

UNREAL ENGINE 5

ШАГ ЗА ШАГОМ ОТ ПОЛНОГО НОВИЧКА ДО РАЗРАБОТЧИКА ИГР

ГАЙД ПО ИЗУЧЕНИЮ



В данном файле я расскажу про изучение Unreal Engine шаг за шагом именно в контексте разработки игр.

Двигаясь постепенно по этим этапам вы научитесь создавать свои игры. Плюс будут интересные задания:) Дополнительно отмечу, что в тексте будет много ссылок на различные видеоуроки как на мои, так и на других авторов, потому что некоторые вещи проще объяснить в видеоформате.

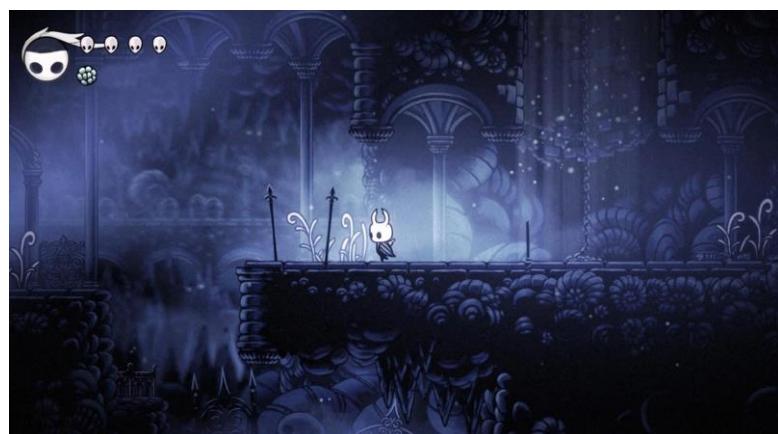
p.s. все нижеприведенные шаги – это сугубо личный опыт автора

ПЕРВЫЙ УРОВЕНЬ

(Учимся навигации и работой с объектами, чтобы смастерить дом)

1. Зачем вам нужен Unreal Engine?

В первую очередь спросите себя: «А зачем мне нужен Unreal Engine?» Почему это важно? Если у вас стоит выбор между BMW и Mercedes, то почему вы выберете, скажем, BMW? Мы знаем, что BMW и Mercedes – это отличные автомобили. То же самое и с игровыми движками. Ведь помимо Unreal Engine есть ещё Unity, на котором делается много классных игр. Например, Hollow Knight. А также много других игровых движков.



Согласитесь, не хотелось бы тратить время на продукт, который вам в итоге не подошёл? Чтобы такого избежать, в автосалонах есть тест-драйвы (вспоминаем вышеуказанный пример). Поэтому выбирайте движок с умом.

Лично я для себя выбрал этот движок потому, что он для меня:

- а) удобен;
- б) на нём сделаны мои любимые игры;

в) он популярен (даже CDPR после возни с Cyberpunk 2077 решили перейти со своего движка на Unreal Engine для Ведьмака 4).

Если вы поняли для себя, зачем вам нужен Unreal Engine – супер! Вы сделали осознанный выбор, а не навязанный людьми с Ютуба или онлайн-школами. Ведь осознанный выбор – это важный психологический момент.

Разработчики из CD Projekt объяснили, почему они перешли на Unreal Engine 5 при разработке следующего Ведьмака[©]



monk70

5 апреля 2022 Новости

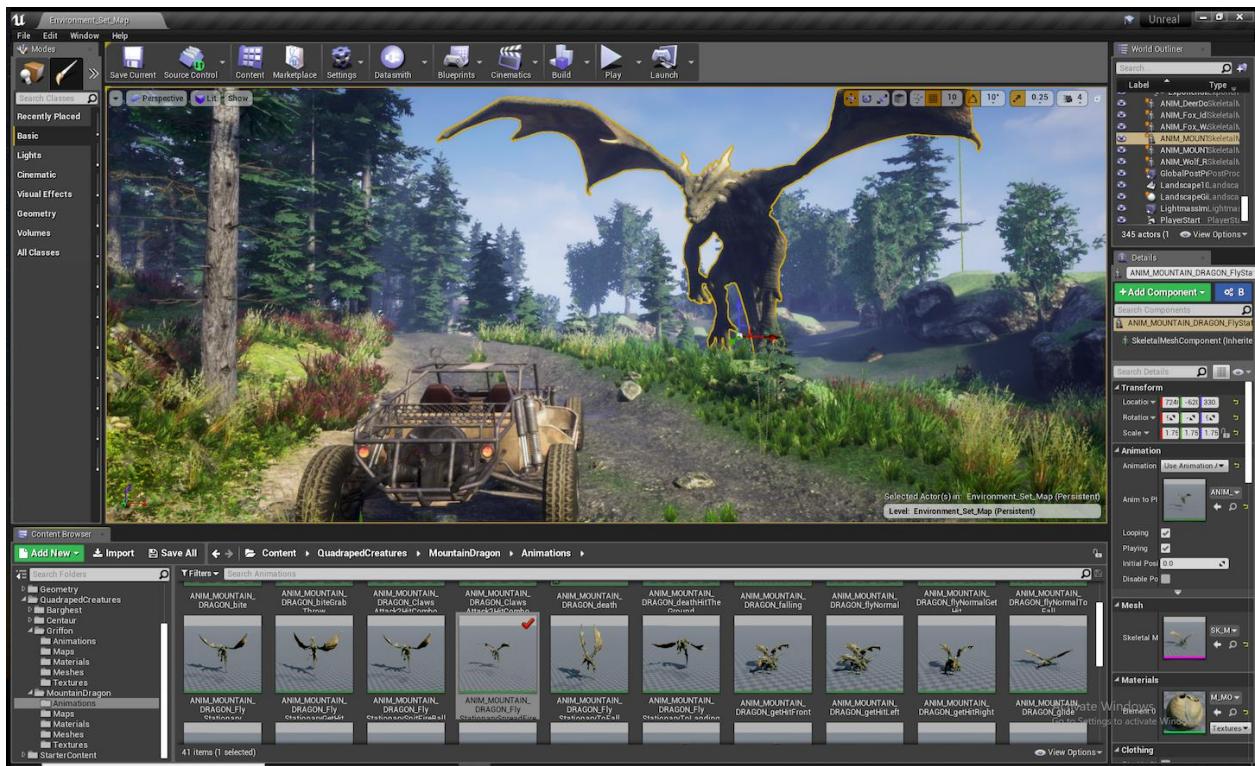
2. Навигация в движке

Не ленитесь запоминать горячие клавиши и шорткаты. Они очень сильно упростят вам работу.

У меня есть [видео](#) по самым важным шорткатам в Unreal Engine.

Также в Интернете полно видео и статей о том, как передвигаться по сцене в Unreal Engine.

Поверьте, когда вам нужно будет быстро приблизиться к объекту среди других десятков (а то и сотни) объектов на сцене, то вам будет проще нажать клавишу F, чем делать это вручную.



Задача: выпишите себе десять шорткатов и основные клавиши навигации в Unreal Engine (движение по сцене). И пусть эти клавиши будут у вас перед глазами, когда вы будете работать в движке. Можете повесить их перед собой, добавить на второй монитор и т.д.

Доказано, что зрительная память и визуальные образы лучше запоминаются и воспроизводятся, нежели слуховая.

3. Работа с объектами

Когда вы научитесь перемещаться по сцене и выпишете себе шорткаты и кнопки навигации, то следующее, что вам нужно обязательно сделать – это понять, как изменять объекты в Unreal Engine. В Unreal вы можете взаимодействовать с объектами тремя способами (имеется в виду, не через код):

- Двигать и поднимать объекты (Move);

- крутить (Rotate);
- Изменять размер (Scale).

Можете посмотреть здесь хорошее и понятное [видео](#).

Казалось бы, всё легко и понятно. Да, пока вы не столкнётесь с ситуацией, когда крутится объект не так, как вам нужно, или двигать объект становится неудобно. Тут на помощь вам придут координаты, о чём ниже.



