**AI模块设计**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 编辑 | 说明 |
| 1.0 | 2019.04.15 | great90 | 创建文档 |
|  |  |  |  |

目录

[1. AI模块的大致规划 1](#_Toc6262688)

[1.1 当前的AI方案 1](#_Toc6262689)

[1.2 为什么选择状态机 1](#_Toc6262690)

[1.3 AI框架的整体设计 1](#_Toc6262691)

[2. 有限状态机与分层有限状态机 2](#_Toc6262692)

[2.1 有限状态机 2](#_Toc6262693)

[2.2 分层有限状态机 2](#_Toc6262694)

[3. AI状态设计 3](#_Toc6262695)

[3.1 移动类 3](#_Toc6262696)

[3.2 攻击类 3](#_Toc6262697)

[3.3 防御类 3](#_Toc6262698)

[3.4 其他行为 3](#_Toc6262699)

[4. AIControl 4](#_Toc6262700)

[4.1 AI状态等数据的保存 4](#_Toc6262701)

[4.2 目标选取支持 4](#_Toc6262702)

[4.3 与其他模块的交互 4](#_Toc6262703)

[5. 配置信息 5](#_Toc6262704)

[5.1 领地范围 5](#_Toc6262705)

[5.2 警戒范围与攻击范围和攻击技能 5](#_Toc6262706)

[5.3 其他智能行为 5](#_Toc6262707)

# 1. AI模块的大致规划

## 1.1 当前的AI方案

当前的AI方案参照了MC1.6中的AI设计，每个AI实体包含多个不同类型的AI行为，每种AI行为实现了一个独立的AI功能，例如游泳、释放技能等等，甚至选择目标都是一个AI行为。每个AI行为之间存在优先级和互斥性的问题，同一时间只能有一个行为执行，某些事件发生时，满足执行条件的高优先级AI行为可以打断低优先级的AI行为抢先执行。行为之间的互斥关系采用位掩码的方式实现。当增加一个行为时，为了维护与之前已有行为的关系，可能需要再对原有的行为设置进行修改，位掩码的互斥实现显得不那么方便，也不太容易理解。

## 1.2 为什么选择状态机

基于上述所说的AI行为增多的时候，维护起来显得并不容易。行为之间的联系较为松散，层次性不明显，可控性比较差。当要处理一些事件时，事件的派发也不能明确指定。采用状态机的设计方案，状态的实现和上述AI行为的实现区别不大，但不再需要掩码表达互斥性，优先级也不是必要的。行为增多时，可以利用分层有限状态机来组织状态实现层次化的管理，更易于理解和维护。

## 1.3 AI框架的整体设计

AI模块的整体设计大致分成两大部分：基本的状态机部分实现通用的常见AI功能，例如寻敌、攻击和回家等功能。另外，仇恨值，寻路和目标选取等作为状态机的支撑功能也包含在其中。另一部分对于负责的特定功能，如某些Boss大招技能，则可以通过Trigger-Action的行为树机制来实现更方便灵活。

# 2. 有限状态机与分层有限状态机

## 2.1 有限状态机

对于某个AI对象的来说，其行为可进行归类成数量有限的几种。例如一只羊在草地上吃草，还会随机或跟随头羊走动，也有可能到水边喝水，遇到危险时会跟随羊群逃跑。上面列举的四种行为对应着羊的四种状态，基本能描述出一只羊的智能出来。当状态确定后，还需要建立状态间的转换，以及某一状态的持续执行。因此需用一个有限状态机（Finite State Machine简称FSM）来进行驱动，每个状态内部只需要负责各自的进入、执行和退出操作，状态机负责对他们进行调用。状态间的转换关系可以通过建立状态转换映射表来配置实现。

## 2.2 分层有限状态机

当AI状态增涨到一定程度的时候，维护他们就有难度了。将他们按照层次性或互斥的功能相似性进行组织，分层划分是一个非常好的办法。分层有限状态机（Hierarchical Finite State Machine简称HFSM）就是做这件事的，它不仅降低了整个状态机的复杂度，也让整个系统更易于扩展和维护。

