**1 ОБЗОР ЛИТЕРАТУРЫ**

* 1. **Обзор существующих аналогов**

На сегодняшний день существует огромное количество игр, в которых в том или ином виде представлен развлекательный игровой искусственный интеллект. Различаются лишь подходы к реализации развлекательной задачи. Ниже будут представлены три примера наиболее похожих на создаваемый искусственный интеллект. Первые два будут схожи по цели, но с различной реализацией в игре. При этом эти две игры со схожим названием, развивающими одну игровую вселенную. С помощью третьего примера будет показано то, каким создают искусственный интеллект в многопользовательских играх с одиночной компанией.

**1.1.1 DooM 1993**

Игра Doom была создана в 1993 года компанией id Software. Она является одной из самых значимых игр в истории индустрии. Именно она определила вектор развития шутеров от первого лица[x]. В ней игрок поочередно исследует уровни-комнаты разной сложности, решает различные загадки для дальнейшего прохождения, может находить тайники и, конечно, уничтожать монстров. Углубляться в устройство уровней не имеет смысла, так как это не является темой данного дипломного проекта и не связано с искусственным интеллектом. Противники не могут пользоваться окружением, что обусловлено недостаточной вычислительной мощностью техники того времени.

Стоит указать возможные действия со стороны игрока, связанные с взаимодействием с противниками. Предшественником данной игры был Wolfenstein 3D, но искусственный интеллект был значительно ослаблен. Например, пропала возможность противников позвать на помощь из другой комнаты или попытаться зайти игроку за спину.

Основным недостатком, за который, однако не стоит винить саму игру, являются, как и написано выше, невозможность использования окружения, кооперации противников для достижения цели. Также примитивность данного искусственного интеллекта не предполагает какую-либо очередность атаки противников, что не оставит игроку шанса на победу. Более подробную информацию про искусственный интеллект можно получить в источнике[x].

**1.1.2 DooM 2016**

Данная уже мультиплатформенная игра по той же вселенной, что и DooM 1993 года, разработана компанией id Software совместно со студией Certain Affinity и издана Bethesda Softworks. Вышла 13 мая 2016 года на Windows, Xbox One и PlayStation 4.

В данной версии игры разработчики старались учесть все недочеты прошлых игр этой игровой вселенной. Основной задумкой разработчиков являлось то, что игра является шутером от первого лица. А это означало для них, что она должна быть динамичнее, чем обычные игры. Динамичность игры достигалась за счет особого подхода к прохождению игры. Игрок для более интересного и относительно простого прохождения не должен был стоять на месте. Иначе противники начинали точнее стрелять, разумеется, также приближаться к стоящему игроку и атаковать. Тут стоит отметить, что, в отличие от DooM 1993 года, противникам добавляли ограничения на одновременную стрельбу, что одновременно и упрощало игру и делало ее более приятной для прохождения. Как отмечалось ранее, искусственный интеллект, если не ограничивать его, станет непроходимым препятствием и вряд ли будет интересен при прохождении. Более подробно про искусственный интеллект в данной игре можно узнать из источника[x].

Так же, хоть это и не напрямую относится к основной логике интеллекта, было значительно увеличено количество анимаций за счет динамического скелета персонажей. Это делает игру более живой с точки зрения наполнения. Это косвенно, но помогает улучшить противников и также улучшить восприятие игры.

**1.1.3 Star wars Battlefront II**

Существует несколько игр с таким названием, однако речь пойдет про игру, разработанную в 2017 году. Расчет в данной игре сделан на многопользовательский режим. В нем существует несколько видов персонажей, за которых может сыграть игрок или который может управляться искусственным интеллектом. Специализации каждого из них уникальны, так как каждый класс заточен под особую задачу. Стоит отметить, что искусственный интеллект в данной игре опять же создан прежде всего для развлечения игроков. Искусственный интеллект, как говорят разработчики, придает сражениям на планетах ощущение масштабности за счет увеличения количества бойцов, одновременно играющих на одной карте.

