Character 文件添加内容

如图 1，在角色头文件（Homework2Character.h）中设置受保护的积分属性，用于记录角色当前的总得分，并使用第三节课学习到的方法将其设置为所有均可见。

```
protected:
    /** Called for movement input */
    void Move(const FInputActionValue& Value);

    /** Called for looking input */
    void Look(const FInputActionValue& Value);

    // 新增积分属性
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Score")
    int32 Score;
```

图 1 积分属性

如图 2，在角色文件（Homework2Character.cpp）的构造函数中初始化积分值为 0。

```
AHomework2Character::AHomework2Character()
{
    // Set size for collision capsule
    GetCapsuleComponent()->InitCapsuleSize(55.f, 96.0f);

    // Create a CameraComponent
    FirstPersonCameraComponent = CreateDefaultSubobject<UCameraComponent>(TEXT("FirstPersonCamera"));
    FirstPersonCameraComponent->SetupAttachment(GetCapsuleComponent());
    FirstPersonCameraComponent->SetRelativeLocation(FVector(-10.f, 0.f, 60.f)); // Position the camera
    FirstPersonCameraComponent->bUsePawnControlRotation = true;

    // Create a mesh component that will be used when being viewed from a '1st person' view (when controlling this pawn)
    Mesh1P = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("CharacterMesh1P"));
    Mesh1P->SetOnlyOwnerSee(true);
    Mesh1P->SetupAttachment(FirstPersonCameraComponent);
    Mesh1P->bCastDynamicShadow = false;
    Mesh1P->CastShadow = false;
    Mesh1P->SetRelativeLocation(FVector(-30.f, 0.f, -150.f));

    // 初始化积分值
    Score = 0;
}
```

图 2 初始化积分值

如图 3，在角色头文件（Homework2Character.h）中声明公共的积分累加函数和积分获取函数，并按照图 4 在角色文件（Homework2Character.cpp）中实现该积分函数的功能。

```
public:
    /** Returns Mesh1P subobject **/
    USkeletalMeshComponent* GetMesh1P() const { return Mesh1P; }
    /** Returns FirstPersonCameraComponent subobject **/
    UCameraComponent* GetFirstPersonCameraComponent() const { return FirstPersonCameraComponent; }

    /** 增加积分的函数 */
    UFUNCTION(BlueprintCallable, Category = "Score")
    void AddScore(int32 N);

    /** 积分获取函数 */
    UFUNCTION(BlueprintCallable, Category = "Score")
    int GetScore() {
        return Score;
    }
```

图 3 声明积分增加函数和积分获取函数

```
// 积分函数增加并输入
void AHomework2Character::AddScore(int32 N)
{
    Score += N;
    UE_LOG(LogTemplateCharacter, Log, TEXT("Score added: %d, Current Score: %d"), N, Score);
}
```

图 4 实现积分增加函数

Projectile 文件添加内容

在碰撞检测头文件（Homework2Projectile.h）的公共属性中声明命中后为角色增加的积分值 ScoreAmount（如图 5），并且将其设置为在编辑器及蓝图中可修改。

```
30  public:
31      AHomework2Projectile();
32
33
34      /** called when projectile hits something */
35      UFUNCTION()
        0 个 Blueprint 引用
36      void OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, FVector NormalImpulse, const FHitResult& Hit);
37
38      /** Returns CollisionComp subobject **/
39      USphereComponent* GetCollisionComp() const { return CollisionComp; }
40      /** Returns ProjectileMovement subobject **/
41      UProjectileMovementComponent* GetProjectileMovement() const { return ProjectileMovement; }
42
43      // 蓝图中可修改的积分值
44      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Projectile")
        已在 0 个 Blueprints 中更改
45      int32 ScoreAmount; // 在蓝图中设置积分
```

图 5 声明可修改的累加分值

在碰撞检测头文件（Homework2Projectile.h）的公共属性中添加命中特殊物品可获取积分的倍数 Times（如图 6）。

```
43      // 蓝图中可修改的积分值
44      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Projectile")
        已在 0 个 Blueprints 中更改
45      int32 ScoreAmount; // 在蓝图中设置积分
46
47      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Projectile")
        已在 0 个 Blueprints 中更改
48      int32 Times; // 在蓝图中设置倍数
49
```

