**21. Packaging and debugging of weapon logic**

1. Добавим несколько проверок на пули и обоймы (в каком cpp-файле?) с помощью нового макроса – чем отличается?

2. Добавим проверку на количество оружия (в каком cpp-файле?) с помощью того же макроса (не будем создавать магическое число – как поступим)?

3. В нашем коде так же большое значение имеют AnimNotify, какой макрос использовали для них? Как работает?

4. Убрали нашу шаблонную функцию FindNotifyByClass – куда, почему и как? Как надо пофиксить ее вызов в нашем коде?

5. Добавим еще один ЗФ для различных структур и делегатов – зачем, куда и как? Какой надо здесь не забыть подключить ЗФ и для чего?

1. Добавим несколько проверок.

Будем использовать макрос checkf, он ведет себя как обычный check, единственное отличие – он позволяет добавить пользовательское сообщение (аналогично assert и static\_assert).

Сделали проверку на количество пуль и обойм (вдруг кто-то в БП выставит некорректное число).

Изображение выглядит как текст

Автоматически созданное описание

2. В классе WeaponComponent добавим проверку на количество оружия, так как наш текущий класс поддерживает только два оружия:

Изображение выглядит как текст

Автоматически созданное описание

3. В нашем коде так же большое значение имеют AnimNotify, если анимации с ними не будут установлены, то наша логика перезарядки и смены оружия просто не будет работать, так как анимации будут всегда в прогрессе:

Изображение выглядит как текст

Автоматически созданное описание

Здесь используется макрос checkNoEntry() – он эквивалентен вызову макроса check с аргументом false – то есть данный check срабатывает всегда, если до него доходит поток выполнения программы.

4. Уберем FindNotifyByClass (шаблонную) из WeaponComponent в специализированный ЗФ AnimUtils.h и помести его в Animations. Данная функция чисто вспомогательная, она не имеет отношения к WeaponComponent, поэтому логично выделить ее в некоторый класс утилит, к тому же данной функцией сможет воспользоваться другой класс, который работает с анимацией:

Изображение выглядит как текст

Автоматически созданное описание

Сделали так же функцию статичной, так как она не зависит от какого-то экземпляра.

В классе WeaponComponent добавляем новый ЗФ и добавляем оператор расширения области видимости перед вызовом соответствующих функций:

Изображение выглядит как текст

Автоматически созданное описание

5. Добавим еще один ЗФ STUCoreTypes и положим его в папку Public. Мы задекларировали в разных частях достаточно много различных структур и делегатов, их все можно поместить в одно место. Все они будут располагаться в одном файле, что сильно повысит читаемость:

Изображение выглядит как текст

Автоматически созданное описание

Мы забрали структуры и делегаты из WeaponComponent, BaseWeapon и HealthComponent поэтому там надо подключить наш ЗФ.

А так же в самом верху мы подключили ЗФ , чтобы UnrealHeaderTool корректно сгенерировал весь код, который относится к макросам структуры и делегатов.