Игрок, встречающий таких ботов, управляемых искусственным интеллектом, может почувствовать себя более существенным на поле боя и так или иначе повысить настроение играющему. Внедрение таких ботов позволило веселиться без необходимости соперничества с другими людьми. Как пишет разработчик: «Визуальные скрипты позволяют ставить перед ИИ следующие цели: идти, защищать, атаковать, взаимодействовать, использовать, искать и уничтожить, а также следовать. В конечном итоге мозг ИИ ищет правильные «клавиши» — стрельба, смещение, рыскание, наклон, прыжок и так далее — для каждого кадра. Это, как если бы робот играл на контроллере и нажимал физические кнопки — только на концептуальном уровне. В случае подключения к игровому движку этот инструмент становится крайне универсальным и может использоваться почти в любой игре…».

В случае данного искусственного интеллекта, который старается симулировать самого игрока, а не просто развлекать своим присутствием, сильно увеличивается вовлеченность реальных игроков. В первую очередь потому, что в игре с такой динамичностью потребуется либо большой опыт игры, либо очень большие старания чтобы просто различить реального человека и бота, управляющего персонажем. Это, как и говорилось выше, повышает ощущение массовости при боях и ощущение того, что игрок является «героем».

Разумеется, в игре присутствует и одиночный режим, где используются боты с таким искусственным интеллектом. Их использование в повествовании сюжета никак не отягощает прохождение самой игры, ведь ощущение живости в совокупности с использованием отличной графики окружения, использованием достаточного количества классов персонажей в игре, приводит к тому, что игрок захочет играть в игру больше. Это и является целью искусственного интеллекта в конечном счете – увеличивать шанс возвращения людей в игру.

* 1. **Unreal Engine 4**

**1.2.1 Игровой движок**

Unreal Engine 4 – игровой движок[x], который разрабатывается компанией Epic Games. Впервые был выпущен в 1998 году и изначальным предназначением которого являлось создание игр от первого лица. После разработки дальнейших версий, стал применяться для создания игр различного жанра.

Благодаря данному базовому программному обеспечению и большому количеству людей, создающих игры на нем, разработчик может при относительно малых затратах во времени на изучение всех аспектов разработки начать создавать игру. При этом не обязательно знать многие вещи связанные с созданием окружения, звуковых эффектов и графического содержимого игры. Разработку также облегчает наличие свободно распространяемых материалов для создания игры. В данном дипломном проекте они также будут использоваться в связи с недостаточным временем, выделенным на разработку, а значит и на изучение движка. Однако вещи, косвенно связанные с разработкой искусственного интеллекта или созданием того, с чем может взаимодействовать персонаж, управляемый им, будут по возможности создаваться без использования готовых решений.

**1.2.2 Система визуальных сценариев Blueprint**

Система визуальных сценариев Blueprint в Unreal Engine — это полноценная система сценариев игрового процесса, основанная на концепции использования интерфейса, где за основу взяты узлы для создания элементов игрового процесса из Unreal Editor. Как и многие распространенные языки сценариев, он используется для определения объектно-ориентированных классов или объектов в движке.

Данный подход позволяет разработчику использовать все инструменты, обычно доступные лишь программистам. Стоит отметить, что количество возможных ошибок, которые может допустить разработчик при создании проекта, сводится к минимуму. Это достигается тем, что следить за тем, что происходит в логике задачи легче, нежели в привычных языках программирования.

Существует несколько типов blueprint, создание каждого из которых преследует различные цели. Далее будут кратко описаны типы: blueprint class, data-only blueprint, level blueprint, blueprint interface, blueprint macro library и blueprint utilities.

Blueprint class, обычно используется сокращение до Blueprint. В общем случае, это добавление функционала для уже существующих классов в игровом процессе. Создаются обычно визуально, что упоминалось ранее, а не путем ввода кода. Они определяют новый класс или тип Actor для последующего размещения на сцене как экземпляры, которые будут вести себя, как и другие экземпляры типа Actor, за тем лишь исключением, что добавленная логика скорее всего добавляет функционал, расширяющий его возможности на карте.

Data-only blueprint является классом Blueprint, содержащий только код, опять же оформленный в виде графов узлов, необходимые переменные и компоненты, унаследованные от родителя. Данные тип позволяет только настраивать и изменять уже существующее, но не добавлять новые элементы.

Level blueprint действует как глобальный график событий уровня. Все уровни в проекте имеют свой план уровня, который создается по умолчанию, его можно редактировать, но новые level blueprint не получится создать через интерфейс редактора. Все события, которые имеют отношение к уровню или экземплярам объектов, типа Actor, будут использоваться для запуска последовательностей действий в виде вызовов функций или операций по управлению потоком. Такие чертежи предоставляют механизмы для управления потоковой передачи уровней и Sequencer и имеют привязки событий к экземплярам классов типа Actor, которые размещены на уровне.