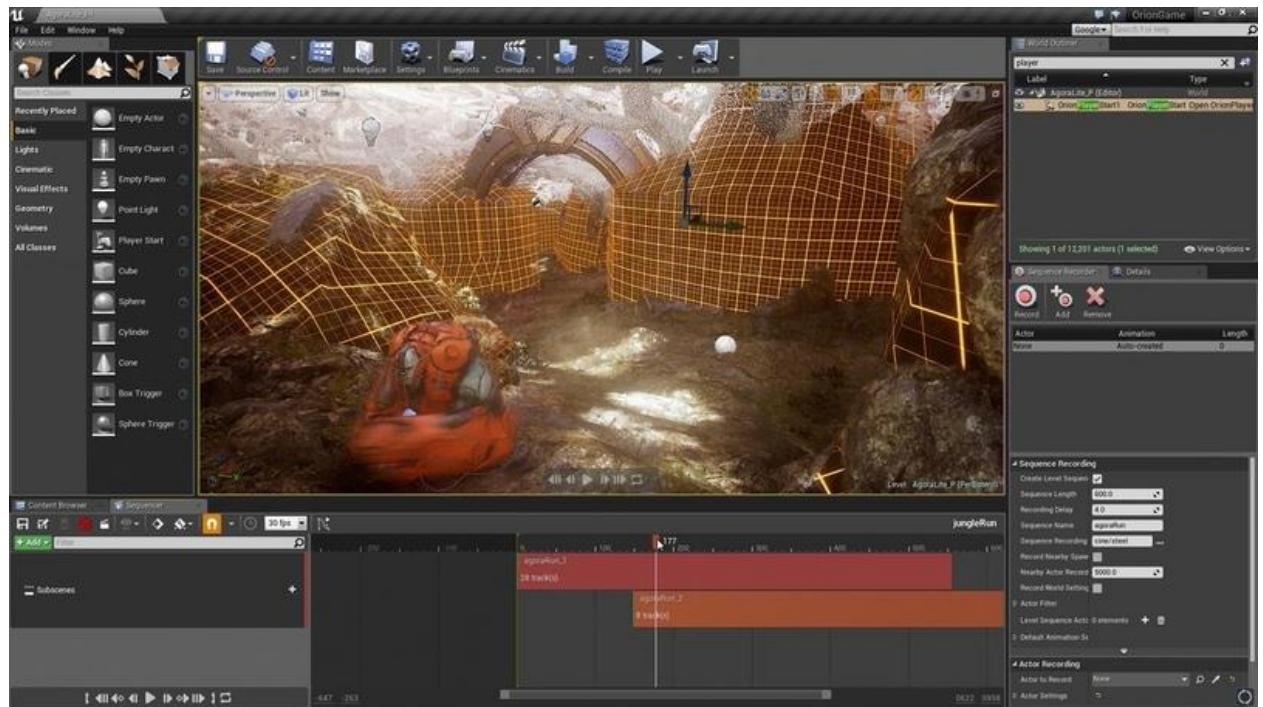
4. Локальные и мировые координаты

В один момент вы столкнётесь с ситуацией, что объекты не всегда удобно двигать, не всегда поворачиваются туда, куда вам нужно и будет множество других неудобств.

Для этого вам нужно понять, что такое World (мировые) и Relative (иногда – Local, локальные) координаты.

На ютубе есть много видео на этот счёт. Введите в поиске: отличия мировых и локальных координат в Unreal Engine.

Это несложная тема, но поняв отличие этих координат – вы станете круче точно. Почему? А потому что эти координаты используются не только в Анриле, но и в любом ином движке и даже в 3D моделировании. Поэтому поняв координаты сейчас – будете знать как ими пользоваться вообще практически в любом софте.

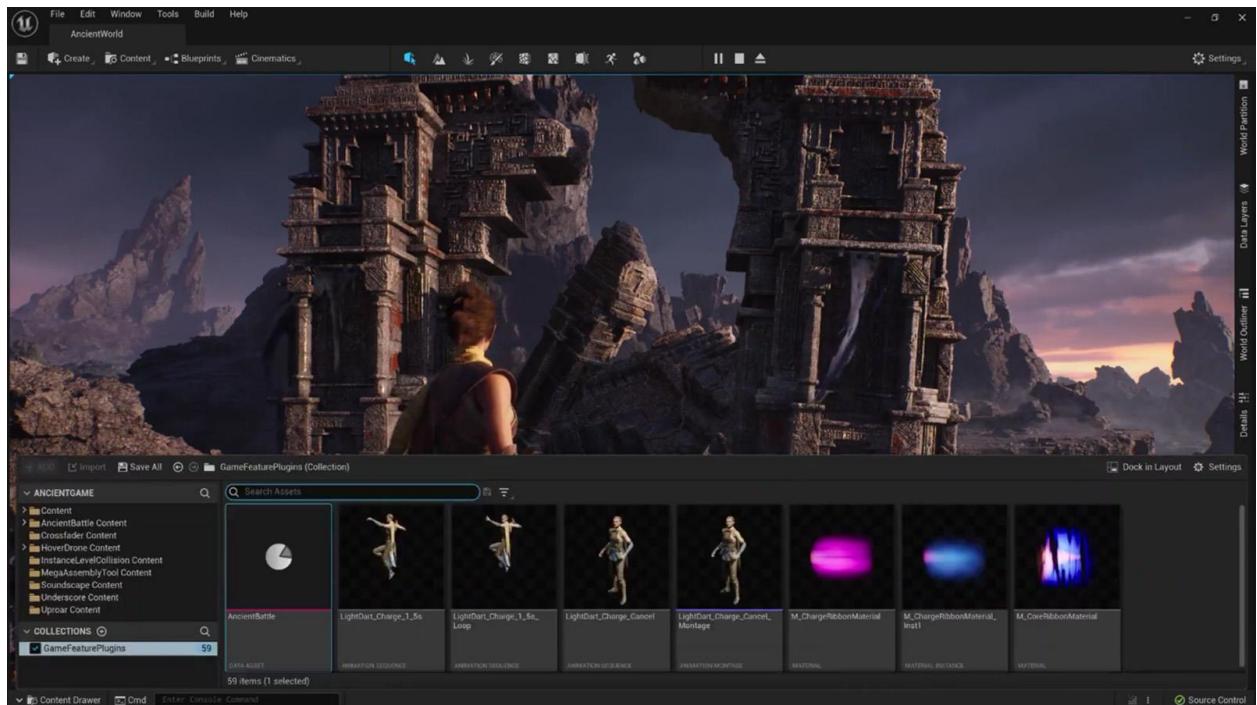


5. Outliner и Content Drawer/Browser

Важные вещи. Очень.

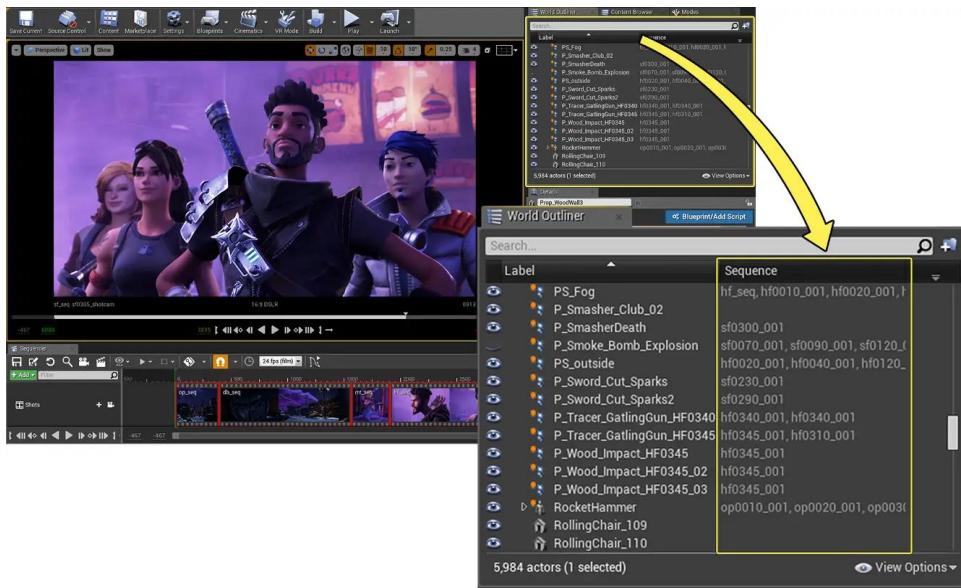
- Content Drawer/Browser («Ящик с контентом») – это виртуальный ящик, где хранятся все ваши ассеты. Ассеты – это ВСЁ, из чего состоит ваша игра: персонажи, предметы, текстуры, эффекты. В общем, всё.

На ютубе есть видео о том, как работать с Content Drawer. В этом «ящике» будет собрано всё, что есть у вас в проекте. Поверьте, этот «ящик» вы будете открывать крайне часто.



- Outliner – своего рода «структуризатор». Постараюсь проще. То что есть у вас в «ящике с контентом» – вы можете где-нибудь размещать. Например, у вас в «ящике» есть ваш игровой персонаж. Вы можете его разместить на игровой сцене. И всё то, что вы разместите на сцене – появится в Outliner.

На Ютубе тоже есть уроки, посвященные Outliner.

