Содержание

Введение

Объем рынка игр растет с каждым годом [17]. В первую очередь развитие игровых приложений обязано стремительному прогрессу, но также важная роль в разработке игр ложится на плечи программистов. Именно с их помощью реализуются идеи геймдизайнеров, именно они собирают воедино воображения художников с целью создания новой игры.

Программисту, в настоящее время, чтобы написать игру, что называется, «с нуля» необходимо позаботиться о многих важных аспектах: игровая платформа, графическая система, аудио система, системы моделирования игровой физики и виртуального интеллекта игры. Каждая из этих областей объемна и потребуется много времени, чтобы связать воедино все системы в соответствии с игровой логикой.

Для того, чтобы упростить и ускорить работу над созданием игры, было написано множество библиотек для работы с графическими системами и для моделирования игровой физики. Такие библиотеки, как Lightweight Java Game Library [15], предоставляют инструменты для работы с графической и аудио системами, а библиотека box2d [11] предоставляет инструменты для моделирования реалистичной физики в плоском пространстве. Однако для упрощения реализации виртуального интеллекта долгое время не было придумано, по большому счету, ничего. Над созданием искусственного интеллекта в игре программист трудился самостоятельно, так как этот процесс не был сложным или утомительным. В конечном итоге модуль виртуального интеллекта содержал в себе некоторое количество условных операторов.

Такое решение обнажило бы свою слабую сторону в играх с развитым поведением персонажей потому, что условных операторов становилось очень много, что затрудняло отладку и дальнейшую поддержку игры. Пример такой игры – Spore [16]. В ней все игровые персонажи имели если не уникальное, то редко повторяющееся поведение, для создания которого в Spore использовался совершенно

другой подход.

Только в последнее десятилетие (примерно с момента выхода игры Spore) виртуальный интеллект в играх стал развиваться и появились некоторые библиотеки AIEngine (Artificial Intelligence Engine), которые обобщали накопленные программистами знания об интеллекте в играх. Эти библиотеки предоставляют инструменты для создания конечных автоматов поведения, для обработки взаимодействия автономных объектов. Но конечные автоматы в общем виде сложны и часто запутаны. Чтобы стратегия объекта была ясной были созданы деревья поведения – конечные автоматы древовидной структуры, состоящие из вершин трех типов: вершины-действия, вершины-условия и управляющие вершины. Вершина-действие содержит в себе некоторое возможное действие объекта (бежать, искать, стрелять), вершина-условие содержит в себе некоторый предикат (есть патроны?, враг рядом?), в зависимости от которого выбирается следующее состояние, управляющая вершина организует порядок обхода дочерних вершин (параллельно, до первой успешной вершины, до первой неуспешной вершины).[1]

Цель данной работы - проектирование библиотеки и реализация визуального средства для создания стратегий поведения виртуальных игровых персонажей на основе деревьев поведений.

[1] Деревья поведения подробно описаны в главе 1.

1 Обзор существующих инструментов

1.1 Общие сведения

Для разработки виртуальных игровых объектов необходимо создать и поддерживать большой набор их поведений. Например, в военных играх виртуальному персонажу необходимо распознавать опасность и убегать в укрытие, когда уровень здоровья ниже 10%. От количества разнообразных вариантов действий, которые могут использовать игровые персонажи, зависит количество игровых ситуаций, которые могут быть распознаны и приняты во внимание. Чем больше различного поведения игроки будут встречать в играх даже от несущественных (фоновых) объектов[2], тем интереснее будет игра.

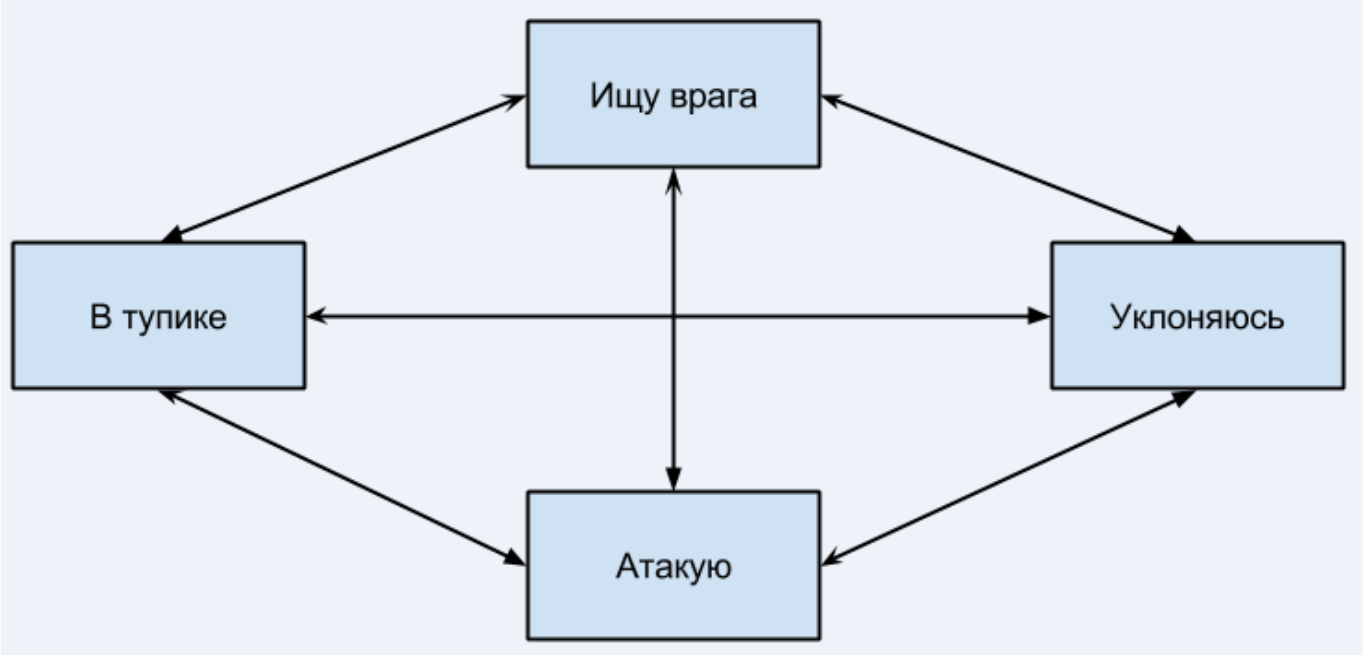

Рисунок 1.1 – Пример конечного автомата поведения игрового объекта

Для создания поведения долгое время использовались конечные автоматы, где каждое поведение может быть представлено графически (рисунок 1.1). Одновременно автомат может находиться в одном состоянии, которое представляет

---

[2] Несущественные объекты – это игровые объекты, такие как массовка, животные, птицы, исключая ситуации, когда именно эти персонажи – главные герои в играх. Поведение несущественных объектов никак не влияет на ход игры.

поведение объекта. Каждое состояние имеет логику переходов с соответствующими проверками. Например, если игрок находится в состоянии «В тупике» и обнаружил врага, и здоровье игрока больше 50%, то следующим состоянием будет «Атакую».

Мы сделали анализ и пришли к выводу, что такой подход к реализации принятия решений игровыми объектами имеет ряд недостатков:

- Расширяемость: конечный автомат с большим количеством состояний теряет преимущество графического представления, со временем такое поведение станет невозможно понять.
- Изменяемость: при добавлении/удалении поведения (состояния) необходимо изменить все другие состояние, которые связаны с новым/старым поведением; большие изменения могут приводить к ошибкам логики поведения объекта в целом.
- Параллельные алгоритмы: запускать состояния автомата параллельно не представляется возможным.

В 2005 году был создан более эффективный способ принятия решения игровыми объектами по сравнению с конечными автоматами – деревья поведения.

Дерево поведения [1-2] – это направленный связный ациклический[3] граф, имеющий единственную вершину, в которую не входят ребра – корень дерева. Из пары вершин, соединенных ребром, та, из которой выходит ребро, называется родительской вершиной, а другая дочерней вершиной. Вершина, не имеющая дочерних, называются листом. Каждое поддерево дерева поведения определяет различное поведение. Вершины, находящиеся между корнем дерева и листами могут быть двух типов – декораторами или композитами. Корень дерева поведения периодично генерирует сигнал, который передает дочерним вершинам, заставляя их выполнять алгоритм, определенный типом вершины. Как только сигнал достигнет

[3] В данном случае дерево поведения не должно содержать циклов даже если не учитывать направления ребер.

листа, то лист произведет некоторые вычисления и вернет одно из 4 состояний: «успешно» (success), «не успешно» (failure), «запущено» (running), «ошибка» (error). Возвращенное состояние передастся родительским вершинам, для принятия решений в соответствии с типом вершины. Процесс закончится тогда, когда корневая вершина вернет некоторое состояние.

1.2 Типы вершин

Все вершины дерева поведения по способу обработки дочерних вершин делятся на три типа: декоратор, композит, лист.

1.2.1 Лист

Лист – это неделимая часть дерева поведения. Эти вершины не имеют дочерних вершин, они принимают сигнал, производят некоторые вычисления и возвращают результат родительской вершине.

Существует два вида листовых вершин: лист-условие и лист-действие. Лист-условие выполняет проверку некоторого условия и возвращает соответствующий результат (успешно, не успешно или ошибка). Лист-действие выполняет действие и возвращает результат успешно, запущено или ошибка.

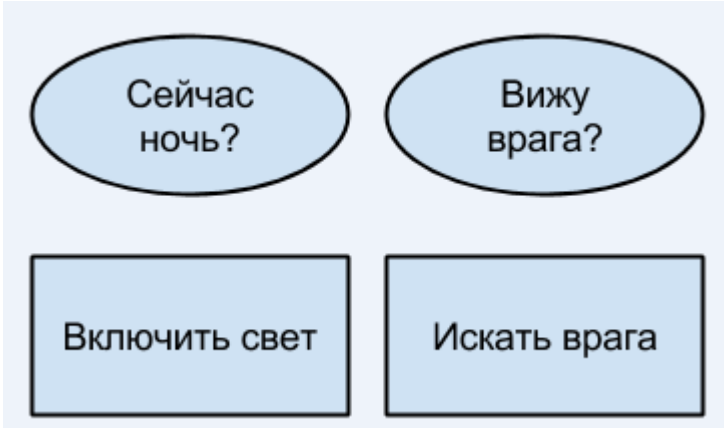

Рисунок 1.2 - Графическое представление листовых вершин

Графически лист-условие изображается овалом, лист-действие прямоугольником (рисунок 1.2).

1.2.2 Композит

Композит имеет одну или больше дочерних вершин. Он принимает и передает сигнал дочерним вершинам в некотором порядке, и также решает какое и когда вернуть состояние. Композит всегда возвращает одно из трех состояний: «успешно», «не успешно» или «ошибка». Все композитные вершины изображаются в виде квадрата со специальным символом внутри.

Существует три вида композитной вершины (рисунок 1.3): композит-селектор, композит-последовательность и параллельный композит. Композит-селектор обрабатывает дочерние вершины до тех пор, пока дочерняя вершина возвращает результат «не успешно», затем пробрасывает полученный результат родительской вершине и заканчивает выполнение. Если все дочерние вершины вернули результат «не успешно», то композит-селектор вернет результат «не успешно». Специальный символ для композита-селектора – знак вопроса.

Композит-последовательность обрабатывает дочерние вершины до тех пор, пока они возвращают результат «успешно», затем пробрасывает полученный результат родительской вершине и заканчивает выполнение. Если все дочерние вершины вернули результат «успешно», то композит-последовательность вернет результат «успешно». Специальный символ для композита-последовательности – стрелка вправо.

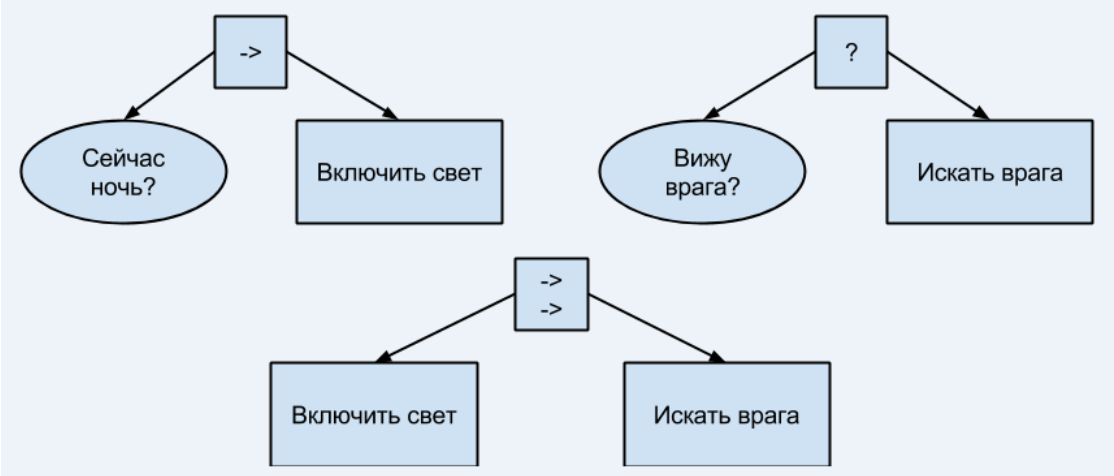

Рисунок 1.3 – Примеры вершин типа «композит»

Параллельный композит обрабатывает все вершины одновременно, возвращает

«успешно», если количество дочерних вершин с результатом «успешно» превышает некоторую константу S (которая может быть различна для разных параллельных композитов), возвращает результат «не успешно», если количество дочерних вершин с результатом «не успешно» превышает некоторую константу F, которая так же может быть определена для конкретного параллельного композита, иначе возвращает результат «запущено». Специальный символ для параллельного композита – две стрелки вправо.

1.2.3 Декоратор

Декоратор – это специальная вершина, которая имеет ровно одну дочернюю вершину. Цель, которую преследует декоратор, - изменить возвращаемое дочерней вершиной значение, или повлиять на частоту передаваемого сигнала дочерней вершины. Например, декоратор может делать инверсию возвращаемого значения, а может повторить сигнал, передаваемый дочерней вершине 3 раза. Декоратор изображается в виде ромба с пояснением внутри.

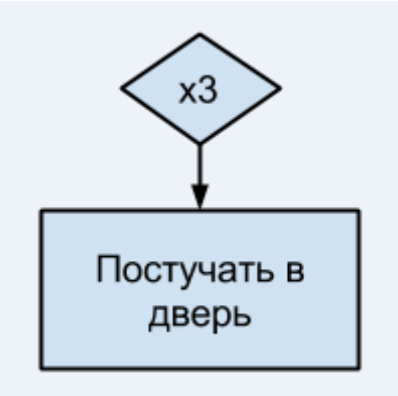

Рисунок 1.4 – Пример вершины-декоратора, который три раза передаст сигнал дочерней вершине

1.2.4 Взаимодействие различных вершин

Автономным объектам необходимо в процессе принятия решения хранить промежуточную информацию об окружающем мире. Эта информация формирует систему «восприятия» мира для объекта, она может включать, например, последнюю видимую позицию врага, количество видимых объектов, последнее совершенное действие или любые другие вычисленные данные. Таким образом дерево поведения

некоторого игрового объекта должно хранить и использовать некоторую информацию о мире.

Для решения данной задачи применяется blackboard [3] (рисунок 1.5), который активно используется вершинами дерева поведения для чтения и записи информации. Blackboard – это ассоциативный массив, к которому имеют доступ все вершины дерева поведения.

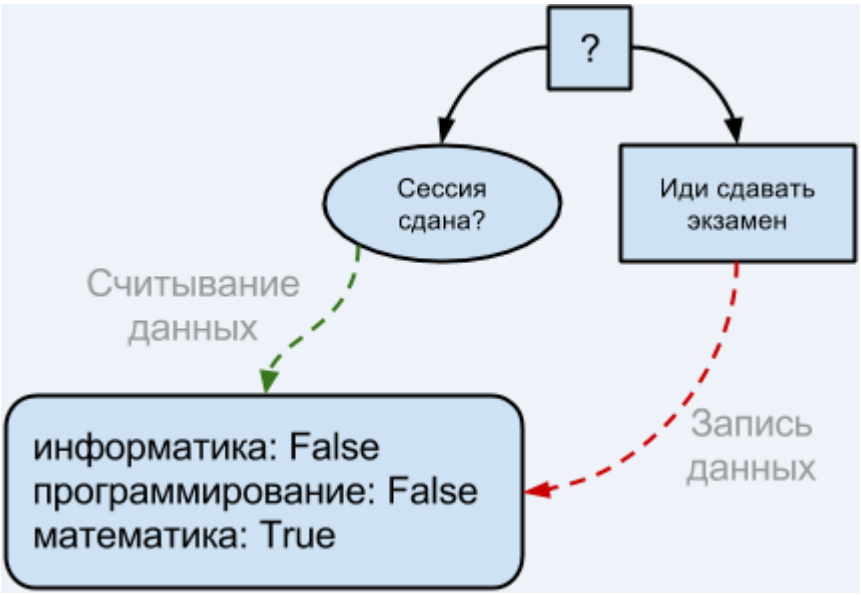

Рисунок 1.5 - Использование blackboard

1.3 Деревья поведения в современных фреймворках

В некоторых достаточно крупных игровых фреймворках существует возможность создания стратегии персонажей с помощью деревьев поведения. Мы рассмотрим три фреймворка, использующих эту технологию: Unreal Engine, LibGDX, Unity3D.

1.3.1 Unreal Engine

UnrealEngine – игровой движок, разрабатываемый компанией Epic Games. Различные версии этого фреймворка были использованы во многих современных играх.

Для реализации стратегии для автономных объектов в UnrealEngine можно

использовать деревья поведения [4], причем в данном игровом движке есть возможность создать дерево поведения, используя графический интерфейс, при этом не написав ни строчки кода.

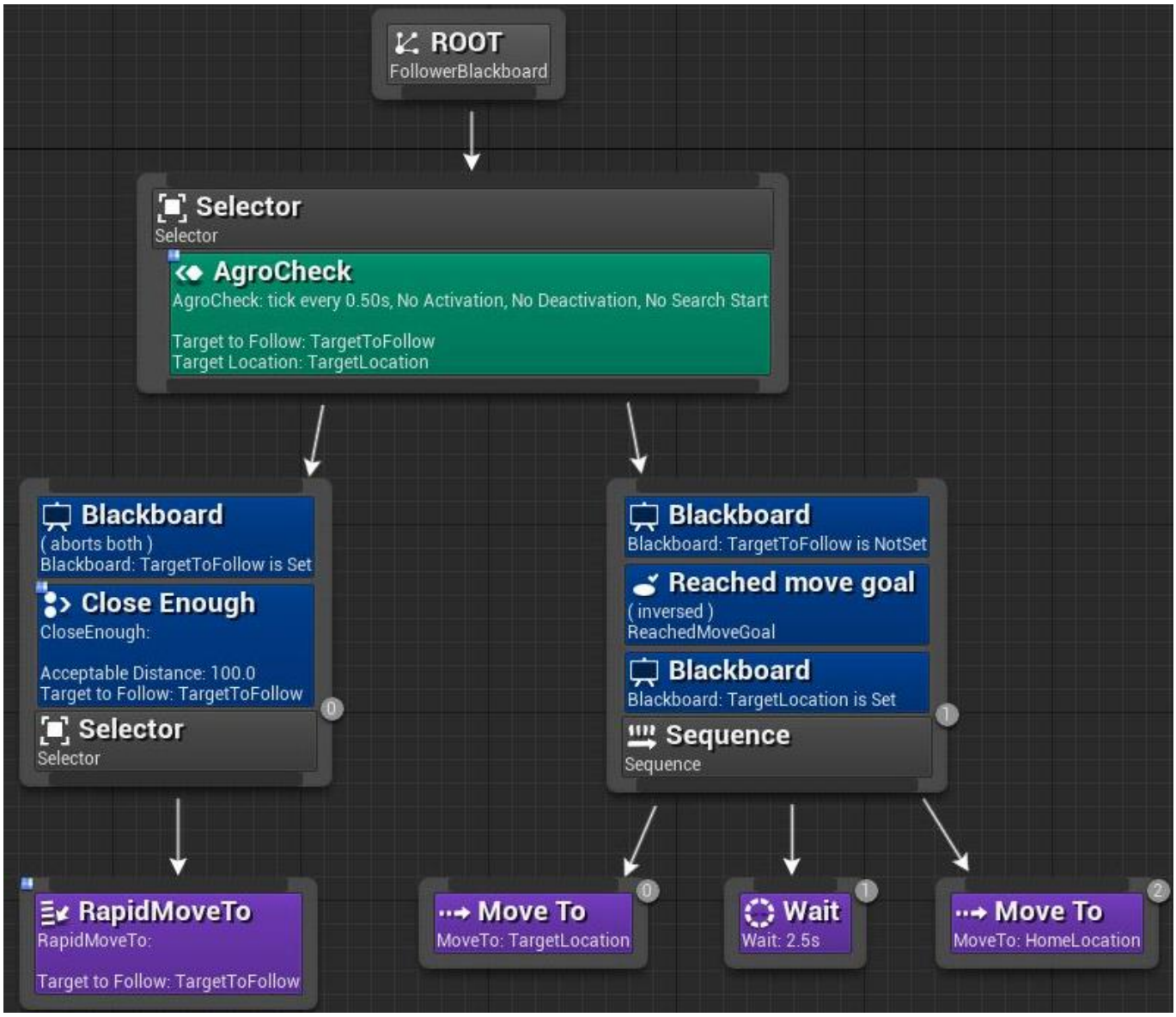

Рисунок 1.6 - Реализация дерева поведения в UnrealEngine

Рассмотрим использование деревьев поведения в UnrealEngine на примере: имеется игровой мир в виде комнаты со стенами, в котором находятся два персонажа, один из них – человек, другой – виртуальный интеллект, задача виртуального интеллекта – найти человека и подойти к нему. Для реализации такого рода стратегии в UnrealEngine необходимо построить дерево поведения (рисунок 1.6).

Данное дерево состоит из двух вершин-селекторов, вершины-

последовательности, четырех вершин-действий. В первой вершине-селектор находится дополнительное действие «AgroCheck», которое реализует «зрение» виртуального интеллекта, устанавливая, в частности, две переменные: позиция и ссылка на персонажа, которого необходимо отыскать. Композит-селектор слева включает в себя два декоратора, которые проверяют установлена ли ссылка на персонажа и приемлема ли до него дистанция, при верном ответе на эти два вопроса выполняется действие – быстрое движение к искомому объекту. Если же объект пропадет из поля видимости, то тогда сигнал пойдет к левому композиту-последовательности, который содержит три декоратора, проверяющих, что цель не установлена, цель не достигнута и некоторая позиция установлена, в этом случае будут выполняться следующие действия. Сначала виртуальный интеллект переместится до установленной позиции (скорее всего это та позиция, где он последний раз видел другого персонажа), затем подождет 2.5 секунды и отправится на исходную точку.

1.3.2 LibGDX

LibGDX – кроссплатформенный игровой фреймворк, написанный на языке Java и C++. LibGDX используют для написания мобильных приложений и игр. В своем составе этот фреймворк имеет модуль AI [5], который реализует алгоритмы нахождения кратчайших путей, взаимодействия автономных объектов (steering behavior), а также алгоритмы принятия решений на основе behavior tree.

Рассмотрим создание дерева поведения на том же примере, что и в предыдущем разделе. Так как в деревьях поведения LibGDX AI нет декораторов, проверяющих некоторое условие, то аналогичное дерево поведения будет выглядеть как показано на рисунке 1.7.

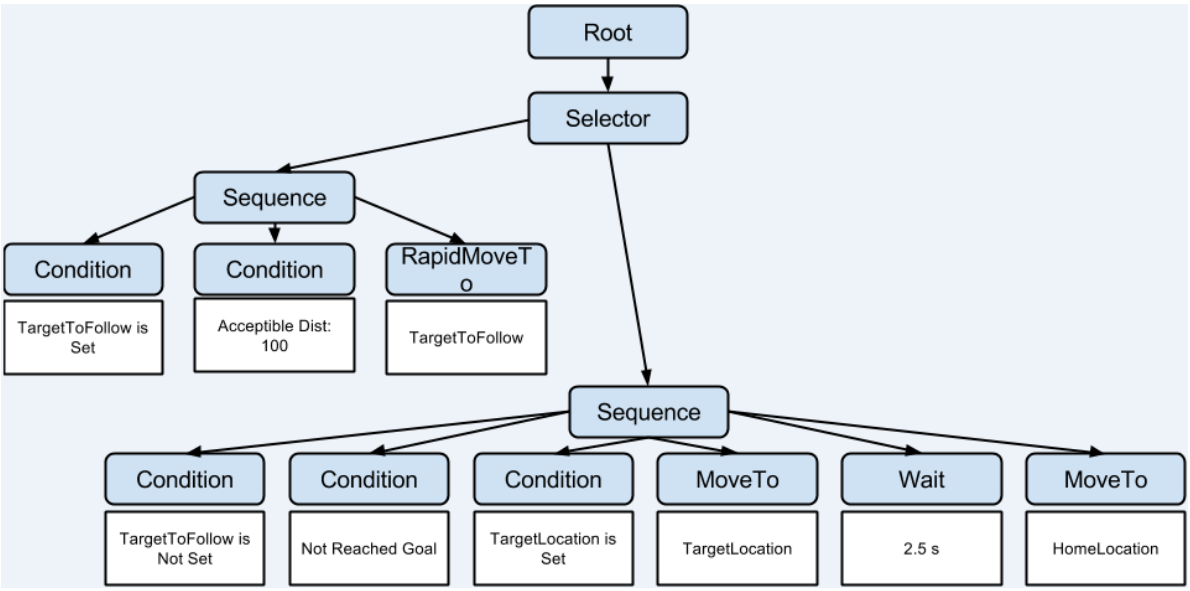

Рисунок 1.7 - Дерево поведения с использованием модуля LibGDX AI

Так как в LibGDX AI не предусмотрено графического средства для создания деревьев поведения, то полученное на рисунке 1.7 дерево необходимо создать непосредственно в коде.

```
private BehaviorTree<Blackboard> createHeroBehavior() {
            ConditionTask<Blackboard> targetToFollowIsSet = new ConditionTask<>(
                        bb -> bb.targetToFollow != null);
            ConditionTask<Blackboard> acceptibleDist = new ConditionTask<>(
                        bb -> dist(bb.targetToFollow.location, me.location) <= 100);
            RapidMoveTo<Blackboard> rapidMoveTo =
                         new RapidMoveTo<>(bb.targetToFollow.location);
      Sequence<Blackboard> sequenceSeeMan = new Sequence<>(
                        targetToFollowIsSet, acceptibleDist, rapidMoveTo);

            ConditionTask<Blackboard> targetToFollowIsNotSet = new ConditionTask<>(
                        bb -> bb.targetToFollow == null);
            ConditionTask<Blackboard> notreachedGoal = new ConditionTask<>(
                        bb -> dist(bb.targetLocation, me.location) > 1);
            ConditionTask<Blackboard> targetLocationIsSet = new ConditionTask<>(
                        bb -> bb.targetLocation != null);
```

```
            MoveTo<Blackboard> moveToTargetLocation = new MoveTo<>(bb.targetLocation);
            Wait<Blackboard> wait = new Wait<>(2500);
            MoveTo<Blackboard> moveToHomeLocation = new MoveTo<>(bb.homeLocation);
        Sequence<Blackboard> sequenceNotSeeMan = new Sequence<>(
                    targetToFollowIsNotSet,
                    notreachedGoal,
                    targetLocationIsSet,
                    moveToTargetLocation,
                    wait,
                    moveToHomeLocation
        );

        Selector<Blackboard> selector = new Selector<>(sequenceSeeMan, sequenceNotSeeMan);
        Selector<Blackboard> root = new Selector<Blackboard>(selector);
        BehaviorTree<Blackboard> bt = new BehaviorTree<>(root, blackboard);
        return bt;
}
```

В LibGDX для каждой вершины дерева поведения есть общий класс – Task. Наследуясь от него можно создавать пользовательские типы вершин, которые не предусмотрены в базовой структуре классов. Так, лист-условие `Condition`, листы-действия RapidMoveTo, MoveTo и Wait необходимо создать и реализовать их логику. Так как композит-последовательность передает сигнал на исполнение дочерним вершинами до тех пор, пока они возвращают «успешно», то данное дерево будет аналогичным дереву в разделе 1.3.1. Действительно, если некоторый лист-действие вернет «не успешно», то композит-селектор прекратит передачу сигнала на исполнение и следующим дочерним вершинам сигнал не будет передан.

В методе `createHeroBehavior` описан процесс создания дерева поведения с использованием LibGDX AI. Лист-условие Condition задается предикатом с одним параметром – blackboard, с помощью которого и определяется собственно истинность условия. Листы-действия RapidMoveTo и MoveTo задаются с помощью функции, которая возвращает позицию, к которой необходимо двигаться. Лист-действие Wait задается длительностью интервала задержки в миллисекундах.

1.3.3 Unity3D

Unity3D – игровой движок, разрабатываемый компанией Unity Technologies. В силу наличия бесплатной версии и огромного количества поддерживаемых платформ [7] этот движок весьма популярен среди многих крупных разработчиков игр (Blizzard, EA, QuartSoft, Ubisoft).

В составе Unity3D отсутствуют инструменты для работы с искусственным интеллектом, но в магазине плагинов [8] можно найти дополнение, позволяющее создавать деревья поведения для игровых объектов. Один из таких плагинов – Behaviour Machine Free. Рассмотрим создание дерева поведения с использованием данного плагина на примере из предыдущего раздела.

В Behaviour Machine Free много стандартных вершин, таких как Translate (переместить объект по указанному направлению на указанную длину), IsFloatLess/IsFloatGreater (проверить, что некоторое число с плавающей точкой меньше/больше заданного числа), GetDistance (получить расстояние между двумя объектами и записать результат в blackboard), IsSee (проверить, видит один объект другой или нет).

Таким образом, для того чтобы создать необходимое поведение для персонажа, нужно в графическом интерфейсе для определения способа принятия решений создать структуру дерева поведения и структуру blackboard (рисунок 1.8), а затем задать параметры для каждой вершины дерева. Так, для `Agro Check` необходимо задать объекты, между которыми нужно проверять видимость, для Acceptible Dist необходимо задать переменную и число, чтобы проверять, что переменная меньше числа, для Wait необходимо задать время задержки в миллисекундах и т.д.