图 6 设置特殊组件提供的积分倍数

在碰撞检测头文件（Homework2Projectile.h）的公共属性中添加记录发射物的 Owner 的属性，并通过 SetFiringCharacter()函数将 Owner 传输给碰撞检测文件（Home-work2Projectile.cpp），用于确定给谁加分（如图 7）。

```
46
47      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Projectile")
        已在 0 个 Blueprints 中更改
48      int32 Times; // 在蓝图中设置倍数
49
50      // 保存发射该弹丸的角色
51      UPROPERTY()
        已在 0 个 Blueprints 中更改
52      AHomework2Character* FiringCharacter;
53
54      // 在 Homework2WeaponComponent.cpp 中调用
55      void SetFiringCharacter(AHomework2Character* Character)
56      {
57          FiringCharacter = Character;
58      }
```

图 7 找到并保存发射物的 Owner

在碰撞检测头文件（Homework2Projectile.h）的公共属性中添加记录被命中物体受到的设计次数的属性 HitCounter，同时添加可在编辑器和蓝图中修改的物体缩小倍数的属性 SmallTimes（如图 8）。

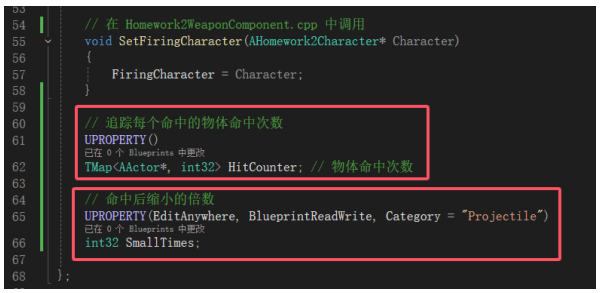

图 8 记录命中次数并设置缩小倍数

为了能够实现玩家命中物体后，根据命中物是否为特殊物品给玩家加分 ScoreAmount 或 ScoreAmount*Times 的分数，对首次命中物的体积缩小 SmallTimes 倍，二次命中的物体直接销毁，并且保证子弹在命中目标或 3 秒后自动销毁的功能，这里主要修改了 OnHit 函数。

备注：由于提供的 OnHit 已经实现了子弹销毁功能，在这里不再截图展示。

在碰撞检测文件（Homework2Projectile.cpp）的构造函数中对如图 9 中的三个属性设置默认值。

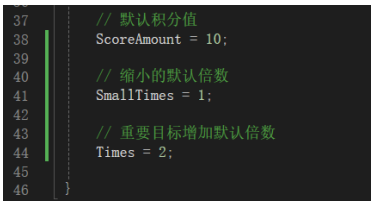

图 9 默认积分值、默认缩小倍数、默认积分倍增倍数

在碰撞检测文件（Homework2Projectile.cpp）的 OnHit 函数中默认提供的判断是否命中可积分方块的 if 判断语句中添加如图 10 的内容，判断是否能够找到进行射击的角色。

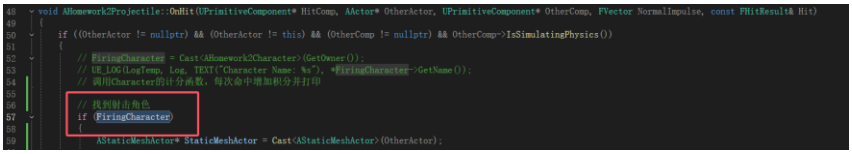

图 10 判断是否能找到释放射击的角色

碰撞检测文件（Homework2Projectile.cpp）如图 10 的判断语句中添加判断命中目标是否是静态网格体且能够调用静态网格体的组件的判断语句（如图 11）。

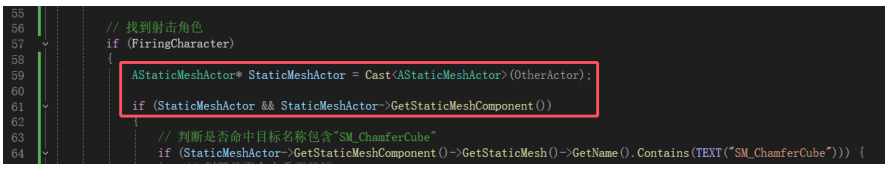

图 11 判断是否为静态网格体且能正常使用组件

在碰撞检测文件（Homework2Projectile.cpp）如图 11 的判断语句中继续添加判断目标是否是我们想要击中的组件，并根据是否命中重要目标分别设置为

