**4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

Так как в дипломном проекте основными разрабатываемыми сущностями являлись дрон, бот и турель, то их описание является приоритетным в записке. Логика поведения дрона и бота была подробно разобрана ранее, в третьем разделе. Там же было приведено дерево поведения дрона на соответствующем рисунке и описание всех ветвей-задач у дрона. У бота также были описаны все функции и дерево поведений с написанными для него задачами, сервисами, декораторами и логикой построения самого дерева. Подробное описание турели будет приводится именно в этом разделе. Причиной этому является то, что данный персонаж разрабатывался первым и при его реализации, соответственно, было изучено множество необходимых функций и параметров, уже встроенных в классы actor, pawn и character. В данном разделе будут описаны события и функции, прикрепленные к ним.

Сперва стоит описать функции, которые будут прикреплены к событиям для выполнения. Тут же будет кратко описана работа с чертежами в Unreal Engine 4 для лучшего понимания.

Первой рассматриваемой функцией является FindTarget. Для лучшего понимания, она будет изображена на рисунке 4.1. Как упоминалось при описании турели в функциональном проектировании, у турели отсутствуют встроенные органы чувств. Он может ли получать информацию о сущностях типа actor, находящихся в радиусе. Получение данной информации происходит после вызовы функции SphereOverlapActors. Задается центр сферы, ее радиус, объекты, которые необходимо искать и опционально можно настроить список сущностей, которые стоит исключать. Последний параметр не использовался. Далее происходит поиск сущностей, которые ближе всего расположены к турели и находятся также в зоне видимости, чтобы в последствии атаковать.

На рисунках белыми линиями показан порядок выполнения функций. Узел с названием FindTarget является начальным, что и видно на рисунке. Это своего рода начало выполнения функции. Прочие цветные линии — это переменные связи переменных, которые могут быть как локальными, так и глобальными. Под локальной переменной в данном случае стоит подразумевается то, что они объявлены разработчиком. Локальные же наоборот нельзя записать в память, они исчезают сразу после того, как были использованы функциями или иными узлами. К сожалению, для демонстрации кода в записке, было необходимо уменьшить расстояние между узлами, что сильно ухудшило читаемость кода.

Тут стоит кратко рассказать, что в основном, для создания игровых персонажей используются классы типа Pawnили Character, который наследуется от класса Pawn. По этой причине вход ObjectTypes в функции SphereOverlapActors использует как маску лишь классы типа Pawn. По этой же причине турель атакует всех – ботов, дронов, игрока.

Далее, после сбора информации обо всех сущностях в радиусе, начинается фильтрация по двум признакам. Первый, как уже было сказано – возможность турели стрелять по цели. Второй – радиус атаки. У турели есть как радиус видимости, при котором он начнет сводить башню к цели, так и радиус атаки, при нахождении персонажей в котором турель начнет атаковать цель. Опять же, для демонстрации пришлось уменьшать расстояние между узлами.