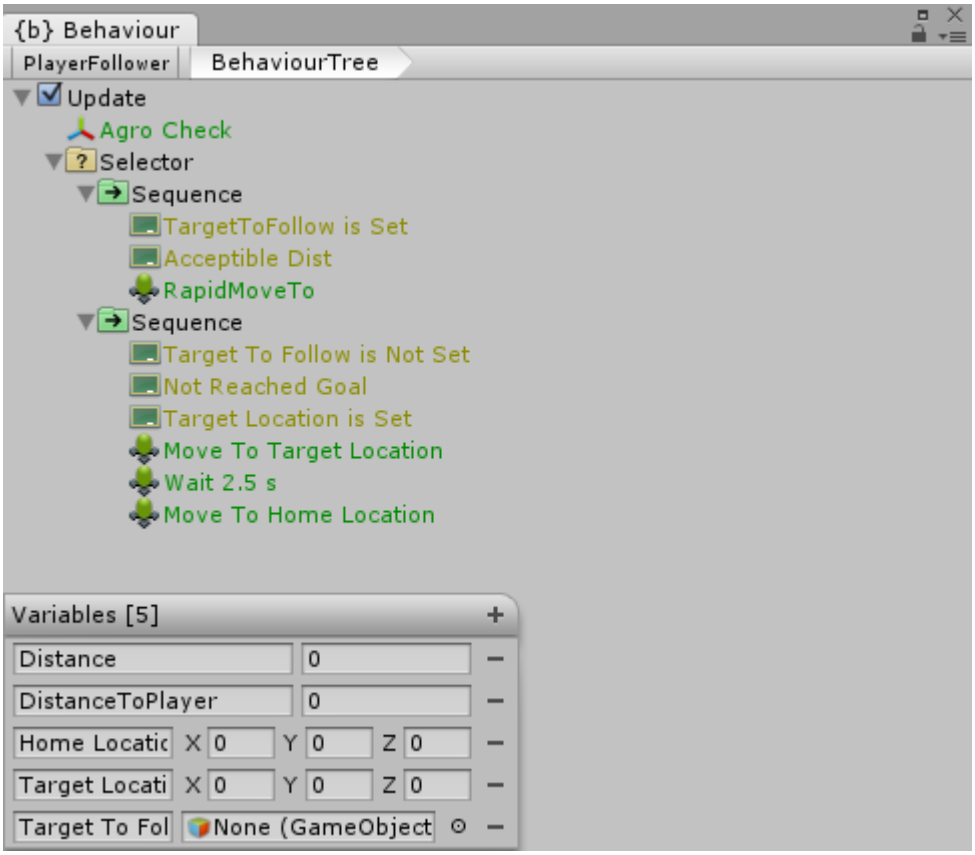

Рисунок 1.8 - Дерево поведения с использованием Behaviour Machines на Unity3D

1.4 Итоги

Таким образом, мы рассмотрели три крупных игровых фреймворка, в которых есть возможность использовать деревья поведения для создания логики принятия решений виртуальным интеллектом, и решили одну задачу с помощью этих фреймворков. Были отмечены следующие недостатки: использовать деревья поведения UnrealEngine или Behaviour Machine Free в проектах, написанных на

других фреймворках не представляется возможным, а для модуля Behavior Tree из LibGDX AI не существует визуального средства создания деревьев поведения.

1.5 Постановка задачи

Необходимо разработать кроссплатформенную библиотеку и визуальное средство для создания деревьев поведения, которые не будут зависимы от конкретного игрового фреймворка. Библиотека должна обладать стандартизированным и расширяемым программным интерфейсом. Также необходимо разработать демонстрационный пример, созданный при помощи визуального средства проектирования деревьев поведения, показывающий основные возможности библиотеки.

2 Библиотека Behavior Tree

2.1 Обоснование выбора языка программирования

Для реализации библиотеки Behavior Tree мы выбрали язык Java 8 по нескольким причинам:

- Java – кроссплатформенный язык программирования
- Наличие анонимных методов
- Низкий порог вхождения

2.2 Проектирование схемы классов

На рисунке 2.1 частично представлена разработанная структура классов библиотеки Behavior Tree.

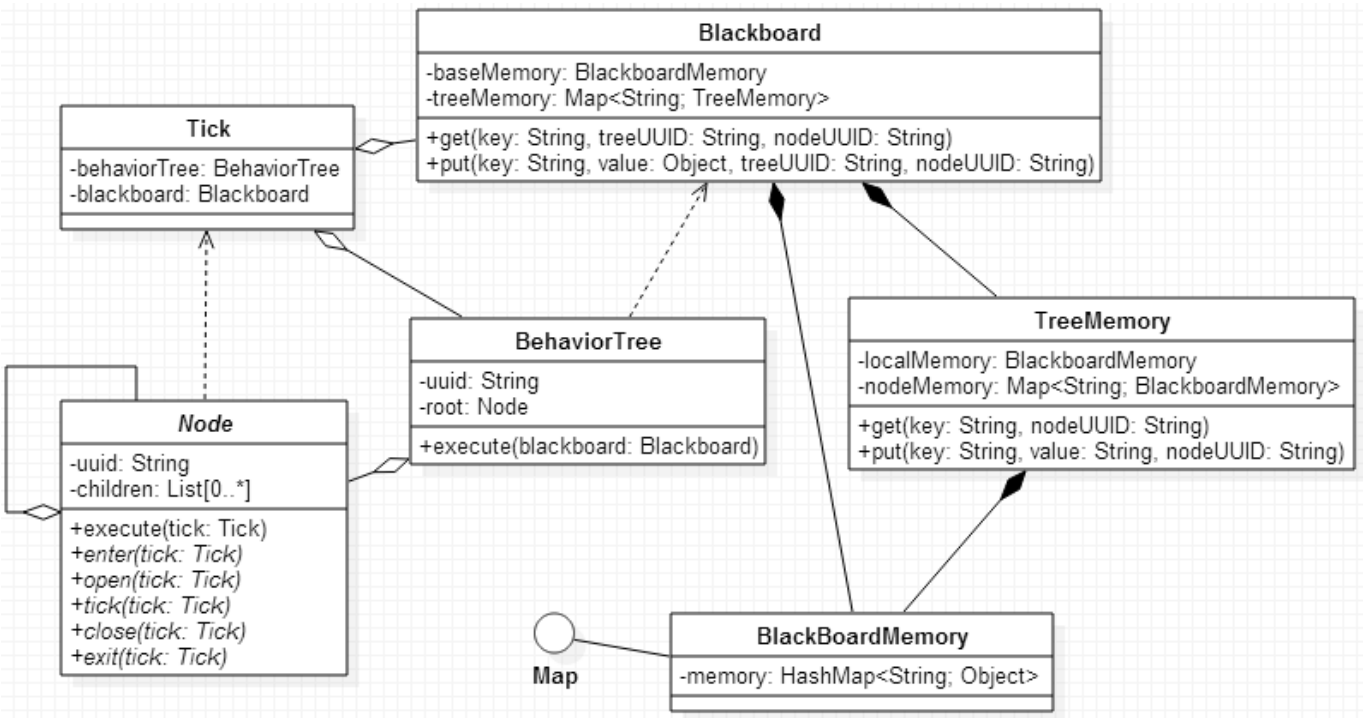

Рисунок 2.1 - Схема классов библиотеки Behavior Tree

Рассмотрим подробно структуру Blackboard. Для эффективного хранения/чтения/записи переменных мы разбили «память» на несколько областей: глобальная область, область дерева, область вершины в дереве.

Глобальная область содержит переменные, логически не связанные ни с одним поведением (baseMemory). Такие переменные могут отражать некоторую общую

характеристику персонажа, такую, как количество забитых мячей в матче или текущее состояние здоровья.

Область дерева (TreeMemory) содержит переменные, логически связанные с одним поведением. Такие переменные могут хранить информацию о поведении, например, «защита»: количество видимых врагов, место ближайшего укрытия и количество дееспособных объектов в своем отряде. К области памяти дерева поведения можно получить доступ при указании уникального идентификатора дерева treeUUID.

Область вершины в дереве содержит локальную информацию, характерную только для конкретной вершины в конкретном поведении. Мы используем эту область для хранения информации о последней запущенной вершине в композитах с запоминанием. Пользователю рекомендуется здесь хранить ключевую информацию, используемую в пользовательских листах-действие. К области вершины в дереве можно получить доступ при указании уникальных идентификаторов дерева (treeUUID) и вершины (nodeUUID) в нем.

Каждая вершина и каждое дерево поведения имеют уникальные идентификаторы для определения участка памяти для хранения переменных в Blackboard. Эти уникальные идентификаторы генерируются по стандарту идентификации UUID [10] при создании объекта.

Для того, чтобы каждая вершина имела доступ к набору переменных и дереву поведения, мы используем объект класса Tick, который как сигнал на исполнение передается дочерним вершинам. Объект класса Tick создается один раз на принятие одного решения, то есть этот объект создается в методе execute класса BehaviorTree и передается корневой вершине дерева поведения.

Все вершины, представленные на рисунке 2.2, наследуются от класса Node и реализуют некоторые его абстрактные функции, необходимые для обеспечения логики вершины.

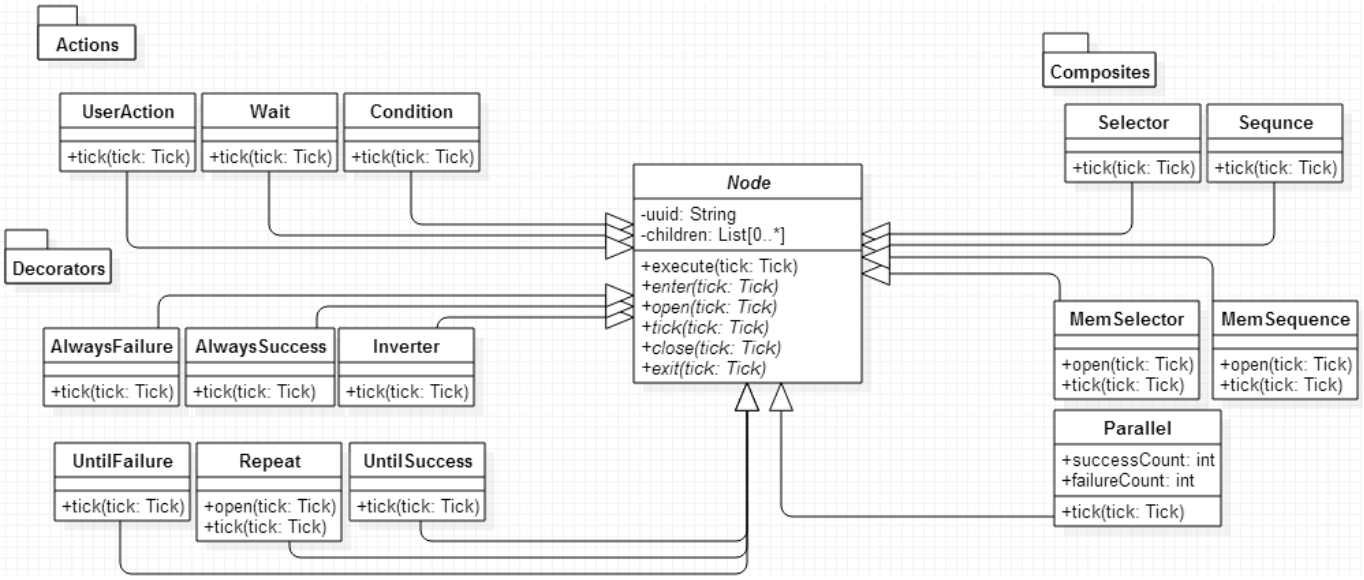

Рисунок 2.2 - Различные классы вершин

Так сигнал на исполнение запускает метод tick(), которая реализует ядро логики вершины. Например, вершина-действие Wait в этом методе проверяет, прошло ли достаточное время с момента первого запуска. Абстрактный метод enter() класса Node запускается всякий раз, когда сигнал на исполнение пришел в вершину. Метод open() запускается только в том случае, если после последнего запуска данной вершины она вернула результат не «запущено». Метод close() запускается в том случае, если текущий результат не «запущено». Таким образом, в один момент принятия решения могут быть ситуации, когда метод open был запущен, а метод close() – нет, или наоборот. Метод exit() исполняется всякий раз перед возвратом результата исполнения логики вершины.

Вышеописанная логика реализована в методе execute() класса Node. Именно этот метод запускается, когда необходимо передать сигнал на исполнение вершине.

2.3 Описание реализованных классов и их методов

Описание реализованных классов и методов разработанной библиотеки выполнено в виде Doxygen [9] комментариев и приведено в приложении Д. Мы выбрали Doxygen комментарии, так как они являются стандартом де-факто комментирования кода на различных языках.

2.4 Разработка примера использования библиотеки Behavior Tree

Для демонстрации примера использования разработанной библиотеки решим задачу из раздела 1.3.1. При решении задачи будем использовать игровой фреймворк LibGDX [5] для подсистемы ввода/вывода, так как он имеет простой в использовании интерфейс, и физический движок box2d [11] для обработки взаимодействия объектов. Разработанная UML диаграмма классов представлена на рисунке 2.3.

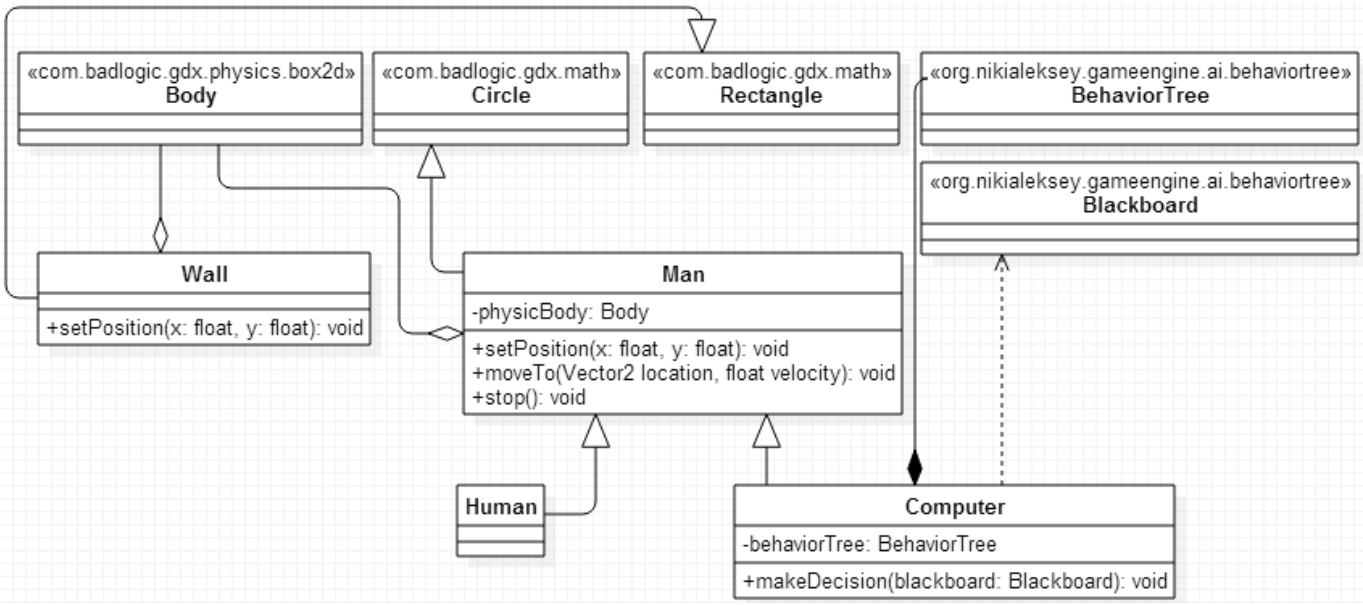

Рисунок 2.3 - UML диграмма классов демонстрационного примера

Виртуальный персонаж (далее компьютер) и персонаж, управляемый человеком (далее человек) будут иметь общий метод – двигаться к некоторой точке с некоторой скоростью, поэтому создадим общий для них класс Man:

```
class Man extends Circle {
    Body physicBody;

    public Man(Body physicBody, float x, float y) {
        this.physicBody = physicBody;
        this.setPosition(x, y);
        this.setRadius(manRadius);
    }
    @Override
    public void setPosition(float x, float y) {
        super.setPosition(x, y);
```

```
        this.physicBody.setTransform(x, y, 0);
    }
    public void moveTo(Vector2 location, float velocity) {
        float manAng = physicBody.getAngle();
        float manToLocAng = (float) Math.atan2(
                location.y - physicBody.getPosition().y,
                location.x - physicBody.getPosition().x
        );
        if (Math.abs(manAng - manToLocAng) > 0.1) {
            physicBody.setLinearVelocity(Vector2.Zero);
            physicBody.setAngularVelocity(manAng > manToLocAng ? -5 : 5);
        } else {
            physicBody.setLinearVelocity(
                    new Vector2(
                            location.x - physicBody.getPosition().x,
                            location.y - physicBody.getPosition().y
                    ).nor().scl(velocity)
            );
            physicBody.setAngularVelocity(0);
        }
    }
    public void stop() {
        physicBody.setAngularVelocity(0);
        physicBody.setLinearVelocity(Vector2.Zero);
    }
}
```

Метод moveTo(Vector2 location, float velocity) сначала разворачивает персонажа по направлению к точке location, а затем двигает вдоль этого направления со скоростью velocity. Метод stop() устанавливает линейную и угловую скорость значением ноль.

Класс Human, реализующий функциональность человек, не будет иметь каких-либо дополнительных методов, так как его задача – двигаться туда, куда укажет пользователь:

```
class Human extends Man {
    public Human(Body physicBody, float x, float y) {
        super(physicBody, x, y);
    }
```

```
}
```

Класс Computer, реализующий функциональность компьютер, должен принимать решения о том, куда двигаться дальше. Поэтому в его конструкторе создадим дерево поведения с необходимой логикой и добавим метод принятия решения:

```
class Computer extends Man {
    BehaviorTree behaviorTree;
    public Computer(Body physicBody, float x, float y) {
        super(physicBody, x, y);
        behaviorTree = new BehaviorTree(
                new Selector(
                        new AlwaysFailure(new UserAction(tick -> {
                             boolean seeHuman = isComputerSeeHuman();
                             tick.getBlackboard()
                                 .put("targetToFollow", seeHuman ? human : null);
                             if (seeHuman) {
                                 Vector2 hPos = human.physicBody.getPosition();
                                 tick.getBlackboard()
                                     .put("targetLocation", new Vector2(hPos.x, hPos.y));
                             }
                             tick.getBlackboard().put("computerLocation", null);
                        })),
                        new Sequence(
                                new Condition(
                           tick -> tick.getBlackboard().get("targetToFollow") != null
                                ),
                                new UserAction(tick -> {
                                    Human human =
                            (Human) tick.getBlackboard().get("targetToFollow");
                                    tick.getBlackboard()
                           .put("computerLocation", human.physicBody.getPosition());
                                })
                        ),
                        new Sequence(
                                new Condition(tick ->
                                    tick.getBlackboard().get("targetToFollow") == null),
                                new Condition(tick ->
                                    tick.getBlackboard().get("targetLocation") != null),
```

```
                                new UserAction(tick -> {
                                    Vector2 position = (Vector2)
                                        tick.getBlackboard().get("targetLocation");
                                    tick.getBlackboard().put("computerLocation", position);
                                })
                        )
                )
        );
    }
    public void makeDecision(Blackboard blackboard) {
        behaviorTree.execute(blackboard);
    }
}
```

Дерево поведения, реализующее поведение компьютера, представлено на рисунке 2.4 (рисунок получен с использованием разработанной программы для построения деревьев поведений).

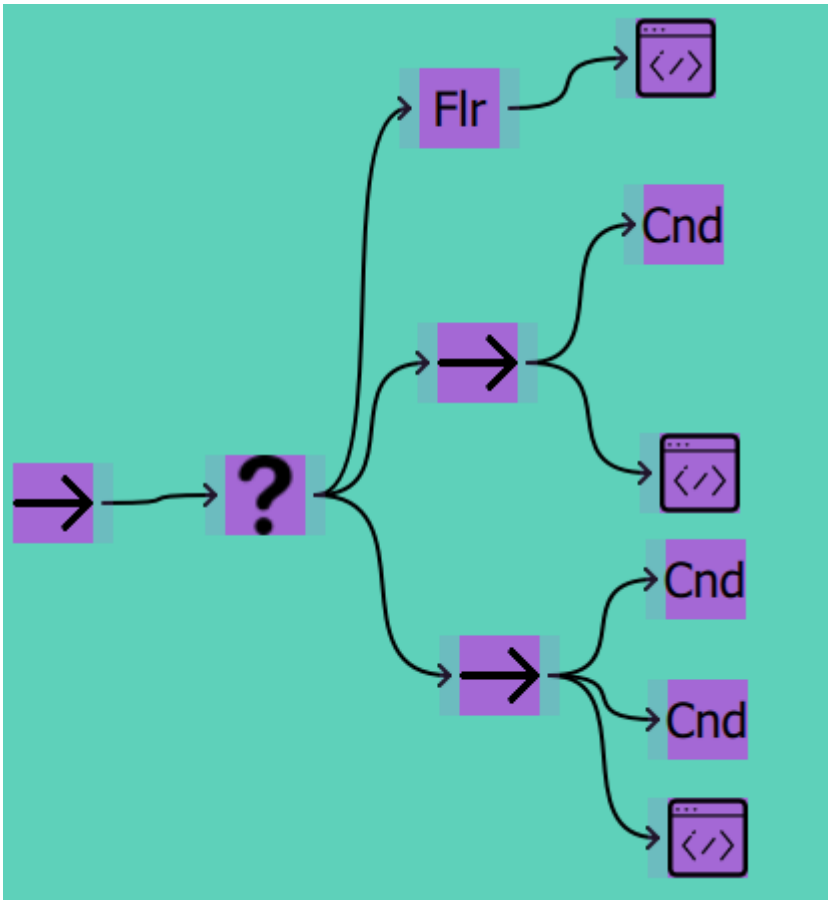

Рисунок 2.4 - Дерево поведения компьютера

Корень дерева поведения – композит-селектор, поэтому, если дочерняя

вершина корня возвращает результат «не успешно», то сигнал передается следующей вершине. Первая дочерняя вершина у корня – это лист-действие, обернутый в декоратор, который всегда возвращает «не успешно», поэтому действия, описанные в этом листе, будут выполняться каждый раз при принятии решения. Данный лист действие устанавливает переменные targetToFollow – объект класса Human, если компьютер его видит (нет никаких препятствий на отрезке, соединяющем центры кругов, объектов человека и компьютера соответственно), targetToLocation – последнее место, где был виден человек, computerLocation – точка, куда будет двигаться компьютер после принятия решения.

Следующая дочерняя вершина – композит-последовательность, поддерево с корнем в этой вершине определяет поведение в случае, если компьютер видит человека (первое условие), в этом случае вторая дочерняя вершина установит computerLocation точкой, где в данный момент находится человек.

Следующая дочерняя вершина – тоже композит последовательность, и определяет поведение компьютера в случае, когда он не видит человека. В этом случае computerLocation будет указывать на точку, где последний раз был виден человек.

Чтобы проверить видит ли компьютер человека или нет, необходимо провести отрезок между центрами кругов определяющих объекты компьютера и человека и проверить, не пересекает ли этот отрезок какое-нибудь препятствие. В нашем случае все препятствия – это прямоугольники. Функция, проверяющая, видит ли компьютер человека:

```
boolean isComputerSeeHuman() {
    Vector2 a = computer.physicBody.getPosition();
    Vector2 b = human.physicBody.getPosition();
    boolean isIntersect = false;
    for (Wall wall : walls) {
        float w = wall.getWidth();
        float h = wall.getHeight();
        float wallX = wall.getX();
        float wallY = wall.getY();
```

```
        isIntersect |=
                Intersector.intersectSegments(
                        a.x, a.y, b.x, b.y,
                        wallX - w / 2, wallY + h / 2, wallX + w / 2, wallY + h / 2,
                        new Vector2()
                )
                        || Intersector.intersectSegments(
                        a.x, a.y, b.x, b.y,
                        wallX - w / 2, wallY - h / 2, wallX + w / 2, wallY - h / 2,
                        new Vector2()
                )
                        || Intersector.intersectSegments(
                        a.x, a.y, b.x, b.y,
                        wallX - w / 2, wallY + h / 2, wallX - w / 2, wallY - h / 2,
                        new Vector2()
                )
                        || Intersector.intersectSegments(
                        a.x, a.y, b.x, b.y,
                        wallX + w / 2, wallY + h / 2, wallX + w / 2, wallY - h / 2,
                        new Vector2()
                );
    }
    return !isIntersect;
}
```

Множество стен определяется классом Wall, который представляет прямоугольник:

```
class Wall extends Rectangle {
    Body physicBody;
    public Wall(Body physicBody, float x, float y) {
        this.physicBody = physicBody;
        this.setSize(wallWidth, wallHeigh);
        this.setPosition(x, y);
    }
    @Override
    public Rectangle setPosition(float x, float y) {
        Rectangle rect = super.setPosition(x, y);
        this.physicBody.setTransform(x, y, 0);
        return rect;
    }
```

```
}
```

Теперь необходимо создать все объекты в переопределенном методе класса LibGDX ApplicationAdapter.create():

```
@Override
public void create() {
    int w = Gdx.graphics.getWidth();
    int h = Gdx.graphics.getHeight();
    batch = new SpriteBatch();
    debugRenderer = new Box2DDebugRenderer();
    camera = new OrthographicCamera(w, h);
    camera.translate(0, 0);
    world = new World(new Vector2(0, 0), true);
    // создаем человека
    BodyDef def = new BodyDef();
    def.type = BodyDef.BodyType.DynamicBody;
    Body circle = world.createBody(def);
    CircleShape circleShape = new CircleShape();
    circleShape.setRadius(0.1f);
    circle.createFixture(circleShape, 0.1f);
    human = new Human(circle, 0, 0);
    // создаем компьютер
    circle = world.createBody(def);
    def.type = BodyDef.BodyType.DynamicBody;
    circleShape = new CircleShape();
    circleShape.setRadius(0.1f);
    circle.createFixture(circleShape, 0.1f);
    computer = new Computer(circle, 1.5f, 1);
    blackboard = new Blackboard();
    // создаем стены
    def.type = BodyDef.BodyType.StaticBody;
    PolygonShape polygonShape = new PolygonShape();
    polygonShape.setAsBox(0.1f, 1f);
    Body rectangle = world.createBody(def);
    rectangle.createFixture(polygonShape, 0.1f);
    walls.add(new Wall(rectangle, -1, 0));
    rectangle = world.createBody(def);
    rectangle.createFixture(polygonShape, 0.1f);
    walls.add(new Wall(rectangle, 1, 0));
    // устанавливаем обработчик на мышь
```

```
    Gdx.input.setInputProcessor(new InputProcessor() {
        @Override
        public boolean keyDown(int keycode) {
            return false;
        }
        @Override
        public boolean keyUp(int keycode) {
            return false;
        }
        @Override
        public boolean keyTyped(char character) {
            return false;
        }
        @Override
        public boolean touchDown(int screenX, int screenY, int pointer, int button) {
            int w = Gdx.graphics.getWidth();
            int h = Gdx.graphics.getHeight();
            humanLocation = new Vector2(
                                (screenX - w / 2) / 100.0f, (h / 2 - screenY) / 100.0f);
            return false;
        }
        @Override
        public boolean touchUp(int screenX, int screenY, int pointer, int button) {
            return false;
        }
        @Override
        public boolean touchDragged(int screenX, int screenY, int pointer) {
            return false;
        }
        @Override
        public boolean mouseMoved(int screenX, int screenY) {
            return false;
        }
        @Override
        public boolean scrolled(int amount) {
            return false;
        }
    });
}
```

На каждой отрисовке кадра игры необходимо обновлять ее логику. Сделать это можно, переопределив метод ApplicationAdapter.render():

```
@Override
public void render() {
    …
    human.moveTo(humanLocation, 2);
    computer.makeDecision(blackboard);
    Vector2 computerLocation = (Vector2) blackboard.get("computerLocation");
    computer.moveTo(computerLocation, 1);
    …
}
```

Результат работы примера представлен на рисунках 2.5 и 2.6.

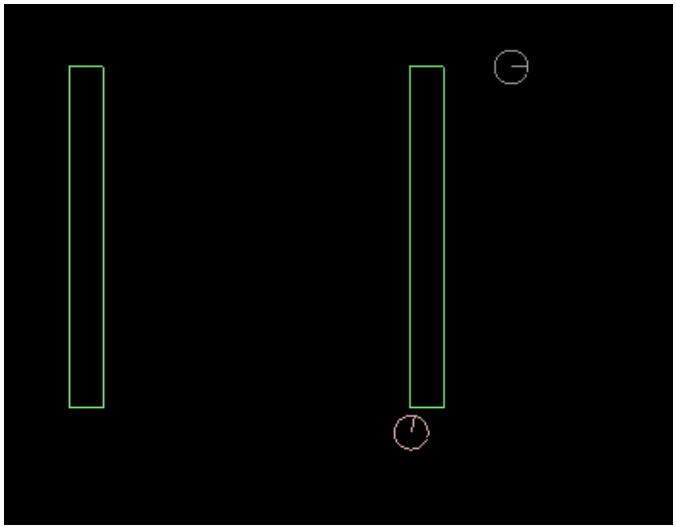

Рисунок 2.5 - Человек прячется от компьютера

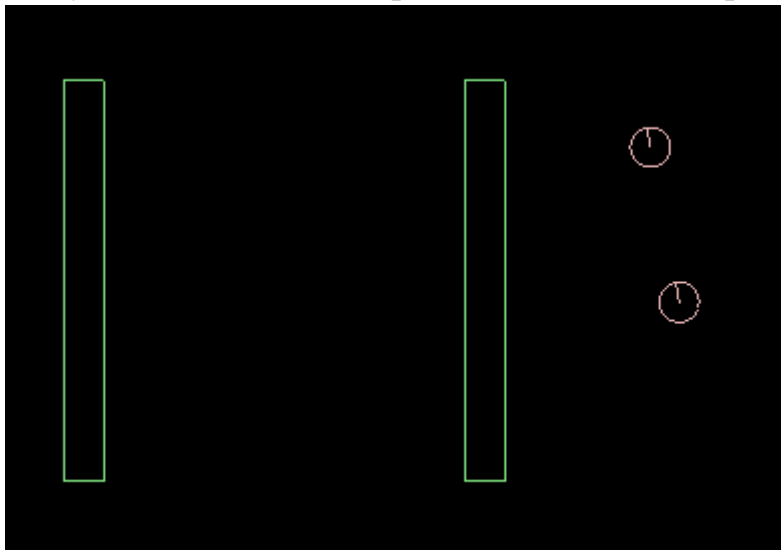

Рисунок 2.6 - Компьютер преследует человека

На рисунке 2.5 круг под прямоугольником представляет объект человека. В данном случае компьютер не видит человека, поэтому находится на месте. На рисунке 2.6 компьютер видит человека и идет вслед за ним.

3 Визуальный редактор стратегий

3.1 Выбор инструментов и технологий

Для реализации визуального средства создания деревьев поведений мы выбрали декларативный язык программирования QML с использованием скриптов на JavaScript под управлением и интеграцией компонентов Python с использованием библиотеки PyQt5.

Код на языке QML занимает меньше места, в отличие от других декларативных языков, таких как Xaml, HTML, FXML. Данный факт объясняется тем, что язык QML имеет много общего с языком JSON, тогда как большинство декларативных языков похожи на XML.

Альтернатива Python в качестве бэкенда для QML всего одна – это C++. Выбор был сделан в пользу Python по следующей причине: производительность Python-программиста выше производительности c++ программиста [13].

3.2 Проектирование схемы компонентов

В QML есть поддержка наследования компонентов [12]. На рисунке 3.1 представлена разработанная UML-диаграмма компонентов.