角色累加的积分值，并且根据物体被命中次数决定是缩小物体还是销毁物体（如图 12）。其中 SM_ChamferCubeChange 是在 UE 引擎中根据原有的 SM_ChamferCube 复制生成的仅改变了颜色的全新静态网格体，主要用于在 GameMode 中修改特殊物体在游戏中的表现。

```
61          if (StaticMeshActor && StaticMeshActor->GetStaticMeshComponent())
62          {
63              // 判断是否命中目标名称包含"SM_ChamferCube"
64              if (StaticMeshActor->GetStaticMeshComponent()->GetStaticMesh()->GetName().Contains(TEXT("SM_ChamferCube"))) {
65                  // 判断是否命中重要目标
66                  if (StaticMeshActor->GetStaticMeshComponent()->GetStaticMesh()->GetName() == TEXT("SM_ChamferCubeChange"))
67                  {
68                      int32& HitCount = HitCounter.FindOrAdd(OtherActor); // 获取或初始化该物体的命中次数
69
70                      if (HitCount == 0)
71                      {
72                          // 第一次命中，缩小物体大小 Y 倍
73                          FVector NewScale = StaticMeshActor->GetActorScale3D() * FMath::Pow(0.5f, SmallTimes);
74                          StaticMeshActor->SetActorScale3D(NewScale);
75                      }
76                      else
77                      {
78                          StaticMeshActor->Destroy();
79                      }
80
81                      FiringCharacter->AddScore(Times * ScoreAmount);
82                  }
83                  else
84                  {
85                      int32& HitCount = HitCounter.FindOrAdd(OtherActor); // 获取或初始化该物体的命中次数
86
87                      if (HitCount == 0)
88                      {
89                          // 第一次命中，缩小物体大小一倍
90                          FVector NewScale = StaticMeshActor->GetActorScale3D() * FMath::Pow(0.5f, SmallTimes);
91                          StaticMeshActor->SetActorScale3D(NewScale);
92                      }
93                      else
94                      {
95                          StaticMeshActor->Destroy();
96                      }
97                      FiringCharacter->AddScore(ScoreAmount);
98                  }
99              }
100
101         }
102         // 只加分，不缩小
103         // 判断能否获得静态网格体
```

图 12 OnHit 函数中的主要修改内容

WeaponComponent 文件添加功能

Projectile 文件需要的开火角色获取的函数 SetFiringCharacter()要在武器文件（Homework2WeaponComonent.cpp）中 UHomework2WeaponComponent::Fire()函数内实现，将其添加在如图 13 所示的位置，获取武器的拥有者，将值传输到碰撞检测文件（Homework2Projectile.cpp）中去。

```
55          // Spawn the projectile at the muzzle
56          // World->SpawnActor<AHomework2Projectile>(ProjectileClass, SpawnLocation, SpawnRotation, ActorSpawnParams);
57          AHomework2Projectile* SpawnedProjectile = World->SpawnActor<AHomework2Projectile>(ProjectileClass, SpawnLocation, SpawnRotation, ActorSpawnParams);
58
59          // 将发射的角色传输给 AHomework2Projectil.cpp
60          if (SpawnedProjectile)
61          {
62              SpawnedProjectile->SetFiringCharacter(Character);  // 设置发射者为当前角色
63          }
```

图 13 在 WeaponComponent 中将 Owner 传输给 Projectile

GameMode 文件添加功能

如图 14 在内容浏览器的“内容/LevelPrototyping/Meshes”中找到蓝色的 SM_ChamferCube，使用 Ctrl+D 复制得到一个同样的静态网格体，双击进入

后，按照图 15 对复制得到的新的静态网格体的颜色进行修改。

图 14 生成新的静态网格体

图 15 修改静态网格体颜色

如图 16，在游戏模式头文件（Homework2GameMode.h）中的 protected 内添加重写 BeginPlay 的方法，并给出时间句柄和游戏结束倒计时的时间变量 TimeToEnd-Game，使其能够在编辑器和蓝图中修改。

```
18  public:
19      AHomework2GameMode();
20
21  protected:
22      // 重写BeginPlay方法
23      virtual void BeginPlay() override;
24
25      // 时间计时器句柄
26      FTimerHandle EndGameTimerHandle;
27
28      // 游戏结束倒计时的时间（可以在编辑器中设置）
29      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game Settings")
        已在 0 个 Blueprints 中更改
30      float TimeToEndGame;
```

