch (type)
```

```
2 个引用
public void ClearDraw()
{
    if (pos.y < 0) return;//屏幕外就不绘制
    Console.SetCursorPosition(pos.x, pos.y);
    Console.Write("  ");
```

```
        new DrawObject(type),
    };
    //需要初始化方块位置
    //原点位置，随机自定义 方块List中的0个就是我们的原点方块
    blocks[0].pos = new Position(24, -5);
    //其他三个方块的位置
```

2、就是判断如何结束游戏

```
2 个引用
public void AddWalls(List<DrawObject> walls)
{
    for (int i = 0; i <walls.Count ; i++)
    {
        //传递方块进来时，把其类型改为墙壁
        walls[i].ChangeType(E_DrawType.Wall);
        dynamicWalls.Add(walls[i]);//添加到动态墙壁列表中

        //==============================结束逻辑==============================
        //在动态墙壁添加处 发现位置顶满了，就结束
        if (walls[i].pos.y <= 0)   添加的方块位置到顶了就结束游戏
        {
            //关闭输入监听线程
            nowScene.StopThread(); 停止监听按键线程
            //场景切换
            Game.changeScene(E_SceneType.End); 切换到结束游戏场景
            return;
        }

        //进行添加动态墙壁的计数
        //根据索引来得到行
        // h就是 game.h-6 那我们的第一行是game.h-7 所以需要-1
        //y最大就是Game.h-7
        //且不会重复，因为重合会停下，所以每一行都有最大值(Game.w-2)最大值就表示一行满了
        recordInfo[h - walls[i].pos.y - 1] +=1;//这里就可以得到最开始是从0行开始的（也就是小方形y轴的最大值）
    }



    //检查是否需要跨层
    //1、先移除动态方块（就算真的不需要迁移，后面还会绘制出来
    ClearDraw();
    //2、检测移除
    CheckClear();
    //3、在绘制动态方块
    Draw();
}
```

```
//检测输入线程
Thread inputThread;
bool  isRunning;
```

```
1 个引用
public void CheckInputThread()
{
    while (isRunning)
    {
        if (Console.KeyAvailable)
        {

            lock (woker)
            {
                switch (Console.ReadKey(true).Key)
                {
                    case ConsoleKey.A:
```

```
1 个引用
public GameScene()
{
    score = 0;
    layer = 0;
    map = new Map(this);
    woker = new BlockWorker();

    inputThread = new Thread(CheckInputThread);
    //设置后台线程，声明周期随主程序决定
    inputThread.IsBackground = true;
    //开启线程
    inputThread.Start();
    isRunning = true;
}
1 个引用
```

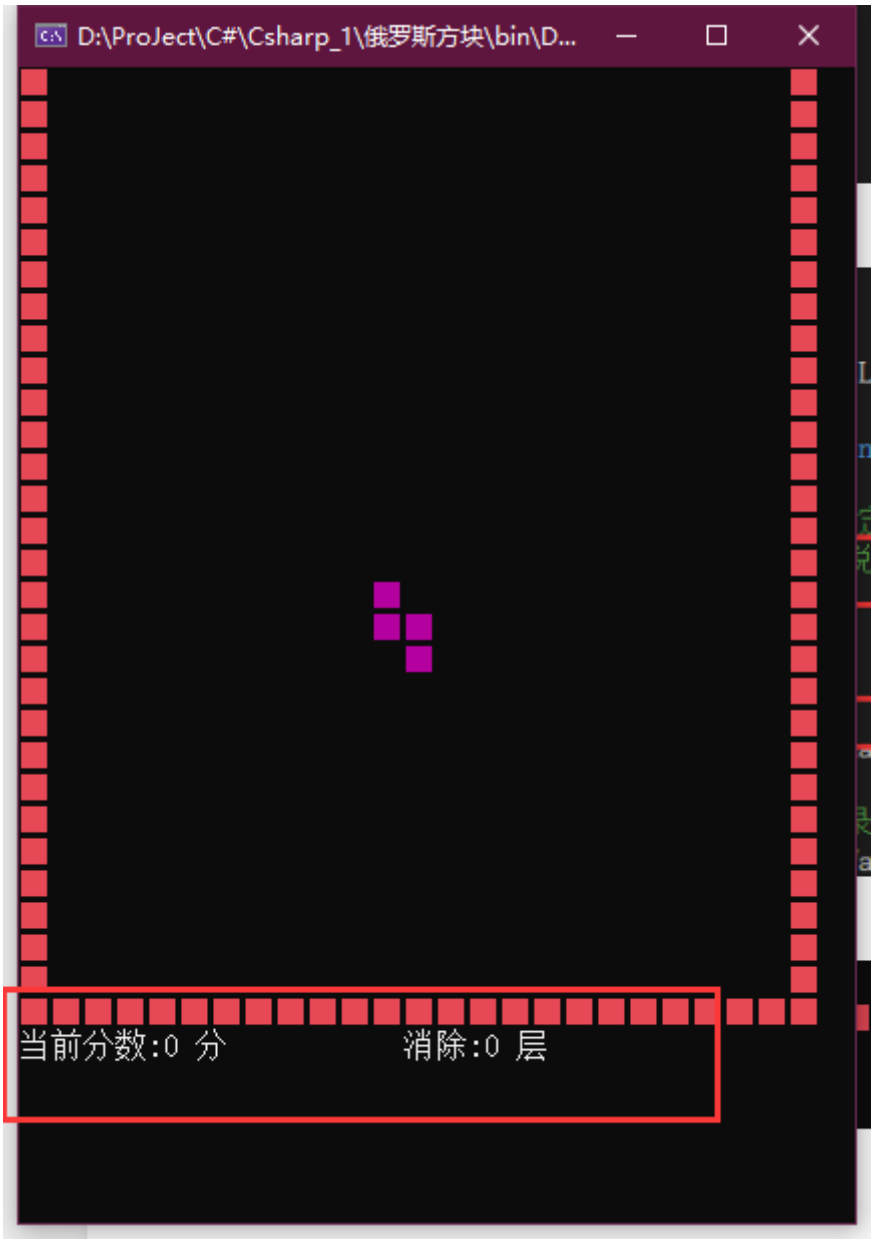
```
//关闭监听按键线程
1 个引用
public void StopThread()
{
    isRunning = false;
    inputThread = null;
}
```