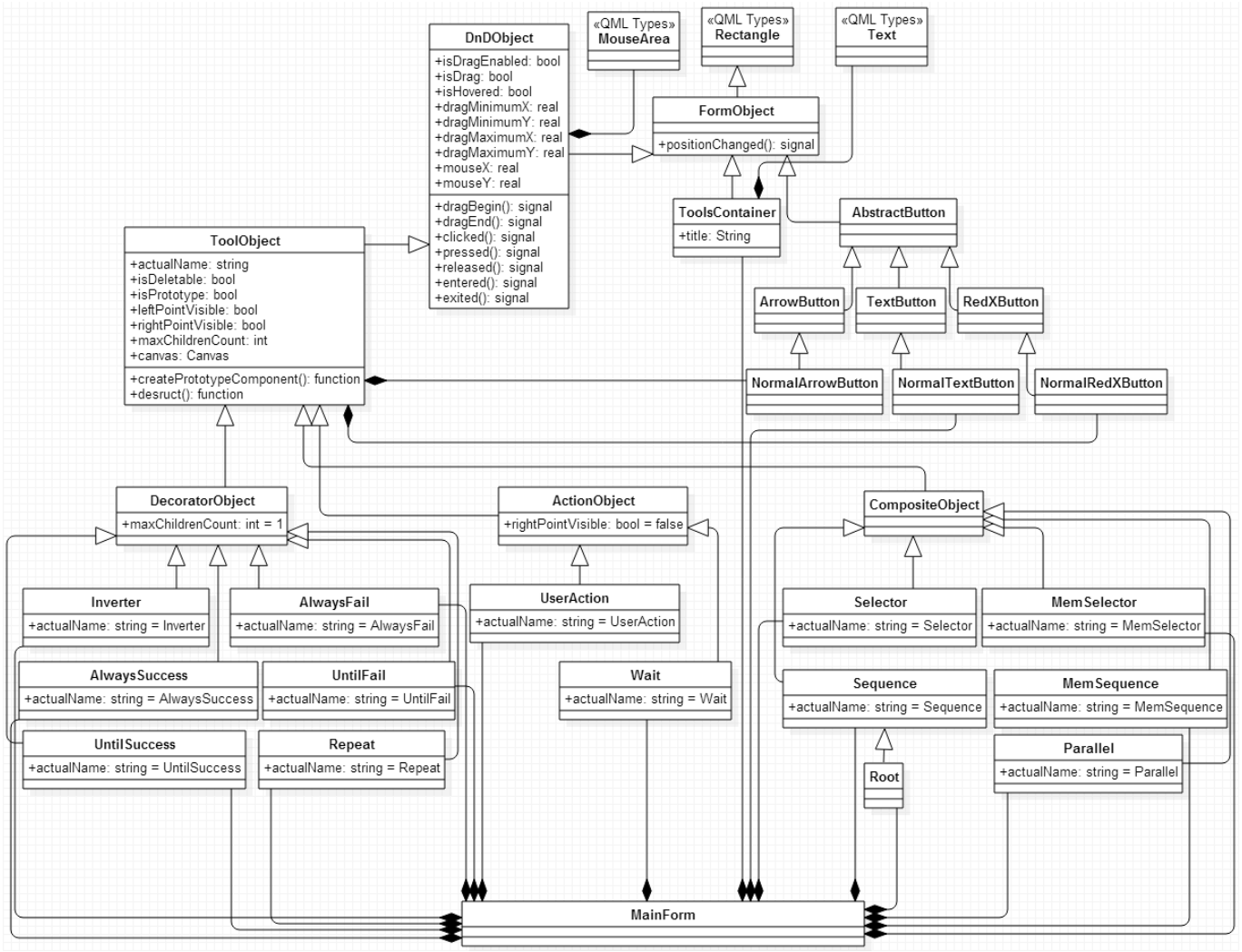

Рисунок 3.1 - UML-диаграмма компонентов QML

3.3 Описание реализованных компонентов

Корнем наследования является компонент FormObject, который наследуется от стандартного компонента Rectangle. Данный компонент содержит единственный сигнал, которого нет в Rectangle – onPositionChanged. Этот сигнал возбуждается при изменении положения объекта.

Компонент DnDObject представляет объекты, которыми можно оперировать при помощи Drag-and-drop способа. Описание компонента DnDObject представлено в таблице 3.1.

Таблица 3.1 – Описание компонента DnDObject

Тип элемента	Наименование	Описание
Поле	isDragEnabled	Bool: true, если манипулирование при помощи Drag-and-drop разрешено
Поле	isDrag	Bool: true, если эелемнт перетаскивается мышью
Поле	isHovered	Bool: true, если мышь находится над элементом
Поле	dragMinimumX	Real: минимальное значение координаты x объекта
Поле	dragMaximumX	Real: максимальное значение координаты x объекта
Поле	dragMinimumY	Real: минимальное значение координаты y объекта
Поле	dragMaximumY	Real: максимальное значение координаты y объекта
Поле	mouseX	Real: значение x-координаты курсора мыши относительно данного объекта
Поле	mouseY	Real: значение y-координаты курсора мыши относительно данного объекта
Сигнал	dragBegin	возбуждается на начало манипулирования при помощи Drag-and-drop
Сигнал	dragEnd	возбуждается после окончания манипулирования при помощи Drag-and-drop
Сигнал	Clicked	возбуждается после клика мышью на объект
Сигнал	Pressed	возбуждается при нажатии мышью на объект
Сигнал	Released	возбуждается после отжатия мыши на объекте
Сигнал	Entered	возбуждается при входе курсора мыши в поле объекта
Сигнал	Exited	возбуждается при выходе курсора мыши за поле

Продолжение таблицы 3.1

Тип элемента	Наименование	Описание
		объекта

Компонент ToolObject представляет объекты вершин дерева поведения, расположенные на левой панели инструментов. Полное описание ToolObject приведено в таблице 3.2.

Таблица 3.2 – Описание компонента ToolObject

Тип элемента	Наименование	Описание
Поле	actualName	String: имя элемента дерева поведения, используется при генерации дерева
Поле	isDeletable	Bool: true, если на объекте появляется кнопка для удаления
Поле	isPrototype	Bool: true, если объект находится на панеле инструментов
Поле	leftPointVisible	Bool: true, если левая кнопка активна в данный момент
Поле	rightPointVisible	Bool: true, если правая кнопка активна в данный момент
Поле	maxChildrenCount	Int: максимальное количество присоединенных объектов
Функция	createPrototype	Возвращает созданный объект ToolObjece
Функция	destruct	Удаляет вершину и связные соединяющие кривые

Компонент CompositeObject является родительским ко всем компонентам, определяющим вершины композиты. На данный момент этот компонент не реализует внутри себя ничего. В дальнейшем он будет определять некоторые общие параметры для всех вершин композитов.

Компонент ActoinObject является родительским ко всем компонентам,

определяющим вершины листы. Так как у всех листов нет дочерних вершин, то в этом компоненте определена переменная rightPointVisible значением false.

Компонент DecoratorObject является родительским ко всем компонентам, определяющим вершины декораторы. Так как вершина декоратор имеет только одну дочернюю вершину, то в данном компоненте определена переменная maxChildrenCount значением 1.

Дочерние компоненты к DecortorObject, ActionObject, CompositeObject определяют лишь переменную actualName соответствующей строкой (Inverter – «Inverter», AlwaysFail – «AlwaysFail», AlwaysSuccess – «AlwaysSuccess» и т.д.), и содержат идентифицирующую иконку или текст, для того, чтобы можно было визуально отличить один компонент от другого.

Компонент ToolsContainer определяет прямоугольник с подписью, который используется как контейнер для вершин различных типов. Подробное описание ToolsContainer приведено в таблице 3.3.

Таблица 3.3 – Описание компонента ToolsContainer

Тип элемента	Наименование	Описание
Поле	Title	String: название контейнера, расположено левом верхнем углу

Компонент AbstractButton служит родительским для всех кнопок и содержит необходимые сигналы. Подробное описание компонента AbstractButton приведено в таблице 3.4.

Таблица 3.4 – Описание компонента AbstractButton

Тип элемента	Наименование	Описание
Поле	isHovered	Bool: true, если мышь находится над элементом
Поле	isPressed	Bool: true, если мышь была нажата, находясь над элементом
Поле	mouseX	Real: значение x-координаты курсора мыши

Продолжение таблицы 3.4

Тип элемента	Наименование	Описание
		относительно данного объекта
Поле	mouseY	Real: значение y-координаты курсора мыши относительно данного объекта
Сигнал	Clicked	возбуждается после клика мышью на объект
Сигнал	Pressed	возбуждается при нажатии мышью на объект
Сигнал	Released	возбуждается после отжатия мыши на объекте
Сигнал	Entered	возбуждается при входе курсора мыши в поле объекта
Сигнал	Exited	возбуждается при выходе курсора мыши за поле объекта
Сигнал	changeMousePosition	возбуждается, при движении мыши над объектом

Компонент ArrowButton представляет объект, который используется в ToolObject как кнопка, в которую могут входить соединительные кривые, и из которой могут выходить соединительные кривые. Подробное описание объекта ArrowButton представлено в таблице 3.5

Таблица 3.5 – Описание компонента ArrowButton

Тип элемента	Наименование	Описание
Сигнал	drawArrow	Возбуждается при перемещении курсора мыши при условии, что была нажата кнопка мыши в поле объекта и не отжата
Сигнал	drawArrowBegin	Возбуждается при первом перемещении курсора мыши при условии, что была нажата кнопка мыши в поле объекта и не отжата

Продолжение таблицы 3.5

Тип элемента	Наименование	Описание
Сигнал	drawArrowEnd	Возбуждается после отжатия кнопки мыши при условии, что был возбужден сигнал drawArrowBegin ранее

Компонент TextButton представляет объекты кнопок, на которых написан по центру текст. Подробное описание TextButton представлено в таблице 3.6.

Таблица 3.6 – Описание компонента TextButton

Тип элемента	Наименование	Описание
Поле	Text	String: текст, написанный на кнопке

Компонент RedXButton представляет объекты кнопок с текстом «X».

Компоненты с приставкой Normal* определяют цвета, используемые на то или иное событие (нажатие кнопки мыши, наведение курсора мыши).

Для реализации соединительных кривых мы написали расширение для QML на Python – BezierCurve. В данном расширении мы переопределили стандартный метод paint для отрисовки кривой Безье. Полное описание данного расширения представлено в таблице 3.7.

Таблица 3.7 – Описание компонента BezierCurve

Поле	Наименование	Описание
Поле	startX	Real: x-координата точки начала кривой
Поле	startY	Real: y-координата точки начала кривой
Поле	endX	Real: x-координата точки конца кривой
Поле	endY	Real: y-координата точки конца кривой
Поле	curveWidth	Int: ширина кривой
Поле	startArrow	Bool: true, если есть стрелочка в начале кривой
Поле	endArrow	Bool: true, если есть стрелочка в конце кривой
Сигнал	startXChnaged	Возбуждается при изменении startX
Сигнал	startYChnaged	Возбуждается при изменении startY

Продолжение таблицы 3.7

Поле	Наименование	Описание
Сигнал	endXChnaged	Возбуждается при изменении endX
Сигнал	endYChnaged	Возбуждается при изменении endY

3.4 Процесс внедрения разработанного ПО

Разработанное программное обеспечение можно внедрить в любое пользовательское приложение, написанное на языке Java 8, способом, описанным на рисунке 3.2.

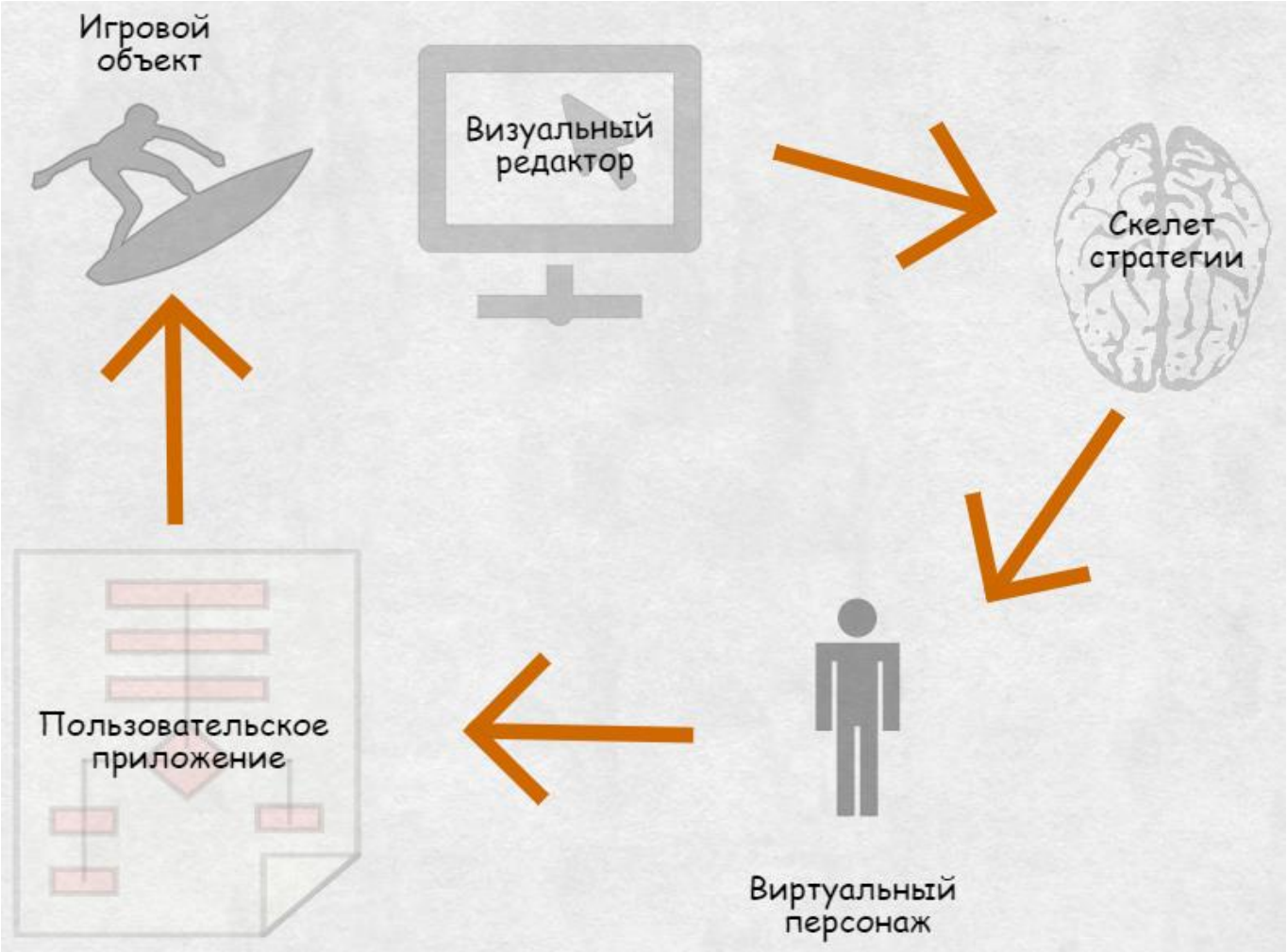

Рисунок 3.2 - Процесс создания игрового объекта и внедрения его в конечное приложение

Сначала выбирается игровой объект, нуждающийся в системе принятия решений. Затем с помощью визуального редактора создается скелет дерева

поведения. Затем этот скелет попадает в конкретный участок кода, где настраивается программный интерфейс и создается объект Blackboard для обмена данными между стратегией, объектом и некоторым игровым миром. Так разработана именно **библиотека** деревьев поведения, то не требуется никакого специального программного интерфейса пользовательского приложения для внедрения стратегий.

3.5 Анализ полученных результатов

Разработанное программное обеспечение ускорит процесс создания игр. Так же данное ПО можно использовать в продуктах, отличных от игровых, например, для проектирования и создания стратегий в робототехнике.

Следующим шагом в развитии данной системы принятия решений виртуальными игровыми персонажами будет создание динамически преобразовывающихся деревьев поведений. Дерево поведения сможет адаптироваться к изменениям в окружающем мире. Так как деревья поведения лучше всего представляют мыслительный процесс в человеческом мозге [14], то данное направление развития выглядит очень актуальным.

Заключение

Таким образом, поставленные задачи были решены. Спроектирована и реализована масштабируемая со стандартизированным программным интерфейсом библиотека для создания деревьев поведения. Спроектирован и реализован визуальный редактор для деревьев поведения, который поддерживает возможность генерации кода дерева поведения с использованием разработанной библиотеки. Так же создан демонстрационный пример, показывающий основные возможности библиотеки.

Список использованных источников

1. Towards a Unified Behavior Trees Framework for Robot Control / A. Marzinotto [и др.] .- Swedish Research Council and the European Union Project, 2013 .- 8 с. [Электронный ресурс] / Режим доступа - http://www.csc.kth.se/~miccol/Michele_Colledanchise/Publications_files/2013_ICRA_mcko.pdf
2. Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees / P. Ogren .- Swedish Defence Research Agency, Stockholm, 2012 .- 8 с. [Электронный ресурс] / Режим доступа - http://www.cas.kth.se/~petter/Publications/ogren2012bt.pdf
3. Behavior Trees for Hierarchical RTS AI / S. Delmer .- Plano: The Guildhall at SMU, 2013.- 10 с. [Электронный ресурс] / Режим доступа - http://www.smu.edu/~/media/Site/guildhall/Documents/Theses/Delmer_Stephan_Thesis_Final.ashx?la=en
4. Unreal Engine 4 Documentation [Электронный ресурс] / Режим доступа- https://docs.unrealengine.com/latest/INT/
5. LibGDX Documentation [Электронный ресурс] / Режим доступа - https://github.com/libgdx/libgdx/wiki
6. Unity3D Documentation [Электронный ресурс] / Режим доступа - http://docs.unity3d.com/Manual/index.html
7. Unity3D site [Электронный ресурс] / Режим доступа – https://unity3d.com
8. Unity3D Asset Store [Электронный ресурс] / Режим доступа - https://www.assetstore.unity3d.com
9. Doxygen [Электронный ресурс]
10. A Universal Unique Identifier (UUID) URN Namesapce [Электронный ресурс] / Режим доступа - https://tools.ietf.org/html/rfc4122
11. Box2D Manual [Электронный ресурс] / Режим доступа - http://box2d.org/manual.pdf
12. QML Documentation [Электронный ресурс] / Режим доступа - http://doc.qt.io/qt-5/qtqml-index.html

13. Phil Thompson Talks About PyQt [Электронный ресурс] / Режим доступа - https://dot.kde.org/2006/08/09/phil-thompson-talks-about-pyqt
14. Об интеллекте / Сандра Блейкси, Джефф Хоккинс [Электронный ресурс] / Режим доступа - http://archism.narod.ru/lib/bleiksli_sandra_ob_intellekte.pdf
15. LWJGL [Электронный ресурс] / Режим доступа -http://www.lwjgl.org/
16. Spore [Электронный ресурс] / Режим доступа -http://www.spore.com/
17. Рынок игр в России и в мире, 2010 – 2016 гг. [Электронный ресурс] / Режим доступа - http://json.tv/ict_telecom_analytics_view/rynok-igr-v-rossii-i-mire-2010-2016-gg-20141121113425

Приложение А

Задание на выполнение бакалаврской работы

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

«Алтайский государственный технический университет им. И. И. Ползунова»

УТВЕРЖДАЮ

Заведующий кафедрой ПМ

_____ Кантор С.А.

подпись *ФИО*

ЗАДАНИЕ № 26

НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

по направлению подготовки 09.03.04 «Программная инженерия»

по профилю Разработка программно-информационных систем

студенту группы Никитину Алексею Александровичу

фамилия, имя, отчество

Тема Проектирование библиотеки и реализация визуального средства создания стратегий поведения виртуальных игровых персонажей

Утверждена приказом ректора от 02.04.15 № Л-1127

Срок выполнения работы 15.06.15

Задание принял к исполнению: _____ Никитин А.А

подпись *ФИО*

Барнаул 2015 г.

1 Исходные данные

Задание на выполнение, техническая документация для Java, Qml, Python, ресурсы internet.

2. Содержание разделов работы

Наименование разделов работы и их содержание	Трудо-ёмкость, %	Срок выполнения	Консультант (Ф.И.О., подпись)
Предметная область	30	17.03.2015	Старолетов С.М.
Проектирование	40	20.04.2015	Старолетов С.М.
Реализация	30	25.05.2015	Старолетов С.М.

3. Научно-библиографический поиск

3.1. По научно-технической литературе просмотреть Реферативные журналы «Кибернетика», «Программное обеспечение» за последние 2 года и научно-технические журналы «Программирование», «Программная инженерия» за последние 2 года.

3.2. По нормативной литературе просмотреть указатели государственных и отраслевых стандартов за последний год.

3.3. Патентный поиск провести за 3 лет по странам Россия

Руководитель работы: Старолетов С.М._____

Ф.И.О. *подпись*

Приложение Б

Руководство пользователя

Визуально программам разделена на три области (рисунок Б.1): область

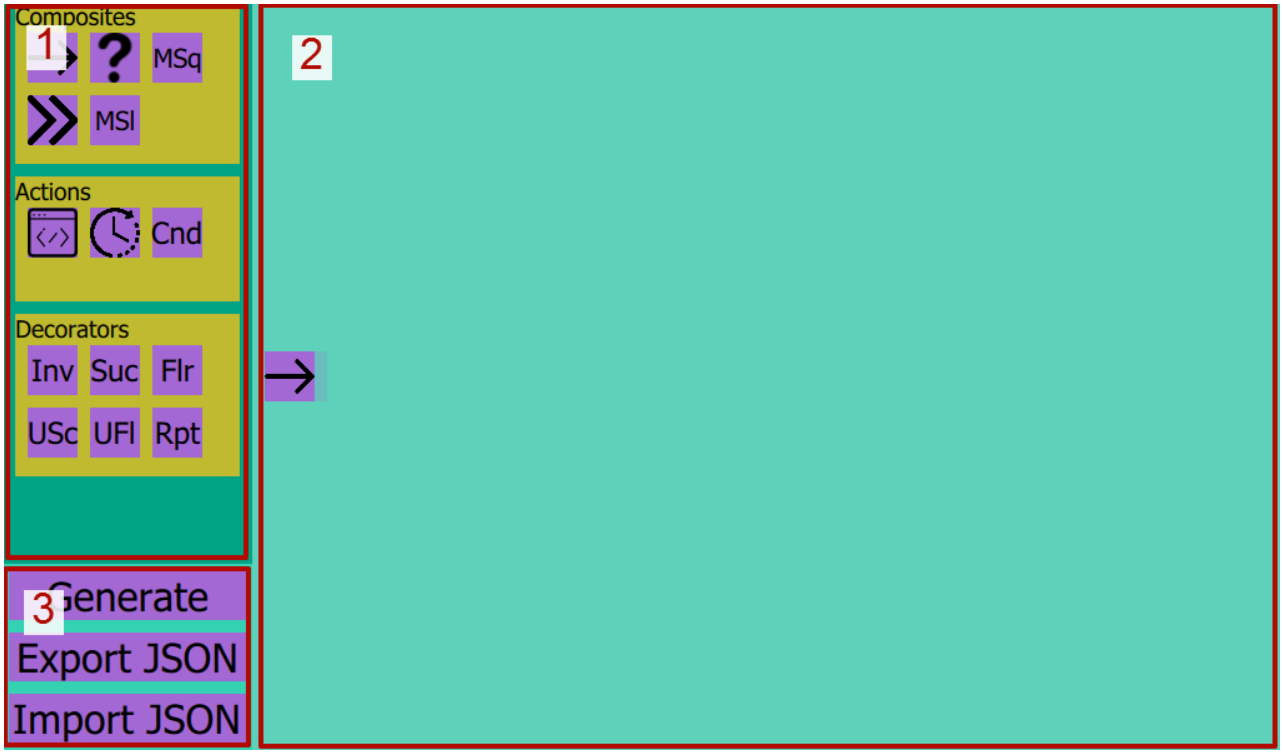

Рисунок Б.1 - Главное окно программы

инструментов – 1, область управления 3 и область построения – 2.

Область инструментов содержит все различные вершины дерева поведения, которые можно с помощью Drag-and-drop способа переносить в область построения (рисунок Б.2).

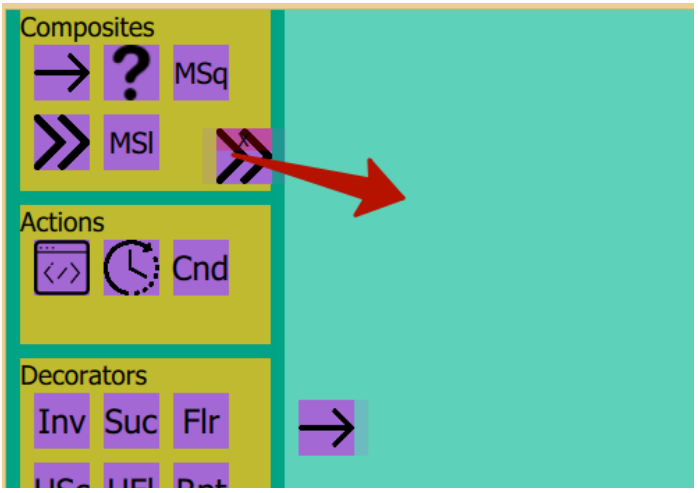

Рисунок Б.2 - Перенос параллельного композита в область построения

В области управления разрешено передвигать вершины методом Drag-and-drop, связывать вершины соединительными кривыми (рисунок Б.3), удалять

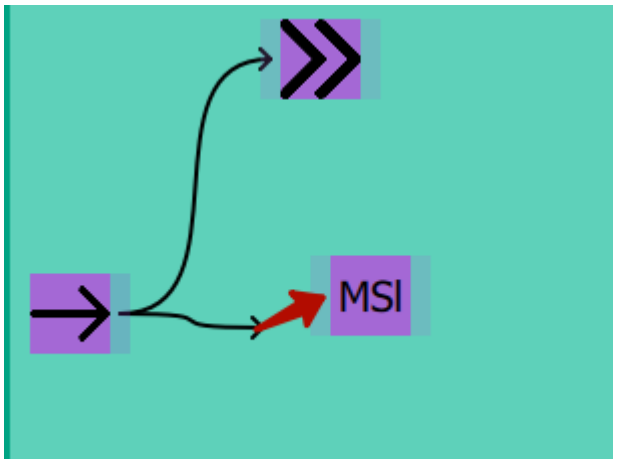

Рисунок Б.3- Связываение композита-последовательности и композита с запоминанием соединительной кривой

соединительные кривые (рисунок Б.4) и удалять вершины (рисунок Б.5).

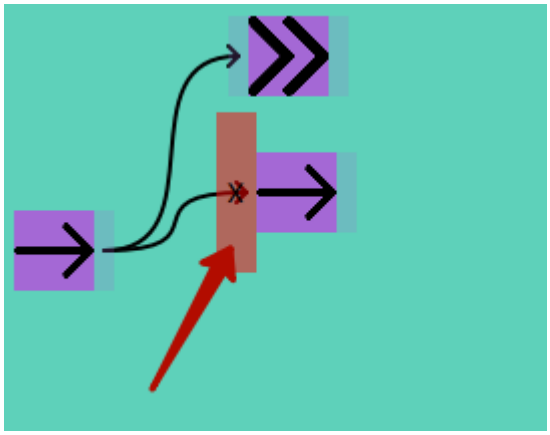

Рисунок Б.4 - Удаление соединительной кривой

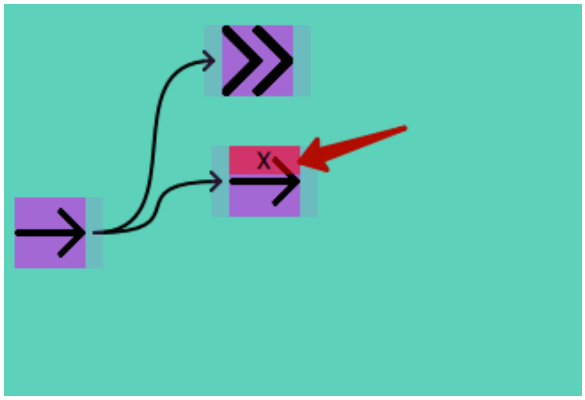

Рисунок Б.5 - Удаление вершины

В области управления находятся три кнопки:

- «Generate» - генерирует Java-код соответствующий дереву поведения в файл, имя которого указывает пользователь. Например, для дерева поведения на рисунке 2.3 сгенерированный файл будет выглядеть следующим образом:

```
import org.nikialeksey.gameengine.ai.behaviortree.*;
import org.nikialeksey.gameengine.ai.behaviortree.Actions.*;
```

```
import org.nikialeksey.gameengine.ai.behaviortree.Composites.*;
import org.nikialeksey.gameengine.ai.behaviortree.Decorators.*;

class BehaviorTreeBuilder {
    public static BehaviorTree buildBehaviorTree() {
        Node root =
            new Sequence(
                new Selector(
                    new AlwaysFailure(
                        new UserAction(
                        )
                    ),
                    new Sequence(
                        new Condition(
                        ),
                        new UserAction(
                        )
                    ),
                    new Sequence(
                        new Condition(
                        ),
                        new Condition(
                        ),
                        new UserAction(
                        )
                    )
                )
            );
        BehaviorTree behaviorTree = new BehaviorTree(root);
        return behaviorTree;
    }
}
```

- «Export JSON» - экспортирует дерево поведения в формат JSON и записывает в файл, имя которого определили пользователь. Например, для дерева поведения на рисунке Б.6 JSON объект будет выглядеть следующим образом:

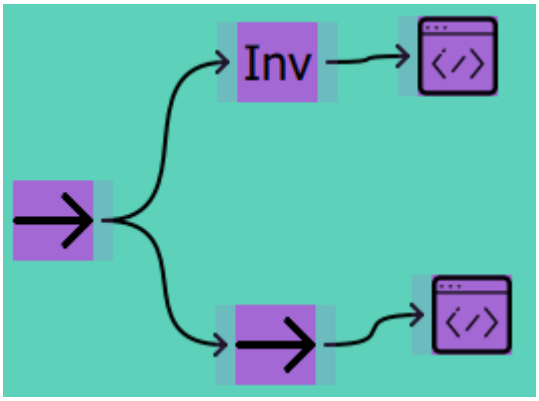

Рисунок Б.6 - Пример дерева повдения

```
{
    'name': 'Sequence',
    'x': 10.0,
    'y': 280.0,
    'children': [
        {
            'name': 'Inverter',
            'x': 122.0,
            'y': 201.0,
            'children': [
                {
                    'name': 'UserAction',
                    'x': 212.0,
                    'y': 198.0,
                    'children': [
                    ],
                },
            ],
        },
        {
            'name': 'Sequence',
            'x': 121.0,
            'y': 342.0,
            'children': [
                {
                    'name': 'UserAction',
                    'x': 219.0,
                    'y': 327.0,
```

```
                'children': [
                ],
            },
        ],
    },
],
}
```

- «Import JSON» загружает дерево поведения из JSON документа, созданного при помощи кнопки «Export JSON».

Приложение В

Документация к библиотеке Behavior Tree

Класс Decorators.AlwaysFailure

Граф наследования Decorators.AlwaysFailure представлен на рисунке Д.1.

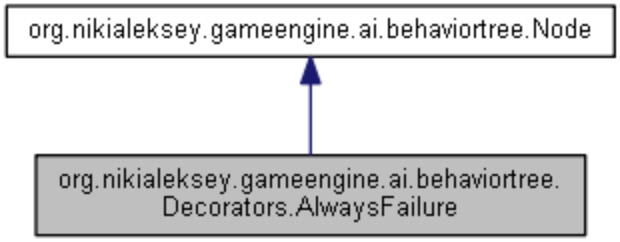

Рисунок Д.1 - Граф наследования Decorators.AlwaysFailure

Граф связей класса Decorators.AlwaysFailure представлен на рисунке Д.2.

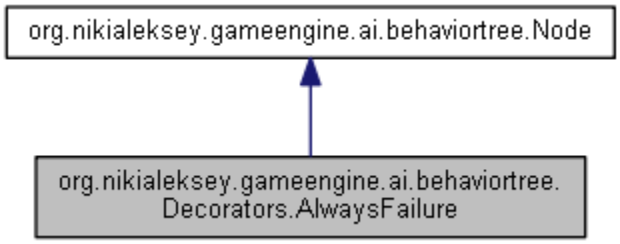

Рисунок Д.2 - Граф связей класса Decorators.AlwaysFailure

Открытые члены

- **AlwaysFailure** (**Node** node)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет вершину декоратор, который после отправки сгнала на исполнение дочерней вершине всегда возвращает статус FAILURE.

Автор:

Alexey Nikitin

Конструктор(ы)

Decorators.AlwaysFailure.AlwaysFailure (Node *node*)

Конструктор.