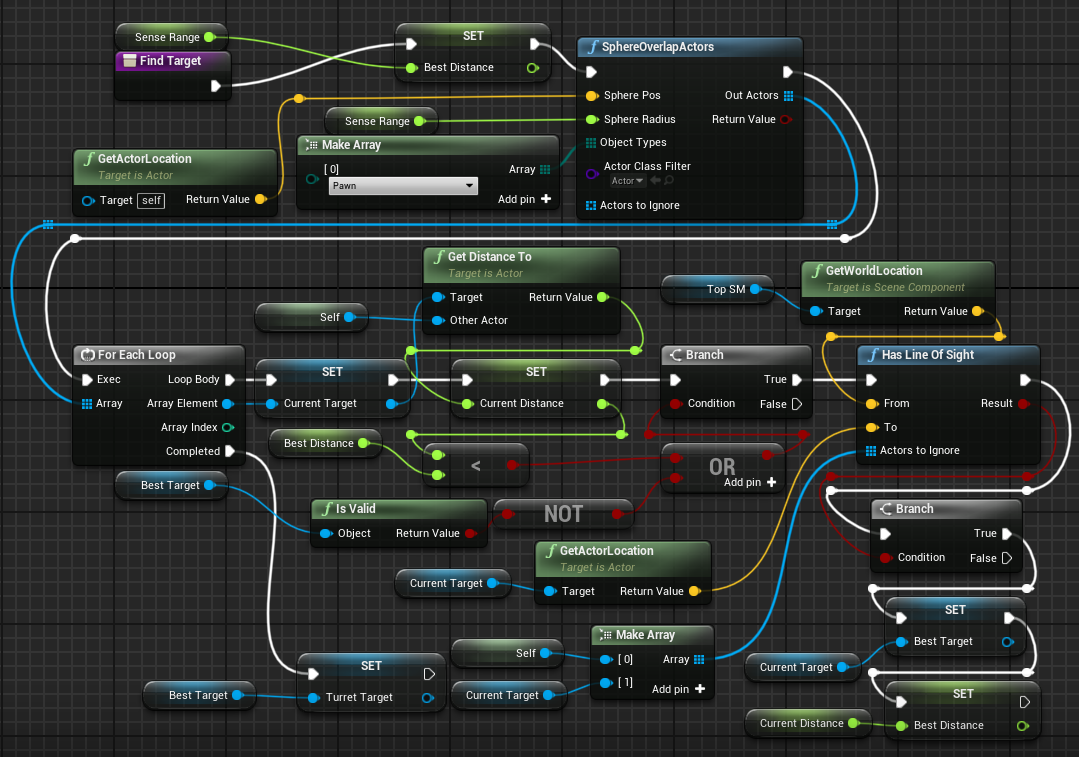


Рисунок 4.1 – Функция FindTarget

Следующая функция RotateTurret была необходима для вращения главной части турели – башни. Так как для данной сущности не предусмотрено никаких анимаций, что является также и избыточным, то возникла необходимость контролировать повороты башни вручную. На рисунке 4.2 показана реализация данной функции.

Логика довольно проста – башня поворачивается каждый тик в направлении текущей цели. Скорость поворота и ее плавность контролируется переменными RotateSpeedи DeltaSeconds. DeltaSeconds контролируется параметром класса, уже описанным в функциональном проектировании – TickInterval. Реализация основана на одной функции – RinterpToConstant, которая плавно изменяет значение одной переменной со временем. Значение, которое должно плавно изменяться, берется как результат работы функции FindLookAtRotation. Угол поворота высчитывается между поворотом башни и целью.

Необходимо также учесть, что турель может вращаться только в определенных границах. Границы выставляются соответственно логике – турель не должна застревать в своих же текстурах, в своей модели. Для этого, при помощи функции ClampAngle**,** был ограничен поворот башни по оси Y. Значения ограничений приведены на рисунке 4.2 в параметрах функции.