Blueprint interface представляет собой набор одной или нескольких функций без реализации, которые можно добавлять в другие чертежи. Это похоже на идею интерфейсов в общем программировании, позволяющая различным типам объектов взаимодействовать между собой. При этом стоит учитывать, что добавление новых компонентов, переменных или изменение графов невозможно в интерфейсах.

Blueprint macro library является библиотекой, где можно создавать часто используемые функции. Они могут использоваться в других чертежах. Подробно описывать данный тип чертежей не имеет смысла из-за чрезвычайной схожести с библиотеками в языках программирования.

Blueprint utility или сокращенно blutility, используется только для редактора, используются для выполнения задач редактором или простого расширения функционала того же редактора.

Так как не представляется возможным показать все основные узлы на чертежах, ниже будет кратко описан принцип программирования используя blueprint. Это будет показано на примере оператора перехода. Его логика проста для понимания и отлично подходит для объяснения основных принципов. У данного узла есть 2 входа, один из которых является исполняемым, а второй используется для выбора задействования выходов, типом данного входа является Boolean. Как и в других языках программирования, тип Boolean может иметь два значения, правда или ложь. При этом, для тестирования или, в случае использования других узлов, можно выставлять константные значения, посылаемые на входы узлов. Исполняемый вход используется для построения самой логики, все так или иначе сводится к последовательному исполнению кода, что означает, что исполняемые выходы узлов можно соединять с исполняемыми входами других узлов, с помощью чего и строятся функции и иные конструкции в blueprint. Разумеется, стоит учитывать, что пусть читабельность написанного таким образом кода относительно выше по сравнению с кодом написанным, например, на языке С++, который будет описан в пункте 1.2.3, в некоторых случаях количество ведущих в узлы переменных может достигать большого количества. Что стоит учитывать при создании функций.

Рассмотрение всех типов переменных в данном дипломном проекте не будет производится в связи с тем, что количество использованных встроенных типов данных слишком велико и не сможет быть описано в пояснительной записке. Ознакомится с документацией к blueprint можно в источнике[x].

**1.2.3 Язык программирования C++**

В разрабатываемом проекте данный способ разработки почти не использовался, но изучался как альтернативный способ создания игровых персонажей, написания необходимых для чертежей функций и прочего. Главной причиной является то, что при переносе проекта с использованием средств разработки на С++ на новые версии игрового движка могут возникнуть дополнительные трудности, которые скорее всего не смогут решиться автоматически. Но стоит отметить, что после точного определения для какой версии игрового движка будет создаваться приложение, появилась возможность использовать некоторый бесплатный контент. Примером такого контента в разрабатываемом проекте является встраиваемый плагин для нахождения пути в пространстве. Он используется в первую очередь для повышения интеллекта на данный момент единственного представленного в игре летающего персонажа, дрона. Плагин был разработан сторонним разработчиком и предоставлен для использования в библиотеке в приложении компании Epic Games на бесплатной основе. Написан данный плагин как раз с помощью средств разработки на языке С++. Написание некоторых функций и плагинов полностью на основе чертежей может составить значительные трудности как в плане отладки, так и трудностей, связанных со временем разработки.

Стоит отметить, что программирование на C++ в Unreal Engine 4 имеет свои особенности. К сожалению, в связи с огромным количеством особенностей и, как следствие, невозможностью описать их, а так же по причине малого опыта разработки на данном языке программирования, описание в записке приводится не будет. Ознакомится с основами, ключевыми понятиями и особенностями разработки, можно в источнике[x].

**1.2.4 BSP геометрия**

Geometry brushes один из удобнейших инструментов для проектирования уровней внутри Unreal Engine. Напрямую с разработкой искусственного интеллекта они не связаны, но для возможности интеллекта корректно пользоваться окружением и для упрощения проектирования большого количества игровых локаций разработчиком, знание BSP геометрии необходимо. Основной задачей такой геометрии является прежде всего создание именно начального уровня, его геометрии. Данный инструмент больше подходит для создания объектов, способствующих ускоренному тестированию механик игры. Прототип уровня может претерпевать множество правок, а изменение уровня после создания готовой модели затруднено или вовсе невозможно без дополнительных расширений.