Аргументы:

- node - дочерняя вершина

Методы

Status Decorators.AlwaysFailure.tick (Tick *tick*)

Передает сигнал на исполнение дочерней вершине, всегда возвращает статус FAILURE

Аргументы:

- tick - объект тика (содержит ссылки на BehaviorTree и Blackboard)

Возвращает:

FAILURE

Граф вызовов представлен на рисунке Д.3:

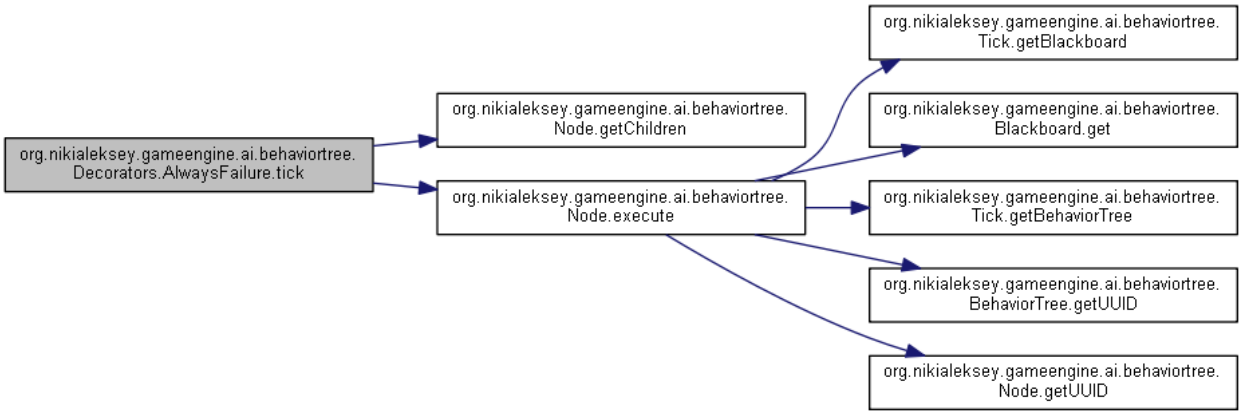

Рисунок Д.3 - Граф вызовов метода AlwaysFailure.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Decorators/AlwaysFailure.java

Класс Decorators.AlwaysSuccess

Граф наследования Decorators.AlwaysSuccess представлен на рисунке Д.4:

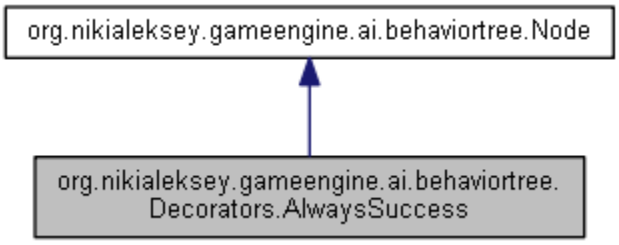

Рисунок Д.4 - Граф наследования Decorators.AlwaysSuccess

Граф связей класса Decorators.AlwaysSuccess представлен на рисунке Д.5:

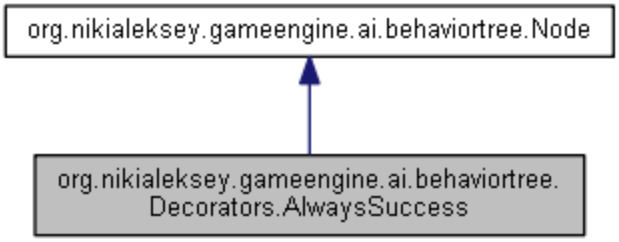

Рисунок Д.5 - Граф связей класса Decorators.AlwaysSuccess

Открытые члены

- **AlwaysSuccess** (**Node** node)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет декоратор, который после отправки сгнала на исполнение дочерней вершине всегда возвращает статус SUCCESS.

Автор:

Alexey Nikitin

Конструктор(ы)

Decorators.AlwaysSuccess.AlwaysSuccess (Node *node*)

Конструктор.

Аргументы:

- node - дочерняя вершина

Методы

Status Decorators.AlwaysSuccess.tick (Tick *tick*)

Передает сигнал на исполнение дочерней вершине, всегда возвращает статус SUCCESS

Аргументы:

- tick - объект тика

Возвращает:

SUCCESS

Граф вызовов представлен на рисунке Д.6:

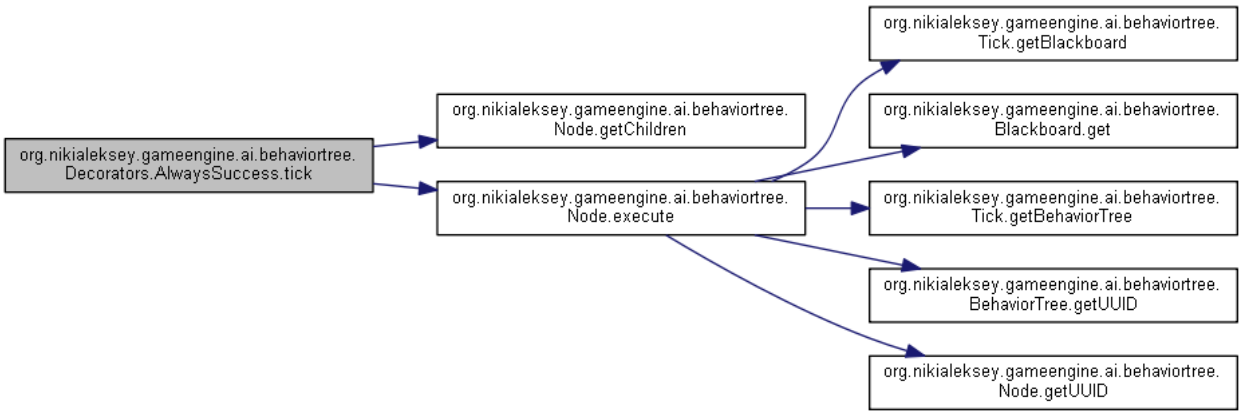

Рисунок Д.6 - Граф вызовов метода AlwaysSuccess.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Decorators/AlwaysSuccess.java

Класс BehaviorTree

Открытые члены

- **BehaviorTree** (**Node** root)
- **Status execute** (**Blackboard** blackboard)
- String **getUUID** ()

Подробное описание

Класс представляет дерево поведения.

Автор:

Alexey Nikitin

Конструктор(ы)

BehaviorTree.BehaviorTree (Node *root*)

Конструктор дерева поведения. Назначает уникальный идентификатор и устанавливает ссылку на корень дерева поведнеия.

Аргументы:

- root - корень дерева поведения, именно этой вершине будут передаваться сигналы на исполнение

Методы

Status BehaviorTree.execute (Blackboard *blackboard*)

Создает объект тика и передает сигнал на исполнение корню дерева поведения

Аргументы:

- blackboard - память для принятия решения

Возвращает:

объект статуса

Граф вызовов представлен на рисунке Д.7:

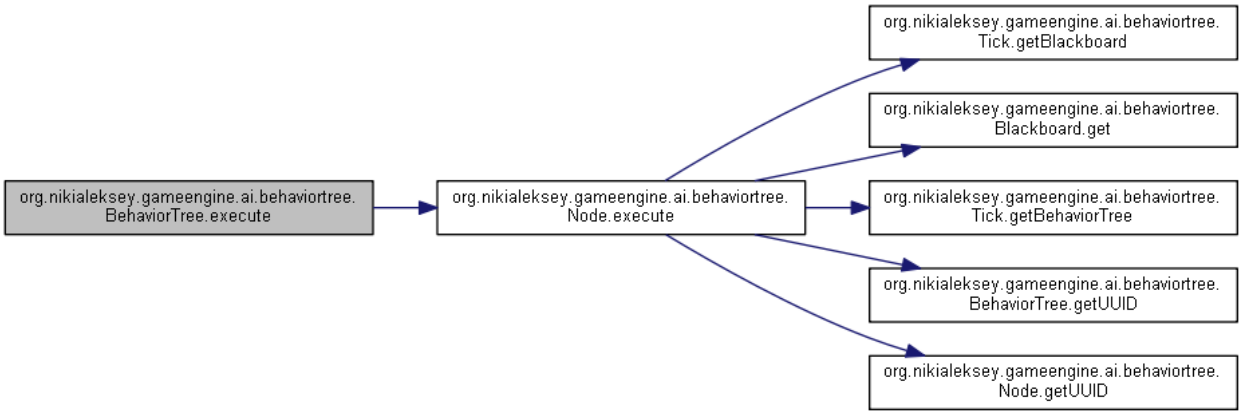

Рисунок Д.7 - Граф вызовов метода BehaviorTree.execute()

String BehaviorTree.getUUID ()

Возвращает уникальный идентификатор дерева поведения.

Возвращает:

строка, представляющая уникальный идентификатор дерева поведения.

Граф вызова представлен на рисунке Д.8:

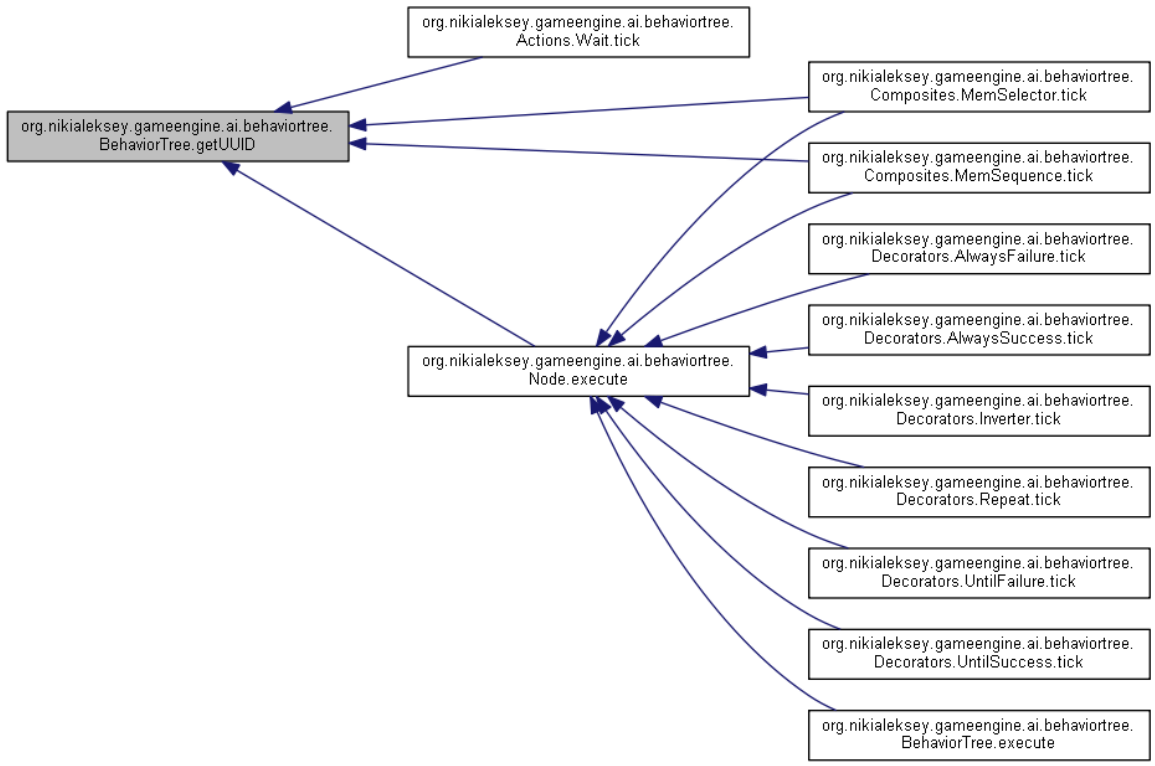

Рисунок Д.8 - Граф вызовов метода BehaviorTree.getUUID()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/BehaviorTree.java

Класс Blackboard

Классы

- class **BlackboardMemory**
- class **TreeMemory**

Открытые члены

- **Blackboard** ()
- Object **get** (String key)
- Object **get** (String key, String treeUUID)
- Object **get** (String key, String treeUUID, String nodeUUID)
- void **put** (String key, Object value)
- void **put** (String key, Object value, String treeUUID)
- void **put** (String key, Object value, String treeUUID, String nodeUUID)

Подробное описание

Класс представляет структуру памяти для наших персонажей, которую будут использовать вершины дерева поведения. Информация, хранящаяся в

```
Blackboard
```

структурирована следующим образом: глобальная информация(доступная из любого места), информация о дереве (досутпная для всех вершин одного дерева), информация о вершине (доступная только вершине).

Память для простототы можно изобразить в виде JSON документа:

```
{    'global key 1': globalObject1,                    #
     'global key 2': globalObject2,                    #  глобальная информация
     ...                                               #
     'tree1UUID': {
         'tree key 1': treeObject1,                    #
         'tree key 2': treeObject2,                    #  информация, доступная для всех
                                                       #  вершин одного дерева
         ...                                           #
         'nodeMemory': {
             'node key 1': nodeObject1,                #
             'node key 2': nodeObject2,                #  информация, доступная для одной вершины
             ...                                       #
         },
     },
     'tree2UUID': {
          ...
     },
     ...
}
```

Автор:

Alexey Nikitin

Конструктор(ы)

Blackboard.Blackboard ()

Конструктор. Инициализирует память.

Методы

Object Blackboard.get (String *key*)

Возвращает объект из глобальной памяти.

Аргументы:

- key - ключ

Возвращает:

объект из глобальной памяти

Граф вызова функции представлен на рисунке Д.9:

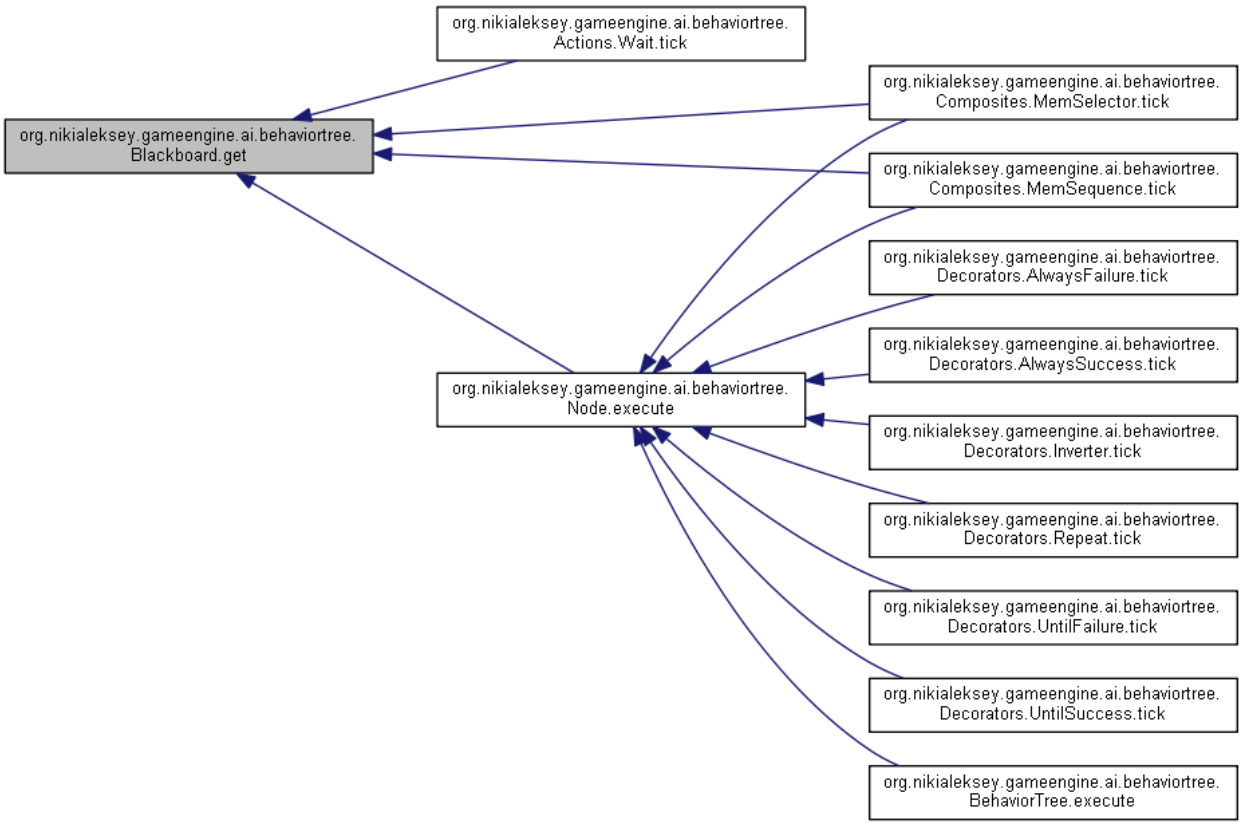

Рисунок Д.9 - Граф вызова метода Blackboard.get()

Object Blackboard.get (String *key*, String *treeUUID*)

Возвращает объект из памяти дерева с идентификатором

`treeUUID`

.

Аргументы:

- key – ключ
- treeUUID - уникальный идентификатор дерева

Возвращает:

объект из памяти дерева с идентификатором `treeUUID`.

Object Blackboard.get (String *key*, String *treeUUID*, String *nodeUUID*)

Возвращает объект из вершины дерева.

Аргументы:

- key – ключ
- treeUUID - уникальный идентификатор дерева
- nodeUUID - уникальный идентификатор вершины

Возвращает:

объект из вершины дерева

void Blackboard.put (String *key*, Object *value*)

Кладет объект в глобальную память.

Аргументы:

- key – ключ
- value - объект

Граф вызова функции представлен на рисунке Д.10:

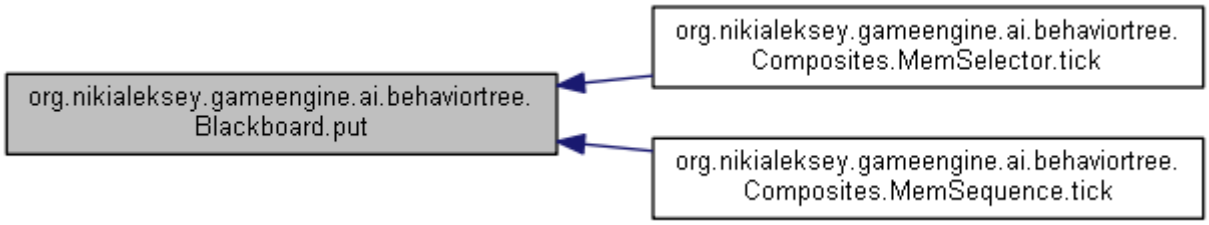

Рисунок Д.10 - Граф вызова метода Blackboard.put()

void Blackboard.put (String *key*, Object *value*, String *treeUUID*)

Кладет объект в память для дерева.

Аргументы:

- key – ключ
- value – объект
- treeUUID - уникальный идентификатор дерева

void Blackboard.put (String *key*, Object *value*, String *treeUUID*, String *nodeUUID*)

Кладет объект в память вершины.

Аргументы:

- key – ключ
- value – объект
- treeUUID - уникальный идентификатор дерева
- nodeUUID - уникальный идентификатор вершины

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Blackboard.java

Класс Decorators.Inverter

Граф наследования Decorators.Inverter представлен на рисунке Д.11:

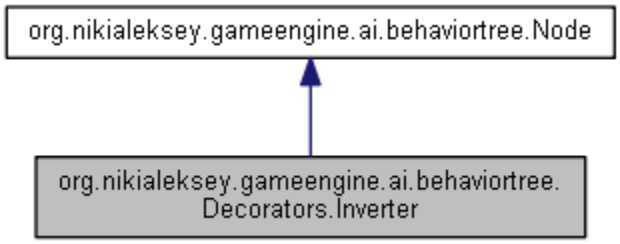

Рисунок Д.11 - Граф наследования Decorators.Inverter

Граф связей класса Decorators.Inverter представлен на рисунке Д.12:

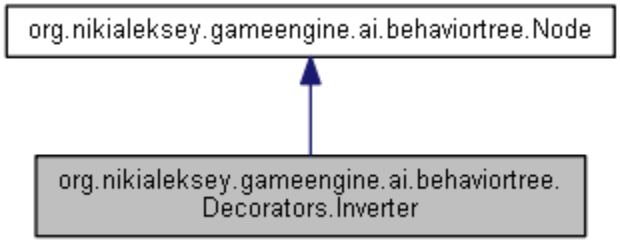

Рисунок Д.12 - Граф связей класса Decorators.Inverter

Открытые члены

- **Inverter** (**Node** node)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Представляет декоратор инвертер.

Автор:

Alexey Nikitin

Конструктор(ы)

Decorators.Inverter.Inverter (Node *node*)

Конструктор.

Аргументы:

- node – дочерняя вершина

Методы

Status Decorators.Inverter.tick (Tick *tick*)

Передает сигнал на исполнение дочерней вершине.

Аргументы:

- tick – объект тика

Возвращает:

SUCCESS, если дочерняя вершина вернула результат FAILURE; FAILURE, если дочерняя вершина вернула результат SUCCESS; иначе тот результат, который вернула дочерняя вершина.

Граф вызовов представлен на рисунке Д.13:

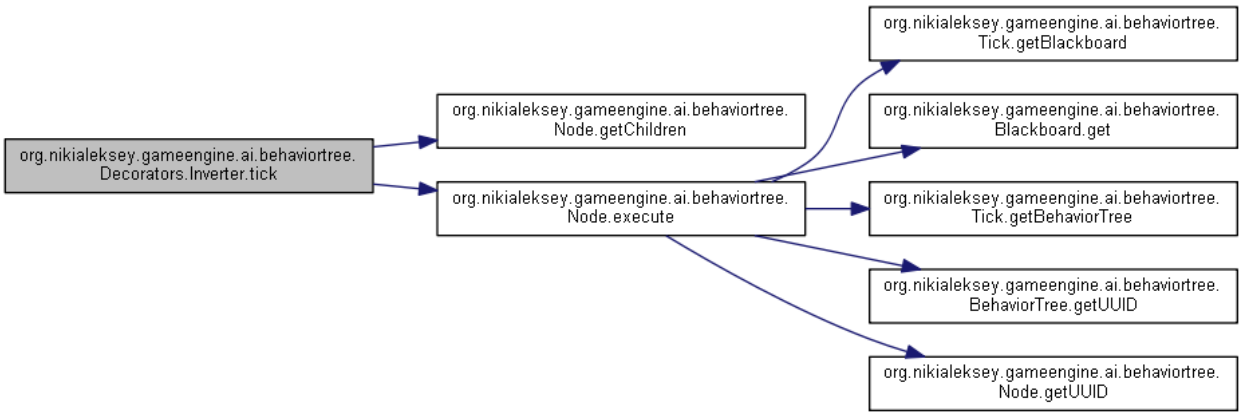

Рисунок Д.13 - Граф вызова метода Inverter.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Decorators/Inverter.java

Класс Composites.MemSelector

Граф наследования Composites.MemSelector представлен на рисунке Д.14:

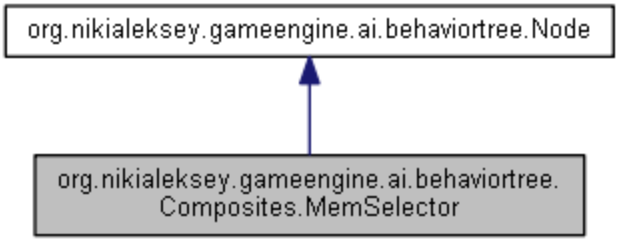

Рисунок Д.14 - Граф наследования Composites.MemSelector

Граф связей класса Composites.MemSelector представлен на рисунке Д.15:

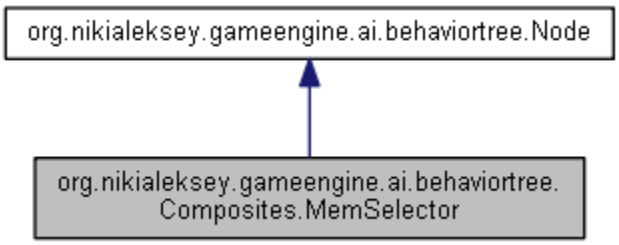

Рисунок Д.15 - Граф связей класса Composites.MemSelector

Открытые члены

- **MemSelector** (Node...nodes)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет композит-селектор с запоминанием вершины, которая вернула результат RUNNING. На следующем тике данная вершина начнет давать сигнал на исполнение как раз последней запущенной дочерней вершне, а не с начала спика дочерних вершин.

Автор:

Alexey Nikitin

Конструктор(ы)

Composites.MemSelector.MemSelector (Node... *nodes*)

Конструктор.

Аргументы:

- nodes – список дочерних вершин

Методы

Status Composites.MemSelector.tick (Tick *tick*)

Передает сигнал на исполнение дочерним вершинам до тех пор, пока они возвращают статус FAILURE. Как только дочерняя вершина вернет статус, отличный от FAILURE, этот статус будет сразу возвращен этой вершиной и выполнение закончится. Если это был статус RUNNING, то в таком случае будет запомнен в blackboard индекс данной дочерней вершины и в следующий раз передача сигнала на исполнение продолжится именно с этой вершины.

Аргументы:

- tick – объект тика

Возвращает:

либо статус дочерней вершины, которая вернула результат, отличный от FAILURE, либо FAILURE

Граф вызовов представлен на рисунке Д.16:

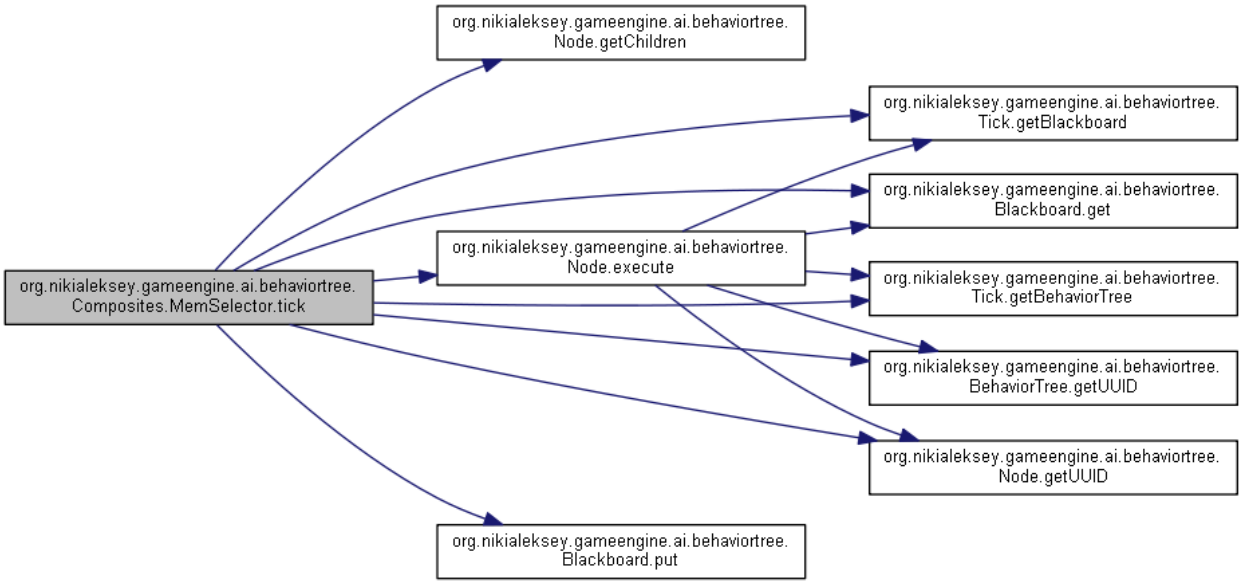

Рисунок Д.16 - Граф вызовов метода MemSelector.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Composites/MemSelector.java

Класс Composites.MemSequence

Граф наследования: Composites.MemSequence представлен на рисунке Д.17:

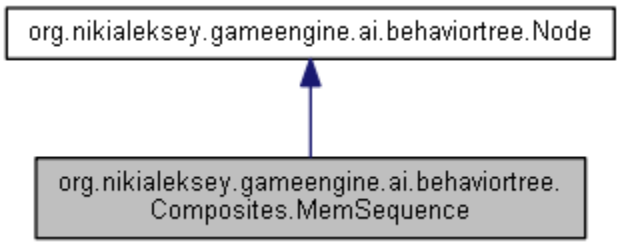

Рисунок Д.17 - Граф наследования Composites.MemSequence

Граф связей класса Composites.MemSequence представлен на рисунке Д.18:

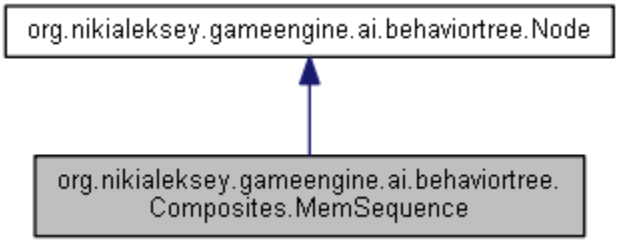

Рисунок Д.18 - Граф связей класса Composites.MemSequence

Открытые члены

- **MemSequence** (Node...nodes)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет композит-последовательность с запоминанием вершины, которая вернула результат RUNNING. На следующем тике данная вершина начнет давать сигнал на исполнение как раз последней запущенной дочерней вершне, а не с начала спика дочерних вершин.

Автор:

Alexey Nikitin

Конструктор(ы)

Composites.MemSequence.MemSequence (Node... *nodes*)

Конструктор.

Аргументы:

- nodes – список дочерних вершин

Методы

Status Composites.MemSequence.tick (Tick *tick*)

Передает сигнал на исполнение дочерним вершинам до тех пор, пока они возвращают статус SUCCESS. Как только дочерняя вершина вернет статус, отличный от SUCCESS, этот статус будет сразу возращен этой вершиной и выполнение закончится. Если это был статус RUNNING, то в таком случае будет запомнен в blackboard индекс данной дочерней вершины и в следующий раз передача сигнала на исполнение продолжится именно с этой вершины.

