**实 验 四 报 告**

**（2020-2021学年第二学期）**

**3D游戏引擎架构设计基础**

**（Foundations of 3D Game Engine Architecture Design）**

|  |  |  |
| --- | --- | --- |
| 学生姓名： | | 年级 |
| 其他组员姓名： | | 小组成绩 |
| 个人成绩： | 任课教师签名： | 日期： |
| 实验题目： | | |
| 目录：   1. 实验内容描述: 引擎名称和版本,将分析的引擎模块名称 2. 将分析的引擎模块的主要功能描述 3. 将分析的引擎模块的主要类和类关系描述 4. 类的数据成员和成员函数描述 5. 总结 | | |
| 实验报告：   1. 实验内容：分析Unreal 4.27.2引擎Tick 相关机制 2. 主要功能：Tick是引擎运行时的重要机制，负责驱动引擎中的各个模块和场景中各个实体 3. 主要类及类关系：   FEngineLoop：游戏引擎主循环，存在一个循环调用的Tick函数负责驱动其它所有类  UWorld：游戏世界，存在Tick函数负责调用下列三种Tick主体  FTimerManager：负责管理各种定时器  FTickFunction：虚基类，包装类各种可以Tick的调用，比如AActor  FTickableGameObject：TickableObject的基类，同时储存了子类实例，提供Tick函数调用   1. 类的数据成员和成员函数描述   执行Tick流程中主要涉及的核心函数、成员如下：  [1] void FEngineLoop::Tick() Tick入口，调用其他Tick类  在GuardedMian中被调用：  int32 GuardedMain( const TCHAR\* CmdLine )  {  /\* ... \*/  BootTimingPoint("Tick loop starting");  DumpBootTiming();  // Don't tick if we're running an embedded engine - we rely on the outer  // application ticking us instead.  if (!GUELibraryOverrideSettings.bIsEmbedded)  {  while( !IsEngineExitRequested() )  {  EngineTick();  }  }  TRACE\_BOOKMARK(TEXT("Tick loop end"));  /\* ... \*/  }  [2] void UWorld::Tick( ELevelTick TickType, float DeltaSeconds ) 负责调用其他三种Tick  在FEngineLoop::Tick()中间接调用：  LAUNCH\_API void EngineTick( void )  {  GEngineLoop.Tick();  }  void FEngineLoop::Tick()  {  /\* ... \*/  {  /\* ... \*/  GEngine->Tick(FApp::GetDeltaTime(), bIdleMode);  /\* ... \*/  }  /\* ... \*/  }  void UGameEngine::Tick( float DeltaSeconds, bool bIdleMode )  {  /\* ... \*/  for (int32 WorldIdx = 0; WorldIdx < WorldList.Num(); ++WorldIdx)  {  FWorldContext &Context = WorldList[WorldIdx];  if (Context.World() == NULL || !Context.World()->ShouldTick())  {  continue;  }  GWorld = Context.World();  // Tick all travel and Pending NetGames (Seamless, server, client)  {  QUICK\_SCOPE\_CYCLE\_COUNTER(STAT\_UGameEngine\_Tick\_TickWorldTravel);  TickWorldTravel(Context, DeltaSeconds);  }  if (!bIdleMode)  {  SCOPE\_TIME\_GUARD(TEXT("UGameEngine::Tick - WorldTick"));  // Tick the world.  Context.World()->Tick( LEVELTICK\_All, DeltaSeconds );  }  /\* ... \*/  }  /\* ... \*/  }  [3] void FTimerManager::Tick(float DeltaTime) 调用当前管理的所有Timer  出现在UWorld::Tick中  void UWorld::Tick( ELevelTick TickType, float DeltaSeconds )  {  /\* ... \*/  if (TickType != LEVELTICK\_TimeOnly && !bIsPaused)  {  SCOPE\_TIME\_GUARD\_MS(TEXT("UWorld::Tick - TimerManager"), 5);  STAT(FScopeCycleCounter Context(GetTimerManager().GetStatId());)  GetTimerManager().Tick(DeltaSeconds);  }  }  [4] FTimerManager::TSparseArray<FTimerData> Timers; 当前的所有Timer  [5] virtual void FTickTaskManagerInterface::RunTickGroup(ETickingGroup Group, bool bBlockTillComplete ) 运行当前Group所有FTickFunction  出现在UWorld::Tick中  void UWorld::Tick( ELevelTick TickType, float DeltaSeconds )  {  /\* ... \*/  RunTickGroup(TG\_PrePhysics);  RunTickGroup(TG\_StartPhysics);  RunTickGroup(TG\_DuringPhysics, false);  RunTickGroup(TG\_EndPhysics);  RunTickGroup(TG\_PostPhysics);  RunTickGroup(TG\_PostUpdateWork);  RunTickGroup(TG\_LastDemotable);  }  void UWorld::RunTickGroup(ETickingGroup Group, bool bBlockTillComplete = true)  {  check(TickGroup == Group); // this should already be at the correct value, but we want to make sure things are happening in the right order  FTickTaskManagerInterface::Get().RunTickGroup(Group, bBlockTillComplete);  TickGroup = ETickingGroup(TickGroup + 1); // new actors go into the next tick group because this one is already gone  }  [6] void FTickableGameObject::TickObjects(UWorld\* World, const int32 InTickType, const bool bIsPaused, const float DeltaSeconds)运行当前World下所有Tickable类  出现在UWorld::Tick中  void UWorld::Tick( ELevelTick TickType, float DeltaSeconds )  {  FTickableGameObject::TickObjects(this, TickType, bIsPaused, DeltaSeconds);  }  附：AActor如何添加到UWorld中的TickGroup  [1]在UWorld启动时会调用所有AActor的BeginPlay  [2] AActor::BeginPlay会调用AActor::RegisterAllActorTickFunctions  [3] AActor::RegisterAllActorTickFunctions 会调用 AActor::RegisterActorTickFunctions  [4] AActor::RegisterActorTickFunctions中会创建一个 FActorTickFunction 并添加到ULevel中   1. 总结   Tick机制作为一个贯穿引擎的机制，从虚幻4的Tick设计的逻辑、结构设计上，我看到了一个大型引擎的复杂但清晰的设计，从中我学到了规划代码架构的重要性。 | | |