图 16 重写 BeginPlay 并给出及时相关内容

如图 17，在游戏模式头文件（Homework2GameMode.h）声明保存所有玩家分数的变量和当前关卡名称，并声明游戏结束的函数。

```
32      // 声明保存所有玩家总分的变量
33      int32 TotalScore;
34
35      // 当前关卡名称（可以在编辑器中设置）
36      UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game Settings")
37      FString CurrentLevelName;
38
39      // 用于显示玩家积分和总积分的 UMG 控件
40      /*UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "UI")
41      class UUserWidget* ScoreboardWidget;*/
42
43      // 结束游戏
44      void EndGame();
```

图 17 声明玩家总分数、关卡名称和游戏结束函数

如图 18，在游戏模式头文件（Homework2GameMode.h）的 private 中声明修改颜色的函数 ChangeColorOfCubes，在 public 中声明特殊方块的个数 SqCubeNum，其中 N 是可调节的方块数，在函数现实调时用了 SpCubeNum。

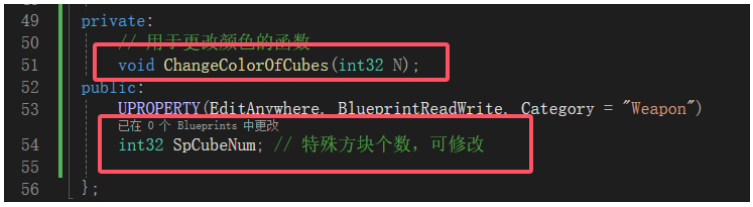

图 18 声明生成特殊方块的函数和可调节的方块数

在游戏模式文件（Homework2GameMode.cpp）的构造函数中给出各变量的初始值（如图 19）。

图 19 各变量初始值

在游戏模式文件（Homework2GameMode.cpp）中重写 BeginPlay()方法如图 20，实现设置特殊方块，并设置计时器，用于结束游戏。

图 20 调用修改颜色的函数并控制游戏结束

在游戏模式文件（Homework2GameMode.cpp）中 EndGame()主要实现了对所有角色的积分进行日志输出，并统计所有角色的总分数对其进行输出。为了能够更好地发现日志信息，这里将日志类型设置为“waring”。最后将游戏退出（如图 21）。