```
2 个引用
public void upData()//游戏界面(主要的逻辑处理更新)
{
    lock (woker)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.SetCursorPosition(0, Game.h - 5);
        Console.Write("当前分数:{0} 分\t\t消除:{1} 层", score, layer);
        //地图更新绘制//每一帧都绘制一次
        map.Draw();
        //绘制砖块
        woker.Draw();
        if (woker.CanMove(map))
            woker.AutoMove();
    }   Thread.Sleep(200);
}
```

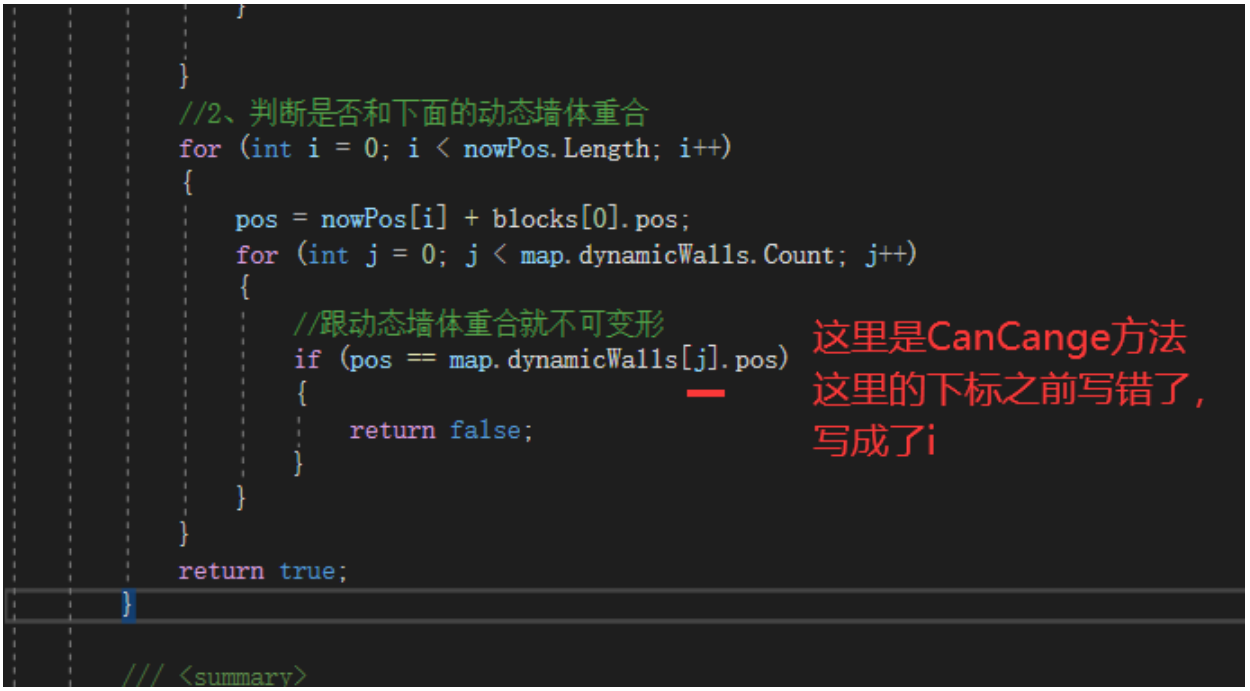
添加积分显示

```
public void CheckClear()
{
    for (int i = 0; i < recordInfo.Length; i++)
    {
        List<DrawObject> delList = new List<DrawObject>();

        //这里的w已经去掉了两天的固定墙壁
        if(recordInfo[i]==w)//成立说明一行满了
        {
            nowScene.Scoce += 10;
            nowScene.Layer += 1;
            //1、这一行的小方块移除
            for (int j = 0; j < dynamicWalls.Count; j++)
            {
                //当前行号与我们记录数组的行号一致说明这一行满了
                if (i == h-dynamicWalls[j].pos.y-1)
```



错误纠正：


}
//2、判断是否和下面的动态墙体重合
for (int i = 0; i < nowPos.Length; i++)
{
pos = nowPos[i] + blocks[0].pos;
for (int j = 0; j < map.dynamicWalls.Count; j++)
{
//跟动态墙体重合就不可变形
if (pos == map.dynamicWalls[j].pos)
{
return false;
}
}
}
return true;
}
/// <summary>
这里是CanCange方法
这里的下标之前写错了，
写成了i