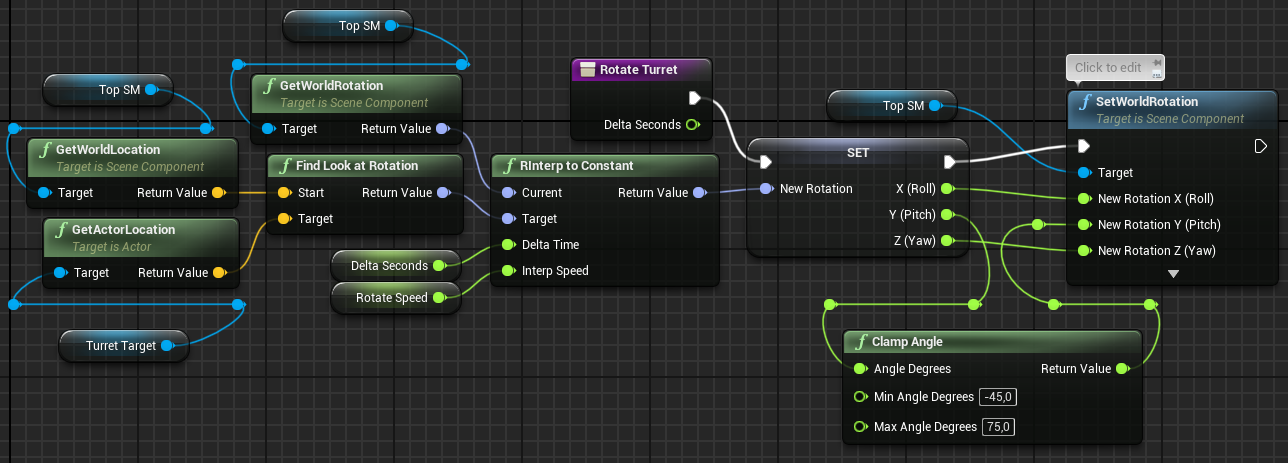


Рисунок 4.2 – Функция RotateTurret

Функция FireFrom отвечает за стрельбу турели. Существует множество способов реализации стрельбы. Основной используемый способ – создание отдельного класса снаряда. В него заносится начальная скорость, наносимый урон и иные эффекты, например анимация попадания или создание системы партиклов в месте попадания.

В данном проекте было принято решение сделать стрельбу турели при помощи моментального получения урона. Это достигается при использовании функции LineTrace и ей подобных. Данная функция в общем случае пускает луч или несколько лучей. Далее проверяется пересечения лучей с объектами на карте по маске, которая задается опять же в параметрах функции. Тут также привязана и работа с лазером. Лазер был реализован, но не является важной частью проекта, он служит лишь для визуального выделения стрельбы.

Как и во всех функциях, существует начальный узел, он выделяется фиолетовым цветом и соответствующим названием. Данная функция вызывается после наведения на игрока или иную цель, а значит можно сразу начинать стрельбу, без последующей фильтрации полученной информации – не важно, попала турель по цели или нет, так как нанесение урона тем же стенам должно предусматривать обработку ей же.

После генерации луча, берется информация об итоге работы функции LineTrace и в зависимости от исхода, отображается лазер и генерируется звук стрельбы. Стоит отметить, что звук генерируется не только обычный, который может слышать игрок, но и технический, при помощи MakeNoise для возможного информирования других персонажей.

Так как лазер должен заканчиваться только при попадании в цель или, в случае если не было попадания, заканчиваться после определенной длины, то в конце присутствует небольшое ветвление кода. Ветвление происходит потому, что рисоваться лазер должен либо до какой-то определенной точки, выдаваемой в TraceEnd, либо до цели, координаты попадания в этом случае будут указаны в HitLocation. Как и в случае других функций, код будет представлен на рисунке 4.3.