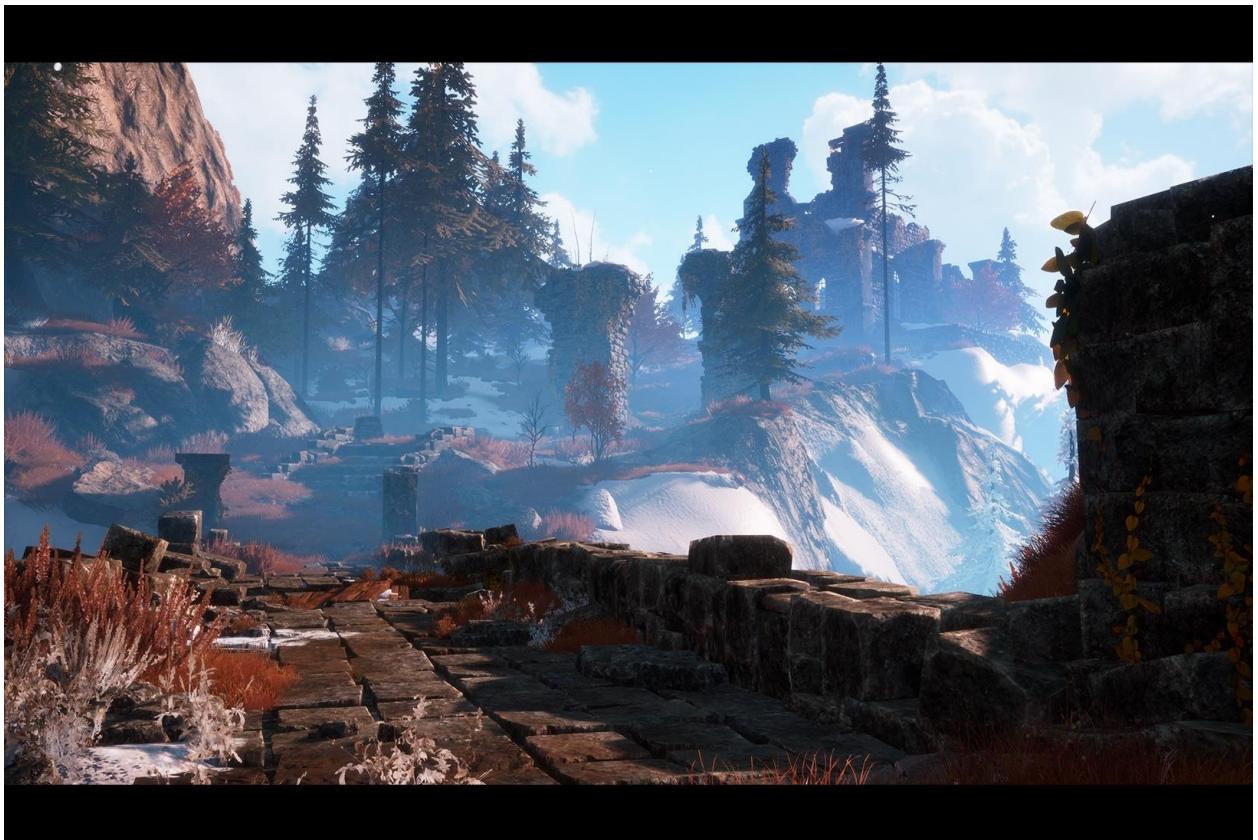


6. Работа с Level

Каждая игра состоит из уровней. Даже главное меню в играх – это уровень.

Вам нужно научиться создавать свой Level (именно файл уровня, а не всякие леса и прочее, о чём поговорим позднее) и устанавливать конкретные уровни по умолчанию. Например, если вы этого не сделаете, то может случиться неприятная ситуация – что вы создадите что-то на уровне, а потом, закрыв движок и войдя в него обратно – вы увидите, что ваш прогресс пропал!

На самом деле прогресс не пропал. Возможно, что он просто сохранился на другом уровне, а по умолчанию у вас открывается другой уровень.



7. Сохранения

Это звучит может быть глупо, но новички не умеют сохраняться в Unreal. Казалось бы, вы можете мне сейчас возразить, что есть же шорткат `ctrl+s`, который произведёт сохранение. И да, вы будете правы. Но вот в чём проблема: `ctrl+s` сохранит лишь то окно, где вы находитесь в настоящий момент.

А если вы произвели десятки изменений в разных объектах? Будете заходить в каждое окно и везде жать `ctrl+s`? Это дико неудобно.

Поэтому вам нужно использовать `ctrl+shift+s` чтобы сохранить абсолютно все изменения в вашем проекте.



8. ИГРОВАЯ ЗАДАЧА: ПОСТРОЙТЕ ДОМ

Итак, у вас уже есть знания, чтобы создавать какие-то примитивные уровни и объекты (блокинг, который используется в играх на ранних этапах).

Поэтому теперь вы можете создать какой-нибудь простенький дом из примитивов (Cube, Plane, Sphere и так далее).

Советы:

1. Установите ваш уровень по умолчанию в проекте;
2. Не забывайте сохранять всё;
3. Чтобы добавлять примитивы на сцену, то советую посмотреть это [видео](#).

ВТОРОЙ УРОВЕНЬ

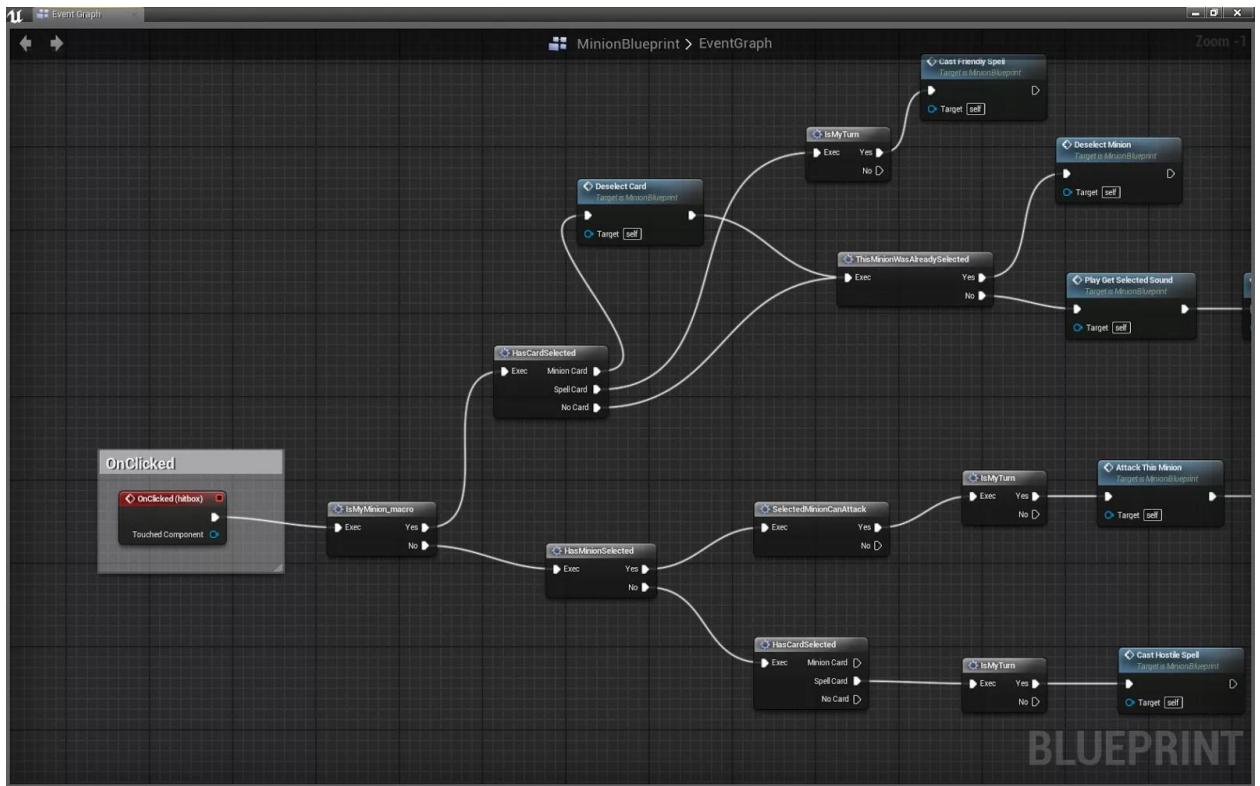
(Учимся работать с простым кодом, чтобы создать свой первый платформер)

9. Блюпринты (Blueprints)

Ну что ж, друзья. Пожалуй, начинается самое сложное, но необходимое. Работа с Blueprints. С английского языка блюпринт переводится как «чертёж». Помните миссию в GTA San Andreas, где нужно было в здании сфоткать план здания на синей схеме с белыми линиями? Вот Blueprints в Unreal Engine – это практически то же самое.



Блюпринты – это всё то, где будет происходить ваша логика. Вы будете в них наносить урон врагам, считать очки персонажа и так далее. В общем, блюпринты – это уже код.



И хочу сказать две ВАЖНЫЕ вещи:

1. Какие-то моменты в блюпринтах я буду далее упускать, потому что в блюпринтах есть такие вещи, о которых не то что написать можно, но даже на Ютубе нет уроков по таким вещам. Поэтому по блюпринтам пройдёмся по верхам, потому что здесь важна только практика, практика, практика. Даже люди, которые работают в Unreal десятилетиями до сих пор могут о чём-то не знать (видел это на официальном форуме Unreal Engine).
2. Поэтому, пожалуйста, во время изучения блюпринтов – не нервничайте. Это сложно, но возможно. Смотрите различные видеоуроки и повторяйте. Если чувствуете силы какую-то логику добавить – делайте. У меня на канале MakeYourGame! на момент написания этого текста

почти 300 видеоуроков как для новичков, так и совсем сложная логика. Вы найдёте своё. Самый оптимальный способ изучения, как я уже сказал ранее: смотрите видеоуроки разные, повторяйте, попробуйте понять код и по возможности добавьте что-то своё. Не получается? Ничего страшного. Повторяйте.