Аргументы:

- tick – объект тика

Возвращает:

либо статус дочерней вершины, которая вернула результат, отличный от SUCCESS, либо SUCCESS

Граф вызовов представлен на рисунке Д.19:

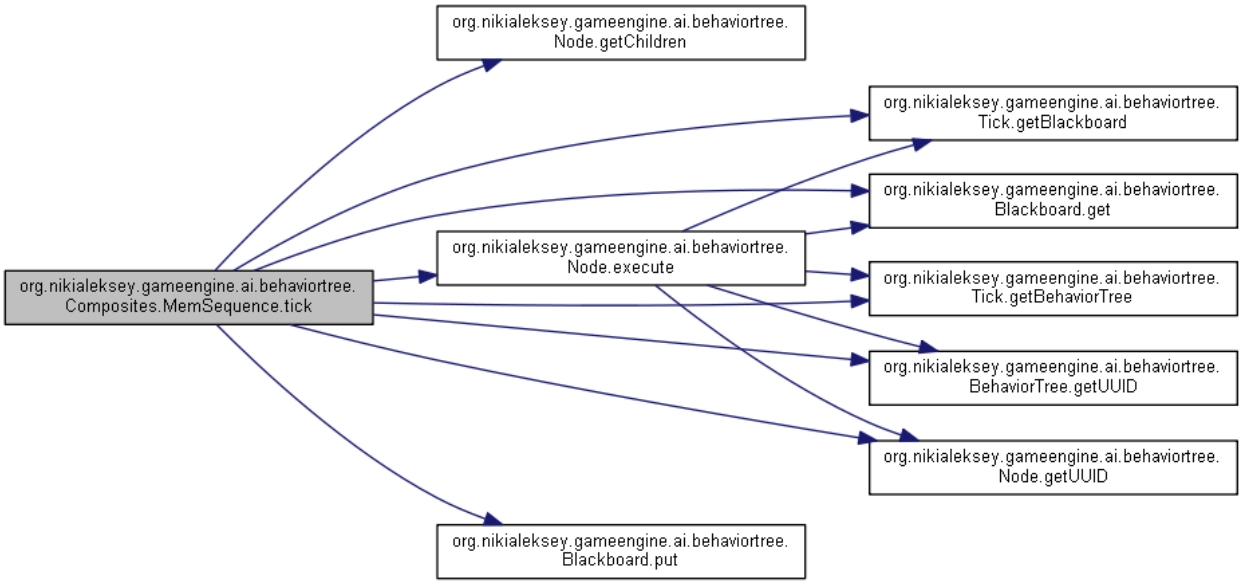

Рисунок Д.19 - Граф вызовов метода MemSequnce.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Composites/MemSequence.java

Класс Node

Граф наследования Node представлен на рисунке Д.20:

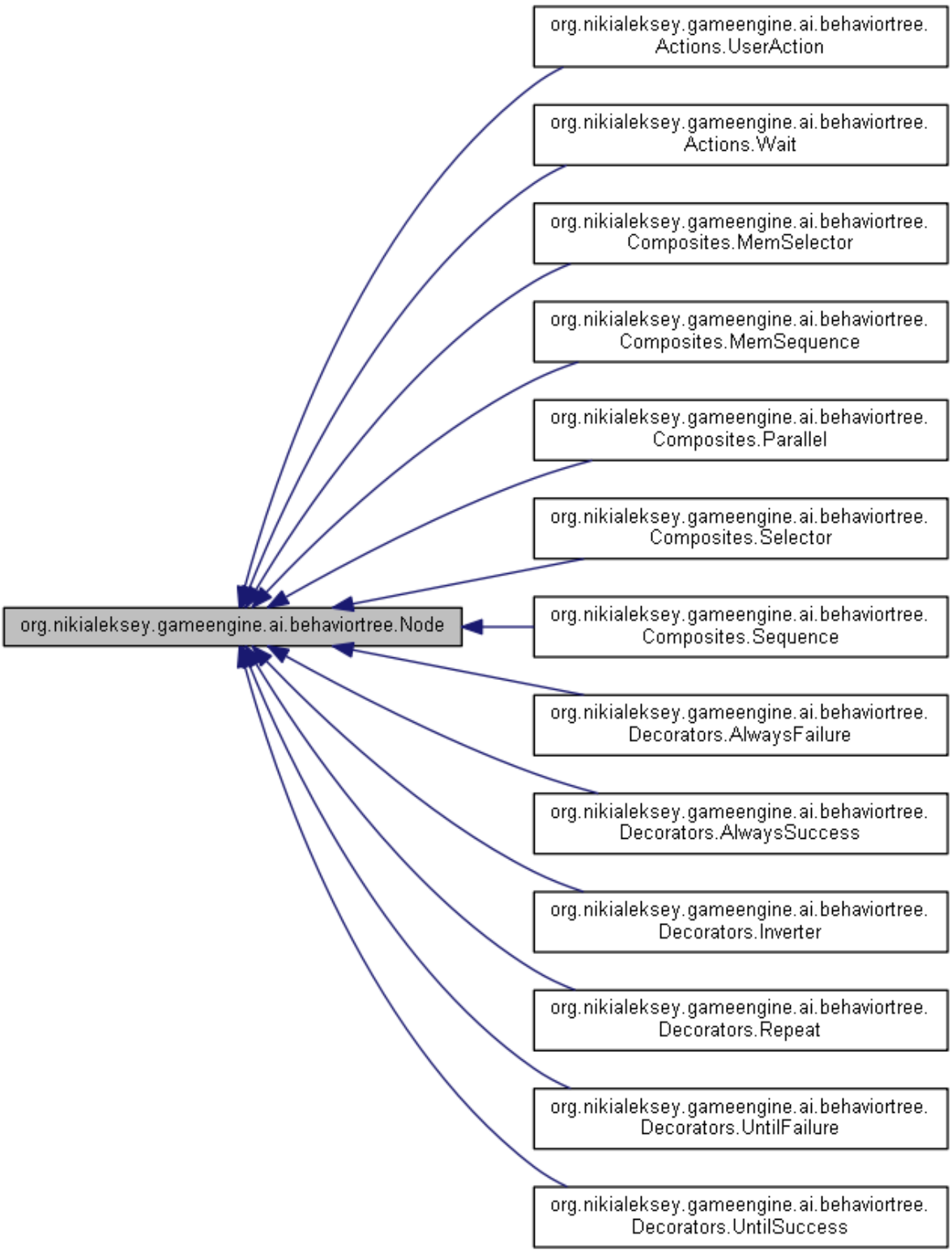

Рисунок Д.20 - Граф наследования Node

Открытые члены

- **Node** (Node...nodes)
- ArrayList< **Node** > **getChildren** ()

- String **getUUID** ()
- **Status execute** (**Tick tick**)
- abstract void **enter** (**Tick tick**)
- abstract void **open** (**Tick tick**)
- abstract **Status tick** (**Tick** tick)
- abstract void **close** (**Tick tick**)
- abstract void **exit** (**Tick tick**)

Подробное описание

Абстрактный класс вершины дерева повдения. Содержит необходимые методы для выполнения логики вершины.

Автор:

Alexey Nikitin

Конструктор(ы)

Node.Node (Node... *nodes*)

Конструктор.

Аргументы:

- nodes – дочерние вершины

Методы

abstract void Node.close (Tick *tick*) `[abstract]`

Вызывается, при осуществлении закрытия вершины (не вызывается после выполнении логики, если статус

```
RUNNING
```

)

Аргументы:

- tick – объект тика

abstract void Node.enter (Tick *tick*) `[abstract]`

Вызывается, при осуществлении входа в вершину

Аргументы:

- tick – объект тика

Status Node.execute (Tick *tick*)

Выполняет логику вершины.

Аргументы:

- tick – объект тика

Возвращает:

статус, после выполнения логики

Граф вызовов представлен на рисунке Д.21:

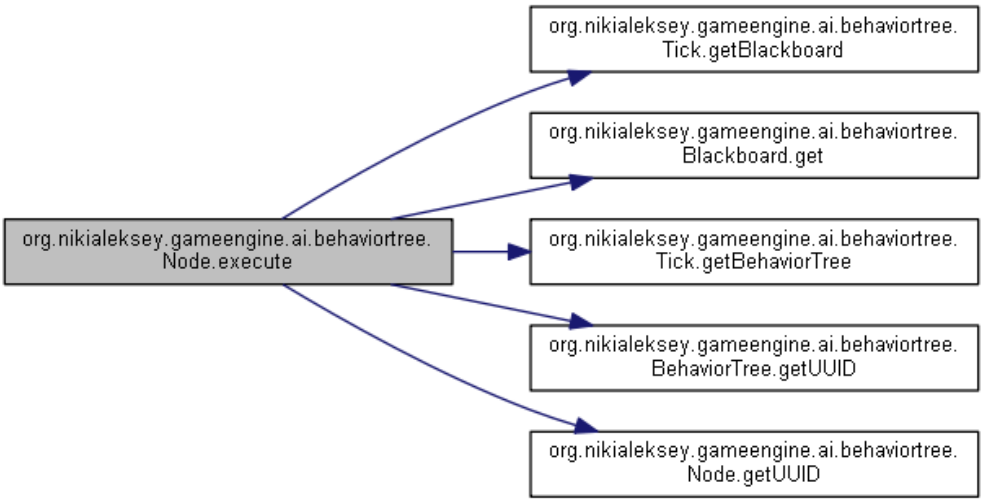

Рисунок Д.21 - Граф вызовов метода Node.execute()

Граф вызова функции представлен на рисунке Д.22:

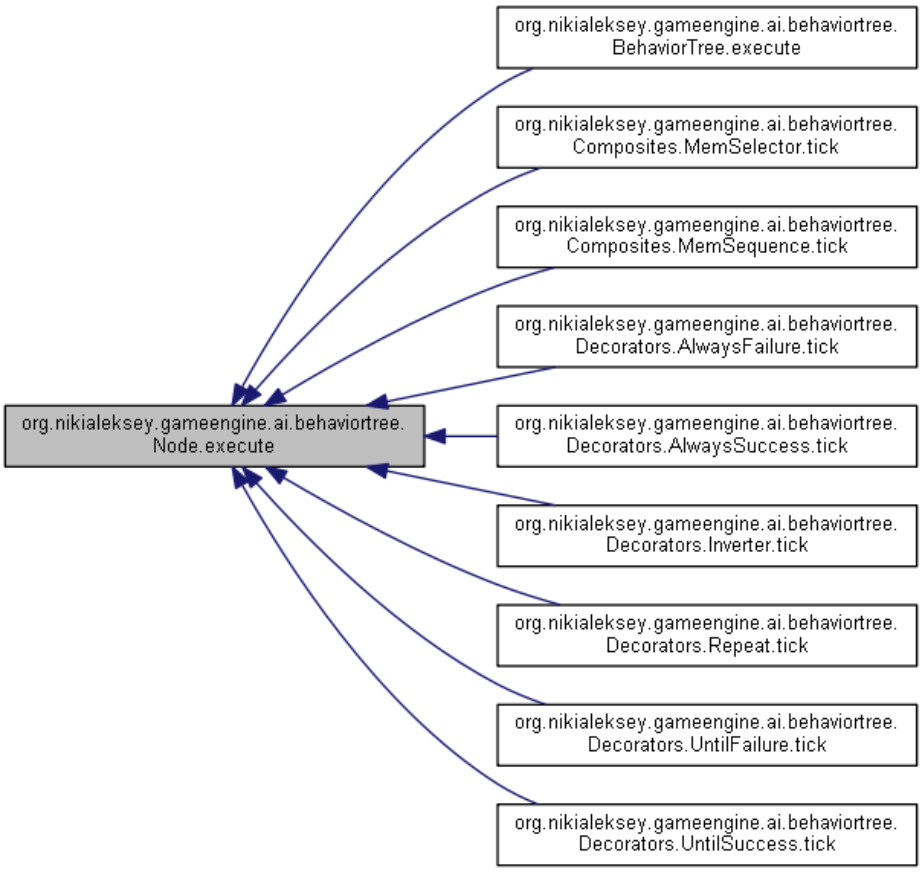

Рисунок Д.22 - Граф вызова метода Node.execute

abstract void Node.exit (Tick *tick*) `[abstract]`

Вызывается, при осуществлении выхода из вершины (всегда после выполнения логики, но позже закрытия вершины, если оно было)

Аргументы:

- tick – объект тика

ArrayList<Node> Node.getChildren ()

Возвращает список дочерних вершин

Возвращает:

список дочерних вершин

Граф вызова функции представлен на рисунке Д.23:

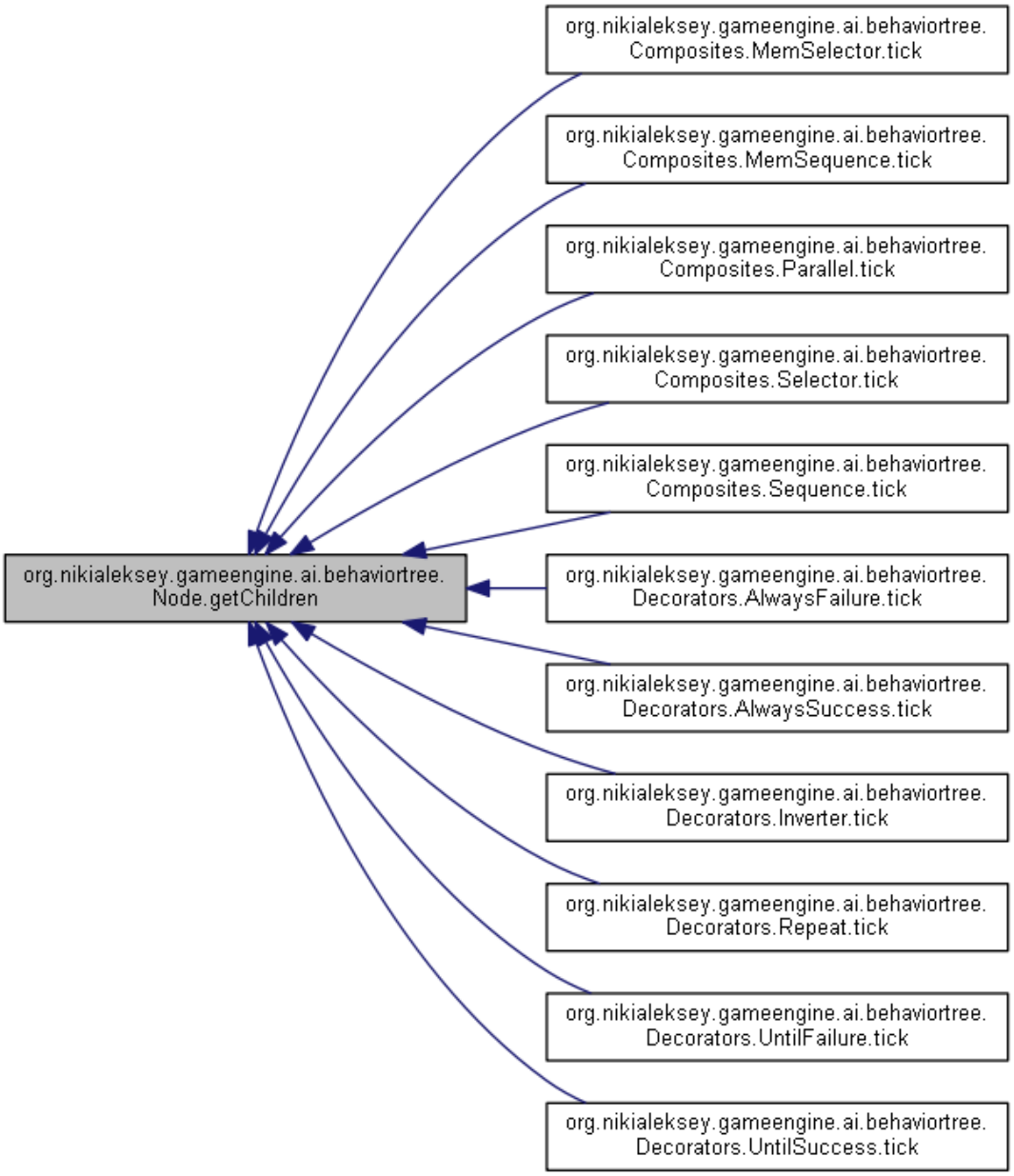

Рисунок Д.23 - Граф вызова метода Node.getChildren()

String Node.getUUID ()

Возвращает уникальный идентификатор вершины

Возвращает:

уникальный идентификатор вершины

Граф вызова функции представлен на рисунке Д.24:

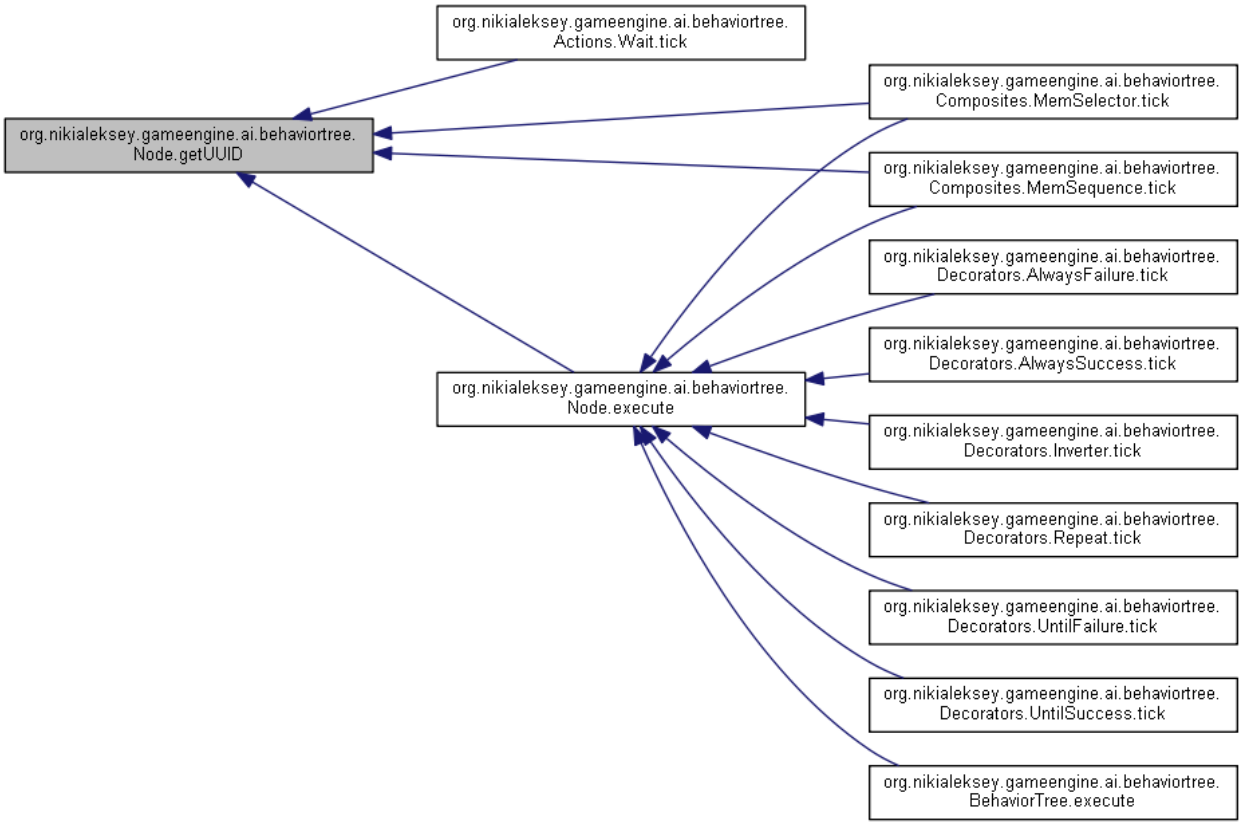

Рисунок Д.24 - Граф вызова метода Node.getUUID()

abstract void Node.open (Tick *tick*) `[abstract]`

Вызывается, при осуществлении открытия вершины (всегда после входа, но не всегда открытая вершина закрывается после выполнения логики)

Аргументы:

- tick – объект тика

abstract Status Node.tick (Tick *tick*) `[abstract]`

Содержит код логики вершины, после отработки возвращает один из четырех статусов: RUNNING, WAIT, FAILURE, SUCCESS

Аргументы:

- tick – объект тика

Возвращает:

один из четырех статусов: RUNNING, WAIT, FAILURE, SUCCESS

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Node.java

Класс Composites.Parallel

Граф наследования Composites.Parallel представлен на рисунке Д.25:

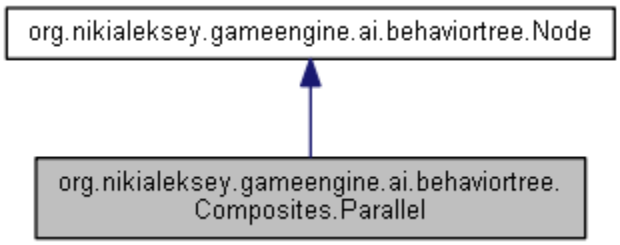

Рисунок Д.25 - Граф наследования Composites.Parallel

Граф связей класса Composites.Parallel представлен на рисунке Д.26:

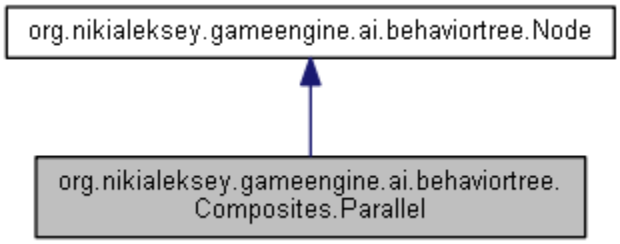

Рисунок Д.26 - Граф связей класса Composites.Parallel

Открытые члены

- **Parallel** (int successCount, int failureCount, Node...nodes)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет параллельный композит.

Автор:

Alexey Nikitin

Конструктор(ы)

Composites.Parallel.Parallel (int *successCount*, int *failureCount*, Node... *nodes*)

Конструктор.

Аргументы:

- successCount - количество дочерних вершин, которое должно вернуть SUCCESS, чтобы данный композит вернул SUCCESS
- failureCount - количество дочерних вершин, которое должно вернуть FAILURE, чтобы данный композит вернул FAILURE
- nodes - список дочеирних вершин

Методы

Status Composites.Parallel.tick (Tick *tick*)

Передает сигнал на исполнение всем дочерним вершинам одновременно.

Аргументы:

- tick – объект тика

Возвращает:

если количество дочерних вершин, вернувших результат SUCCESS больше порогового значение `successCount` и, при этом, количество вершин, вернувших результат FAILURE меньше порогового значения `failureCount` , то SUCCESS; если количество дочерних вершин, вернувших результат FAILURE больше порогового значение `failureCount` и, при этом, количество вершин, вернувших результат SUCCESS меньше порогового значения `successCount` , то FAILURE; иначе ERROR

Граф вызовов представлен на рисунке Д.27:

Рисунок Д.27 - Граф вызовов метода Parallel.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Composites/Parallel.java

Класс Decorators.Repeat

Граф наследования Decorators.Repeat представлен на рисунке Д.28:

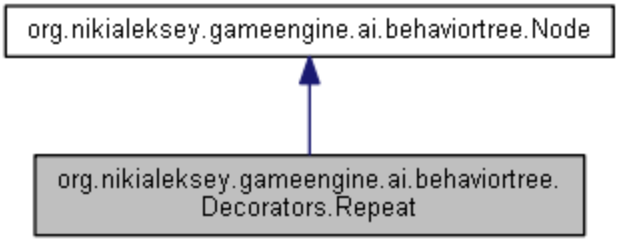

Рисунок Д.28 - Граф наследования Decorators.Repeat

Граф связей класса Decorators.Repeat представлен на рисунке Д.29:

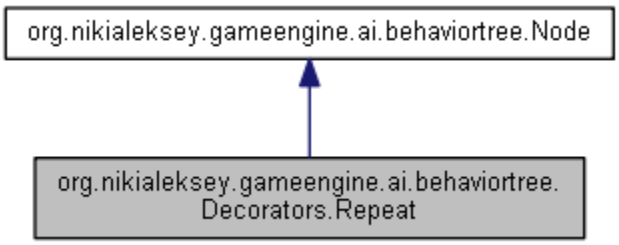

Рисунок Д.30 - Граф связей класса Decorators.Repeat

Открытые члены

- **Repeat** (int repeatCount, **Node** node)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет декоратор, который передает сигнал на исполнение дочерней вершине заданное количество раз.

Автор:

Alexey Nikitin

Конструктор(ы)

Decorators.Repeat.Repeat (int *repeatCount*, Node *node*)

Конструктор.

Аргументы:

- repeatCount - количество передач сигнала на исполнение дочерней вершине
- node - дочерняя вершина

Методы

Status Decorators.Repeat.tick (Tick *tick*)

Передает сигнал на исполнение дочерней вершине заданное число раз.

Аргументы:

- tick – объект тика

Возвращает:

ERROR, если дочерней вершины нет; последний статус, который вернула дочерняя вершина, если количество повторений было не нулевое; если количество повторений было нулевое, то SUCCESS.

Граф вызовов представлен на рисунке Д.31:

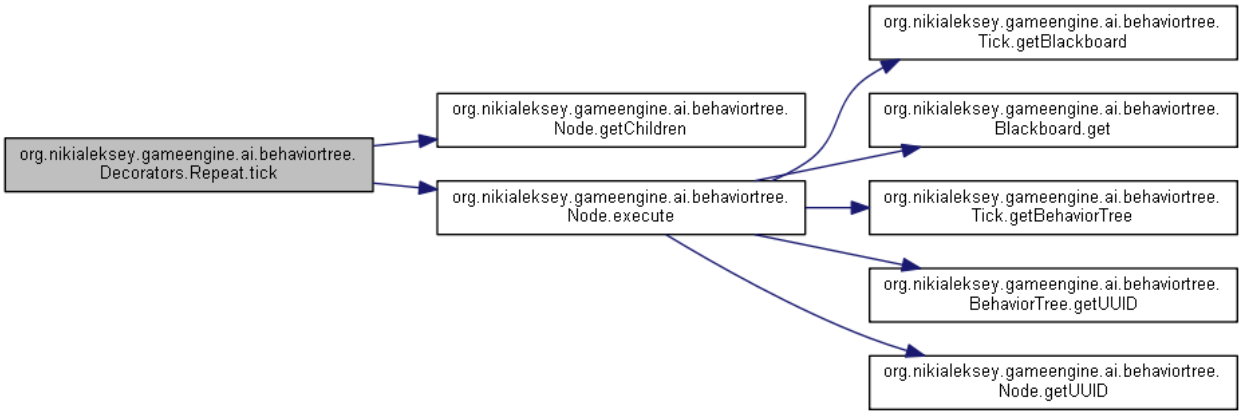

Рисунок Д.31 - Граф вызовов метода Repeat.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Decorators/Repeat.java

Класс Composites.Selector

Граф наследования Composites.Selector представлен на рисунке Д.32:

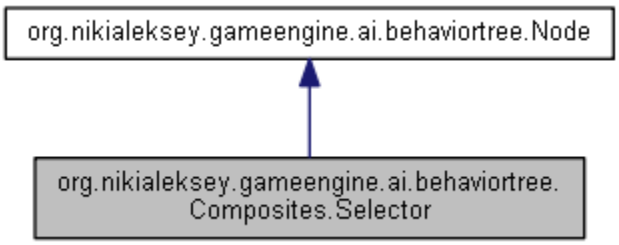

Рисунок Д.32 - Граф наследования Composites.Selector

Граф связей класса Composites.Selector представлен на рисунке Д.33:

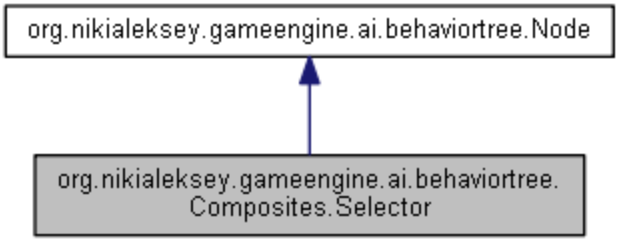

Рисунок Д.33 - Граф связей класса Composites.Selector

Открытые члены

- **Selector** (Node...nodes)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет композит-селектор. Выполняет все свои дочерние вершины по порядку, до тех пор, пока они возвращают результат FAILURE

Автор:

Alexey Nikitin

Конструктор(ы)

Composites.Selector.Selector (Node... *nodes*)

Конструктор.

Аргументы:

- nodes - список дочеирних вершин

Методы

Status Composites.Selector.tick (Tick *tick*)

Передает сигнал на исполнение всем дочерним вершинам, до тех пор пока они возвращают FAILURE

Аргументы:

- tick – объект тика

Возвращает:

либо статус той вершины, которая вернула результат, отличный от FAILURE, либо FAILURE

Граф вызовов представлен на рисунке Д.34:

Рисунок Д.34 - Граф вызовов метода Selector.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Composites/Selector.java

Класс Composites.Sequence

Граф наследования Composites.Sequence представлен на рисунке Д.35:

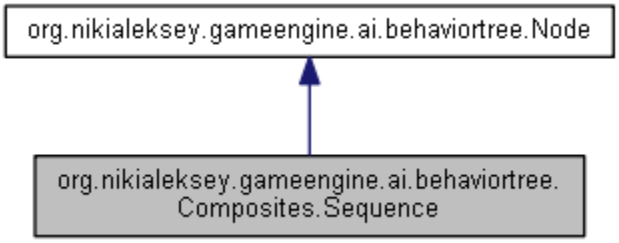

Рисунок Д.35 - Граф наследования Composites.Sequence

Граф связей класса Composites.Sequence представлен на рисунке Д.36:

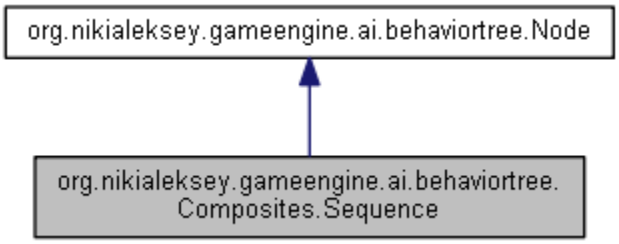

Рисунок Д.36 - Граф связей класса Composites.Sequence

Открытые члены

- **Sequence** (Node...nodes)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет композит-последовательность. Выполняет всех своих детей до тех пор, пока они возвращают SUCCESS

Автор:

Alexey Nikitin

Конструктор(ы)

Composites.Sequence.Sequence (Node... *nodes*)

Конструктор.

Аргументы:

- nodes - список дочеирних вершин

Методы

Status Composites.Sequence.tick (Tick *tick*)

Передает сигнал на исполнение всем своим дочерним вершинам до тех пор, пока они возвращают SUCCESS

Аргументы:

- tick – объект тика

Возвращает:

либо статус дочерней вершины, которая вернула результат, отличный от SUCCESS, либо SUCCESS

Граф вызовов представлен на рисунке Д.37:

Рисунок Д.37 - Граф вызовов метода Sequence.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Composites/Sequence.java

Status Ссылки на перечисление

Открытые атрибуты

- **RUNNING**
- **ERROR**
- **FAILURE**
- **SUCCESS**

Подробное описание

Класс представляет результаты, которые возвращают вершины после исполнения логики.

Автор:

Alexey Nikitin

Данные класса

Status.ERROR

Ошибка. Результат возвращается, когда произошла ошибка при исполнении логики вершины.

Status.FAILURE

Не успешно. Сигнал возвращается, когда исполнение вершины завершилось не успешно.

Status.RUNNING

Запущено. Результат возвращается, когда вершина не завершила действие.

Status.SUCCESS

Успешно Сигнал возвращается, когда исполнение вершины завершилось успешно.

Документация для этого перечисления сгенерерирована из файла:

- src/org/nikialeksey/gameengine/ai/behaviortree/Status.java

Класс Tick

Открытые члены

- **Tick** (**BehaviorTree** behaviorTree, **Blackboard** blackboard)
- **Blackboard getBlackboard** ()
- **BehaviorTree getBehaviorTree** ()
- void **enterNode** (**Node** node)
- void **openNode** (**Node** node)
- void **tickNode** (**Node** node)
- void **closeNode** (**Node** node)
- void **exitNode** (**Node** node)

Подробное описание

Класс содержит ссылки на blackboard и behaviorTree. Используется, когда сигнал на исполнение поступает в дерево поведения и пробрасывается дочерним вершинам, чтобы дочерние вершины могли пользоваться blackboard'ом.

Так же удобно здесь писать отладочный код.

Автор:

Alexey Nikitin

Конструктор(ы)

Tick.Tick (BehaviorTree *behaviorTree*, Blackboard *blackboard*)

Конструктор. Сохранает ссылки на объект дерева поведения и на общую память.

Аргументы:

- behaviorTree - ссылка на дерево поведения, в котором создали объект Tick
- blackboard - ссылка на общую память

Методы

void Tick.closeNode (Node *node*)

Исполняется перед закрытием вершины.

Аргументы:

- node - закрываемая вершина

void Tick.enterNode (Node *node*)

Исполняется при входе в вершину.

Аргументы:

- node - вершина, в которую осуществился вход

void Tick.exitNode (Node *node*)

Исполняется перед выходом из вершины.

Аргументы:

- node - вершина, из которой осуществляется выход

BehaviorTree Tick.getBehaviorTree ()

Возвращает ссылку на дерево поведения.

Возвращает:

ссылку на дерево поведения.

