Плагин BehaviourTree позволяет создавать сложный ИИ с помощью дерева поведения. Для того, чтобы полностью понять текст документации, нужно быть знакомым с концепцией дерева поведения. Для ознакомления будет достаточно этого видео:

<https://www.youtube.com/watch?v=KeShMInMjro>

Первое, что нужно сделать для создания ИИ – создать ноду BehaviourTree. Как только это будет сделано, откроется вкаладка создания дерева, с помощью которой можно добавлять базовые ноды дерева. Все ноды в ней наследуют Composite, Decorator или Leaf, которые, в свою очередь, наследуются от класса Task. С учетом этого, далее будем называть все элементы дерева задачами. Порядок исполнения в дереве такой же, как и в годоте, если не используются рандомные композиты. За один физический кадр по умолчанию выполняется только одна задача. Увеличить число исполняемых задач можно увеличив значение TickSpeed.

**Задачи**

Композиты:

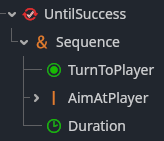
* Sequence – работает как &&;
* Selector – работает как ||;
* Random – исполняет случайную задачу и возвращает ее результат;
* RandomSequence – Sequence с случайным порядком исполнения;
* RandomSelector – Selector с случайным порядком исполнения;
* ParallelSequence – Sequence, который сначала выполняет все задачи, а потом уже возвращает результат. Работает в один тик ИИ (плохо для производительности);
* ParallelSelector – Selector, который сначала выполняет все задачи, а потом уже возвращает результат. Работает в один тик ИИ (плохо для производительности).

Декораторы:

* AlwaysFail – всегда вовращает провал;
* AlwaysSucceed – всегда возвращает успех;
* Invert – работает как !;
* Limit – ограничивает число успехов подряд. После достижения предела возвращает провал;
* Repeat – повторяет задачу заданное число раз и возвращает успех;
* UntilFail – повторяет задачу, пока не будет достигнут провал;
* UntilSuccess – повторяет задачу, пока не будет достигнут успех.

Листья:

* Print – выводит в консоль сообщение, возвращает успех;
* Wait – останавливает исполнение дерева на заданное число секунд;
* BlackboardDebug – выводит в консоль значение с доски;
* PlayAnimation – проигрывает анимацию. Если WaitUntilEnd = true, то на время проигрывания анимации дерево перестанет исполнять свою логику. Если PlayPart = true, проиграется часть анимации, выбранная параметрами ниже.
* Cooldown – возвращает успех, затем возвращает провал, пока не пройдет заданное число секунд.
* Duration – при первом вызове возвращает провал и запускает таймер на заданное число секунд. Продолжает возвращать провал, пока не пройдет заданное число секунд, потом возвращает успех и сбрасывает свое состояние. Можно использовать вместе с UntilSuccess(), чтобы добавить задержку с выполнением какого-то действия:



**Создание новой задачи**

При создании новой задачи в первую очередь нужно создать новый класс и унаследоваться от Composite, Decorator или Leaf в зависимости от типа задачи. Если листок возыращает всегда один и тот же результат, то имя класса должно оканчиваться на Task. Если же листок делает проверку и возвращает разные результаты, то имя должно оканчиваться на Check. Для написания логики используются следующие методы:

Методы для доопределения:

* Run(Dictionary blackboard) – исполняется каждый тик дерева, если оно активно;
* ChildSuccess() – исполняется, если ребенок вернул успех;
* ChildFail() – исполняется, если ребенок вернул провал;
* ChildRunning() – исполняется, если ребенок продолжает исполняться.

Методы для использования:

* SetSuccess() – возвращает успех родителю задачи;
* SetFail() – возвращает провал родителю задачи;
* SetRunning() – возвращает продолжение исполнения родителю задачи.

В общем случае нужно использовать один из трех методов выше, иначе дерево остановится, причем использовать их нужно только в методах для доопределения. Если же остановка дерева необходима, можно использовать методы возвращения результата в методах для переопределения годота, но нужно быть осторожным, чтобы не вернуть результат дважды. Возвращение результата в момент, когда задача не исполняется, тоже приведет к непредсказуемому поведению.

**Доски**

Доска – это динамически типизированный словарь, который используется для передачи данных внутри дерева. Каждое дерево содержит собственную доску, которая может быть использованно в качестве памяти ИИ или для связи с другими ИИ с помощью ноды SharedBlackboard. Для подключения дерева к SharedBlackboard нужно использовать метод ConnectTree, для отключения – DisconnectTree.

**Машины состояний**

Плагин также позволяет создавать конечные и иерархические машины состояний. Видео для ознакомления с концепцией:

https://youtu.be/-ZP2Xm-mY4E?feature=shared

<https://youtu.be/DPxIMVC0oZA?feature=shared>

Состояния для машин нужно создавать, наследуясь от класса State. Каждое состояние имеет методы для доопределения: PhysicsProcess(), Process(), Enter(), Exit() – в этих методах должна содержаться логика состояния. Абстратная функция CheckTransitions() должна содержать условие перехода к другому состоянию. Для перехода используется метод Fsm.TransitionTo(string stateName), где stateName – имя в редакторе ноды состояния, в которое требуется перейти. Для создание иерархической машины состояний необходимо добавить к ноде состояния конечную машину состояний в качестве ребенка – теперь добавленные к новой машине состояния будут работать лишь тогда, когда работает состояние-родитель.