10. C++ или Blueprints

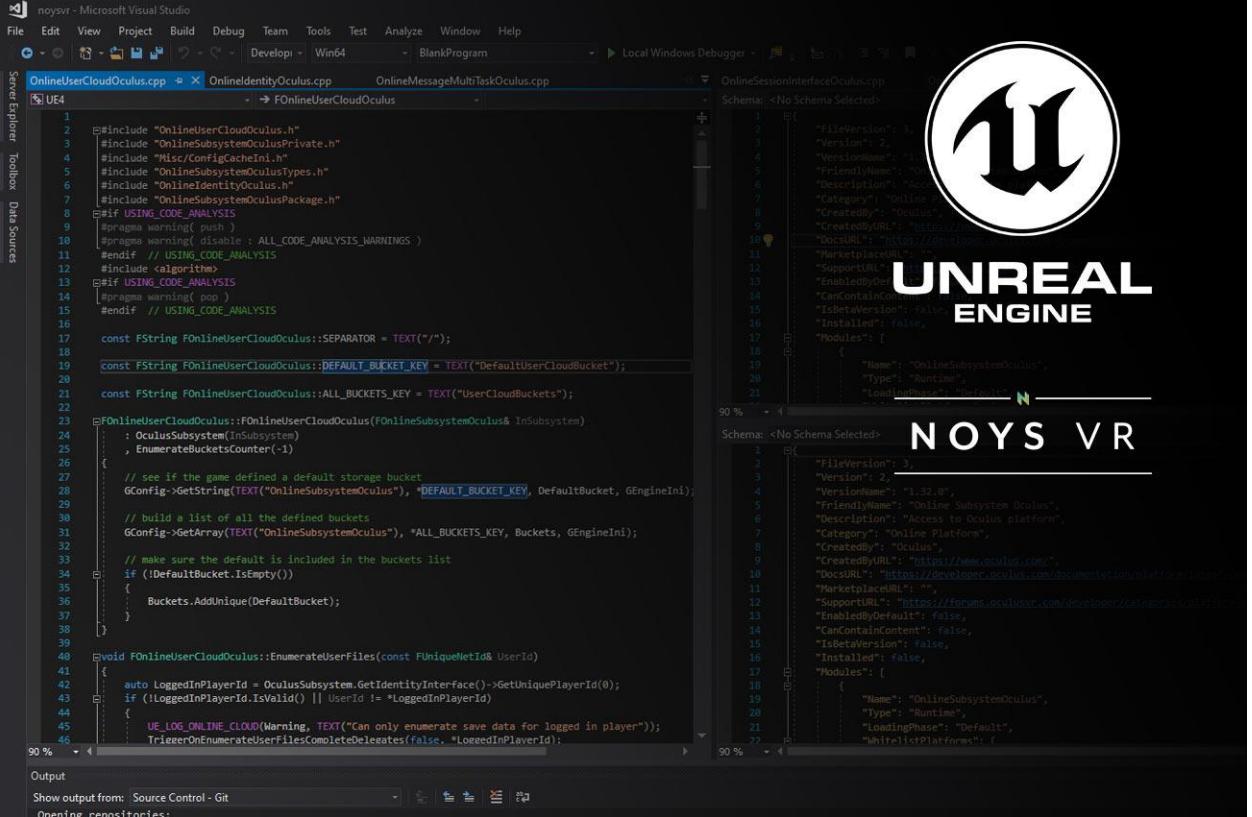
В Unreal Engine вы можете программировать как на C++, так и на Blueprints. Самый лучший подход – это работать в связке C++ и Blueprints. Но даже если вы не хотите влезать в C++ (это довольно сложный язык), то вы можете без проблем писать весь свой код на Blueprints. Сейчас есть даже игры, которые полностью написаны на Blueprints (Ark: Survival Evolved) и (Fortnite, в которой 80% написано на блюпринтах, а 20% – в C++, в частности мультиплеерная часть).

Но бывает ситуация, когда человек начинает изучать Анрил, но у него уже есть опыт работы с C++, либо же новичок хочет всех удивить и сразу начать писать на C++.

Я этого КРАЙНЕ не рекомендую для обоих типов людей.

Дело в том, что C++ в контексте Анрила становится в разы сложнее. За счёт чего? А дело в том, что разработчиками Анрила уже написаны различные макросы, функции, структуры и классы. И их удобнее всего изучать именно в Blueprints, а не в C++. И если вы не знаете, например, как работает довольно популярное событие ActorBeginOverlap, которое есть по умолчанию во многих блюпринтах в момент их создания,

то в C++ для изучения этого события вы будете тратить в несколько раз больше времени, чем если будете изучать это в блюпринтах.



The screenshot shows a Microsoft Visual Studio interface with several windows open. On the left, the 'Server Explorer' and 'Toolbox' are visible. The main code editor window displays C++ code for the 'FOnlineUserCloudOculus' class, specifically the 'EnumerateUserFiles' method. The code includes various #include directives and logic for handling user files. To the right of the code editor is a large circular Unreal Engine logo. Below the logo, the text 'UNREAL ENGINE' is displayed. Further down, there is another code editor window showing JSON-like schema definitions for 'OnlineSessionInterfaceOculus' and 'OnlineSubsystemOculus'. The bottom of the screen shows the 'Output' window and some repository information.

```

1 // #include "OnlineUserCloudOculus.h"
2 // #include "OnlineSubsystemsOculusPrivate.h"
3 // #include "Misc/ConfigCacheIni.h"
4 // #include "OnlineSubsystemsOculusTypes.h"
5 // #include "OnlineIdentityOculus.h"
6 // #include "OnlineSubsystemsOculusPackage.h"
7 // #if USING_CODE_ANALYSIS
8 // #pragma warning( disable : ALL_CODE_ANALYSTS_WARNINGS )
9 // #endif // USING_CODE_ANALYSIS
10 // #if USING_CODE_ANALYSIS
11 // #pragma warning( disable : ALL_CODE_ANALYSTS_WARNINGS )
12 // #include <algorithm>
13 // #endif // USING_CODE_ANALYSIS
14 // #pragma warning( pop )
15 // #endif // USING_CODE_ANALYSIS
16
17 const FString FOnlineUserCloudOculus::SEPARATOR = TEXT("/");
18
19 const FString FOnlineUserCloudOculus::DEFAULT_BUCKET_KEY = TEXT("DefaultUserCloudBucket");
20
21 const FString FOnlineUserCloudOculus::ALL_BUCKETS_KEY = TEXT("UserCloudBuckets");
22
23 FOnlineUserCloudOculus::FOnlineUserCloudOculus(FOnlineSubsystemOculus& InSubsystem)
24     : OculusSubsystem(InSubsystem)
25     , EnumerateBucketsCounter(-1)
26 {
27     // see if the game defined a default storage bucket
28     GConfig->GetString(TEXT("OnlineSubsystemOculus"), *DEFAULT_BUCKET_KEY, DefaultBucket, GEngineIni);
29
30     // build a list of all the defined buckets
31     GConfig->GetArray(TEXT("OnlineSubsystemOculus"), *ALL_BUCKETS_KEY, Buckets, GEngineIni);
32
33     // make sure the default is included in the buckets list
34     if (!DefaultBucket.IsEmpty())
35     {
36         Buckets.AddUnique(DefaultBucket);
37     }
38 }
39
40 void FOnlineUserCloudOculus::EnumerateUserFiles(const FUniqueNetId& UserId)
41 {
42     auto LoggedInPlayerId = OculusSubsystem.GetIdentityInterface()->GetUniquePlayerId();
43     if (!LoggedInPlayerId.IsValid() || UserId != *LoggedInPlayerId)
44     {
45         UE_LOG_ONLINE_CLOUD(Warning, TEXT("Can only enumerate save data for logged in player"));
46         TriggerOnEnumerateUserFilesCompleteDelegates(false, *LoggedInPlayerId);
47     }
}

```

А вот когда вы уже уверенно научитесь пользоваться блюпринтами – то C++ пойдёт проще.

11. Actor, Pawn, Character

В Анериле есть множество классов. Я бы даже сказал, что очень много. Но самые важные – это Actor, Pawn, Character. Особо останавливаться здесь не буду, советую посмотреть моё [видео](#) по их отличиям.

12. Begin Play

При создании вышеуказанных классов в графе Graphs у вас будут три события. Начнём с Begin Play. Это событие, которое отвечает за то, что должно у вас произойти в момент начала игры. Например, спавн врагов.

13. Event Tick

Второе событие – Tick. Тик – это очень важная вещь для оптимизации. И довольно сложная. Советую посмотреть видеоуроки на эту тему.