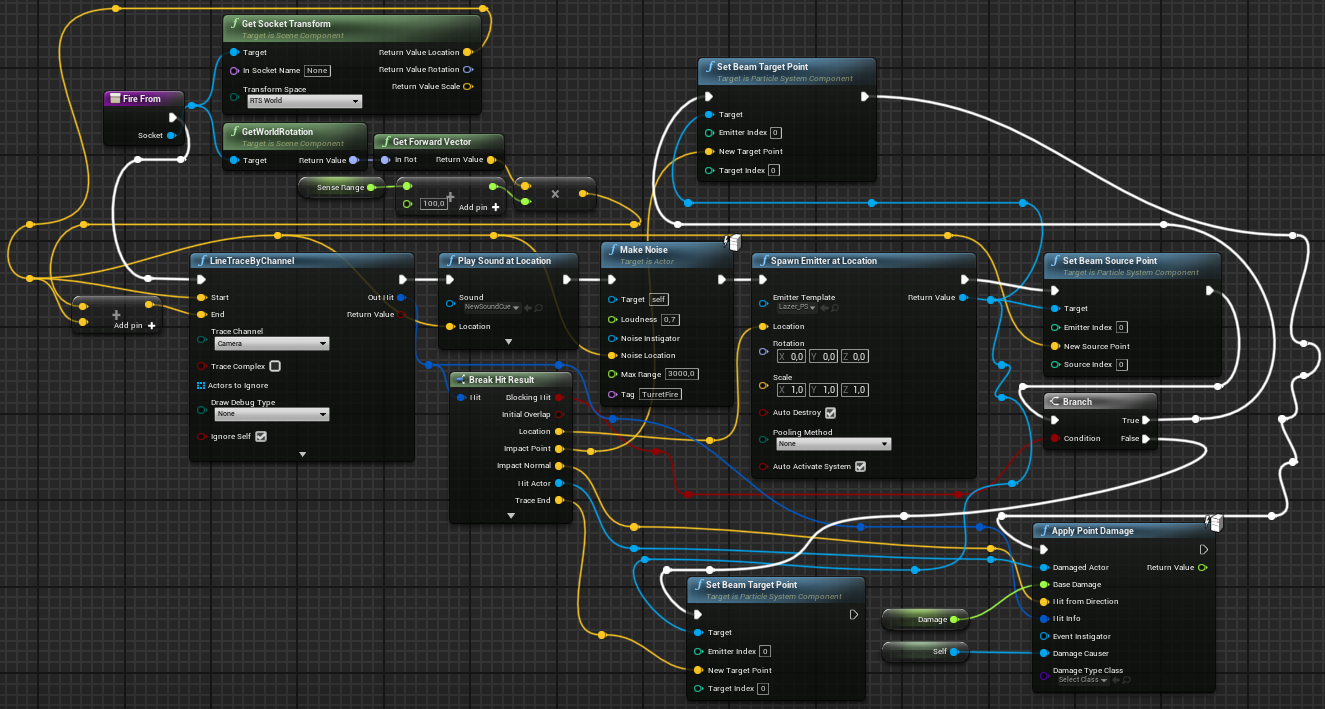


Рисунок 4.3 – Функция FireFrom

Последняя функция – FireTurret. Она отвечает за сам запуск стрельбы. Так как стрельба может производиться не чаще чем указано FireRate, то тут происходит проверка по времени. Также проходит проверка на то, наведено ли орудие на цель. Берется вращение башни и угол поворота, оставшийся до полного наведения на цель. В случае совпадения углов, стрельба разрешается. Код представлен на рисунке 4.4.

Отдельно стоит упомянуть лазер, так как он используется и в оружии, и в дроне с турелью. Внешне он выглядит, как и обычный лазер. Он не имеет коллизии и является системой частиц. Для создания необходимо указать глобальные координаты для размещения и две точки – начала лазера и его конца. Его время жизни контролируется в настройках самой системы частиц. Значение времени жизни было выставлено минимальное.

Система частиц уничтожается по окончанию отображения лазера, что позволяет в некотором роде экономить память и прочие ресурсы компьютера. Цвет лазера также является изменяемым и может меняться в зависимости от нужд разработчика. В будущем планируется добавление кооперативного режима и значительное расширение возможностей игрока. В том числе кастомизации. Так же цвет можно использовать для определения того, кем был произведен выстрел, какой персонаж или команда.

Лазер также можно использовать и в окружении, но на данный момент это не представлено в проекте из-за отсутствия надобности в построении более сложного окружения.

При разработке лазера также был использован отдельный материал, но он является довольно легковесным, так как по своей сути является обыкновенным прямоугольником, который и растягивается по всей длине лазера.