Граф вызова функции представлен на рисунке Д.38:

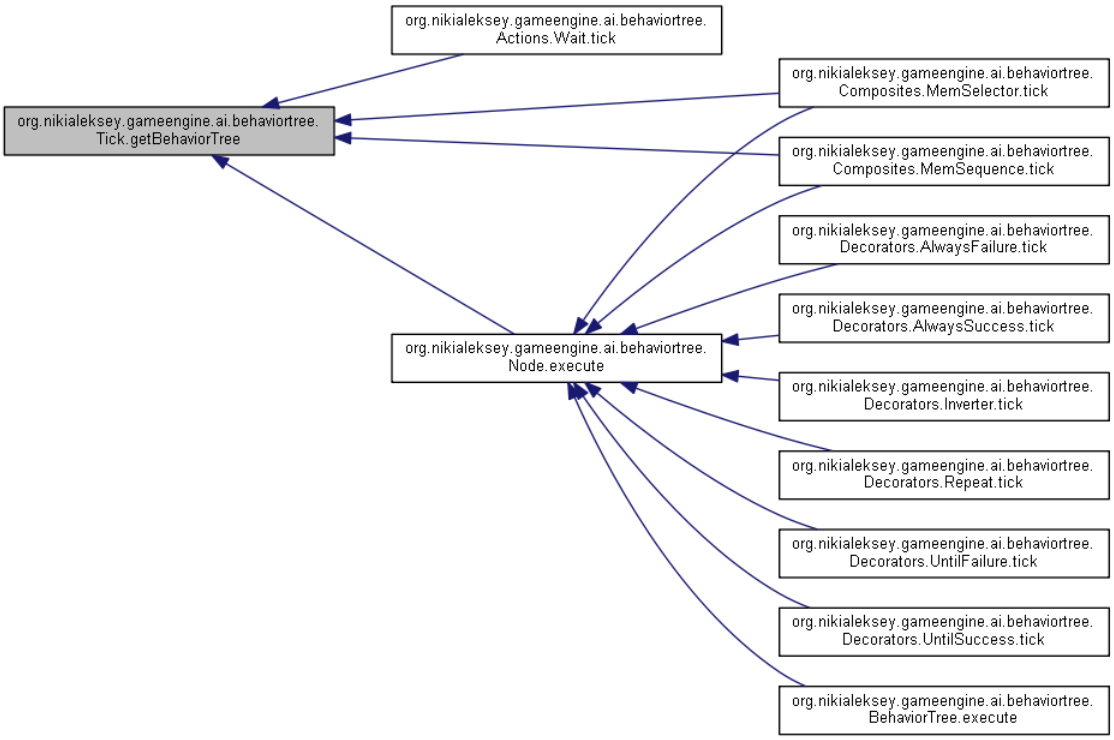

Рисунок Д.38 - Граф вызова метода Tick.getBehaviorTree()

Blackboard Tick.getBlackboard ()

Возвращает ссылку на объект общей памяти.

Возвращает:

ссылку на объект общей памяти.

Граф вызова функции представлен на рисунке Д.39:

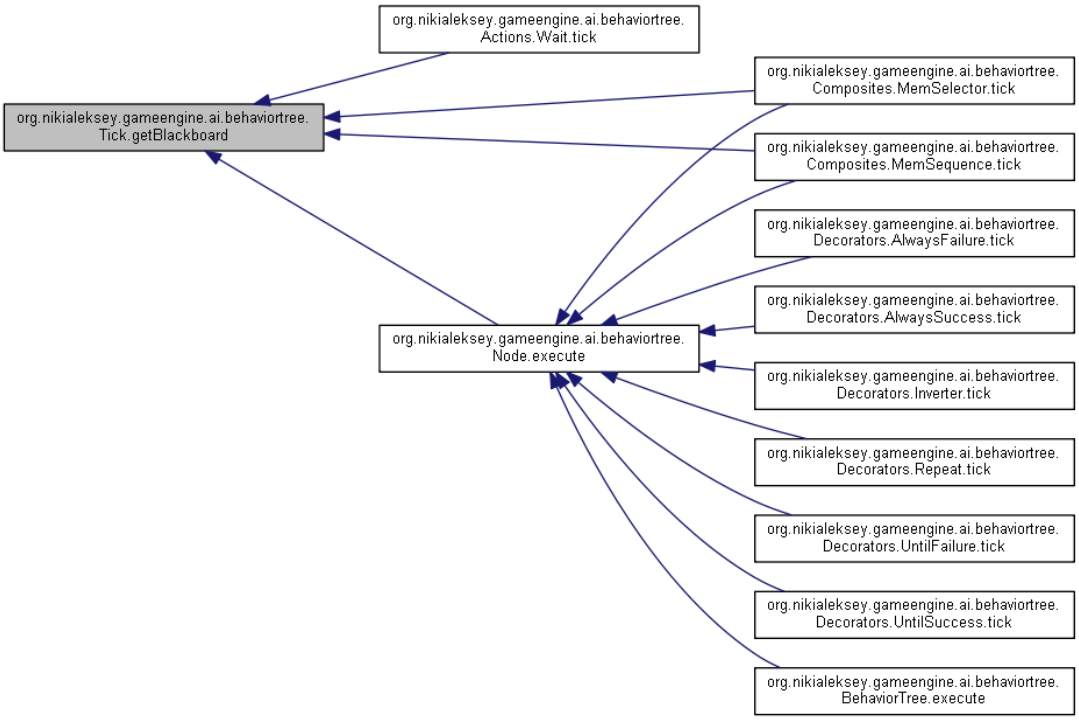

Рисунок Д.39 - Граф вызова метода Tick.getBlackboard()

void Tick.openNode (Node *node*)

Исполняется при открытии вершины.

Аргументы:

- node - открываемая вершина

void Tick.tickNode (Node *node*)

Исполняется перед выполнением логики вершины.

Аргументы:

- node - вершина, в которую зашли, возможно открыли, и еще не началось выполнение логики

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Tick.java

Класс Decorators.UntilFailure

Граф наследования Decorators.UntilFailure представлен на рисунке Д.40:

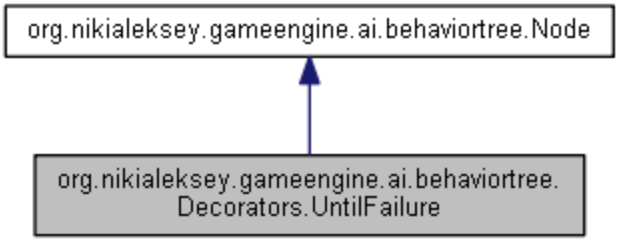

Рисунок Д.40 - Граф наследования Decorators.UntilFailure

Граф связей класса Decorators.UntilFailure представлен на рисунке Д.41:

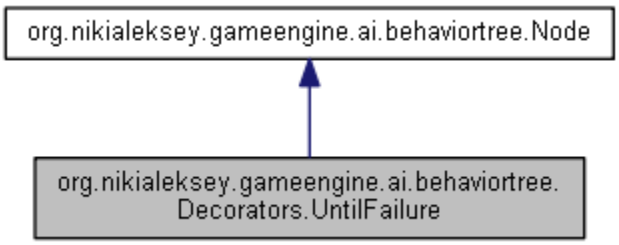

Рисунок Д.41 - Граф связей класса Decorators.UntilFailure

Открытые члены

- **UntilFailure** (**Node** node)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет декоратор, который передает сигнал на исполнение дочерней вершине до тех пор, пока она не вернет статус FAILURE

Автор:

Alexey Nikitin

Конструктор(ы)

Decorators.UntilFailure.UntilFailure (Node *node*)

Конструктор.

Аргументы:

- *node* - дочерняя вершина

Методы

Status Decorators.UntilFailure.tick (Tick *tick*)

Передает сигнал на исполнение дочерней вершине до тех пор, пока она возвращает статус, отличный от FAILURE.

Аргументы:

- tick - объект тика

Возвращает:

ERROR, если нет дочерней вершины, FAILURE иначе.

Граф вызовов представлен на рисунке Д.42:

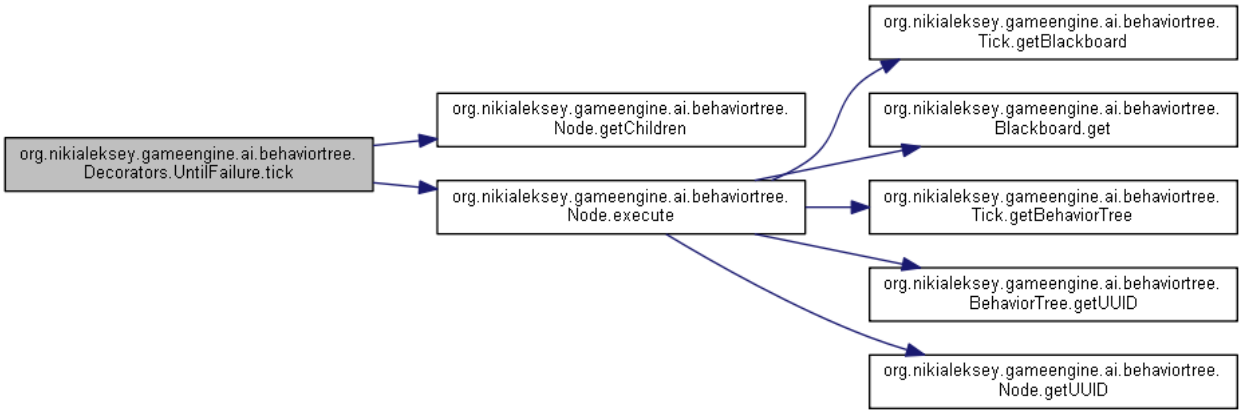

Рисунок Д.42 - Граф вызовов метода UntilFailure.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Decorators/UntilFailure.java

Класс Decorators.UntilSuccess

Граф наследования Decorators.UntilSuccess представлен на рисунке Д.43:

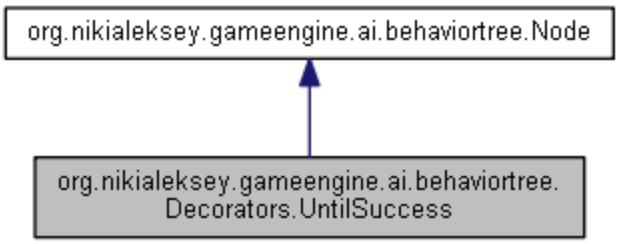

Рисунок Д.43 - Граф наследования Decorators.UntilSuccess

Граф связей класса Decorators.UntilSuccess представлен на рисунке Д.44:

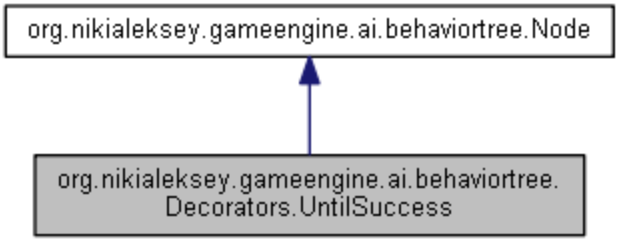

Рисунок Д.44 - Граф связей класса Decorators.UntilSuccess

Открытые члены

- **UntilSuccess** (**Node** node)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет декоратор, который передает сигнал на исполнение дочерней вершине до тех пор, пока она не вернет статус SUCCESS

Автор:

Alexey Nikitin

Конструктор(ы)

Decorators.UntilSuccess.UntilSuccess (Node *node*)

Конструктор.

Аргументы:

- node - дочерняя вершина

Методы

Status Decorators.UntilSuccess.tick (Tick *tick*)

Передает сигнал на исполнение дочерней вершине до тех пор, пока она возвращает статус, отличный от SUCCESS.

Аргументы:

- tick - объект тика

Возвращает:

ERROR, если нет дочерней вершины, SUCCESS иначе.

Граф вызовов представлен на рисунке Д.45:

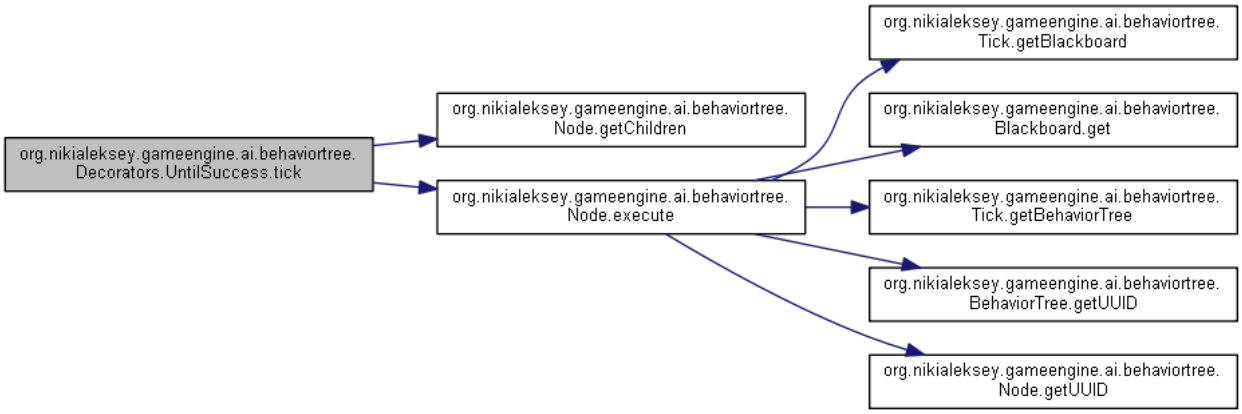

Рисунок Д.45 - Граф вызовов метода UntilSuccess.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Decorators/UntilSuccess.java

Класс Actions.UserAction

Граф наследования Actions.UserAction представлен на рисунке Д.46:

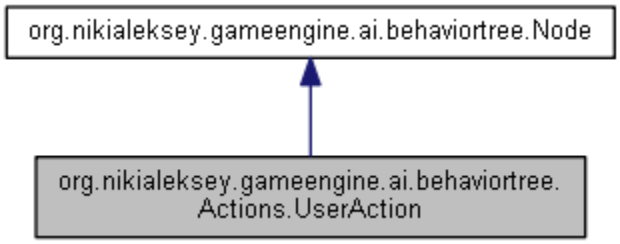

Рисунок Д.46 - Граф наследования Actions.UserAction

Граф связей класса Actions.UserAction представлен на рисунке Д.47:

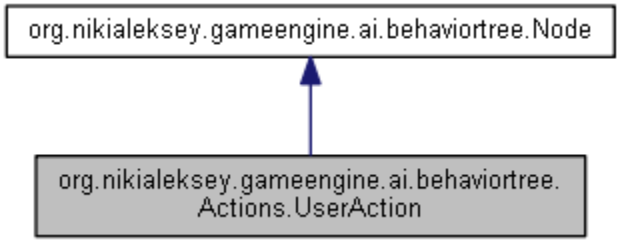

Рисунок Д.47 - Граф связей класса Actions.UserAction

Открытые члены

- **UserAction** (Consumer< **Tick** > action)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)

- void **exit** (**Tick tick**)

Подробное описание

Класс представляет лист-действие, определяемое пользователем.

Автор:

Alexey Nikitin

Конструктор(ы)

Actions.UserAction.UserAction (Consumer< Tick > *action*)

Конструктор.

Аргументы:

- action - дейстие, определяемое пользователем

Методы

Status Actions.UserAction.tick (Tick *tick*)

Выполняет пользовательское действие и возвращает SUCCESS.

Аргументы:

- tick - объект тика

Возвращает:

SUCCESS

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Actions/UserAction.java

Класс Actions.Wait

Граф наследования Actions.Wait представлен на рисунке Д.48:

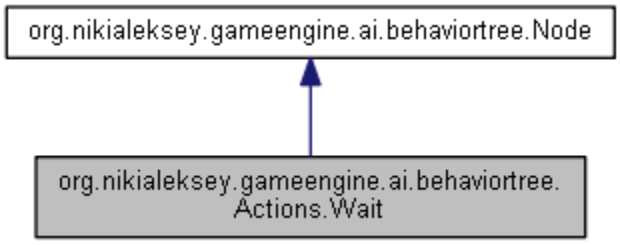

Рисунок Д.48 - Граф наследования Actions.Wait

Граф связей класса Actions.Wait представлен на рисунке Д.49:

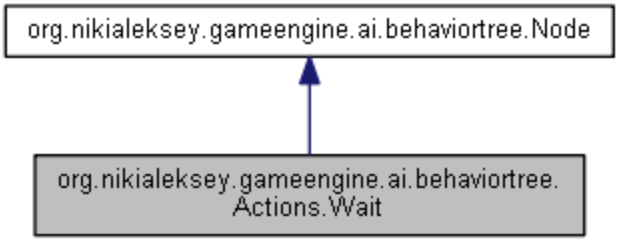

Рисунок Д.49 - Граф связей класса Actions.Wait

Открытые члены

- **Wait** (long milliseconds)
- void **enter** (**Tick tick**)
- void **open** (**Tick tick**)
- **Status tick** (**Tick** tick)
- void **close** (**Tick tick**)
- void **exit** (**Tick tick**)

Подробное описание

Класс представляет лист-действие "ожидание".

Автор:

Alexey Nikitin

Конструктор(ы)

Actions.Wait.Wait (long *milliseconds*)

Конструктор.

Аргументы:

- *milliseconds* - количество миллисекунд, которое необходимо подождать

Методы

Status Actions.Wait.tick (Tick *tick*)

Проверяет, сколько времени прошло с первого вызова и если прошло времени больше, чем заданное количество, то вернет SUCCESS.

Аргументы:

- tick - объект тика

Возвращает:

ERROR, если время первого запуска не было записано; RUNNING, если время ожидания еще не прошло; SUCCESS, если время ожидания превышает или совпает с заданным значением.

Граф вызовов представлен на рисунке Д.50:

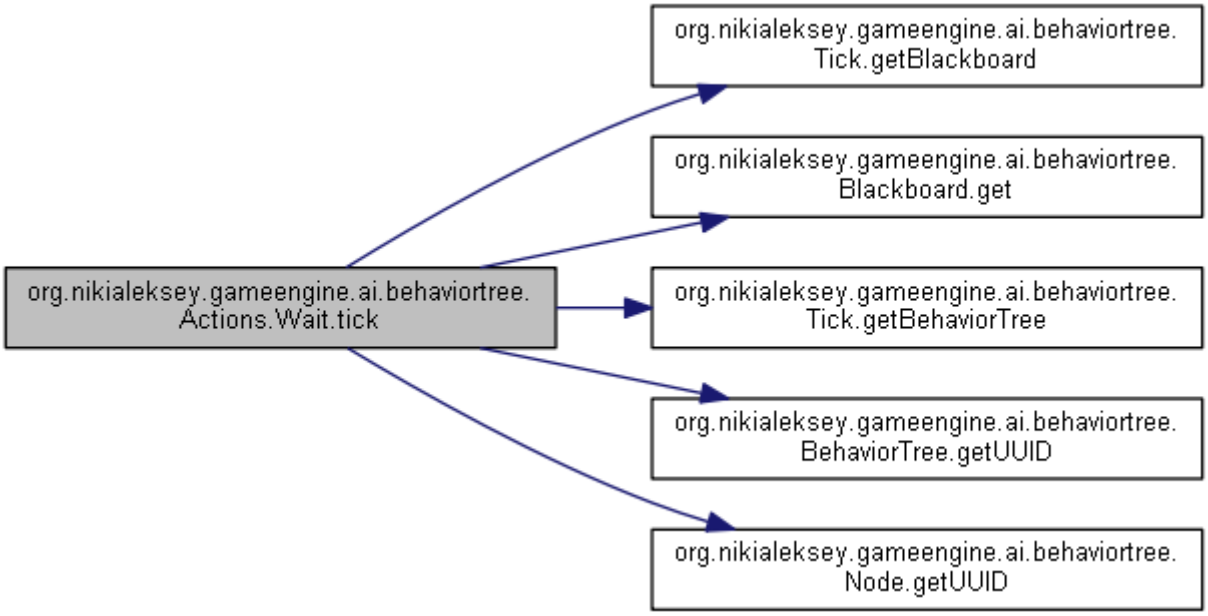

Рисунок Д.50 - Граф вызовов метода Wait.tick()

Объявления и описания членов класса находятся в файле:

- src/org/nikialeksey/gameengine/ai/behaviortree/Actions/Wait.java

Приложение Г

Исходный код программы

Библиотека BehaviorTree:

BehaviorTree.java

```
package org.nikialeksey.gameengine.ai.behaviortree;

import java.util.UUID;

/**
 * Класс представляет дерево поведения.
 * @author Alexey Nikitin
 */
public class BehaviorTree {

    /**
     * корневая вершина
     */
    private Node root;
    /**
     * уникальный идентификатор дерева поведения
     */
    private String uuid;

    /**
     * Конструктор дерева поведения.
     * Назначает уникальный идентификатор и устанавливает ссылку на корень дерева
поведнеия.
     * @param root корень дерева поведения, именно этой вершине будут передаваться
сигналы на исполнение.
     */
    public BehaviorTree(Node root) {
        this.uuid = UUID.randomUUID().toString();
        this.root = root;
    }

    /**
     * Создает объект тика и передает сигнал на исполнение корню дерева поведения
     * @param blackboard память для принятия решения
     * @return объект статуса
     */
    public Status execute(Blackboard blackboard) {
        Tick tick = new Tick(this, blackboard);

        return this.root.execute(tick);

    }

    /**
     * Возвращает уникальный идентификатор дерева поведения.
     * @return строка, представляющая уникальный идентификатор дерева поведения.
     */
    public String getUUID() {return this.uuid;}
```

```
}
```

Blackboard.java

```
package org.nikialeksey.gameengine.ai.behaviortree;

import java.util.*;

/**
 * Класс представляет структуру памяти для наших персонажей, которую будут
использовать вершины дерева поведения.
 * Информация, хранящаяся в {@code Blackboard} структурирована следующим образом:
глобальная информация(доступная
 * из любого места), информация о дереве (досутпная для всех вершин одного дерева),
информация о вершине (доступная
 * только вершине).
 *
 * Память для простототы можно изобразить в виде JSON документа:
 * <pre>
 * {
 *     'global key 1': globalObject1,                    #
 *     'global key 2': globalObject2,                    #  глобальная информация
 *     ...                                               #
 *
 *     'tree1UUID': {
 *         'tree key 1': treeObject1,                    #
 *         'tree key 2': treeObject2,                    #  информация, доступная
для всех вершин одного дерева
 *         ...                                           #
 *
 *         'nodeMemory': {
 *             'node key 1': nodeObject1,                #
 *             'node key 2': nodeObject2,                #  информация, доступная
для одной вершины
 *             ...                                       #
 *         },
 *         'openNodes': [
 *             ...
 *         ]
 *     },
 *     'tree2UUID': {
 *          ...
 *     },
 *     ...
 * }
 * </pre>
 *
 * @author Alexey Nikitin
 */
public class Blackboard {
    private BlackboardMemory baseMemory;
    private Map<String, TreeMemory> treeMemory;

    /**
     * Конструктор.
     * Инициализирует память.
```

```
     */
    public Blackboard() {
        baseMemory = new BlackboardMemory();
        treeMemory = new HashMap<String, TreeMemory>();
    }

    private BlackboardMemory getMemory() {
        return baseMemory;
    }

    private TreeMemory getMemory(String treeUUID) {
        if (!treeMemory.containsKey(treeUUID))
            treeMemory.put(treeUUID, new TreeMemory());
        return treeMemory.get(treeUUID);
    }

    private BlackboardMemory getMemory(String treeUUID, String nodeUUID) {
        return getMemory(treeUUID).getMemory(nodeUUID);
    }

    /**
     * Возвращает объект из глобальной памяти.
     * @param key ключ
     * @return объект из глобальной памяти
     */
    public Object get(String key) {
        return this.getMemory().get(key);
    }

    /**
     * Возвращает объект из памяти дерева с идентификатором {@code treeUUID}.
     * @param key ключ
     * @param treeUUID уникальный идентификатор дерева
     * @return объект из памяти дерева с идентификатором {@code treeUUID}.
     */
    public Object get(String key, String treeUUID) {
        return getMemory(treeUUID).get(key);
    }

    /**
     * Возвращает объект из вершины дерева.
     * @param key ключ
     * @param treeUUID уникальный идентификатор дерева
     * @param nodeUUID уникальный идентификатор вершины
     * @return объект из вершины дерева
     */
    public Object get(String key, String treeUUID, String nodeUUID) {
        return getMemory(treeUUID, nodeUUID).get(key);
    }

    /**
     * Кладет объект в глобальную память.
     * @param key ключ
     * @param value объект
     */
    public void put(String key, Object value) {
        getMemory().put(key, value);
    }
```

```
/**
 * Кладет объект в память для дерева.
 * @param key ключ
 * @param value объект
 * @param treeUUID уникальный идентмификатор дерева
 */
public void put(String key, Object value, String treeUUID) {
    getMemory(treeUUID).put(key, value);
}

/**
 * Кладет объект в память вершины.
 * @param key ключ
 * @param value объект
 * @param treeUUID уникальный идентификатор дерева
 * @param nodeUUID уникальный идентификатор вершины
 */
public void put(String key, Object value, String treeUUID, String nodeUUID) {
    getMemory(treeUUID, nodeUUID).put(key, value);
}

/**
 * Класс представляет память дерева.
 */
public static class TreeMemory {
    private BlackboardMemory localMemory;
    private Map<String, BlackboardMemory> nodeMemory;
    private ArrayList<Object> openNodes;

    public TreeMemory() {
        localMemory = new BlackboardMemory();
        nodeMemory = new HashMap<String, BlackboardMemory>();
        openNodes = new ArrayList<Object>();
    }

    private BlackboardMemory getMemory() {
        return localMemory;
    }

    private BlackboardMemory getMemory(String nodeUUID) {
        if (!nodeMemory.containsKey(nodeUUID))
            nodeMemory.put(nodeUUID, new BlackboardMemory());
        return nodeMemory.get(nodeUUID);
    }

    /**
     * Возвращает объект дерева.
     * @param key ключ
     * @return объект дерева
     */
    public Object get(String key) {
        return getMemory().get(key);
    }

    /**
     * Возвращает объект из вершины данного дерева.
     * @param key ключ
```

```
         * @param nodeUUID уникальный идентификатор вершины
         * @return объект из вершины данного дерева
         */
        public Object get(String key, String nodeUUID) {
            return getMemory(nodeUUID).get(key);
        }

        /**
         * Кладет объект в память дерева.
         * @param key ключ
         * @param value объект
         */
        public void put(String key, Object value) {
            getMemory().put(key, value);
        }

        /**
         * Кладет объект в память вершины.
         * @param key ключ
         * @param value объект
         * @param nodeUUID уникальный идентификатор вершины
         */
        public void put(String key, Object value, String nodeUUID) {
            getMemory(nodeUUID).put(key, value);
        }
    }

    /**
     * Класс представляет память.
     */
    public static class BlackboardMemory implements Map<String, Object> {
        private HashMap<String, Object> memory;

        public BlackboardMemory() {
            memory = new HashMap<String, Object>();
        }

        @Override
        public int size() {
            return memory.size();
        }

        @Override
        public boolean isEmpty() {
            return memory.isEmpty();
        }

        @Override
        public boolean containsKey(Object key) {
            return memory.containsKey(key);
        }

        @Override
        public boolean containsValue(Object value) {
            return memory.containsValue(value);
        }

        @Override
```

```
        public Object get(Object key) {
            return memory.get(key);
        }

        @Override
        public Object put(String key, Object value) {
            return memory.put(key, value);
        }

        @Override
        public Object remove(Object key) {
            return memory.remove(key);
        }

        @Override
        public void putAll(Map<? extends String, ?> m) {
            memory.putAll(m);
        }

        @Override
        public void clear() {
            memory.clear();
        }

        @Override
        public Set<String> keySet() {
            return memory.keySet();
        }

        @Override
        public Collection<Object> values() {
            return memory.values();
        }

        @Override
        public Set<Entry<String, Object>> entrySet() {
            return memory.entrySet();
        }
    }

}
```

Node.java

```
package org.nikialeksey.gameengine.ai.behaviortree;

import java.util.ArrayList;
import java.util.Collections;
import java.util.UUID;

/**
 * Абстрактный класс вершины дерева повдения.
 * Содержит необходимые методы для выполнения логики вершины.
 * @author Alexey Nikitin
 */
public abstract class Node {

    private String uuid;
```

```
    private ArrayList<Node> children;

    /**
     * Конструктор.
     * @param nodes дочерние вершины
     */
    public Node(Node... nodes) {
        this.uuid = UUID.randomUUID().toString();
        this.children = new ArrayList<Node>(nodes.length);
        Collections.addAll(children, nodes);
    }

    /**
     * Возвращает список дочерних вершин
     * @return список дочерних вершин
     */
    public ArrayList<Node> getChildren() {
        return this.children;
    }

    /**
     * Возвращает уникальный идентификатор вершины
     * @return уникальный идентификатор вершины
     */
    public String getUUID(){return this.uuid;}

    /**
     * Выполняет логику вершины.
     * @param tick объект тика
     * @return статус, после выполнения логики
     */
    public Status execute(Tick tick) {
        this.btEnter(tick);

        Boolean isOpen = (Boolean)tick.getBlackboard().get("isOpen",
tick.getBehaviorTree().getUUID(), this.getUUID());
        if (isOpen == null || !isOpen) {
            this.btOpen(tick);
        }

        Status status = this.btTick(tick);

        if (status != Status.RUNNING) {
            this.btClose(tick);
        }

        this.btExit(tick);

        return status;
    }

    private void btEnter(Tick tick) {
        tick.enterNode(this);
        this.enter(tick);
    }

    private void btOpen(Tick tick) {
        tick.openNode(this);
```

```
        tick.getBlackboard().put("isOpen", true, tick.getBehaviorTree().getUUID(),
this.getUUID());
        this.open(tick);
    }

    private Status btTick(Tick tick) {
        tick.tickNode(this);
        return this.tick(tick);
    }

    private void btClose(Tick tick) {
        tick.closeNode(this);
        tick.getBlackboard().put("isOpen", false, tick.getBehaviorTree().getUUID(),
this.getUUID());
        this.close(tick);
    }

    private void btExit(Tick tick) {
        tick.exitNode(this);
        this.exit(tick);
    }

    /**
     * Вызывается, при осуществлении входа в вершину
     * @param tick объект тика
     */
    public abstract void enter(Tick tick);

    /**
     * Вызывается, при осуществлении открытия вершины (всегда после входа, но не
всегда открытая вершина закрывается
     * после выполнения логики)
     * @param tick объект тика
     */
    public abstract void open(Tick tick);

    /**
     * Содержит код логики вершины, после отработки возвращает один из четырех
статусов: RUNNING, WAIT, FAILURE, SUCCESS
     * @param tick объект тика
     * @return один из четырех статусов: RUNNING, WAIT, FAILURE, SUCCESS
     */
    public abstract Status tick(Tick tick);

    /**
     * Вызывается, при осуществлении закрытия вершины (не вызывается после
выполнении логики,
     * если статус {@code RUNNING})
     * @param tick объект тика
     */
    public abstract void close(Tick tick);

    /**
     * Вызывается, при осуществлении выхода из вершины (всегда после выполнения
логики, но позже
     * закрытия вершины, если оно было)
     * @param tick объект тика
     */
```

```
    public abstract void exit(Tick tick);
}
```

Status.java

```
package org.nikialeksey.gameengine.ai.behaviortree;

/**
 * Класс представляет результаты, которые возвращают вершины после исполнения логики.
 * @author Alexey Nikitin
 */
public enum Status {
    /**
     * Запущено.
     * Результат возвращается, когда вершина не завершила действие.
     */
    RUNNING,

    /**
     * Ошибка.
     * Результат возвращается, когда произошла ошибка при исполнении логики вершины.
     */
    ERROR,

    /**
     * Не успешно.
     * Сигнал возвращается, когда исполнение вершины завершилось не успешно.
     */
    FAILURE,

    /**
     * Успешно
     * Сигнал возвращается, когда исполнение вершины завершилось успешно.
     */
    SUCCESS
}
```