14. ActorBeginOverlap

Третье событие – BeginOverlap. Важное событие, которое отвечает за то, что делать, если актор будет пересечён другим актором (например, чекпоинт в играх).

15. Коллизии

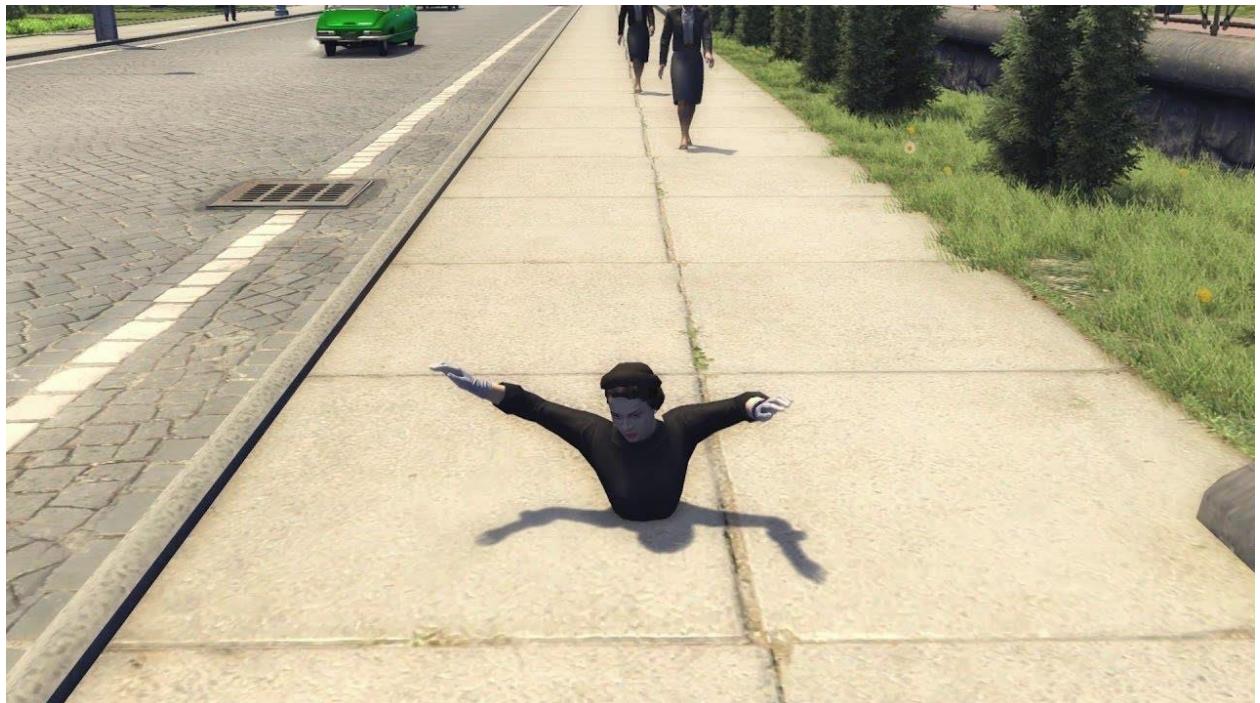
Пожалуй, это мега-мега-мега-мега-мега-мега-мега-мега-мега важная вещь. И крайне сложная. У меня есть большое [видео](#) на эту тему.

Дело в том, что все игры состоят из коллизий. Коллизии – это столкновения с чем-либо. Например, в любой игре персонаж стоит на земле. Но на самом деле происходит коллизия вашего персонажа с землей. Если бы коллизии не было, то ваш персонаж просто падал бы под землю, проходил сквозь стены и так далее.

Наверняка вы помнили неприятные ситуации в играх, когда вы проваливались под землю или застревали в объектах.

Обыватели называют это «провалиться в текстуры», но в текстуры «провалиться» нельзя, потому что текстуры – это по сути картинка. А картинки не имеют коллизии. Поэтому я всегда говорю, в случае таких багов, что я провалился сквозь/через коллизию. Это больше похоже на правду.

Поэтому, пожалуйста, прежде чем идти дальше – попробуйте разобраться с коллизиями. Эта тема невероятно важная, уж поверьте.



16. ИГРОВАЯ ЗАДАЧА: ПРИЗРАК

Посмотрев вышеуказанное видео или в целом если вы чувствуете, что разобрались с коллизиями, то сделайте, пожалуйста, такую примитивную игру:

Ваш персонаж – призрак. Но призрак крайне плохой. Потому что он может проходить сквозь стены, но не может проходить сквозь какой-то другой объект (придумайте сами).

Поэтому создайте Blueprint с типом Actor, засуньте туда какой-нибудь Cube и сделайте таким образом, чтобы ваш персонаж проходил сквозь него.

И создайте Blueprint другой, тоже с типом Actor. Но пусть там вместо Cube будет какой-нибудь другой объект (например, Sphere),

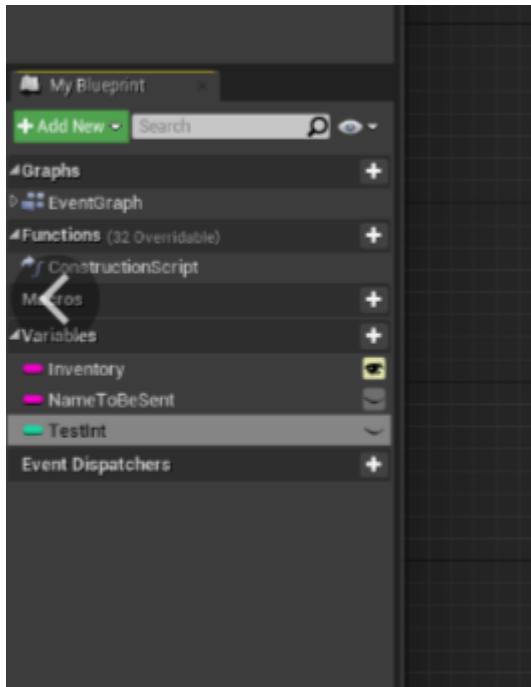
И пусть ваш персонаж проходит через стену, но не проходит через сферу.

Если вы не знаете, как запускать игру, то советую посмотреть это [видео](#).

17. Variables (Переменные)

Если вы программист, то для вас нет ничего страшного в Boolean, Integer, Float и так далее. Если же вы новичок в программировании, то без переменных абсолютно никак. Любая программа (даже ваша кофеварка в квартире или пульт от телевизора) – была сделана с помощью переменных.

Поэтому изучите, пожалуйста, переменные. В Интернете куча уроков по переменным в Unreal Engine. Скажу больше, что эти же переменные используются в **БОЛЬШИНСТВЕ** языков программирования.



Разберётесь с переменными в Unreal Engine – сможете орудовать переменными в C++, Python, Java и так далее. Программист без знания переменных – водитель без педалей. Не уедете.

18. ИГРОВАЯ ЗАДАЧА: ВАШ ПЕРВЫЙ ПЛАТФОРМЕР

Всем новичкам я рекомендую учиться создавать игры именно с платформеров. Они легки в реализации и не требуют множества знаний, как, например, шутеры, где нужны знания компонентов и так далее.

Поэтому теперь вы знаете, как создавать примитивные кубики (из первого уровня этого файла), а также знаете как работают коллизии и переменные (из второго уровня этого файла).

А это означает, что вы способны эти знания объединить.

Задача: создайте простой платформер. Сделайте следующее:

1. Создайте блюпринт Actor со сферой;
2. Создайте блюпринт Actor с «полом» (это может быть куб), по которому ваш персонаж будет ходить;
3. Создайте блюпринт Actor с каким-нибудь «вредным полом». Если ваш персонаж будет вставать или прыгать на этот пол, то персонаж будет падать вниз, потому что у пола такая коллизия, что он пропускает вашего персонажа.

Цель игры: чтобы ваш персонаж собрал три монетки (Sphere) и с помощью Print String было выведено слово «Победа!».

Совет: можете посмотреть моё старенькое [видео](#) по сбору предметов.

19. Что дальше изучать на этом уровне

Как я и говорил вам, то тема блюпринтов – необъятная. Находите уроки и просто повторяйте их. И рано или поздно вы будете там как рыба в воде и без проблем сможете создавать свою логику.

Но дополнительно я бы вам посоветовал ещё изучить следующие вещи, прежде чем идти к следующему уровню (в Интернете есть масса уроков):

- Что такое Static Mesh и Skeletal Mesh, а также в чём их отличия.
- Как подключать свои кнопки и делать управление персонажем с помощью Enhanced Input.