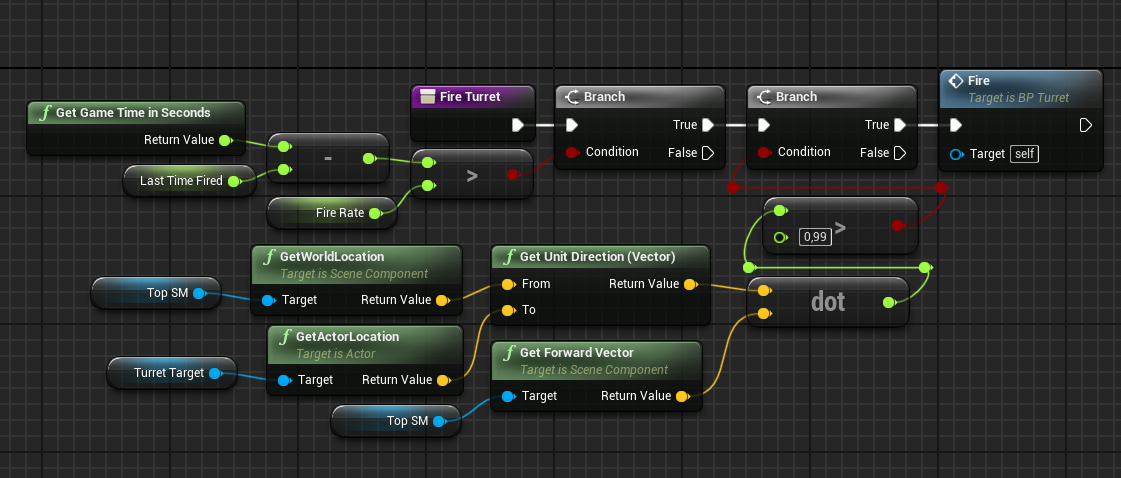


Рисунок 4.4 – Функция **FireTurret**

Далее стоит описать события, которые вызывают данные функции и иные функции, которые уже устроены в движок. Самым важным ивентом является Tick. За данным событием закреплена вся основная логика и данное событие вызывается каждый выделенный тик для турели. Чертеж данного события представлен на рисунке 4.5.

Данное событие встроено в каждый класс, наследуемый от actor. В каждом наследуемом классе есть возможность переопределять данное событие. При построении более сложной системы, как например при создании оружия для использования персонажами, переопределение данных методов необходимо в первую очередь чтобы менять тип стрельбы.

Разумеется, переопределять можно не только события, но и функции. Событий, которые можно использовать, намного больше. И все ивенты можно переопределять. Также можно вызывать закрепленную логику класса, от которого наследуется нужный класс, до или после логики, реализованной разработчиком.

Первым действием, закрепленным за данным событием, является проверка, была ли разрушена турель или была ли она деактивирована иным способом. В случае, если она еще активна, вызывается функция поиска цели, проверяется, валидна ли данная переменная для цели и далее происходит вращение башни турели в сторону цели. Далее вызывается функция стрельбы, в которой сперва проверяется возможность стрельбы, а далее вызывается уже и сама стрельба, если она разрешена.

В случае, если турель была уничтожена или деактивирована извне, необходимо каким-либо образом показать игроку, что турель разрушена или не активна. На данный момент это достигается путем поворота башни к основанию. Часть функционала была взята из функции поворота башни RotateTurret.