Создание брашей происходит простым перетягиванием на сцену необходимого примитива. Существует 6 исходных примитивов, которыми при разработке данного дипломного проекта пришлось ограничиваться для создания прототипов уровней.

Самым простым и часто используемым примитивом является куб. У него есть 6 главных настроек, связанных с его основной геометрией. Среди них размеры по осям X, Y, Z, толщина стенок, которая работает только при включенной опции Hollow, сама опция Hollow, позволяющая создавать полости в кубе и настройка Tessellated, которая позволяет разделять стороны куба на треугольники или квадраты в зависимости от выбранной опции.

Вторым примитивом является конус. У него есть такие настройки, как высота по оси Z. Функция Hollow, кратко описанная выше. Cap Z – высота внутренней области, работает при включенной опции Hollow. Inner и Outer Radius, радиусы основания конуса, при этом Inner radius будет учтен опять же только при включенной функции Hollow. Sides – количество сторон конуса, так как конус не может быть идеально гладким из-за невозможности отрисовки. Align to side – настройки выравнивания объекта.

У цилиндра присутствуют настройки высоты, внутреннего и внешнего радиуса, количество сторон, причина такой настройки коротко объяснена в описании конуса. Так же присутствует функция Hollow и Align to side тоже описанные выше.

Также существует три типа примитивов лестниц различных по настройкам и, соответственно, по применению. Первый тип примитива лестниц – простая прямая лестница. Настройки данного примитива ограничены и представляют собой набор констант, определяющих размеры ступенек, их количество и сколько добавлять к первой ступени. Константы, которые позволяют редактирование ступеней – глубина, высота и ширина одной ступени.

Существует также изогнутая лестница, в ней можно изменять радиус внутреннего цилиндра, вокруг которого образуется лестница. Помимо описанных настроек для прямой лестницы, присутствует характерные только изогнутой лестнице настройки: угол поворота лестницы и обратное вращение лестницы.

Последним типом лестниц является спиральная лестница, помимо настроек, существующих у предыдущих видов, присутствует четыре дополнительные настройки, такие как толщина ступени, возможность плавного спуска, возможность установки гладкой поверхности под ступенями и количество ступеней на полный оборот лестницы.

Последним рассматриваемым примитивом является сфера. Настройка данного примитива проста, так как из встроенных опций есть всего две: радиус и управление количеством сторон самой сферы.

При это всем не получится объединять несколько брашей для создания одного объекта. При создании геометрии, перед перетягиванием на сцену необходимого примитива, можно указать тип браша – добавить или вычесть. При добавлении он добавится на сцену, а при вычитании, он будет вычитать из других объектов пересекаемый с ним объем подобно булевой операции. Помимо выбора типа браша можно устанавливать приоритетность его типа, что может помочь при создании более сложных обьектов.

После создания примитива перетаскиванием его на сцену, появляется возможность изменять его, перетягивая грани, точки или стороны примитива в стороны. Это позволяет сосредоточится на создании проработанного окружения, а не на точности постановки объекта на уровне. После того, как нужные объекты из BSP геометрии расставлены на уровне, есть возможность создать из них статические объекты. Это становится необходимым, если, например, есть необходимость использовать такой же объект несколько раз в игре или при большом количестве геометрии. Большое количество брашей может привести к уменьшению производительности из-за расчета процессором самой геометрии на карте.

**1.2.5 Искусственный интеллект в Unreal Engine 4**

При создании игр часто приходится писать искусственный интеллект для нее. В Unreal Engine 4 присутствуют встроенные классы, функции, макросы и функционал, в целом облегчающий написание интеллекта для игры. Краткое описание того, что необходимо знать для создания и последующей разработки искусственного интеллекта с помощью средств Unreal Engine 4 будет представлено ниже.

Первое, что стоит описать является AIController. Это нефизический Actor, который может контролировать персонажа. С его помощью можно передавать информацию самому персонажу. Если проводить аналогию, то контроллер это голова персонажа, контролируемого искусственным интеллектом. В контроллере принято писать логику, отвечающую за нахождение персонажа, такую как зрение, слух и прочие чувства, которые разработчик сочтет необходимым добавить для улучшенного восприятия игроком интеллекта. Подробная настройка контроллера не приводится по причине ее ситуативности при разработке. Функции, которые могут помочь в написании искусственного интеллекта могут использоваться в контроллере, но это не приветствуется для написания относительно продуманных персонажей. При дальнейшей разработке их использование в контроллере может замедлить разработку и увеличить сложность самого алгоритма.