对于走路、游泳、飞翔和驾驶等移动类的状态，在分层有限状态机中可以统一成移动状态这一大类作为高层次的状态。在这个移动状态中也是一个状态机，它根据不同的移动方式，执行更低层次的具体的移动状态。因为这些移动状态是彼此独立且互斥的，划分到一个新的独立子状态机中是合理的。同理，近战攻击和远程攻击也可以划分到一个攻击状态下。

# 3. AI状态设计

## 3.1 移动类

根据当前需求，需要处理的移动类型有两种：行走和游泳（航海项目中），未来可能还有飞翔和搭乘之类的。行走必须是在可行走方块上，游泳只能在水中，这两种状态的处理比较容易，具体移动的过程交由C++这些，这一点是通过AIControl与C++交互实现的。

## 3.2 攻击类

目前的攻击方式都是通过释放技能实现的，近战技能是MeleeAttack，远程是释放子弹技能，通过子弹命中造成伤害。他们的攻击距离不同，或者说目标选取存在一定的差异，设计AI状态的时候也可以考虑分开。攻击间隔可以理解为技能CD，这个可以直接通过配置来体现。追逐目标也是攻击类状态的一种。

## 3.3 防御类

非战斗状态下，AI总会做一些事情来体现其智能，例如对周围进行警戒巡逻，脱战后迅速回到自己的领地，受伤后逃离攻击者等等。这一类的状态往往有比较多的细节设计。

## 3.4 其他行为

为了增强现实或趣味性，还可以为AI增加一些不太重要又有意思得行为状态，例如动物可以时不时地发出叫声这种等等。

# 4. AIControl

AIControl是管理AI实体的AI相关的功能模块，它负责AI相关数据的保存，为AI状态机提供支持接口，负责驱动AI状态机的执行，还有与其他模块进行交互等功能支持。

## 4.1 AI状态等数据的保存

AI对象在执行AI的过程中会产生很多的数据，这些数据在执行中影响着AI的决策和行为，因此需要一直保存，直到AI的生命周期结束。最基本的就是当前的AI状态，还有仇恨值列表用于攻击对象的选取，有些AI对象还有耐心值，进入战斗时间过长就会自动脱战等等。

## 4.2 目标选取支持

AI执行技能是非常常见的一种状态，技能执行的前提是存在可释放目标。有些技能的目标选取可以直接放在状态中完成，例如自己或周围的某个位置等；另一些技能的目标选取则比较复杂，需要经过筛选或依赖其他的数据，这种目标选取的接口就可以放在AIControl中，实现比较复杂的一些逻辑。

## 4.3 与其他模块的交互

当游戏内某些事件产生，需要通知AI进行相应的处理。例如当AI巡逻发现目标时，需要对目标进行追踪，移动的过程是由C++层完成的，因此需要告知C++模块移动的目标点，这一步骤就是由AIControl模块完成的。这类接口也应该由一个统一的管理者提供支持，AI进行决策，运动层的细节不必了解，更不需要替代它取执行。

# 5. 配置信息

AI状态有引擎编码提供支持，但不同游戏实体的AI行为还是存在一些差别的，这些差别主要是通过配置来实现，最终达到表现效果的不同。

## 5.1 领地范围

几乎每个AI实体都有一个属于自己的领地，一般将自己的出生点当做领地中心，通过配置指定领地半径。AI的正常活动范围就是由上述两个信息所指定的圆。当AI实体脱战后也会向自己的领地中心靠拢。

## 5.2 警戒范围与攻击范围和攻击技能

在AI实体没有确定的攻击对象时，会进行巡逻警戒，一般警戒范围都要比攻击范围大，当发现敌人又打不到攻击范围时，就会进行追逐，这也是体现智能的一个方面。达到攻击范围后，还需要选择可释放的技能进行攻击，不同技能的数值都不太一样，这也能体现出AI行为的多样性。

## 5.3 其他智能行为

对于更高级或更特殊化的智能行为，可以直接在AI实体对象类的BTS脚本中配置Trigger-Action来实现更高级的定制。