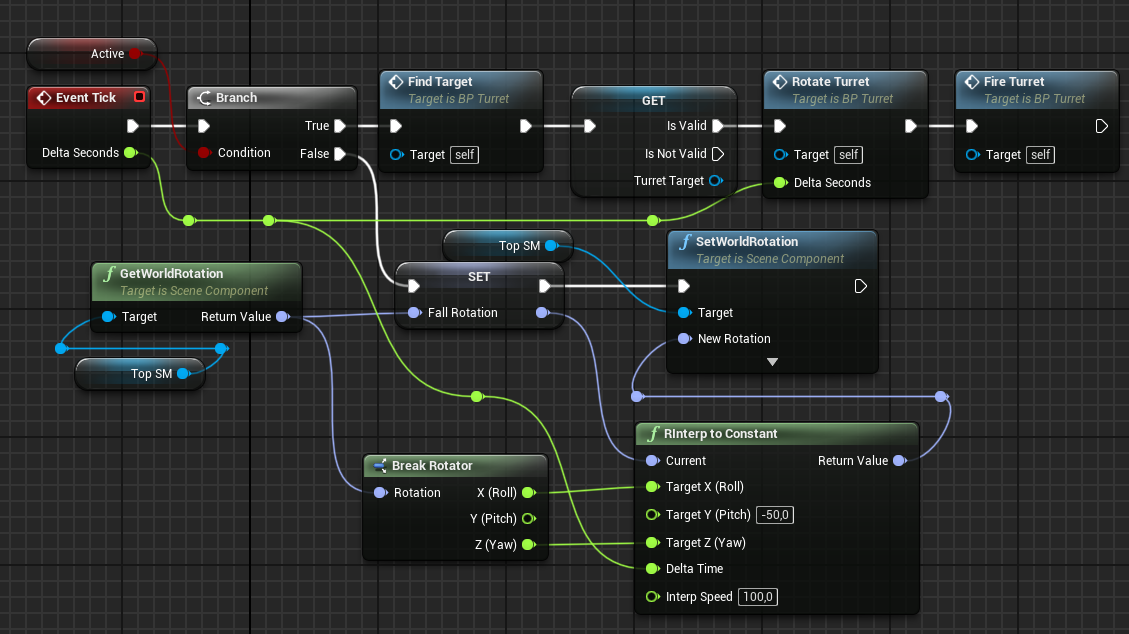


Рисунок 4.5 – Событие Tick

Самым небольшим событием является BeginPlay. Он также встроен в класс благодаря наследованию от класса Actor. Данное событие было крайне полезным при отладке турели, однако в конце вся настройка персонажа была перенесена. Данное событие использовалось в первую очередь для балансировки скорости стрельбы, при изучении функций по типу LineTraceByChannelи SphereOverlapActors. Осталась лишь настройка начального уровня здоровья. На рисунке 4.6 показана логика, связанная с данным событием.



Рисунок 4.6 – Событие BeginPlay

Событие AnyDamage позволяет отлавливать получение урона турелью. Обработка получения урона опять же довольно проста с точки зрения реализации. При начале выполнения необходимо условно нанести урон. Так как событие хранит информацию о типе урона, кем был он нанесен и, разумеется, о количестве наносимого урона, то информация берется из него. Из текущего значения здоровья вычитается значение наносимого урона и сохраняется сразу же в ту же переменную. Далее при помощи простого сравнения проверяется, должна ли уничтожаться турель. В случае если в этом нет необходимости, ничего не происходит и выход False у функции ни с чем не соединяется. Так как анимации получения урона опять же не предусмотрена для турели. Уничтожение турели или точнее ее деактивация происходит в уже описанном событии Tick.

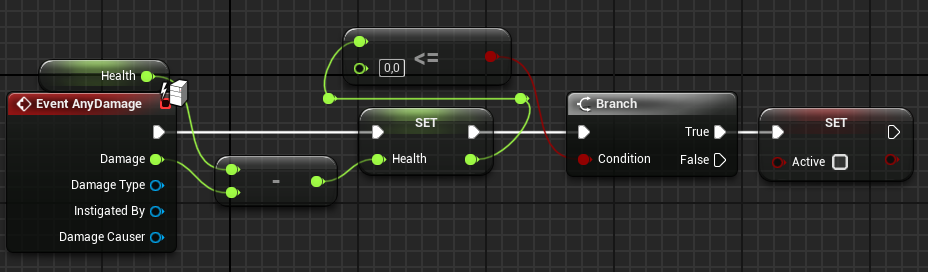


Рисунок 4.7 – Событие AnyDamage

Последнее рассматриваемое событие – Fire. Оно вызывается для стрельбы по цели. При стрельбе также необходимо запомнить время последнего выстрела, что и контролируется переменной LastTimeFired. Далее поочередно вызывается стрельба из двух орудий турели. Вызывается она с некоторой задержкой, равной половине частоты стрельбы. Данная задержка необходима, чтобы корректно обрабатывать попадания и наносить урон.

