**AperiOculos**

**感知框架** - 技术设计文档

**日期: 25.10.4**

**版本:1.0**

**//////////////// 1.项目概述 ////////////////**

**1.1. 项目愿景**

**Aperi Oculos 是一个为Minecraft Java版设计的、基于Forge Modding API的生物感知框架。它旨在彻底取代原版游戏中简单且耦合度高的生物索敌机制，提供一个高性能、高真实感、高度可配置的底层感知系统。**

**1.2. 核心：感知与决策的分离**

**本框架严格遵循“关注点分离”原则，其唯一职责是回答“生物能感知到什么？”这个问题。它本身不包含任何具体的AI行为逻辑。**

**Aperi Oculos (感知层): 负责处理生物的视觉和听觉。它确定生物在特定条件下能看到什么、能听到什么，然后通过Forge事件和公共API将这些纯粹的感知信息广播出去。**

**上层AI Mod (决策层): 其他Mod（如设想中的AI附属）可以依赖本框架，监听其广播的感知事件，并根据自身的逻辑（如脚本、行为树、状态机）来决定生物应该如何行动。**

**这种分离使得AI的开发变得高度模块化、可扩展且易于维护。**

**///////////////// 2. 核心系统 ////////////////**

**Aperi Oculos由两大主动式感知系统构成**

**2.1. 视觉系统**

**该系统负责模拟生物的“视觉”能力，其设计核心是主动式、以玩家为中心的周期性扫描，以解耦感知范围与仇恨范围。**

**触发机制**

*由服务器端的ServerTickEvent驱动，以一个可配置的频率运行（默认为每10游戏刻）。*

*为优化性能，扫描以所有非观察者模式的玩家为中心，仅检查玩家附近（默认64格）的生物，而非迭代服务器上的所有生物。*

**感知逻辑流程**

***对于每一个被扫描的“观察者-目标”对，系统会按以下顺序（从低性能开销到高）执行检查***

**1.距离检查:** 目标是否在观察者的FOLLOW\_RANGE属性范围内?

**2.视野角度(FoV)检查:** 目标是否在观察者前方一个可配置的锥形视野内（默认为180度）？

**3.光照等级检查:** 目标所在位置的光照等级是否在配置的“潜行光照范围”内？此检查可被拥有“夜视能力”的生物豁免。

**4.视线(Line of Sight, LoS)检查:** 从观察者眼睛到目标眼睛之间是否存在阻挡视线的固体方块？

**这一流程的目的是性能优化**

**廉价检查优先:** 昂贵的视线检查仅在前三项检查都通过后才执行。

**视线缓存:** 视线检查的结果会被缓存一个可配置的短暂时间（默认为5游戏刻），以避免在位置变化不大的情况下重复进行昂贵的光线追踪计算。

**输出:** 当一个目标通过所有视觉检查后，系统会广播一个TargetSpottedEvent事件。

**2.2. 听觉系统**

*该系统负责模拟生物的“听觉”能力，其设计核心是基于服务器端GameEvent和VibrationListener的真实模拟。*

**触发机制:**

*由EntityJoinLevelEvent驱动，当一个符合条件的生物（非“失聪”且非玩家）加入世界时，为其动态注入一个自定义的GameEventListener。*

*该监听器会接收到服务器上发生的所有GameEvent（即“振动”）。*

**感知逻辑流程:**

***当监听到一个GameEvent时，系统会为每个潜在的听者执行以下计算：***

**获取基础范围:** 读取GameEvent自身的notificationRadius（通知半径），这是该事件在游戏中最权威的传播距离。

**应用听力乘数:** 获取听者生物的aperioculos:hearing\_multiplier自定义属性值。

**计算有效范围:** 有效听觉距离 = notificationRadius \* hearing\_multiplier。

**距离检查:** 听者与声源的距离是否小于等于“有效听觉距离”？

**遮挡检查:** 从听者位置到声源位置进行一次简单的光线追踪，检查路径上是否有任何固体方块。

**输出:** 如果距离和遮挡检查都通过，系统会广播一个SoundHeardEvent事件。

**////////////////// 3.公共接口 ////////////////**

**3.1. API类: AperiOculosAPI**

***提供一系列公共静态方法，供其他Mod进行即时查询。***

**boolean canSee(LivingEntity observer, LivingEntity target):**

执行一次完整的视觉检查。

**List<LivingEntity> getVisibleTargets(LivingEntity observer):**

获取观察者当前能看到的所有目标。

**boolean isBlind(LivingEntity entity):**

查询实体是否在配置的“失明”列表中。

**boolean isDeaf(LivingEntity entity):**

查询实体是否在配置的“失聪”列表中。

**3.2. 事件**

***所有事件均在服务器端的Forge事件总线上发布。***

**TargetSpottedEvent:**

**触发时机:** 生物通过视觉成功发现一个目标时。

**包含数据:** LivingEntity observer, LivingEntity target。

**SoundHeardEvent:**

**触发时机:** 生物成功听到了一个声音（振动）。

**包含数据:** LivingEntity listener, Vec3 sourcePos, GameEvent gameEvent, @Nullable Entity sourceEntity。

**////////////// 4. 数据驱动配置 //////////////**

**4.1. 配置文件 (aperioculos-common.toml)**

***提供详尽的配置选项，允许整合包作者和服务器管理员精细调整感知系统的行为。***

|  |  |
| --- | --- |
| **vision** | **viewFieldAngle, minStealthLightLevel, maxStealthLightLevel, nightVisionEntities, blindEntities** |
| **hearing** | **defaultHearingMultiplier, deafEntities** |
| **performance** | **visionScanRateTicks, lineOfSightCacheDurationTicks** |

**4.2. 自定义属性 (Attributes)**

**aperioculos:**hearing\_multiplier:

**作用:** 控制生物的听力敏锐度，直接乘以声音的通知半径。

**默认值:** 1.0 (可在配置中修改)。

**自定义: *可以通过JSON数据包为不同类型的生物设置不同的基础值，实现了高度的数据驱动。***

**///////////////// 5. 性能考量 ////////////////**

***性能是本框架设计的核心考量之一。***

**主动扫描优化:** 视觉扫描以玩家为中心，避免了对服务器上所有生物的无效迭代。

**缓存机制:** 视线缓存（LoS Caching）极大地降低了最昂贵的光线追踪操作的频率。

**高效听觉:** 听觉系统基于原版高效的GameEvent和VibrationListener机制，避免了对声音播放事件的低效监听。

**廉价检查优先:** 所有感知逻辑都遵循将低成本检查置于高成本检查之前的原则。

**//////////////// 6. 集成指南 /////////////////**

**添加依赖:** 在build.gradle中添加对aperioculos的依赖。

**监听事件:** 创建一个事件处理器类，订阅TargetSpottedEvent和SoundHeardEvent。

**实现决策逻辑:** 在事件处理器中，根据接收到的感知信息和生物的AI梯队，执行相应的决策逻辑（如设置目标、写入大脑记忆、广播群体警报等）。

**使用API (可选):** 在自定义的AI Goal或Behavior中，可以随时调用AperiOculosAPI中的方法进行即时感知查询。

**////////////// 7. 未来扩展方向 //////////////**

***该框架的模块化设计为未来的功能扩展提供了坚实的基础。***

**嗅觉系统:** 可添加一个基于粒子或路径点的嗅觉系统，用于追踪目标留下的“气味”。

**环境影响:** 可将天气（如下雨会抑制声音传播）、生物状态（如饥饿会提升感知敏锐度）等因素纳入感知计算。

**魔法/科技感知:** 可为特定生物添加感知魔法能量或红石信号的能力。