Tick.java

```
package org.nikialeksey.gameengine.ai.behaviortree;

import java.util.Stack;

/**
 * Класс содержит ссылки на blackboard и behaviorTree.
 * Используется, когда сигнал на исполнение поступает в дерево поведения и
пробрасывается дочерним вершинам, чтобы
 * дочерние вершины могли пользоваться blackboard'ом.
 *
 * Так же удобно здесь писать отладочный код.
 * @author Alexey Nikitin
 */
public class Tick {
    private BehaviorTree behaviorTree;
    private Blackboard blackboard;
    private Stack<Node> openNodes;
```

```
    /**
     * Конструктор.
     * Сохранает ссылки на объект дерева поведения и на общую память.
     * @param behaviorTree ссылка на дерево поведения, в котором создали объект
Tick.
     * @param blackboard ссылка на общую память.
     */
    public Tick(BehaviorTree behaviorTree, Blackboard blackboard) {
        this.behaviorTree = behaviorTree;
        this.blackboard = blackboard;
        this.openNodes = new Stack<Node>();
    }

    /**
     * Возвращает ссылку на объект общей памяти.
     * @return ссылку на объект общей памяти.
     */
    public Blackboard getBlackboard() {
        return this.blackboard;
    }

    /**
     * Возвращает ссылку на дерево поведения.
     * @return ссылку на дерево поведения.
     */
    public BehaviorTree getBehaviorTree() {
        return this.behaviorTree;
    }

    /**
     * Исполняется при входе в вершину.
     * @param node вершина, в которую осуществился вход.
     */
    public void enterNode(Node node) {
    }

    /**
     * Исполняется при открытии вершины.
     * @param node открываемая вершина.
     */
    public void openNode(Node node) {
        this.openNodes.push(node);
    }

    /**
     * Исполняется перед выполнением логики вершины.
     * @param node вершина, в которую зашли, возможно открыли, и еще не началось
выполнение логики.
     */
    public void tickNode(Node node) {

    }

    /**
     * Исполняется перед закрытием вершины.
     * @param node закрываемая вершина.
     */
```

```
    public void closeNode(Node node) {
        this.openNodes.pop();
    }

    /**
     * Исполняется перед выходом из вершины.
     * @param node вершина, из которой осуществляется выход.
     */
    public void exitNode(Node node) {

    }
}
```

Condition.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Actions;

import org.nikialeksey.gameengine.ai.behaviortree.Blackboard;
import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

import java.util.function.Predicate;

/**
 * Класс представляет вершину-условие.
 * В зависимости от результатата проверки условия эта вершина возвратит статус
SUCCESS или FAILURE
 * @author Alexey Nikitin
 */
public class Condition extends Node {
    Predicate<Tick> condition;

    /**
     * Конструктор.
     * @param condition условие
     */
    public Condition(Predicate<Tick> condition) {
        super();
        this.condition = condition;
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Проверяет условие, заданное пользователем, на основании результата проверки
возвращает SUCCESS, если результат
     * положительный, иначе FAILURE
     * @param tick объект тика
```

```
     * @return возвращает SUCCESS, если результат
     *           положительный, иначе FAILURE
     */
    @Override
    public Status tick(Tick tick) {
        boolean result = condition.test(tick);
        return result ? Status.SUCCESS: Status.FAILURE;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

UserAction.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Actions;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

import java.util.function.Consumer;

/**
 * Класс представляет лист-действие, определяемое пользователем.
 * @author Alexey Nikitin
 */
public class UserAction extends Node {

    private Consumer<Tick> action;

    /**
     * Конструктор.
     * @param action дейстие, определяемое пользователем
     */
    public UserAction(Consumer<Tick> action) {
        super();
        this.action = action;
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
```

```
     * Выполняет пользовательское действие и возвращает SUCCESS.
     * @param tick объект тика
     * @return SUCCESS
     */
    @Override
    public Status tick(Tick tick) {
        this.action.accept(tick);

        return Status.SUCCESS;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

Wait.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Actions;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет лист-действие "ожидание".
 * @author Alexey Nikitin
 */
public class Wait extends Node {
    private long waitingTime;

    /**
     * Конструктор.
     * @param milliseconds количество миллисекунд, которое необходимо подождать.
     */
    public Wait(long milliseconds) {
        super();
        this.waitingTime = milliseconds;
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {
        long startTime = System.currentTimeMillis();
        tick.getBlackboard().put("startTime", startTime,
tick.getBehaviorTree().getUUID(), this.getUUID());
    }
```

```
    /**
     * Проверяет, сколько времени прошло с первого вызова и если прошло времени
больше, чем заданное количество, то
     * вернет SUCCESS.
     * @param tick объект тика
     * @return ERROR, если время первого запуска не было записано; RUNNING, если
время ожидания еще не прошло; SUCCESS,
     * если время ожидания превышает или совпает с заданным значением.
     */
    @Override
    public Status tick(Tick tick) {
        long currTime = System.currentTimeMillis();
        Long startTime = (Long)tick.getBlackboard().get("startTime",
tick.getBehaviorTree().getUUID(), this.getUUID());
        if (startTime == null)
            return Status.ERROR;
        if (currTime - startTime >= this.waitingTime)
            return Status.SUCCESS;
        return Status.RUNNING;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

MemSelector.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Composites;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

import java.util.ArrayList;

/**
 * Класс представляет композит-селектор с запоминанием вершины, которая вернула
результат RUNNING.
 * На следующем тике данная вершина начнет давать сигнал на исполнение как раз
последней запущенной
 * дочерней вершне, а не с начала спика дочерних вершин.
 * @author Alexey Nikitin
 */
public class MemSelector extends Node {

    /**
     * Конструктор.
     * @param nodes список дочеирних вершин
     */
    public MemSelector(Node... nodes) {
```

```
        super(nodes);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {
        tick.getBlackboard().put("runningChild", 0,
tick.getBehaviorTree().getUUID(), this.getUUID());
    }

    /**
     * Передает сигнал на исполнение дочерним вершинам до тех пор, пока они
возвращают статус FAILURE. Как только
     * дочерняя вершина вернет статус, отличный от FAILURE, этот статус будет сразу
возвращен этой вершиной и выполнение
     * закончится. Если это был статус RUNNING, то в таком случае будет запомнен в
blackboard индекс данной дочерней
     * вершины и в следующий раз передача сигнала на исполнение продолжится именно с
этой вершины.
     * @param tick объект тика
     * @return либо статус дочерней вершины, которая вернула результат, отличный от
FAILURE, либо FAILURE
     */
    @Override
    public Status tick(Tick tick) {
        ArrayList<Node> children = this.getChildren();
        Integer startIndex = (Integer)tick.getBlackboard().get("runningChild",
tick.getBehaviorTree().getUUID(), this.getUUID());
        for (int i = startIndex == null ? 0 : startIndex; i < children.size(); i++)
{
            Node child = children.get(i);
            Status status = child.execute(tick);

            if (status != Status.FAILURE) {
                if (status == Status.RUNNING) {
                    tick.getBlackboard().put("runningChild", i,
tick.getBehaviorTree().getUUID(), this.getUUID());
                }
                return status;
            }
        }

        return Status.FAILURE;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {
```

```
    }
}
```

MemSequence.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Composites;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

import java.util.ArrayList;

/**
 * Класс представляет композит-последовательность с запоминанием вершины, которая
вернула результат RUNNING.
 * На следующем тике данная вершина начнет давать сигнал на исполнение как раз
последней запущенной
 * дочерней вершне, а не с начала спика дочерних вершин.
 * @author Alexey Nikitin
 */
public class MemSequence extends Node {

    /**
     * Конструктор.
     * @param nodes список дочеирних вершин
     */
    public MemSequence(Node... nodes) {
        super(nodes);
    }
    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {
        tick.getBlackboard().put("runningChild", 0,
tick.getBehaviorTree().getUUID(), this.getUUID());
    }

    /**
     * Передает сигнал на исполнение дочерним вершинам до тех пор, пока они
возвращают статус SUCCESS. Как только
     * дочерняя вершина вернет статус, отличный от SUCCESS, этот статус будет сразу
возращен этой вершиной и выполнение
     * закончится. Если это был статус RUNNING, то в таком случае будет запомнен в
blackboard индекс данной дочерней
     * вершины и в следующий раз передача сигнала на исполнение продолжится именно с
этой вершины.
     * @param tick объект тика
     * @return либо статус дочерней вершины, которая вернула результат, отличный от
SUCCESS, либо SUCCESS
     */
    @Override
    public Status tick(Tick tick) {
        ArrayList<Node> children = this.getChildren();
```

```
        Integer startIndex = (Integer)tick.getBlackboard().get("runningChild",
tick.getBehaviorTree().getUUID(), this.getUUID());
        for (int i = startIndex == null ? 0 : startIndex; i < children.size(); i++)
{
            Node child = children.get(i);
            Status status = child.execute(tick);

            if (status != Status.SUCCESS) {
                if (status == Status.RUNNING) {
                    tick.getBlackboard().put("runningChild", i,
tick.getBehaviorTree().getUUID(), this.getUUID());
                }
                return status;
            }
        }

        return Status.SUCCESS;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

Parallel.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Composites;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет параллельный композит.
 * @author Alexey Nikitin
 */
public class Parallel extends Node {
    private int successCount, failureCount;

    /**
     * Конструктор.
     * @param successCount количество дочерних вершин, которое должно вернуть
SUCCESS,
     *                     чтобы данный композит вернул SUCCESS
     * @param failureCount количество дочерних вершин, которое должно вернуть
FAILURE,
     *                     чтобы данный композит вернул FAILURE
     * @param nodes список дочеирних вершин
     */
    public Parallel(int successCount, int failureCount, Node... nodes) {
        super(nodes);
```

```
        this.successCount = successCount;
        this.failureCount = failureCount;
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение всем дочерним вершинам одновременно.
     * @param tick объект тика
     * @return если количество дочерних вершин, вернувших результат SUCCESS больше
порогового значение
     * {@code successCount} и, при этом, количество вершин, вернувших результат
FAILURE меньше порогового значения
     * {@code failureCount}, то SUCCESS; если количество дочерних вершин, вернувших
результат FAILURE больше порогового значение
     * {@code failureCount} и, при этом, количество вершин, вернувших результат
SUCCESS меньше порогового значения
     * {@code successCount}, то FAILURE; иначе ERROR
     */
    @Override
    public Status tick(Tick tick) {
        int currentSuccessCount = 0,
                currentFailureCount = 0;
        for (Node child: this.getChildren()) {
            Status status = child.execute(tick);
            if (status == Status.SUCCESS)
                currentSuccessCount++;
            else if (status == Status.FAILURE)
                currentFailureCount++;
        }
        if (currentSuccessCount >= successCount && currentFailureCount <
failureCount)
            return Status.SUCCESS;

        if (currentFailureCount >= failureCount && currentSuccessCount <
successCount)
            return Status.FAILURE;

        return Status.ERROR;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {
```

```
    }
}
```

Selector.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Composites;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет композит-селектор.
 * Выполняет все свои дочерние вершины по порядку, до тех пор, пока они возвращают
результат FAILURE
 * @author Alexey Nikitin
 */
public class Selector extends Node {

    /**
     * Конструктор.
     * @param nodes список дочеирних вершин
     */
    public Selector(Node... nodes) {
        super(nodes);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение всем дочерним вершинам, до тех пор пока они
возвращают FAILURE
     * @param tick объект тика
     * @return либо статус той вершины, которая вернула результат, отличный от
FAILURE, либо FAILURE
     */
    @Override
    public Status tick(Tick tick) {
        for (Node child: this.getChildren()) {
            Status status = child.execute(tick);

            if (status != Status.FAILURE)
                return status;
        }

        return Status.FAILURE;
    }

    @Override
```

```
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

Sequence.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Composites;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет композит-последовательность.
 * Выполняет всех своих детей до тех пор, пока они возвращают SUCCESS
 * @author Alexey Nikitin
 */
public class Sequence extends Node {

    /**
     * Конструктор.
     * @param nodes список дочеирних вершин
     */
    public Sequence(Node... nodes) {
        super(nodes);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение всем своим дочерним вершинам до тех пор, пока
они возвращают SUCCESS
     * @param tick объект тика
     * @return либо статус дочерней вершины, которая вернула результат, отличный от
SUCCESS, либо SUCCESS
     */
    @Override
    public Status tick(Tick tick) {
        for (Node child: this.getChildren()) {
            Status status = child.execute(tick);

            if (status != Status.SUCCESS)
                return status;
```

```
        }

        return Status.SUCCESS;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

AlwaysFailure.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Decorators;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет декоратор, который всегда возвращает статус FAILURE.
 * @author Alexey Nikitin
 */
public class AlwaysFailure extends Node {

    /**
     * Конструктор.
     * @param node дочерняя вершина
     */
    public AlwaysFailure(Node node) {
        super(node);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение дочерней вершине, всегда возвращает сигнал
FAILURE
     * @param tick объект тика
     * @return FAILURE
     */
    @Override
    public Status tick(Tick tick) {
        if (this.getChildren().isEmpty())
            return Status.FAILURE;
```

```
        Node child = this.getChildren().get(0);
        if (child == null)
            return Status.FAILURE;

        child.execute(tick);

        return Status.FAILURE;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

AlwaysSuccess.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Decorators;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет декоратор, который всегда возвращает статус SUCCESS.
 * @author Alexey Nikitin
 */
public class AlwaysSuccess extends Node {

    /**
     * Конструктор.
     * @param node дочерняя вершина
     */
    public AlwaysSuccess(Node node) {
        super(node);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение дочерней вершине, всегда возвращает сигнал
SUCCESS
     * @param tick объект тика
     * @return SUCCESS
```

```
     */
    @Override
    public Status tick(Tick tick) {
        if (this.getChildren().isEmpty())
            return Status.SUCCESS;

        Node child = this.getChildren().get(0);
        if (child == null)
            return Status.SUCCESS;

        child.execute(tick);

        return Status.SUCCESS;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

Inverter.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Decorators;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Представляет декоратор инвертер.
 * @author Alexey Nikitin
 */
public class Inverter extends Node {
    /**
     * Конструктор.
     * @param node дочерняя вершина
     */
    public Inverter(Node node) {
        super(node);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
```

```
     * Передает сигнал на исполнение дочерней вершине.
     * @param tick объект тика
     * @return SUCCESS, если дочерняя вершина вернула результат FAILURE; FAILURE,
если дочерняя
     * вершина вернула результат SUCCESS; иначе тот результат, который вернула
дочерняя вершина.
     */
    @Override
    public Status tick(Tick tick) {
        if (this.getChildren().isEmpty())
            return Status.ERROR;

        Node child = this.getChildren().get(0);
        if (child == null)
            return Status.ERROR;

        Status status = child.execute(tick);

        if (status == Status.FAILURE)
            status = Status.SUCCESS;
        else if (status == Status.SUCCESS)
            status = Status.FAILURE;

        return status;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

Repeat.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Decorators;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет декоратор, который передает сигнал на исполнение дочерней
вершине заданное количество раз.
 * @author Alexey Nikitin
 */
public class Repeat extends Node {

    private int repeatCount;

    /**
     * Конструктор.
     * @param repeatCount количество передач сигнала на исполнение дочерней вершине.
```

```
     * @param node дочерняя вершина
     */
    public Repeat(int repeatCount, Node node) {
        super(node);
        this.repeatCount = repeatCount;
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение дочерней вершине заданное число раз.
     * @param tick объект тика
     * @return ERROR, если дочерней вершины нет; последний статус, который вернула дочерняя вершина, если количество
     * повторений было не нулевое; если количество повторений было нулевое, то SUCCESS.
     */
    @Override
    public Status tick(Tick tick) {
        if (this.getChildren().isEmpty())
        return Status.ERROR;

        Node child = this.getChildren().get(0);
        if (child == null)
            return Status.ERROR;

        Status status = Status.SUCCESS;
        for (int iter = 0; iter < this.repeatCount; iter++) {
            status = child.execute(tick);
        }

        return status;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

UntilFailure.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Decorators;
```

```java
import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет декоратор, который передает сигнал на исполнение дочерней вершине до тех пор, пока
 * она не вернет статус FAILURE
 * @author Alexey Nikitin
 */
public class UntilFailure extends Node {

    /**
     * Конструктор.
     * @param node дочерняя вершина
     */
    public UntilFailure(Node node) {
        super(node);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение дочерней вершине до тех пор, пока она возвращает статус, отличный от FAILURE.
     * @param tick объект тика
     * @return ERROR, если нет дочерней вершины, FAILURE иначе.
     */
    @Override
    public Status tick(Tick tick) {
        if (this.getChildren().isEmpty())
            return Status.ERROR;

        Node child = this.getChildren().get(0);
        if (child == null)
            return Status.ERROR;

        while (child.execute(tick) != Status.FAILURE);

        return Status.FAILURE;
    }

    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {
```

```
    }
}
```

UntilSuccess.java

```
package org.nikialeksey.gameengine.ai.behaviortree.Decorators;

import org.nikialeksey.gameengine.ai.behaviortree.Node;
import org.nikialeksey.gameengine.ai.behaviortree.Status;
import org.nikialeksey.gameengine.ai.behaviortree.Tick;

/**
 * Класс представляет декоратор, который передает сигнал на исполнение дочерней
вершине до тех пор, пока
 * она не вернет статус SUCCESS
 * @author Alexey Nikitin
 */
public class UntilSuccess extends Node {

    /**
     * Конструктор.
     * @param node дочерняя вершина
     */
    public UntilSuccess(Node node) {
        super(node);
    }

    @Override
    public void enter(Tick tick) {

    }

    @Override
    public void open(Tick tick) {

    }

    /**
     * Передает сигнал на исполнение дочерней вершине до тех пор, пока она
возвращает статус, отличный от SUCCESS.
     * @param tick объект тика
     * @return ERROR, если нет дочерней вершины, SUCCESS иначе.
     */
    @Override
    public Status tick(Tick tick) {
        if (this.getChildren().isEmpty())
            return Status.ERROR;

        Node child = this.getChildren().get(0);
        if (child == null)
            return Status.ERROR;

        while (child.execute(tick) != Status.SUCCESS);

        return Status.SUCCESS;
    }
```

```
    @Override
    public void close(Tick tick) {

    }

    @Override
    public void exit(Tick tick) {

    }
}
```

Визуальное средство для проектирования деревьев поведений

NormalColorAnimation.qml

```
import QtQuick 2.4
import PyConsole 1.0

ColorAnimation { duration: 500; easing.type: Easing.OutQuint }
```

NormalNumberAnimation.qml

```
import QtQuick 2.4
import PyConsole 1.0

NumberAnimation { duration: 200; easing.type: Easing.OutQuint }
```

DnDObject.qml

```
import QtQuick 2.4
import PyConsole 1.0

FormObject {
    PyConsole {
        id: pyconsole
    }

    property bool isDragEnabled: true // true, если манипулирование при помощи Drag-and-drop разрешено

    property bool isDrag: false // true, если эелемнт перетаскивается мышью
    property bool isHovered: false // true, если мышь находится над элементом

    property real dragMinimumX: x // минимальное значение координаты x объекта
    property real dragMaximumX: x // максимальное значение координаты x объекта

    property real dragMinimumY: y // минимальное значение координаты y объекта
    property real dragMaximumY: y // максимальное значение координаты y объекта

    property real mouseX: mouseArea.mouseX // значение x-координаты курсора мыши относительно данного объекта
    property real mouseY: mouseArea.mouseY // значение y-координаты курсора мыши относительно данного объекта

    Drag.active: mouseArea.drag.active
    Drag.hotSpot.x: width/2
    Drag.hotSpot.y: height/2
```

```
    signal dragBegin // возбуждается на начало манипулирования при помощи Drag-and-
drop
    signal dragEnd // возбуждается после окончания манипулирования при помощи Drag-
and-drop
    signal clicked // возбуждается после клика мышью на объект
    signal pressed // возбуждается при нажатии мышью на объект
    signal released // возбуждается после отжатия мыши на объекте
    signal entered // возбуждается при входе курсора мыши в поле объекта
    signal exited // возбуждается при выходе курсора мыши за поле объекта

    onChildrenChanged: {
        /*if (children.length === 2) {
            // Для того, чтобы все дочерние элементы, кроме тех,
            // что описаны в этом файле, были перенаправлены в
            // mouseArea. Это необходимо для того, чтобы drag
            // работал на всех дочерних элементах
            var childrens = [];
            for(var i = 0; i < mouseArea.children.length; i++) {
                childrens[i] = mouseArea.children[0];
            }
            childrens[mouseArea.children.length] = children[1];
            mouseArea.children = childrens;
            children = [children[0], ];
        }*/
    }

    MouseArea {
        id: mouseArea

        enabled: parent.isDragEnabled

        hoverEnabled: true
        anchors.fill: parent
        drag.target: parent
        drag.filterChildren: true
        drag.minimumX: parent.dragMinimumX
        drag.maximumX: parent.dragMaximumX
        drag.minimumY: parent.dragMinimumY
        drag.maximumY: parent.dragMaximumY

        onPositionChanged: {
            if (mouseArea.pressed && !parent.isDrag) {
                parent.dragBegin();
                parent.isDrag = true;
            }
        }

        onClicked: {
            parent.clicked();
        }

        onPressed: {
            parent.pressed();
        }

        onReleased: {
            if (parent.isDrag) {
                parent.dragEnd();
                parent.isDrag = false;
            }
```

```
            parent.released();
        }

        onEntered: {
            parent.isHovered = true;
            parent.entered();
        }

        onExited: {
            parent.isHovered = false;
            parent.exited();
        }
    }
}
```

FormObject.qml

```
import QtQuick 2.4

Rectangle {
    signal positionChanged

    onXChanged: {
        positionChanged();
    }

    onYChanged: {
        positionChanged();
    }
}
```

AbstractButton.qml

```
import QtQuick 2.4
import "../Base"

FormObject {
    property bool isHovered: false // true, если мышь находится над элементом
    property bool isPressed: mouseArea.pressed // true, если мышь была нажата,
находясь над элементом

    property real mouseX: mouseArea.mouseX // значение x-координаты курсора мыши
относительно данного объекта
    property real mouseY: mouseArea.mouseY // значение y-координаты курсора мыши
относительно данного объекта

    signal clicked
    signal pressed
    signal released
    signal entered
    signal exited
    signal changeMousePosition


    MouseArea {
        id: mouseArea
        anchors.fill: parent
```

```
        hoverEnabled: true

        onPositionChanged: {
            parent.changeMousePosition();
        }

        onClicked: {
            parent.clicked();
        }

        onPressed: {
            parent.pressed();
        }

        onReleased: {
            parent.released();
        }

        onEntered: {
            parent.isHovered = true;
            parent.entered();
        }

        onExited: {
            parent.isHovered = false;
            parent.exited();
        }
    }
}
```

ArrowButton.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../Animations"

AbstractButton {
    PyConsole {
        id: pyconsole
    }

    id: arrowButton
    property string normalColor
    property string pressedColor
    property string hoveredColor
    color: normalColor

    signal drawArrow // Возбуждается при перемещении курсора мыши при условии, что он
была нажата кнопка мыши
                     // в поле объекта и не отжата
    signal drawArrowBegin // Возбуждается при первом перемещении курсора мыши при
условии, что была нажата кнопка мыши в
                          // поле объекта и не отжата
    signal drawArrowEnd // Возбуждается после отжатия кнопки мыши при условии, что
был возбужден сигнал
                        // drawArrowBegin ранее
```

```
    onPressed: {
        arrowButton.drawArrowBegin();
    }

    onReleased: {
        arrowButton.drawArrowEnd();
    }

    onChangeMousePosition: {
        if (isPressed) {
            arrowButton.drawArrow();
        }
    }

    Behavior on color {
        NormalColorAnimation {}
    }

    states: [
        State {
            name: ''
            PropertyChanges {target: arrowButton; color: normalColor}
        },
        State {
            name: 'pressed'
            when: isPressed
            PropertyChanges {target: arrowButton; color: pressedColor}
        },
        State {
            name: 'hovered'
            when: isHovered
            PropertyChanges {target: arrowButton; color: hoveredColor}
        }
    ]
}
```

NormalArrowButton.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../Animations"

ArrowButton {
    property real realButtonWidth
    property real realButtonHeight

    normalColor: '#A468D5'
    pressedColor: '#9240D5'
    hoveredColor: '#582781'

    opacity: 0.2

    onEntered: {
        width *= 2;
        height *= 2;
    }

    onVisibleChanged: {
        if (visible) {
```

```
            width = realButtonWidth;
            height = realButtonHeight;
        }
    }

    onExited: {
        width = realButtonWidth;
        height = realButtonHeight;
    }

    Behavior on width {
        NormalNumberAnimation {}
    }

    Behavior on height {
        NormalNumberAnimation {}
    }

    Behavior on radius {
        NormalNumberAnimation {}
    }

    Behavior on opacity {
        NormalNumberAnimation {}
    }
}
```

NormalRedXButton.qml

```
import QtQuick 2.4

RedXButton {
    normalColor: '#40ff0000'
    hoveredColor: '#80ff0000'
    pressedColor: '#ffff0000'
}
```

NormalTextButton.qml

```
import QtQuick 2.4
import PyConsole 1.0

TextButton {
    colorNormal: '#A468D5'
    colorHovered: '#9240D5'
    colorPressed: '#582781'
}
```

RedXButton.qml

```
import QtQuick 2.4
import "../Animations"

AbstractButton {
    id: redXButton
    property string normalColor
    property string pressedColor
```

```
    property string hoveredColor

    color: normalColor

    width: textInstance.width
    height: textInstance.height

    Text {
        id: textInstance
        text: 'X'
        font.pointSize: 10
        anchors.centerIn: parent
    }

    Behavior on color {
        NormalColorAnimation {}
    }

    Behavior on opacity {
        NormalNumberAnimation {}
    }

    Behavior on x {
        NormalNumberAnimation {}
    }

    Behavior on y {
        NormalNumberAnimation {}
    }

    states: [
        State {
            name: ''
            PropertyChanges {target: redXButton; color: normalColor; opacity: 0}
        },
        State {
            name: 'pressed'
            when: isPressed
            PropertyChanges {target: redXButton; color: pressedColor; opacity: 1}
        },
        State {
            name: 'hovered'
            when: isHovered
            PropertyChanges {target: redXButton; color: hoveredColor; opacity: 1}
        }
    ]
}
```

TextButton.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../Animations"

AbstractButton {
    id: textButton
    property string colorNormal
    property string colorPressed
    property string colorHovered
```

```
    property string text // текст, написанный на кнопке

    color: colorNormal

    width: textInstance.width
    height: textInstance.height

    Text {
        id: textInstance
        text: parent.text
        font.pointSize: 24

        anchors.centerIn: parent
    }

    Behavior on color {
        NormalColorAnimation {}
    }

    states: [
        State {
            name: ''
            PropertyChanges {target: textButton; color: colorNormal}
        },
        State {
            name: 'pressed'
            when: isPressed
            PropertyChanges {target: textButton; color: colorPressed}
        },
        State {
            name: 'hovered'
            when: isHovered
            PropertyChanges {target: textButton; color: colorHovered}
        }
    ]
}
```

ActionObject.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../"

ToolObject {
    rightPointVisible: false

}
```

Condition.qml

```
import QtQuick 2.4
import PyConsole 1.0

ActionObject {

    actualName: 'Condition'

    PyConsole {
```

```
        id: pyconsole
    }

    function createPrototypeComponent() {
        return Qt.createComponent('Condition.qml');
    }

    Text {
        text: 'Cnd'
        font.pointSize: 18
        anchors.centerIn: parent
    }
```

UserAction.qml

```
import QtQuick 2.4
import PyConsole 1.0

ActionObject {

    actualName: 'UserAction'

    PyConsole {
        id: pyconsole
    }

    function createPrototypeComponent() {
        return Qt.createComponent('UserAction.qml');
    }

    Image {
        anchors.fill: parent
        source: "Icons/UserAction.svg"
    }
}
```

Wait.qml

```
import QtQuick 2.4
import PyConsole 1.0

ActionObject {
    actualName: 'Wait'

    function createPrototypeComponent() {
        return Qt.createComponent('Wait.qml');
    }

    Image {
        anchors.fill: parent
        source: "Icons/Wait.svg"
    }
}
```

CompositeObject.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../"

ToolObject {

}
```

MemSelector.qml

```
import QtQuick 2.4
import PyConsole 1.0

CompositeObject {
    actualName: 'MemSelector'

    function createPrototypeComponent() {
        return Qt.createComponent('MemSelector.qml');
    }

    /*Image {
        anchors.fill: parent
        source: "Icons/MemSelector.svg"
    }*/

    Text {
        text: 'MSl'
        font.pointSize: 16
        anchors.centerIn: parent
    }
}
```

MemSequence.qml

```
import QtQuick 2.4
import PyConsole 1.0

CompositeObject {
    actualName: 'MemSequence'

    function createPrototypeComponent() {
        return Qt.createComponent('MemSequence.qml');
    }

    /*Image {
        anchors.fill: parent
        source: "Icons/MemSequence.svg"
    }*/

    Text {
        text: 'MSq'
        font.pointSize: 16
        anchors.centerIn: parent
    }
}
```

Parallel.qml

```
import QtQuick 2.4
import PyConsole 1.0

CompositeObject {
    actualName: 'Parallel'

    function createPrototypeComponent() {
        return Qt.createComponent('Parallel.qml');
    }

    Image {
        anchors.fill: parent
        source: "Icons/Parallel.svg"
    }
}
```

Root.qml

```
import QtQuick 2.4

Sequence {
    function createPrototypeComponent() {
        return Qt.createComponent('Root.qml');
    }

    isDeletable: false
    isDragEnabled: false
    leftPointVisible: false

    Component.onCompleted: {
        _rightPoint.visible = true;
    }
}
```

Selector.qml

```
import QtQuick 2.4
import PyConsole 1.0

CompositeObject {
    actualName: 'Selector'

    function createPrototypeComponent() {
        return Qt.createComponent('Selector.qml');
    }

    Image {
        anchors.fill: parent
        source: "Icons/Selector.svg"
    }
}
```

Sequence.qml

```
import QtQuick 2.4
import PyConsole 1.0
```

```
CompositeObject {
    actualName: 'Sequence'

    function createPrototypeComponent() {
        return Qt.createComponent('Sequence.qml');
    }

    Image {
        anchors.fill: parent
        source: "Icons/Sequence.svg"
    }
}
```

AlwaysFail.qml

```
import QtQuick 2.4
import PyConsole 1.0

DecoratorObject {
    actualName: 'Always Fail'

    function createPrototypeComponent() {
        return Qt.createComponent('AlwaysFail.qml');
    }

    Text {
        text: 'Flr'
        font.pointSize: 18
        anchors.centerIn: parent
    }
}
```

AlwaysSuccess.qml

```
import QtQuick 2.4
import PyConsole 1.0

DecoratorObject {
    actualName: 'Always Success'

    function createPrototypeComponent() {
        return Qt.createComponent('AlwaysSuccess.qml');
    }

    Text {
        text: 'Suc'
        font.pointSize: 18
        anchors.centerIn: parent
    }
}
```

DecoratorObject.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../"
```

```
ToolObject {
    PyConsole {
        id: pyconsole
    }

    maxChildrenCount: 1
}
```

Intevrter.qml

```
import QtQuick 2.4
import PyConsole 1.0

DecoratorObject {
    actualName: 'Inverter'

    function createPrototypeComponent() {
        return Qt.createComponent('Inverter.qml');
    }