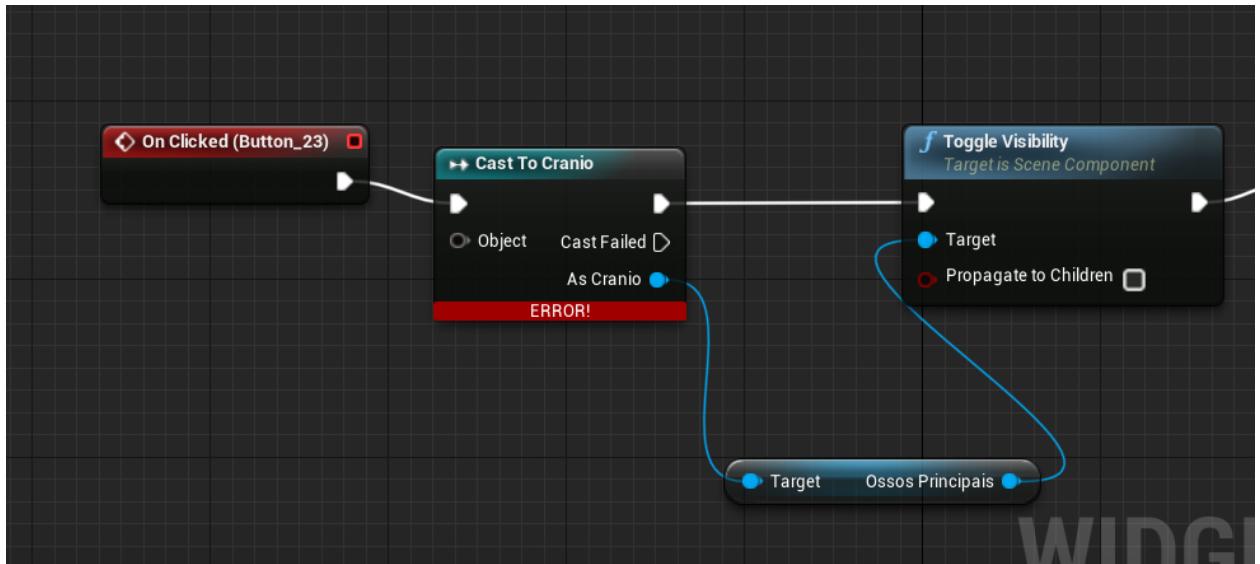
- В чём отличие Functions от Macros. Начните с ними работать.
- Что такое Construction Script.
- Что такое Components в Blueprints.
- Что такое GameMode.
- Timers by Function и Timers by Event.
- Массивы.
- ООП.
- Event Dispatchers.
- Actor Component Blueprint.

ТРЕТИЙ УРОВЕНЬ

(Учимся взаимодействовать с окружающим миром и создаём свой шутер)

20. Cast To и Get Actor Of Class

Итак, теперь мы переходим к самому интересному – к взаимодействию с предметами. Например, вам нужно уничтожить врага. А как нам указать Анрилу, какого именно врага уничтожить? Вот Анрилу и нужно показывать это с помощью Cast To и Get Actor of Class.



Вообще Cast To использовать не всегда нужно, потому что вещь может быть полезна для простой логики, но для сложной логики лучше использовать Blueprint Interface (об этом позже).

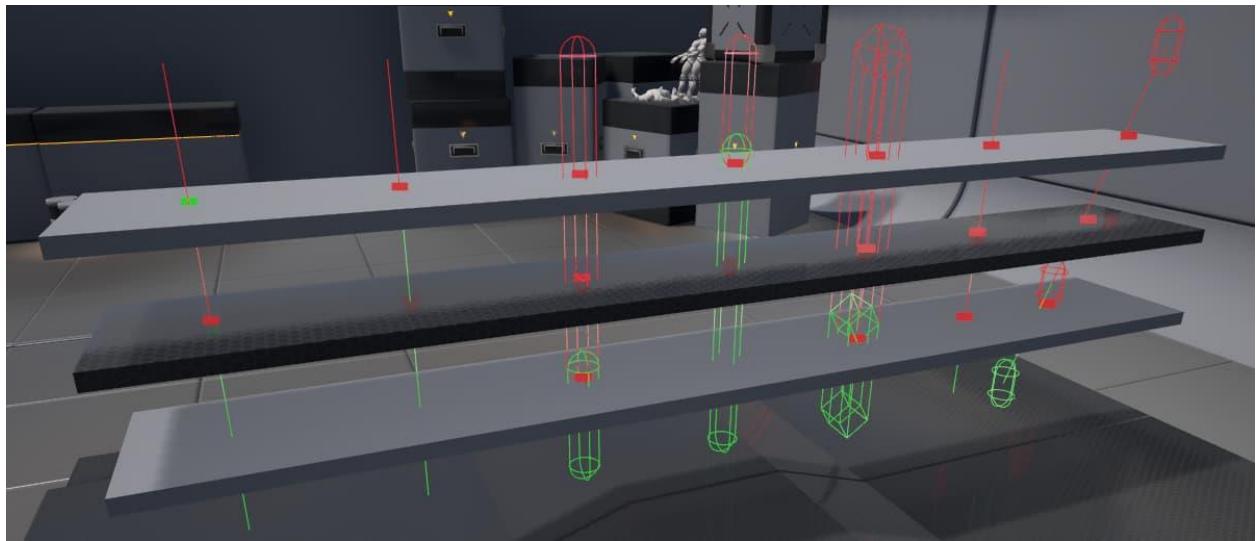
Поэтому советую посмотреть уроки по Cast To и Get Actor Of Class.

21. Trace (Трассировка)

И снова мега-мега-мега-мега-мега важная вещь. Не бывает игр, где нет трейсов. Я НАСТОЯТЕЛЬНО вам говорю о том, что вы ОБЯЗАНЫ знать трейсы.

Что такое трейсы и как они работают – смотрите в моём [видео](#).

Если коротко – то с помощью трейсов определяется, попали ли вы по врагу пулей? Какая поверхность у вас под ногами, чтобы издать соответствующий звук? Достаточное ли у вас расстояние до предмета, чтобы его поднять? И так далее.



22. Blueprint Interface

Вещь, которая будет говорить о вас, как о хорошем разработчике. Blueprint Interface позволит вам настраивать гибкое взаимодействие с окружающим миром.



Казалось бы, почему нельзя использовать Cast To?

Снова отсылаю вас на своё [видео](#), где рассказываю как работать с Cast и Blueprint Interface и в чём их отличия.

23. Sockets (Сокеты)

Итак, мы с вами теперь затронем немногого тему анимаций.



Давайте представим, что вы с помощью Trace подняли оружие, а Blueprint Interface сигнализирует о том, что теперь вы можете стрелять. Но вопрос: а как нам прикрепить оружие или иной другой предмет к персонажу? Способов много, но самый популярный – это сокеты.

Советую посмотреть это [видео](#).

24. ИГРОВАЯ ЗАДАЧА: ПРОСТОЙ ШУТЕР

Ну что ж. Теперь вы можете подбирать предметы (пусть и без анимации) и прикреплять оружие к сокету. Тогда вам задачка сделать простой шутер:

1. Создайте Blueprint Actor, который будет вашим врагом.

2. Создайте Blueprint Actor, который будет оружием.
3. Подберите оружие с помощью LineTrace.
4. Прикрепите к сокету ваше оружие.
5. Добавьте к оружию компонент Projectile Movement (здесь – [видео](#)).
6. С помощью Cast To или Blueprint Interface наносите урон вашему врагу, стреляя по нему.

ЧЕТВЁРТЫЙ УРОВЕНЬ

(Учимся делать разные анимации)

25. Animation Blueprint

Но как нам теперь добавлять анимации нашему персонажу, врагам и другим акторам на сцене? С помощью Animation Blueprint.

Animation Blueprint – это блюпринт, в котором содержатся все условия для анимации бега, приседа, прыжка, подбора оружия и так далее.

Анимации – это крайне сложная тема. Скажу больше, что в игровой разработке больше всего работают над анимациями (туда же входят и лицевые анимации и так далее). Потому что если анимация в процессе игры сломается – ловите баг и «Т-позу».

