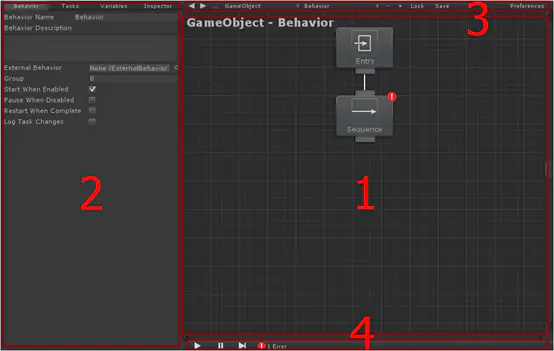
### 简述

Behavior Designer 是一个行为树插件！是为了让设计师，程序员，美术人员方便使用的可视化编辑器！Behavior Designer 提供了强大的 API 可以让你轻松的创建 tasks（任务），配合 uScript 和 PlayMaker 这样的插件，可以不费吹灰之力就能够创建出强大的 AI 系统，而无需写一行代码！

本指南将介绍所有 Behavior Designer 的功能特性！如果你还不了解什么是行为树（behavior trees）请查看"行为树的概述"！依赖于 Behavior Designer你完全可以不用关心什么是行为树（behavior trees）但是，如果你了解了一些 behavior trees！这将有助于你使用 Behavior Designer，包括一些常用组件，例如：tasks（任务），action（行为），composite（复合），conditional（条件），decorator（修饰符）!

### 界面

这里一共分为四大部分：下图中的第一个部分是主要操作区，用来创建你的行为树！第二部分是面板属性区，这里可以编辑一个行为树的特定属性，添加新任务，创建新的变量，或者编辑 tasks（任务）的参数。第三部分是工具栏，你可以添加/删除行为树，锁定当前行为树，查看所有行为树，等等！第四部分是调试工具栏。你可以启动/停止，逐步调试，暂停，查看错误！



## 行为树组件

这个组件记录了你的行为树的结构以及一些 BehaviorDesigner 配置信息！下面的 API 用来启动和停止你的行为树！  
public void EnableBehavior();  
public void DisableBehavior(bool pause = false);  
你可以通过下面的这些方法查找行为树中的相关节点 task 任务  
TaskType FindTask< TaskType >();  
List< TaskType > FindTasks< TaskType >();  
Task FindTaskWithName(string taskName);  
List< Task > FindTasksWithName(string taskName);  
行为树当前的执行状态可以像下面这样获取：  
behaviorTree.ExecutionStatus;  
当行为树运行结束后会有一个状态被返回，返回的接口可能是 Success 成功或者是 Failure 失败，这个结构依赖于行为树中的各个  
子节点 Task 任务的返回值！  
你可以对行为树监听以下事件：  
OnBehaviorStart  
OnBehaviorRestart  
OnBehaviorEnd  
行为树组件包含以下几个属性：  
Behavior Name  
行为树的名称  
Behavior Description  
行为树的描述信息  
External Behavior  
一个外部行为树的资源引用，行为树可以被导出成外部序列化文件（ScriptableObject 文件）单独存储，并被其他行为树引用，或  
者作为子节点任务而使用！方便了行为树的共用！  
Group  
行为树的分组编号，用来将行为树分组！可以用来方便的查找到特定的行为树！  
Start When Enabled  
如果设置为 true，那么当这个行为树组件 enabled 的时候，这个行为树就会被执行！  
Pause When Disabled  
如果设置为 true，那么当这个行为树组件 disabled 的时候，这个行为树就会被暂停！  
Restart When Complete  
如果设置为 true，那么当这个行为树组件执行结束的时候，这个行为树就会被重新执行！  
Reset Values On Restart  
如果设置为 true，那么当这个行为树组件 reset 的时候，这个行为树就会被重新执行！  
Log Task Changes  
当设置为 true 是，这个行为树下只要 task 流程发生变化就会打印一条 log 日志到控制台中！

## 节点

### Composites （复合）节点

Sequence（序列）节点

这个节点是一个"和"的关系，也就是他下面的子节点的执行顺序是一个接着一个的！如果其中一个返回 false。那么后续的子节点不会被执行，这个序列节点返回 false。只有当所有子节点全部完成并返回 success 的时候，这个 Sequence（序列）节点才会返回 success;

Selector（选择）节点

这个节点是"或"的关系，也就是他下面的子节点的执行顺序是一个或另一个的！只有所有子节点返回 false 才会返回 false。只要有一个子节点返回 success，那么这个 Selector 节点就会返回 success，后续的节点不会被执行！

Parallel（并行）节点

这个节点类似于 Sequence（序列）节点。不同的是，Parallel（并行）节点会在同一时间执行所有子节点而不是一个一个的去执行！如果子节点中有任意一个返回 false，则停掉所有子节点并返回 false。只有所有子节点全部返回 success 的时候，才会返回 success。

Parallel Selector（并行选择）节点

类似于 Selector（选择）节点，ParallelSelector(并行选择)节点只要有一个子节点返回 success，那么他就会返回 success！不同于 Selector的一点就是 ParalleSelector（并行选择）节点会在同一时间执行下面的所有子节点，如果有一个节点返回 success，则会停止掉其他所有子节点并返回 success。只有当所有子节点全部 false 的时候才会返回 false！