    Text {
        text: 'Inv'
        font.pointSize: 18
        anchors.centerIn: parent
    }
}
```

Repeat.qml

```
import QtQuick 2.4
import PyConsole 1.0

DecoratorObject {
    actualName: 'Repeat'

    function createPrototypeComponent() {
        return Qt.createComponent('Repeat.qml');
    }

    Text {
        text: 'Rpt'
        font.pointSize: 18
        anchors.centerIn: parent
    }
}
```

UntilFail.qml

```
import QtQuick 2.4
import PyConsole 1.0

DecoratorObject {
    actualName: 'UntilFail'

    function createPrototypeComponent() {
        return Qt.createComponent('UntilFail.qml');
    }
```

```
    Text {
        text: 'UFl'
        font.pointSize: 18
        anchors.centerIn: parent
    }
}
```

UntilSuccess.qml

```
import QtQuick 2.4
import PyConsole 1.0

DecoratorObject {
    actualName: 'UntilSuccess'

    function createPrototypeComponent() {
        return Qt.createComponent('UntilSuccess.qml');
    }

    Text {
        text: 'USc'
        font.pointSize: 18
        anchors.centerIn: parent
    }
}
```

ToolObject.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../Base"
import "../Buttons"
import Curves 1.0

DnDObject {
    PyConsole {
        id: pyconsole
    }

    id: toolObject

    z: 3

    width: 40
    height: 40

    color: "#A468D5"
    dragMinimumX: -10000
    dragMaximumX: 10000
    dragMinimumY: -10000
    dragMaximumY: 10000

    Drag.keys: ['tool', ]

    property string actualName: 'toolObject' // имя элемента дерева поведения,
используется при генерации дерева
```

```
    property bool isDeletable: true // true, если на объекте появляется кнопка для
удаления

    property var _leftPoint: leftPoint
    property var _rightPoint: rightPoint

    property bool isPrototype: true // true, если объект находится на панеле
инструментов
    property real canvasDX: 0
    property real canvasDY: 0
    property string arrowPointColor: '#A69F00'

    property bool leftPointVisible: true // активна ли левая кнопка для стрелочек
    property bool rightPointVisible: true // активна ли правая кнопка для стрелочек
    property int maxChildrenCount: -1 // сколько стрелочек могут входить в правую
кнопку (-1 - бесконечно много)

    property real arrowButtonWidth: 10
    property real arrowButtonHeight: height

    property Canvas canvas

    function createPrototypeComponent() {
        return Qt.createComponent('ToolObject.qml');
    }

    onDragBegin: {
        if (isPrototype) {
            var component = createPrototypeComponent();

            var posX = mouseX;
            var posY = mouseY;

            var incubator = component.incubateObject(
                    toolObject.parent, {
                        x: toolObject.x,
                        y: toolObject.y,
                        canvasDX: toolObject.canvasDX,
                        canvasDY: toolObject.canvasDY,
                        color: toolObject.color,
                        canvas: toolObject.canvas,
                        leftPointVisible: toolObject.leftPointVisible,
                        rightPointVisible: toolObject.rightPointVisible,
                        maxChildrenCount: toolObject.maxChildrenCount,
                        arrowButtonWidth: toolObject.arrowButtonWidth,
                        arrowButtonHeight: toolObject.arrowButtonHeight,
                    }
            );
            incubator.onStatusChanged = function (status) {
                if (status === Component.Ready) {
                    var obj = incubator.object;
                    toolObject.isPrototype = false;
                    toolObject.z = 4;
                    leftPoint.visible = leftPointVisible;
                    rightPoint.visible = rightPointVisible;
                }
            }
        }
    }

    onDragEnd: {
```

```
        if (parent != Drag.target) {
            parent = Drag.target;

            x = x - canvasDX;
            y = y - canvasDY;

            if (parent == null) {
                destruct();
            }
        }
    }

    onEntered: {
        if (!isPrototype && isDeletable) {
            redX.opacity = 1;
        }
    }

    onExited: {
        if (!isPrototype && isDeletable) {
            redX.opacity = 0;
        }
    }

    NormalRedXButton {
        id: redX
        width: parent.width
        z: 1
        opacity: 0

        visible: !parent.isPrototype && parent.isDeletable

        onClicked: {
            parent.destruct();
        }
    }

    Component {
        id: bezierCurvePrototype
        BezierCurve {
            curveWidth: 2
            color: 'black'

            property var rightPoint // то, откуда выходит, то есть где старт
            property var leftPoint // то, куда приходит, то есть где конец
        }
    }

    NormalArrowButton {
        id: leftPoint
        realButtonWidth: parent.arrowButtonWidth
        realButtonHeight: parent.arrowButtonHeight
        x: -width
        y: parent.height / 2 - height / 2
        visible: false

        property var parentArrow
        property var arrow // текущая стрелочка (она в данный момент двигается вместе
с мышкой)

        function getArrowEndX() {
```

```
            return parent.x - parent.arrowButtonWidth / 2;
        }

        function getArrowEndY() {
            return parent.y + parent.height / 2;
        }

        function addArrow(arrow) {
            if (parentArrow) {
                var rightPoint = parentArrow.rightPoint;
                if (rightPoint) {
                    rightPoint.removeArrow(parentArrow);
                }
                parentArrow.destroy();
            }

            arrow.endX = Qt.binding(getArrowEndX);
            arrow.endY = Qt.binding(getArrowEndY);
            arrow.leftPoint = leftPoint;

            parentArrow = arrow;
        }

        function removeArrow(arrow) {
            parentArrow = null;
        }

        onVisibleChanged: {
            if (visible) {
                width = parent.arrowButtonWidth;
                height = parent.arrowButtonHeight;
            }
        }

        onDrawArrowBegin: {
            var incubator = bezierCurvePrototype.incubateObject(
                parent.canvas,
                {
                    'endArrow': true,
                }
            );

            incubator.onStatusChanged = function (status) {
                if (status === Component.Ready) {
                    arrow = incubator.object;
                    arrow.endX = Qt.binding(getArrowEndX);
                    arrow.endY = Qt.binding(getArrowEndY);
                    arrow.startX = parent.x - parent.arrowButtonWidth + mouseX;
                    arrow.startY = parent.y + mouseY;

                    arrow.Drag.active = true;
                    arrow.Drag.keys = ['left', ];
                    arrow.Drag.hotSpot.x = Qt.binding(function() {return
arrow.startX;});
                    arrow.Drag.hotSpot.y = Qt.binding(function() {return
arrow.startY;});
                }
            };
        }

        onDrawArrowEnd: {
```

```
    var dropArea = arrow.Drag.target
    if (dropArea) {
        addArrow(arrow);

        var rightPoint = dropArea.parent;
        rightPoint.addArrow(arrow);

        arrow.Drag.active = false;
    } else {
        arrow.destroy();
    }
    arrow = null;
}

onDrawArrow: {
    arrow.startX = parent.x - parent.arrowButtonWidth + mouseX;
    arrow.startY = parent.y + mouseY;
}

DropArea {
    id: leftPointDrop
    anchors.fill: parent

    keys: ['right', ]
}

states: [
    State {
        when: leftPointDrop.containsDrag
        PropertyChanges {
            target: leftPoint
            opacity: 0.8
        }
    }
]

onEntered: {
    if (parentArrow) {
        textX.visible = true;
        color = '#80ff0000';
        opacity = 1;
    }
}

onExited: {
    if (parentArrow) {
        textX.visible = false;
        opacity = 0.2;
    }
}

onPressed: {
    if (parentArrow) {
        color = '#ffff0000';
        opacity = 1;
    }
}

onReleased: {
    opacity = 0.2;
}
```

```
        onClicked: {
            if (parentArrow) {
                if(parentArrow.rightPoint) {
                    parentArrow.rightPoint.removeArrow(parentArrow);
                }
                parentArrow.destroy();
                parentArrow = null;

                textX.visible = false;
            }
        }

        Text {
            id: textX
            text: 'X'
            font.pointSize: 10
            anchors.centerIn: parent
            visible: false
        }

    }

    NormalArrowButton {
        id: rightPoint
        realButtonWidth: parent.arrowButtonWidth
        realButtonHeight: parent.arrowButtonHeight
        x: parent.width
        y: parent.height / 2 - height / 2
        visible: false

        property var arrows: [] // список всех стрелочек, торчащих из этой кнопки
        property var arrow // текущая стрелочка (она в данный момент двигается вместе
с мышкой)

        function getArrowStartX() {
            return parent.x + parent.width + parent.arrowButtonWidth / 2;
        }

        function getArrowStartY() {
            return parent.y + parent.height / 2;
        }

        function addArrow(arrow) {
            if (arrows.length == parent.maxChildrenCount) {
                var i;
                var leftPoint = arrows[0].leftPoint;
                if (leftPoint) {
                    leftPoint.removeArrow(arrow[0]);
                }
                arrows[0].destroy();
                for (i = 1; i < arrows.length; i++) {
                    arrows[i - 1] = arrows[i];
                }
                arrows.pop();
            }
            arrow.startX = Qt.binding(getArrowStartX);
            arrow.startY = Qt.binding(getArrowStartY);
            arrow.rightPoint = rightPoint;

            arrows.push(arrow);
```

```
        }

        function removeArrow(arrow) {
            var index = arrows.indexOf(arrow);
            if (index >= 0) {
                arrows.splice(index, 1);
            }
        }

        onDrawArrowBegin: {
            var incubator = bezierCurvePrototype.incubateObject(
                parent.canvas,
                {
                    'endArrow': true,
                }
            );

            incubator.onStatusChanged = function (status) {
                if (status === Component.Ready) {
                    arrow = incubator.object;
                    arrow.endX = parent.x + parent.width + mouseX;
                    arrow.endY = parent.y + mouseY;
                    arrow.startX = Qt.binding(getArrowStartX);
                    arrow.startY = Qt.binding(getArrowStartY);

                    arrow.Drag.active = true;
                    arrow.Drag.keys = ['right', ];
                    arrow.Drag.hotSpot.x = Qt.binding(function() {return
arrow.endX;});
                    arrow.Drag.hotSpot.y = Qt.binding(function() {return
arrow.endY;});
                }
            };
        }

        onDrawArrowEnd: {
            var dropArea = arrow.Drag.target;
            if (dropArea) {
                addArrow(arrow);

                var leftPoint = dropArea.parent;
                leftPoint.addArrow(arrow);

                arrow.Drag.active = false;
            } else {
                arrow.destroy();
            }
            arrow = null;
        }

        onDrawArrow: {
            arrow.endX = parent.x + parent.width + mouseX;
            arrow.endY = parent.y + mouseY;
        }

        DropArea {
            id: rightPointDrop
            anchors.fill: parent

            keys: ['left', ]
        }
```

```
        states: [
            State {
                when: rightPointDrop.containsDrag
                PropertyChanges {
                    target: rightPoint
                    opacity: 0.8
                }
            }
        ]
    }

    function destruct() {
        if (leftPoint.parentArrow) {
            var _rightPoint = leftPoint.parentArrow.rightPoint;
            _rightPoint.removeArrow(leftPoint.parentArrow);
            leftPoint.parentArrow.destroy();
        }

        var i;
        for (i = 0; i < rightPoint.arrows.length; i++) {
            var arrow = rightPoint.arrows[i];
            if (arrow && arrow.leftPoint) {
                arrow.leftPoint.removeArrow(arrow);
            }
            arrow.destroy();
        }
        rightPoint.arrows.splice(0, rightPoint.arrows.length);

        parent = null;
    }

    Component.onDestruction: {
    }
}
```

ToolsContainer.qml

```
import QtQuick 2.4
import PyConsole 1.0
import "../Base"

FormObject {
    property string title: "Title" // название контейнера, расположено левом верхнем
углу
    color: '#BFBA30'

    Text {
        id: titleText
        text: title
        font.pointSize: 14
    }
}
```

MainForm.qml

```
import QtQuick 2.4
import QtQuick.Dialogs 1.2
```

```
import PyConsole 1.0
import "Buttons"
import "Animations"
import "Tools"
import "Tools/Actions"
import "Tools/Composites"
import "Tools/Decorators"
import Generators 1.0

Rectangle {
    PyConsole {
        id: pyconsole
    }

    property string toolsPanelColor: '#00A383'
    property string controlsPanelColor: '#34D1B2'
    property string canvasPanelColor: '#5ED1BA'
    property string canvasPanelDropColor: '#1F7A68'
    property real toolsWidth: 200

    id: wrapper
    width: 1024
    height: 600

    Row {
        x: 0; y: 0

        Column {
            Rectangle {
                id: toolsPanel
                width: toolsWidth
                height: wrapper.height - controlsPanel.height
                color: toolsPanelColor

                property real distanceToCanvasX: toolsWidth - 10

                Column {
                    spacing: 10
                    anchors.horizontalCenter: parent.horizontalCenter
                    ToolsContainer {
                        id: compositesToolsContainer
                        title: 'Composites'
                        height: 130
                        width: toolsWidth - parent.spacing * 2

                        Sequence {
                            x: 10; y: 25
                            canvasDX: toolsPanel.distanceToCanvasX
                            canvas: canvas
                        }

                        Selector {
                            x: 60; y: 25
                            canvasDX: toolsPanel.distanceToCanvasX
                            canvas: canvas
                        }

                        Parallel {
                            x: 10; y: 75
                            canvasDX: toolsPanel.distanceToCanvasX
                            canvas: canvas
```

```
                    }

                    MemSelector {
                        x: 60; y: 75
                        canvasDX: toolsPanel.distanceToCanvasX
                        canvas: canvas
                    }

                    MemSequence {
                        x: 110; y: 25
                        canvasDX: toolsPanel.distanceToCanvasX
                        canvas: canvas
                    }
                }

                ToolsContainer {
                    id: actionsToolsContainer
                    title: 'Actions'
                    height: 100
                    width: toolsWidth - parent.spacing * 2
                    property int realPositionY: compositesToolsContainer.height +
parent.spacing

                    UserAction {
                        x: 10; y: 25
                        canvasDX: toolsPanel.distanceToCanvasX
                        canvasDY: -actionsToolsContainer.realPositionY
                        canvas: canvas
                    }

                    Wait {
                        x: 60; y: 25
                        canvasDX: toolsPanel.distanceToCanvasX
                        canvasDY: -actionsToolsContainer.realPositionY
                        canvas: canvas
                    }

                    Condition {
                        x: 110; y: 25
                        canvasDX: toolsPanel.distanceToCanvasX
                        canvasDY: -actionsToolsContainer.realPositionY
                        canvas: canvas
                    }
                }

                ToolsContainer {
                    id: decoratorsToolsContainer
                    title: 'Decorators'
                    height: 130
                    width: toolsWidth - parent.spacing * 2
                    property int realPositionY:
actionsToolsContainer.realPositionY + actionsToolsContainer.height + parent.spacing

                    Inverter {
                        x: 10; y: 25
                        canvasDX: toolsPanel.distanceToCanvasX
                        canvasDY: -decoratorsToolsContainer.realPositionY
                        canvas: canvas
                    }

                    AlwaysSuccess {
```

```
                x: 60; y: 25
                canvasDX: toolsPanel.distanceToCanvasX
                canvasDY: -decoratorsToolsContainer.realPositionY
                canvas: canvas
            }

            AlwaysFail {
                x: 110; y: 25
                canvasDX: toolsPanel.distanceToCanvasX
                canvasDY: -decoratorsToolsContainer.realPositionY
                canvas: canvas
            }

            UntilSuccess {
                x: 10; y: 75
                canvasDX: toolsPanel.distanceToCanvasX
                canvasDY: -decoratorsToolsContainer.realPositionY
                canvas: canvas
            }

            UntilFail {
                x: 60; y: 75
                canvasDX: toolsPanel.distanceToCanvasX
                canvasDY: -decoratorsToolsContainer.realPositionY
                canvas: canvas
            }

            Repeat {
                x: 110; y: 75
                canvasDX: toolsPanel.distanceToCanvasX
                canvasDY: -decoratorsToolsContainer.realPositionY
                canvas: canvas
            }

        }
    }
}

Rectangle {
    id: controlsPanel
    color: controlsPanelColor
    width: toolsWidth
    height: 150

    Column {
        spacing: 10
        anchors.centerIn: parent
        NormalTextButton {
            text: "Generate"
            width: toolsWidth - 10

            onClicked: {
                generateCodeFileDialog.open();
            }
        }

        NormalTextButton {
            text: "Export JSON"
            width: toolsWidth - 10

            onClicked: {
```

```
                        exportJsonFileDialog.open();
                    }
                }

                NormalTextButton {
                    text: "Import JSON"
                    width: toolsWidth - 10
                }
            }
        }
    }


    Rectangle {
        id: canvasRectangle
        color: canvasPanelColor
        width: wrapper.width - toolsWidth
        height: wrapper.height
        z: -1

        Canvas {
            id: canvas
            anchors.fill: parent

            Root {
                id: rootNode
                x: 10
                y: parent.height/2 - height/2
                isPrototype: false
                canvas: canvas
            }
        }

        DropArea {
            id: canvasDrop
            anchors.fill: parent
            keys: ['tool', ]
        }

        Behavior on color {
            NormalColorAnimation {}
        }

        states: [
            State {
                when: canvasDrop.containsDrag
                PropertyChanges {
                    target: canvasRectangle
                    color: canvasPanelDropColor
                }
            }
        ]
    }
}

FileDialog {
    id: exportJsonFileDialog
    title: "Выберете файл для сохранения дерева в json-формате"
    selectExisting: false
    selectFolder: false
    selectMultiple: false
```

```
        nameFilters: ["Json Files (*.json)", ]

        onAccepted: {
            jsonGenerator.fileName = fileUrl;
            jsonGenerator.rootNode = rootNode;
            jsonGenerator.generate();
        }
    }

    FileDialog {
        id: generateCodeFileDialog
        title: "Выберете файл для сохранения дерева в java коде"
        selectExisting: false
        selectFolder: false
        selectMultiple: false
        nameFilters: ["Java (*.java)", ]

        onAccepted: {
            jsonGenerator.fileName = fileUrl;
            jsonGenerator.rootNode = rootNode;
            jsonGenerator.generateCode();
        }
    }

    JsonGenerator {
        id: jsonGenerator
    }
}
```

visionai.qml

```
import QtQuick 2.4
import Curves 1.0

MainForm {}
```

BezierCurve.py

```
from PyQt5.QtQuick import QQuickPaintedItem, QQuickItem
from PyQt5.QtGui import QColor, QPainter, QPainterPath, QPen
from PyQt5.QtCore import pyqtProperty, QRectF, pyqtSignal, Qt
from docutils.nodes import paragraph


class BezierCurve(QQuickPaintedItem):
    """Класс представляет кривую безье, задаваемою двумя точками, толщиной, цветом и
двумя флагами на наличие стартовой
    и концевой стрелочек.
    """
    def __init__(self, parent: QQuickItem=None):
        super(BezierCurve, self).__init__(parent)

        self._color = QColor()
        self._startX = 0
        self._startY = 0
        self._endX = 0
        self._endY = 0
        self._curveWidth = 1
```

```
        self._startArrow = False
        self._endArrow = False

    def updateRectSize(self):
        parent = self.parent()
        if parent is not None:
            self.setWidth(parent.width())
            self.setHeight(parent.height())

    colorChanged = pyqtSignal()

    @pyqtProperty(QColor, notify=colorChanged)
    def color(self):
        return self._color

    @color.setter
    def color(self, color):
        if self._color != color:
            self._color = QColor(color)
            self.colorChanged.emit()
            self.update()

    startXChanged = pyqtSignal()

    @pyqtProperty(float, notify=startXChanged)
    def startX(self):
        return self._startX

    @startX.setter
    def startX(self, startX):
        if self._startX != startX:
            self._startX = startX
            self.startXChanged.emit()
            self.updateRectSize()
            self.update()

    startYChanged = pyqtSignal()

    @pyqtProperty(float, notify=startXChanged)
    def startY(self):
        return self._startY

    @startY.setter
    def startY(self, startY):
        if self._startY != startY:
            self._startY = startY
            self.startYChanged.emit()
            self.updateRectSize()
            self.update()

    endXChanged = pyqtSignal()

    @pyqtProperty(float, notify=endXChanged)
    def endX(self):
        return self._endX

    @endX.setter
    def endX(self, endX):
        if self._endX != endX:
            self._endX = endX
            self.endXChanged.emit()
```

```
            self.updateRectSize()
            self.update()

    endYChanged = pyqtSignal()

    @pyqtProperty(float, notify=endYChanged)
    def endY(self):
        return self._endY

    @endY.setter
    def endY(self, endY):
        if self._endY != endY:
            self._endY = endY
            self.endYChanged.emit()
            self.updateRectSize()
            self.update()

    curveWidthChanged = pyqtSignal()

    @pyqtProperty(int, notify=curveWidthChanged)
    def curveWidth(self):
        return self._curveWidth

    @curveWidth.setter
    def curveWidth(self, curveWidth):
        if self._curveWidth != curveWidth:
            self._curveWidth = curveWidth
            self.curveWidthChanged.emit()
            self.update()

    startArrowChanged = pyqtSignal()

    @pyqtProperty(bool, notify=startArrowChanged)
    def startArrow(self):
        return self._startArrow

    @startArrow.setter
    def startArrow(self, startArrow):
        if self._startArrow != startArrow:
            self._startArrow = startArrow
            self.startArrowChanged.emit()
            self.update()

    endArrowChanged = pyqtSignal()

    @pyqtProperty(bool, notify=endArrowChanged)
    def endArrow(self):
        return self._endArrow

    @endArrow.setter
    def endArrow(self, endArrow):
        if self._endArrow != endArrow:
            self._endArrow = endArrow
            self.endArrowChanged.emit()
            self.update()

    @staticmethod
    def add_arrow_left(x, y, painter_path: QPainterPath):
        painter_path.moveTo(x, y)
        painter_path.lineTo(x + 5, y + 5)
        painter_path.moveTo(x, y)
```

```
        painter_path.lineTo(x + 5, y - 5)

    @staticmethod
    def add_arrow_right(x, y, painter_path: QPainterPath):

        painter_path.moveTo(x, y)
        painter_path.lineTo(x - 5, y + 5)
        painter_path.moveTo(x, y)
        painter_path.lineTo(x - 5, y - 5)

    def add_arrow(self, x, y, direction, painter_path: QPainterPath):
        if direction == 'l':
           self.add_arrow_left(x, y, painter_path)
        elif direction == 'r':
            self.add_arrow_right(x, y, painter_path)

    def paint(self, painter: QPainter):
        painter_path = QPainterPath()
        painter_path.moveTo(self._startX, self._startY)
        x1 = (7 * self._endX + self._startX) / 8
        y1 = self._startY
        x2 = (self._endX + 7 * self._startX) / 8
        y2 = self._endY
        painter_path.cubicTo(x1, y1, x2, y2, self._endX, self._endY)

        if self._startArrow:
            self.add_arrow(self._startX, self._startY, 'l' if self._startX <=
self._endX else 'r',painter_path)

        if self._endArrow:
            self.add_arrow(self._endX, self._endY,  'r' if self._startX <= self._endX
else 'l',painter_path)

        pen = QPen(self._color, self._curveWidth, Qt.SolidLine, Qt.RoundCap,
Qt.RoundJoin)
        painter.setPen(pen)
        painter.setRenderHints(QPainter.Antialiasing, True)
        painter.drawPath(painter_path)
```

JsonGenerator.py

```
import json
import ast

from PyQt5.QtQuick import QQuickItem
from PyQt5.QtCore import pyqtSignal, pyqtProperty, pyqtSlot
from PyQt5.QtQml import QQmlProperty

class JsonGenerator(QQuickItem):

    def __init__(self, parent=None):
        super(JsonGenerator, self).__init__(parent)

        self._fileName = ''
        self._rootNode = None

        self.name_to_method = {

##############################################################################
```

```
######################
            #                                JSON

#############################################################################################
######################
            '_generateSequence': self._generateSequence, '_generateInverter':
self._generateInverter,
            '_generateMemSelector': self._generateMemSelector,
'_generateMemSequence': self._generateMemSequence,
            '_generateParallel': self._generateParallel, '_generateSelector':
self._generateSelector,
            '_generateUserAction': self._generateUserAction, '_generateWait':
self._generateWait,


#############################################################################################
######################
            #                                CODE

#############################################################################################
######################
            '_generateCodeSequence': self._generateCodeSequence,
'_generateCodeInverter': self._generateCodeInverter,
            '_generateCodeMemSelector': self._generateCodeMemSelector,
'_generateCodeMemSequence': self._generateCodeMemSequence,
            '_generateCodeParallel': self._generateCodeParallel,
'_generateCodeSelector': self._generateCodeSelector,
            '_generateCodeUserAction': self._generateCodeUserAction,
'_generateCodeWait': self._generateCodeWait,
        }

    def __getitem__(self, item):
        return self.name_to_method.get(item)

    fileNameChanged = pyqtSignal()

    @pyqtProperty(str, notify=fileNameChanged)
    def fileName(self):
        return self._fileName

    @fileName.setter
    def fileName(self, fileName):
        if self._fileName != fileName:
            self._fileName = fileName
            self.fileNameChanged.emit()

    rootNodeChanged = pyqtSignal()

    @pyqtProperty(QQuickItem, notify=rootNodeChanged)
    def rootNode(self):
        return self._rootNode

    @rootNode.setter
    def rootNode(self, rootNode):
        if self._rootNode != rootNode:
            self._rootNode = rootNode
            self.rootNodeChanged.emit()

    @pyqtSlot()
    def generate(self):
        file = open(self._fileName[8:], 'w')
```

```
        s = self._generateStr()
        file.write(s)
        file.close()

    def _generateStr(self):
        entry = self._generateJson(self._rootNode, 0)
        return entry

    @pyqtSlot()
    def generateCode(self):
        s = self._generateStr()
        # d = json.loads(s.replace("'", '"'))
        d = ast.literal_eval(s.replace("'", '"'))
        c = self._generateCode(d, 3)
        file = open(self._fileName[8:], 'w')
        file.write(
"""
import org.nikialeksey.gameengine.ai.behaviortree.*;
import org.nikialeksey.gameengine.ai.behaviortree.Actions.*;
import org.nikialeksey.gameengine.ai.behaviortree.Composites.*;
import org.nikialeksey.gameengine.ai.behaviortree.Decorators.*;

class BehaviorTreeBuilder {
    public static BehaviorTree buildBehaviorTree() {
        Node root = \n""" + c + """;
        BehaviorTree behaviorTree = new BehaviorTree(root);
        return behaviorTree;
    }
}
"""
        )
        file.close()

    def _generateCode(self, entry: dict, offset):
        s = (' ' * offset * 4) + self['_generateCode' + entry['name']](entry)
        was_iter = False
        for _entry in entry.get('children', []):
            s += (',\n' if was_iter else '') + self._generateCode(_entry, offset + 1)
            was_iter = True
        s += '\n' + (' ' * offset * 4) + ')'
        return s

    def _generateJson(self, node: QQuickItem, offset: int) -> str:
        s = (' ' * (offset) * 4) + '{\n'
        s += (' ' * (offset + 1) * 4) + self['_generate' + QQmlProperty.read(node,
'actualName')](node)

        s += (' ' * (offset + 1) * 4) + "'x': " + str(node.x()) + ',\n'
        s += (' ' * (offset + 1) * 4) + "'y': " + str(node.y()) + ',\n'

        s += (' ' * (offset + 1) * 4) + "'children': [\n"
        rightPoint = QQmlProperty.read(node, '_rightPoint')
        if rightPoint is not None:
            try:
                arrows = QQmlProperty.read(rightPoint, 'arrows')
                if arrows.isArray():
                    for i in range(0, arrows.property('length').toInt()):
                        arrow = arrows.property(i).toQObject()
                        leftPoint = QQmlProperty.read(arrow, 'leftPoint')
                        s += self._generateJson(leftPoint.parent(), offset + 2) +
',\n'
```

```
            except Exception as e:
                print(e)
        s += (' ' * (offset + 1) * 4) + '],\n'
        s += (' ' * (offset) * 4) + '}'
        return s


################################################################################
################################
    #                   SEQUENCE

################################################################################
################################
    def _generateSequence(self, node: QQuickItem) -> str:
        s = "'name': 'Sequence',\n"

        return s

    def _generateCodeSequence(self, entry: dict) -> str:
        s = "new Sequence(\n"

        return s


################################################################################
################################
    #                   SELECTOR

################################################################################
################################
    def _generateSelector(self, node: QQuickItem) -> str:
        s = "'name': 'Selector',\n"

        return s

    def _generateCodeSelector(self, entry: dict) -> str:
        s = "new Selector(\n"

        return s


################################################################################
################################
    #                   PARALLEL

################################################################################
################################
    def _generateParallel(self, node: QQuickItem) -> str:
        s = "'name': 'Parallel',\n"

        return s

    def _generateCodeParallel(self, entry: dict) -> str:
        s = "new Parallel(\n"

        return s


################################################################################
################################
    #                   MEMSEQUENCE
```

```
################################################################################
################################
    def _generateMemSequence(self, node: QQuickItem) -> str:
        s = "'name': 'MemSequence',\n"

        return s

    def _generateCodeMemSequence(self, entry: dict) -> str:
        s = "new MemSequence(\n"

        return s


################################################################################
################################
    #                   MEMSELECTOR

################################################################################
################################
    def _generateMemSelector(self, node: QQuickItem) -> str:
        s = "'name': MemSelector',\n"

        return s

    def _generateCodeMemSelector(self, entry: dict) -> str:
        s = "new MemSelector(\n"

        return s


################################################################################
################################
    #                   USERACTION

################################################################################
################################
    def _generateUserAction(self, node: QQuickItem) -> str:
        s = "'name': 'UserAction',\n"

        return s

    def _generateCodeUserAction(self, entry: dict) -> str:
        s = "new UserAction(\n"

        return s


################################################################################
################################
    #                   WAIT

################################################################################
################################
    def _generateWait(self, node: QQuickItem) -> str:
        s = "'name': 'Wait',\n"

        return s

    def _generateCodeWait(self, entry: dict) -> str:
        s = "new Wait(\n"
```

```
        return s


##########################################################################################
##############################
    #                   INVERTER

##########################################################################################
##############################
    def _generateInverter(self, node: QQuickItem) -> str:
        s = "'name': 'Inverter',\n"

        return s

    def _generateCodeInverter(self, entry: dict) -> str:
        s = "new Inverter(\n"

        return s
```

main.py

```
from PyQt5.QtCore import QUrl, QObject, pyqtSlot
from PyQt5.QtGui import QGuiApplication
from PyQt5.QtQuick import QQuickView
from PyQt5.QtQml import qmlRegisterType

class ConsoleOutput(QObject):
    def __init__(self, parent=None):
        super(ConsoleOutput, self).__init__(parent)

    @pyqtSlot(str)
    def out(self, obj):
        print(obj)

if __name__ == '__main__':
    import sys
    from Curves import BezierCurve
    from Generators import JsonGenerator

    app = QGuiApplication(sys.argv)

    qmlRegisterType(ConsoleOutput, 'PyConsole', 1, 0, 'PyConsole')
    qmlRegisterType(BezierCurve, 'Curves', 1, 0, 'BezierCurve')
    qmlRegisterType(JsonGenerator, 'Generators', 1, 0, 'JsonGenerator')

    view = QQuickView()
    context = view.rootContext()

    view.engine().quit.connect(app.quit)
    view.setResizeMode(QQuickView.SizeRootObjectToView)
    view.setSource(
        QUrl.fromLocalFile('../view/visionai.qml')
    )
    view.show()

    for error in view.errors():
        print(error.toString())
```

```
status = app.exec_()

for error in view.errors():
    print(error.toString())

sys.exit(status)
```