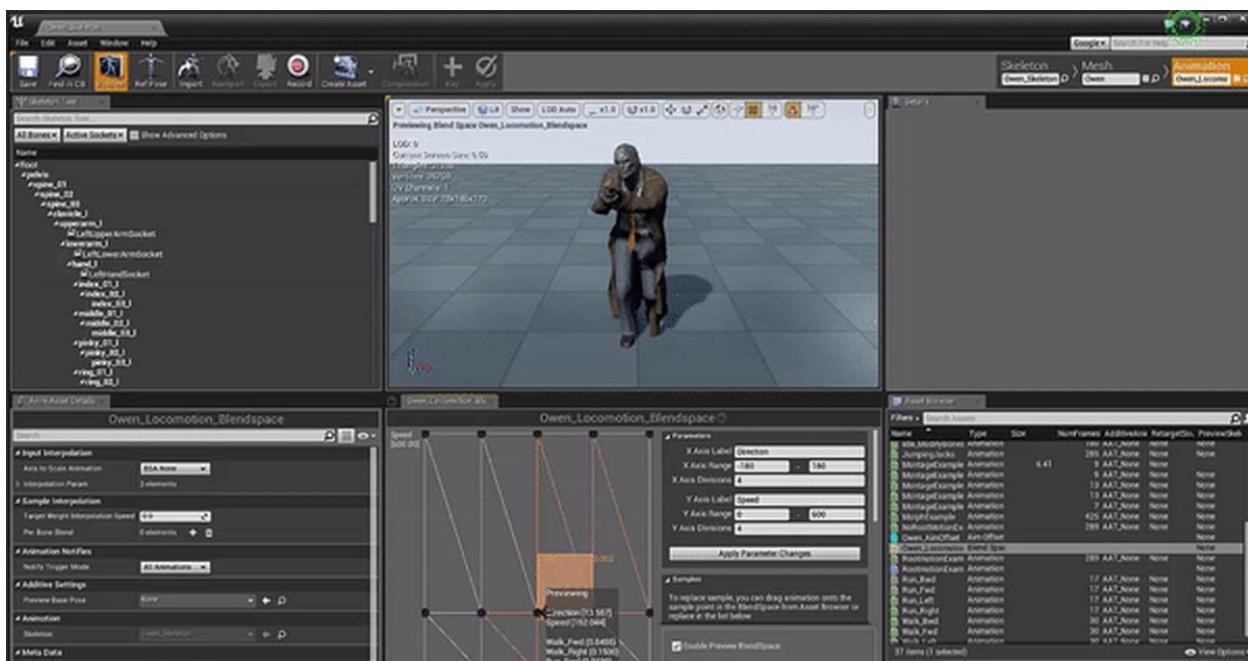


Советую посмотреть это [видео](#). Оно понятное и небольшое по продолжительности.

Как импортировать анимации в Ариал – [видео](#).

26. Blend Space

Вышеуказанное видео тоже рассказывает про принцип работы Blend Space, но всё-таки вынесу это в отдельный раздел.



Хочу отметить, что есть Blend Space 1D и просто Blend Space. Посмотрите отличия в этом [видео](#).

27. Aim Offset

И снова отсылаю вас на [видео](#). Но если коротко, то Aim Offset позволит вам целиться с вашего оружия в разные стороны.

28. Animation Sequence и Animation Montage

Вам нужно изучить отличия. Тема непростая, но в Интернете хватает уроков. Ну и тема важная, естественно. Потому что Animation Montage отвечает за комбо удары, за анимацию подбора предметов и так далее.

29. Root Motion

Тоже сложная тема, которую нужно понять. Уроки также доступны в Сети.

30. ИГРОВАЯ ЗАДАЧА: БЕГИ, ФОРЕСТ, БЕГИ!

Тут на выбор у вас две задачи: простая и сложная.
Выбирайте любую.

Первая задача (простая): реализуйте анимацию ходьбы, лёгкого бега и спрингта. По желанию реализуйте присед и прыжок.

Вторая задача (сложная): если у вас сохранился предыдущий проект с шутером, то можете сделать анимацию стрельбы в приседе, на бегу и стоя.

ПЯТЫЙ УРОВЕНЬ

(Делаем искусственный интеллект)

31. Behavior Tree (Дерево поведения)

В Unreal можно делать искусственный интеллект многими способами. Простая логика для NPC и ботов может поместиться в одном блюпринте. Но если вы будете делать сложную логику для ботов и NPC (патрулирование, стрельба по вам, поиск укрытий и т.д.), то вам нужно изучать Behavior Tree.

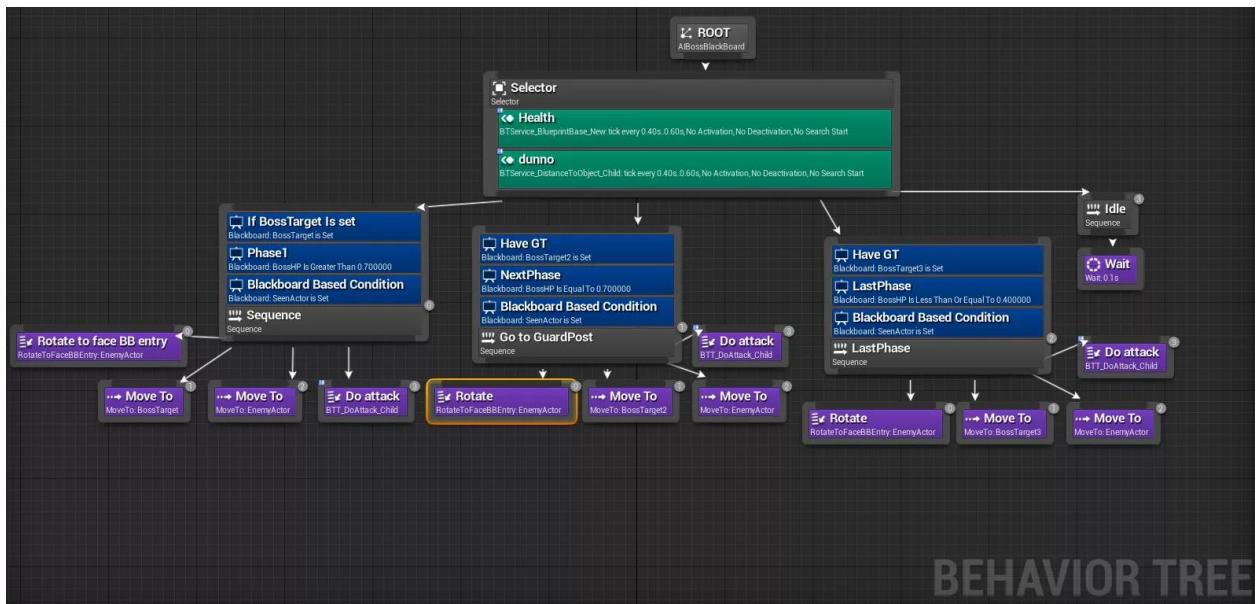
Видеоуроков довольно много в Сети.

32. Blackboard

Blackboard – это некая «доска», которая хранит в себе значения, которые будет использовать Behavior Tree. Советую также посмотреть видеоуроки в Сети.

33. Task, Decorator, Service

К сожалению, мне тут тоже проще вас отправить смотреть видеоуроки, но лучше всего – [документацию](#) Анрила.



34. AI Perception

AI Perception – это набор компонентов, которые позволят добавить боту слух, зрение, возможность предугадывать ваши действия и так далее. Советую посмотреть это [видео](#).

35. Navigation Invoker (опционально)

Вообще, когда вы будете работать с искусственным интеллектом, то в основном в видеоуроках вас научат как кидать навигационную сетку (NavMesh) на сцену, по которой будет двигаться бот, но мало кто говорит о Navigation Invoker, который позволяет динамически менять навигационную сетку.

36. ИГРОВАЯ ЗАДАЧА: НАДОЕДЛИВЫЙ ПОКЛОНИК

Как вы поняли – задача простая. Сделайте бота, который будет бегать по сцене, но как только он увидит вас, то он должен за вами бежать.

ШЕСТОЙ УРОВЕНЬ

(Вещи, которые должен знать хороший разработчик)

37. Structure (Структура)

Структура – это важная вещь, которая есть и в классическом программировании. Она позволяет содержать множество переменных в одной переменной, что очень удобно.

Например, Структура может содержать сведения о выносивости, количестве здоровья персонажа, подобранных предметах и так далее. [Видео](#) здесь.

38. Data Table (Таблица данных)

Тесно связано со структурой, о которой говорил выше. У меня есть [видео](#) на эту тему. Советую посмотреть. Таблица данных позволяет «множить» структуру, добавляя ряды. Например, вы можете прописать одну логику для оружия, а в таблице данных вы сможете гибко менять оружие в руках у персонажа, включая количество патронов и так далее. Также Data Table работает с JSON таблицами.

39. Enumeration (Перечисления)

Тоже пришло к нам из классического программирования. Можете посмотреть [видео](#) на эту тему.