При создании качественных персонажей, контролируемых искусственным интеллектом, принято использовать Behavior tree. Программирование интеллекта в нем упрощено за счет еще лучшего разделения поведения на простые задачи. В дереве поведения бывает четыре типа узлов, называемых по-другому нодов. Первые два – задачи и композиты. При создании Behavior tree в центре экрана уже будет добавлен композит Root. Его нельзя переопределить, удалить или изменить. Это главный корень исполняемой логики, стартовая точка дерева поведений. Единственное, что необходимо сделать для начала выполнения написанного алгоритма – выбрать необходимое дерево поведений и запустить его из контроллера. Обычно это делается при начале игры или сразу после получения контроля над персонажем.

Задачи – узлы, за которыми закреплена какая-либо логика, от простого ожидания некоторого времени на месте до цепочек исполняемых задач. Само дерево поведения требует использования композитов. Оно, дерево, может состоять из огромного множества ветвей, иначе называемых поведениями. В корне, над задачами в ветвях, есть корень, композит. Композит в общем случае выбирает задачу, которая будет исполнена. У главного корня в дереве есть только один выход, от которого может идти сколь угодно большое количество ветвей. Задачи являются конечной точкой, после них нельзя закрепить композитов или других задач ниже. Стоит помнить, что исполнение будет начинаться слева направо, это значит, что в дереве поведений учитывается расположение задач и композитов. Очередность выполнения задач для удобства пишется в правом верхнем углу задач и композитов. На рисунке 1.1 показан простой пример создания дерева поведения.

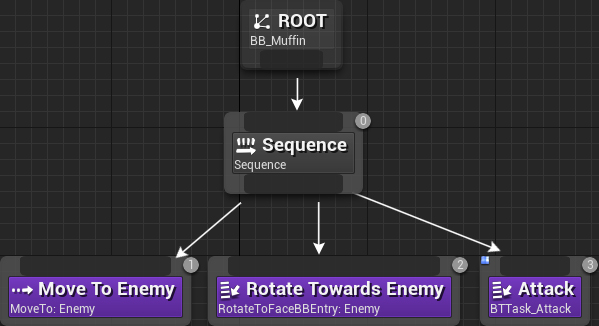


Рисунок 1.1 – Oчередность исполнения задач в Behavior tree

В стандартных деревьях поведений есть уже созданные задачи, которые позволят написать легковесный искусственный интеллект, но в большинстве случаев разработчик будет вынужден писать собственные задачи, что позволяет делать Unreal Engine.

Разумеется, для корректного использования деревьев поведений, требуется какой-либо ресурс для хранения переменных, используемых в задачах. Для этого используется Blackboard. Он прикрепляется непосредственно к дереву поведений. Его использование не обязательно, но может значительно повысить качество разрабатываемого искусственного интеллекта.

Помимо задач и композитов есть еще службы (сервисы), которые можно прикреплять к задачам или композитам. Они также могут использовать и изменять переменные, хранящиеся в blackboard при необходимости. За ними может закрепляться разная логика. Это может быть простое событие при начале выполнения задачи, например перед патрулированием это может быть выбор следующей точки, к которой необходимо пройти, или длящаяся на протяжении всего выполнения ветви для отслеживания или выполнения какой-либо задачи параллельно. Примером параллельной задачи может служить дальние атаки противников при условии, что он в зоне видимости на достаточном удалении от персонажа.

Последним рассматриваемым узлом в деревьях поведений является декораторы. Они также присоединяются к задачам или композитам. Их можно ассоциировать с операторами ветвления. При возвращении декоратором значения true, дальнейшее выполнение ветви продолжается, иначе ветвь блокируется и идет выбор ветвей правее.

Описание всех встроенных узлов можно изучить в официальном источнике[x]

Совместное использование четырех узлов позволяет составлять из задач игровой искусственный интеллект, который в будущем будет возможно дополнять без изменения уже написанной логики дерева поведения.