Priority Selector（优先选择）节点

类似于 Selector（选择）节点，PrioritySelector(并行选择)节点只要有一个子节点返回 success，那么他就会返回 success！不同点在于，子节点的执行顺序不是从左到右的，而是通过优先级来确定的执行顺序！较高的优先级的子节点会被先执行！（译者：优先级在哪里设置的，没有搞清楚，目前测试结果同 Selector 节点，后来还是用我大 Google 搜索到的解决办法！百度就是个垃圾站）需要在 Task 类中覆盖函数，来设置不同的 Priority,原文地址：[http://forum.unity3d.com/threads/behavior-designer-behavior-trees-for-everyone.227497/page-4](https://link.jianshu.com/?t=http://forum.unity3d.com/threads/behavior-designer-behavior-trees-for-everyone.227497/page-4" \t "https://www.jianshu.com/p/_blank)

Random Selector（随机选择）节点

类似于 Sequence（序列）节点，只是他的执行顺序是随机的！只要遇到一个子节点返回 false，RandomSequence（随机序列）就返回错误，直到全部子节点都返回 success，它才会返回 success！在这个节点的属性面板中有：seed（随机种子）的设置，自行使用！

Selector Evaluator（重复判断选择）节点

这个节点每帧都会去重新评估子节点的执行状态并选择。它会执行子节点中优先级最低的子节点！每帧都会这么干！如果当前一个高优先级的节点在运行并且下一帧要执行的子节点优先级比较低，那么它会打断高优先级的节点，去执行优先级低的子节点！ Selector Evaluator（重复判断选择）节点会从低到高的去遍历执行所有子节点，直到遇到一个返回 success 的！如果所有子节点都返回 false，那么它就返回 false！否则只要有一个返回 success，它就会返回 success！这个节点模拟了条件打断功能，如果子节点没有条件节点的话！

Conditionals（条件判断）节点

条件节点的任务是判断游戏的一些属性，比如玩家是否活着，怪物是否在视野距离内！

Random Probability (随机概率)节点

通过设置 successProbability 属性来控制返回 success 的几率（默认 0.5，也就是 50%几率）！另外还有 seed 随机种子的设置等！

Compare Field Value（字段比较）节点

比较指定的值的字段值。 返回成功如果值是相同的。

Has Received Event（是否接收到事件）

Decorators（修饰器）节点

这种节点的功能是用来包装另一个节点！（只能有一个子节点）。Decorators（修饰节点）将改变节点的行为！例如：修饰节点可以再运行时控制子节点直到返回某个特定状态（success 或者是 false）。后者是对子节点返回结果取反（即：success 返回 false，false，返回 success）；

下面来一一介绍 BehaviorDesigner 默认自带的几个 Decorator（装饰器节点）

Conditional Evaluator (条件节点的评估) 装饰节点

参数设置：  
1：reevaluate ：条件节点是否需要每帧都重新评估一次  
2：conditionalTask：要被评估的条件节点，注意：这个节点本身就是个条件节点！  
对设置的条件节点进行评估，如果条件节点返回 success，那么运行子节点并返回子节点的运行结果！如果条件节点没有返回 success那么子节点不会被运行，并且立刻返回 failure！条件节点只会在开始运行的时候被评估一次！

Interrupt（打断）装饰节点

如果打断节点被触发，则打断下面的所有子节点任务的执行！打断命令可以被 Perform interruption（执行打断）节点发起！打断节点在收到打断命令前，不会打断他下面的子节点的执行状态！如果子节点执行完毕还没有收到打断命令，则直接返回子节点的执行结果！

Inverter（取反）装饰节点

子节点的任务完成后返回值，在这个节点会被取反并传递到上一级中！

Repeater（重复/循环）装饰节点

有三个属性设置：执行次数，是否一直重复，运行直到返回错误！

Return Failure （返回失败）装饰节点

只要子节点当前的状态不是 running，也就是子节点执行结果无论是 success 还是 failure，都返回 failure！如果子节点状态是 running的话则返回 running！

Return Success（返回正确）装饰节点

只要子节点当前的状态不是 running，也就是子节点执行结果无论是 success 还是 failure，都返回 success！如果子节点状态是 running的话则返回 running！

Task Guard（任务守卫）装饰节点

类似于多线程互斥操作中使用的 Lock 标记，为了避免公共数据被多次引用！

Until Failure（直到失败）装饰节点

直到子节点返回 failure，否则一直循环执行子节点

Until Success（直到成功）装饰节点  
直到子节点返回 success，否则一直循环执行子节点！

Action接口方法

// OnAwake在行为树启用时被调用一次。把它看作一个构造函数

public virtual void OnAwake();

// OnStart在执行之前被立即调用。它用于设置从上次运行中需要重置的任何变量

public virtual void OnStart();

// OnUpdate运行实际的任务

public virtual TaskStatus OnUpdate();

// OnEnd在执行成功或失败后被调用。

public virtual void OnEnd();

// 当行为暂停和恢复时，调用OnPause

public virtual void OnPause(bool paused);

//优先级选择需要知道该任务的运行优先级

public virtual float GetPriority();

// OnBehaviorComplete在行为树执行完成后被调用

public virtual void OnBehaviorComplete();

// OnReset被检查器调用来重置公共属性

public virtual void OnReset();

// 允许从任务中调用OnDrawGizmos

public virtual void OnDrawGizmos();

// 保持对拥有此任务的行为的引用

public Behavior Owner;