После установки значения в переменную LastTimeFired, запускается узел последовательности, в котором присутствует два исполняемых выхода. Один из них направляется на вход enter цикла while, второй направляется на вход reset того же цикла. К выходу цикла была закреплена сама стрельба из двух орудий. При реализации было решено разделить стрельбу таким образом, чтобы минимизировать повторяющиеся части кода. Для этого и была создана функция FireFrom. Она принимает компонент, из которого необходимо производить стрельбу по цели.

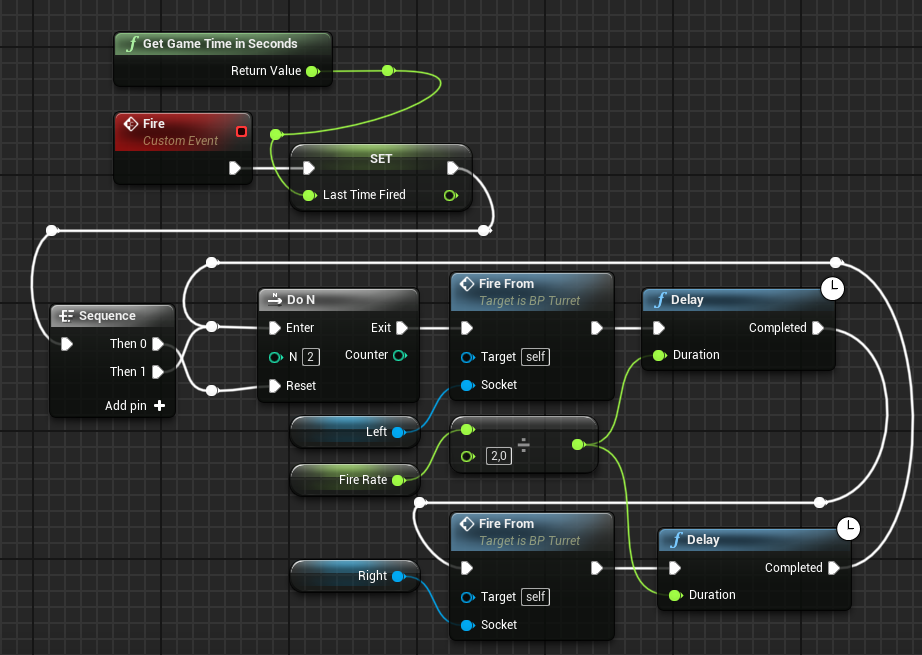


Рисунок 4.8 – Событие Fire

Разумеется, при желании, в будущем возможна модификация турели. А именно добавление или наоборот удаление стволов в ней. Данный персонаж хоть и является завершенным в плане логики, но может улучшаться в будущей доработке проекта.

Одной из доработок может быть балансировка возможностей стрельбы турели. В некоторых играх, особенно в стратегиях реального времени, часто используются турели, которые контролируют территорию не всю вокруг себя с радиусом обзора в триста шестьдесят градусов, а лишь часть, например девяноста градусов. Такая балансировка, возможно, понадобится при доработке проекта.

Одно из возможных применений турели в будущей доработке проекта является добавление класса персонажей типа инженеров. Он находился на момент написания турели в проработке.

Одной из возможностей данного типа персонажа являлась бы установка временной переносной турели в области. Также у игрока появлялась бы возможность устанавливать турели и чинить их при необходимости. К сожалению, из-за малого времени, выделенного на разработку программного дипломного проекта, эту наработку было необходимо отложить.