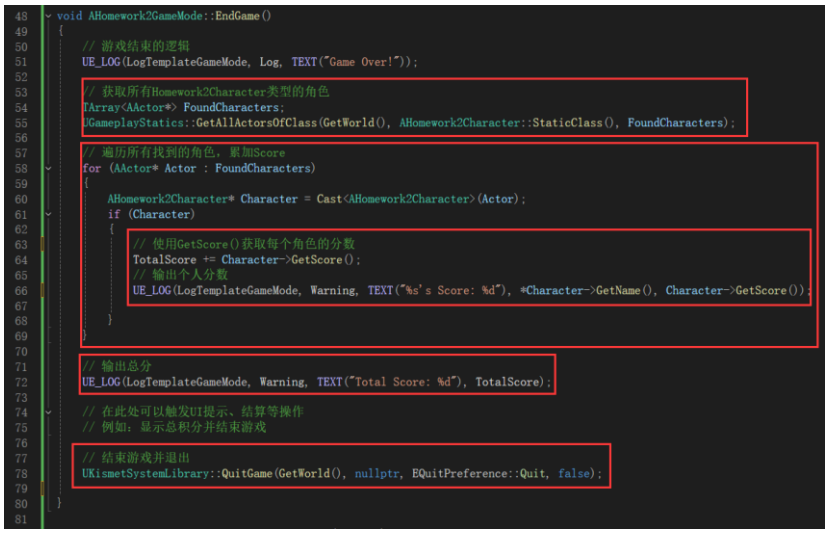

图 21 EndGame()函数

在游戏模式文件（Homework2GameMode.cpp）中 ChangeColorOfCubes()函数主要用于生成特殊方块，在所有的 Actors 中找到使用“SM_ChamferCube”创建的静态网格体，将其纳入待选升级数组。调整需要随机选择的特殊方块的个数，并创建判断方块是否被选过的变量 bIfChoose（如图 22）。

```
82  void AHomework2GameMode::ChangeColorOfCubes(int32 N)
83  {
84      // UE_LOG(LogTemplateGameMode, Log, TEXT("In ChangeColorOfRandomSMChamferCubes"));
85      TArray<AActor*> AllActors;
86      UGameplayStatics::GetAllActorsOfClass(GetWorld(), AStaticMeshActor::StaticClass(), AllActors);
87
88      TArray<AStaticMeshActor*> ChamferCubes;
89
90      // 筛选出符合名称的物体
91      for (AActor* Actor : AllActors)
92      {
93          // UE_LOG(LogTemplateGameMode, Log, TEXT("In Select"));
94          AStaticMeshActor* StaticMeshActor = Cast<AStaticMeshActor>(Actor);
95          if (StaticMeshActor && StaticMeshActor->GetStaticMeshComponent())
96          {
97              if (StaticMeshActor->GetStaticMeshComponent()->GetStaticMesh()->GetName() == TEXT("SM_ChamferCube"))
98              {
99                  ChamferCubes.Add(StaticMeshActor);
100             }
101         }
102     }
103
104     // 如果符合条件的物体少于N个，随机选择所有
105     if (ChamferCubes.Num() <= N)
106     {
107         N = ChamferCubes.Num();
108     }
109
110     // 初始化均为未选中
111     TArray<bool> bIfChoose;
112     for (int32 i = 0; i < ChamferCubes.Num(); ++i)
113     {
114         bIfChoose.Add(false);
115     }
116
117     // 加载新网格体
```

图 22 函数 ChangeColorOfCubes()（1）

加载图 14 中新生成的网格体，进行 N 次随机选取，将所有选取到的未被“升级”的方块的静态网格体进行修改（如图 23）。

```
117     // 加载新网格体
118     UStaticMesh* NewStaticMesh = Cast<UStaticMesh>(StaticLoadObject(UStaticMesh::StaticClass(), nullptr,
119         TEXT("/Game/LevelPrototyping/Meshes/SM_ChamferCubeChange.SM_ChamferCubeChange")));
120     if (!NewStaticMesh)
121     {
122         // UE_LOG(LogTemplateGameMode, Error, TEXT("Failed to load new static mesh!"));
123         return;  // 如果无法加载新网格体，退出函数
124     }
125
126     // 随机选取N个物体并修改颜色
127     for (int32 i = 0; i < N; ++i)
128     {
129         // UE_LOG(LogTemplateGameMode, Log, TEXT("In Choose Color"));
130         AStaticMeshActor* SelectedActor = nullptr;
131         while(1)
132         {
133             // 如果未选过则记录选中的Actor，否则重新选
134             int32 RandomIndex = FMath::RandRange(0, ChamferCubes.Num() - 1);
135             if (!bIfChoose[RandomIndex]) {
136                 SelectedActor = ChamferCubes[RandomIndex];
137                 bIfChoose[RandomIndex] = true;
138                 break;
139             }
140         }
141
142         if (SelectedActor != nullptr)
143         {
144             // UE_LOG(LogTemplateGameMode, Log, TEXT("In Change StaticMesh"));
145             SelectedActor->GetStaticMeshComponent()->SetStaticMesh(NewStaticMesh);  // 设置新网格体
146
147         }
148     }
149 }
```

