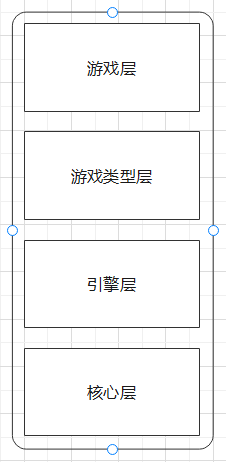
体系结构

按照一般的游戏结构层次划分，我们的游戏可以先大体分为核心层（Core），引擎层（Engine），游戏类型层（Game Genre），游戏层（Game）,一般的商用游戏引擎就用到了引擎层和核心层，而引擎使用者一般都是实现游戏层和游戏类型层。这种分层的架构设计就可以帮助我们把复杂的系统进行解构，从而实现每个子系统或者模块的功能单一化。



但是在实际使用中，我们组则是通过这样的思路，按多层次结构设计了一个核心架构，每个层次都可以更加细致的分派不同的人来负责，该架构分为五层：



第一层：更新世界信息：

这部分主要分为两个设计：一个是知识池（Knowledge Pool），另一个就是感知器（Sensor）。

知识池（Knowledge Pool）：

将世界的数据信息搜集起来，通过key-value的方法存储起来；

感知器（Sensor）：

即获取数据的方法，可以设置一个感知器来获取地图的信息并将信息存入到知识池中；

1、可以通过遍历所有的感知器来获得所有的世界信息

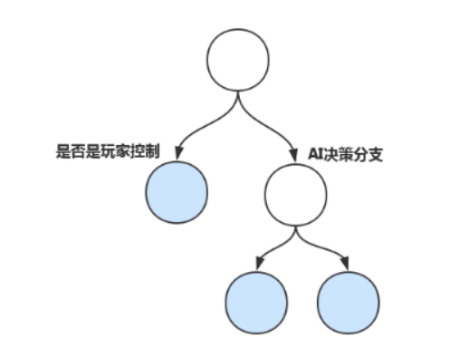
2、感知器设计时可以设计两种不同的感知器，一种是全局感知器，可以将它看作对整个世界、关卡的抽象，还有一种是个体感知器，通过挂载到主角上来实现视野，听力，主角信息捕获等等。

第二层：更新决策层（做什么）：

该层就是用来处理输入，比如当玩家走到怪物攻击范围内时，怪物会产生“做什么”的决策，例如攻击等，而这些“做什么”的决策，会封装到一个“请求”的结构中，然后向下一层传递。

在该部分，我们使用了Unity 的状态机，行为树和神经网络结构。

玩家的输入也属于这一层，因为玩家的输入也是一种决策，这一部分就是通过行为树来进行处理，将玩家输入与AI可以共用：



第三层：更新请求层：

请求层的作用是接受上一层发来的“请求”，然后将该请求经过过滤后传递到下一层中，其实就是一个过滤器，在这个过滤器中，我们设置了一个潜在的优先级，比如当我们角色在执行一个普攻请求时，产生了一个新的请求：释放技能，这时候可以通过潜在优先级的设置，普攻会被释放技能给“打断”，然后优先释放技能。

有了这一层，就可以保证上一层不用担心自己的决策是否会冲突，而只要关心发出的决策就好了，这样做大大降低了上一层实现的复杂度。

第四层：更新行为层（怎么做）：

该层的作用就是用来处理上层的策略，这层的实现同样使用了行为树、状态机，比如在释放一个技能时，我们需要先集气，然后再释放，就可以通过行为树来实现。

在这一层中，会产生一系列的输出，有特效，有动画，还有声音等，然后就是接受运动信息，这部分包括空间信息（位移）和动画信息，在这一层可以进行一些基础的操作，如太过复杂，就需要下一层来实现

第五层：更新运动层：

我们所用的移动方式为：动画跟随物理，比如为了解决移动时动作不协调问题，可以做多个移动动画，然后根据速度做融合，可以减轻不协调性