图 23 ChangeColorOfCubes()（1）

运行结果展示

游戏运行前

如图 24，可以看到游戏运行前各个 SM_ChamferCube 均为蓝色，尚未进行随机选择，并且其余各项均为默认情况。

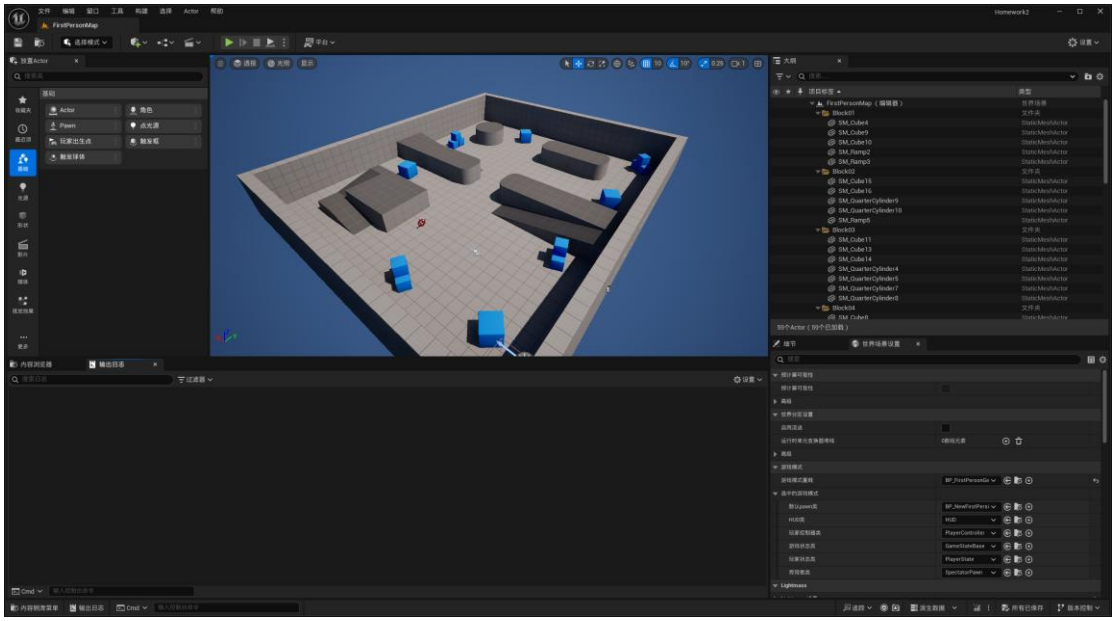

图 24 初始情况

为了方便查看日志信息，我们在程序中为每个角色命中物体后的此次加分和当前得分的日志归类到“LogTemplateCharacter”，并将结束时输出的每个角色的得分和总得分归类到“LogTemplateGameMode”，然后在日志过滤器中进行设置，勾选二者（如图 25）。

图 25 过滤日志信息

游戏运行后

从图 26 能够看到可以成功开启多人游玩（此时尚未进行静态网格体的同步，也未为角色添加他人可见的模型），并且能够发现，游戏中部分物体已经变

为特殊物体（黄色），射击时也能够获得更高的分数，且命中后相关目标会变小。

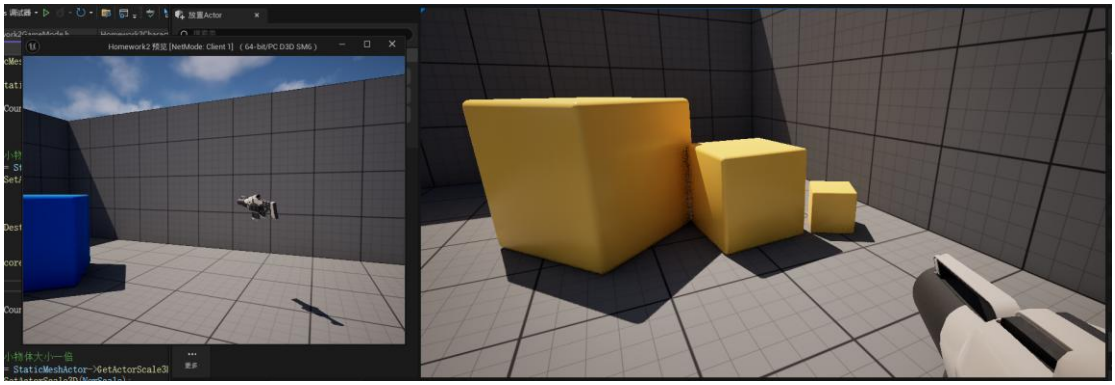

图 27 开启双人运行

同时日志信息也能够打印正确的运行结果（如图 28），在视频中可以看到游戏能够在一分钟后自动结束。

```
LogTemplateCharacter: Character Name: PlayerController_0 Score added: 20, Current Score: 20
LogTemplateGameMode: Game Over!
LogTemplateGameMode: Warning: BP_NewFirstPersionCharacter_C_0's Score: 20
LogTemplateGameMode: Warning: BP_NewFirstPersionCharacter_C_1's Score: 0
LogTemplateGameMode: Warning: Total Score: 20
```

图 28 日志信息

附加题（后续会补充附加题的实现）

支持多人联机

UMG 结算信息