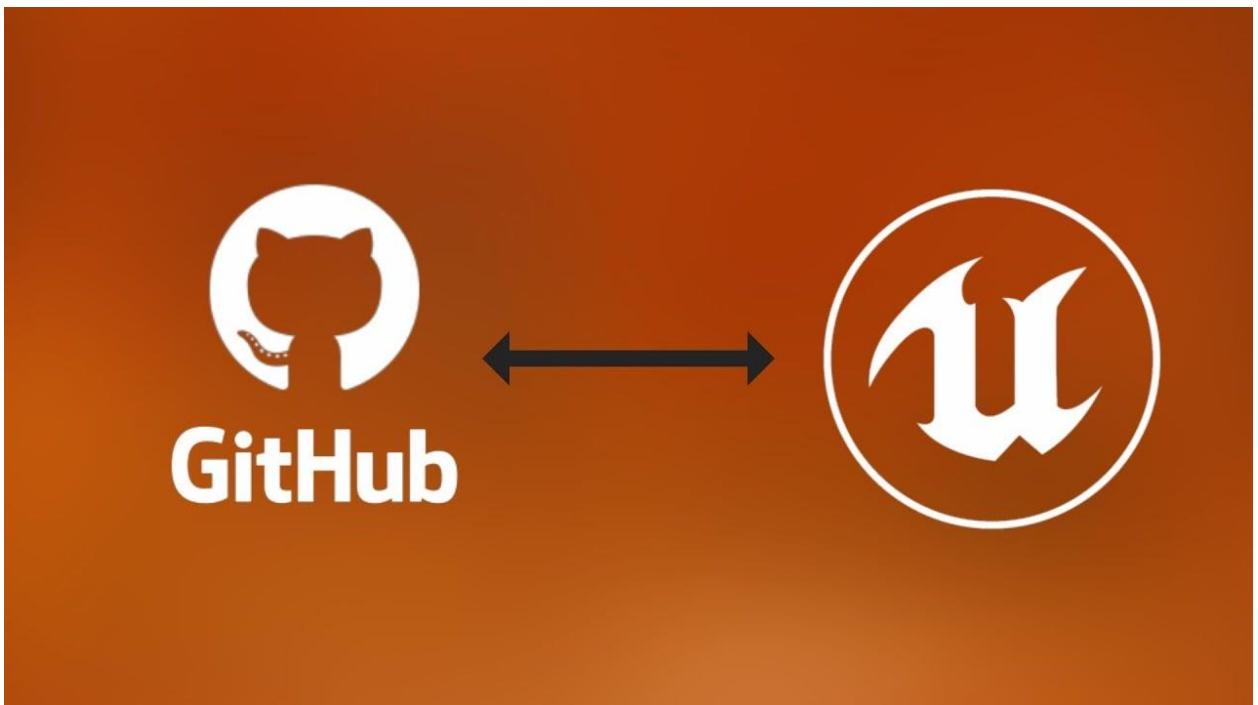
40. Debugging (Находим баги)

Анрил предоставляет множество инструментов для дебагинга и ловли багов, с которыми вы несомненно

будете сталкиваться всегда. И нужно уметь проводить дебагинг. Простое видео на эту тему [здесь](#).

41. GIT для Unreal Engine

GIT для Анрила – крайне важная вещь. Она позволяет «откатывать» ваш проект к предыдущему состоянию, что иногда необходимо, особенно в совместной разработке. Советую посмотреть моё [видео](#) на эту тему.



42. Physics

Скажу так, что физика – это очень сложная вещь, требующая много времени для изучения. Например, RDR2 разрабатывалась долго именно из-за желания сделать революционную физику (у них получилось) и анимаций (тоже получилось).

Вы можете изучать Physics Assets и прочие вещи, связанные с физикой (например, физика компонента

Cable, компонент Physics Constraints, компонент Physics Handle и так далее).

Изучив физику, вы сможете делать классные RagDoll'ы, делать реалистичное покачивание оружия на персонажах и так далее.

43. UMG UI (Unreal Motion Graphics User Interface)

Если вы зайдёте на сайты по поиску вакансий, то вы везде увидите, что разработчик должен знать UMG. Если коротко – это виджеты. Виджет здоровья, инвентаря, главное меню и так далее.



Раньше я этой темы избегал, но если засесть, то где-то за месяц-полтора вы сможете научиться создавать хорошие (пусть и не профессиональные) виджеты.

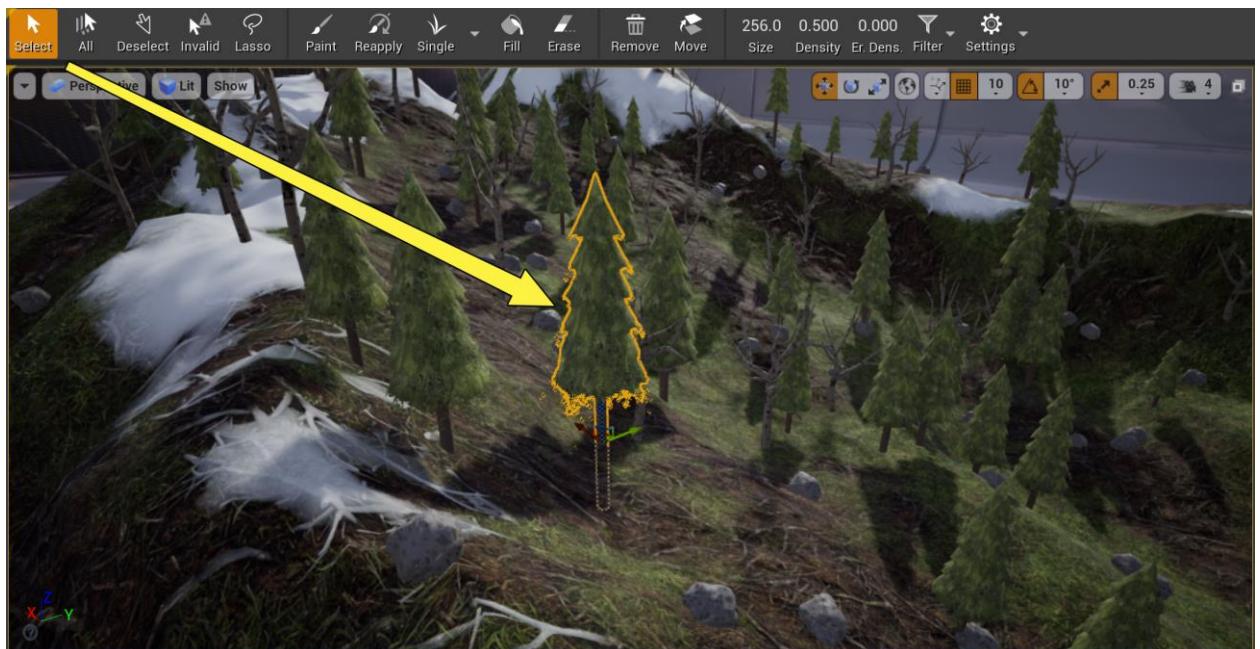
44. Landscape и Foliage (опционально)

Если вы не хотите создавать игровые уровни, поскольку этим занимаются в основе своей художники по уровням,

левел дизайнеры и так далее, то можете этот пункт пропустить.

Landscape – это создание ландшафтов (горы, возможность грузить карты высот и т.д.).

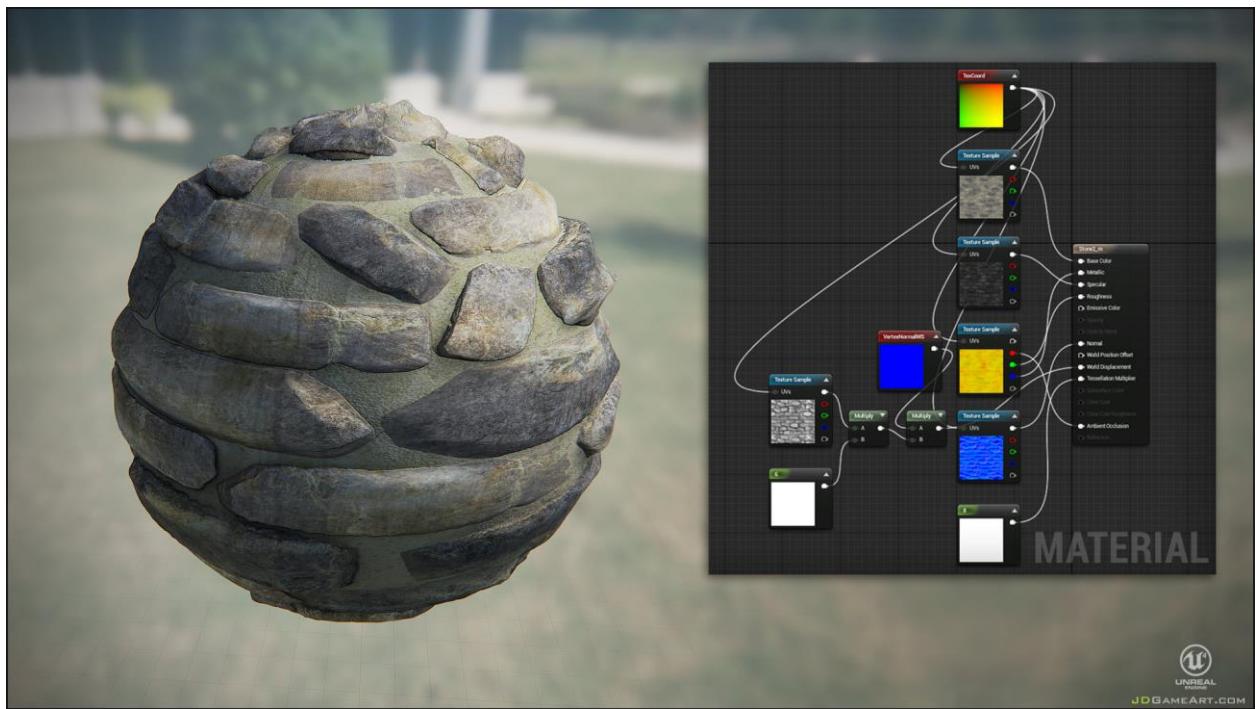
Foliage – это быстрое заполнение ландшафта (камни, деревья, ветки и прочее).



45. Materials (оpционально)

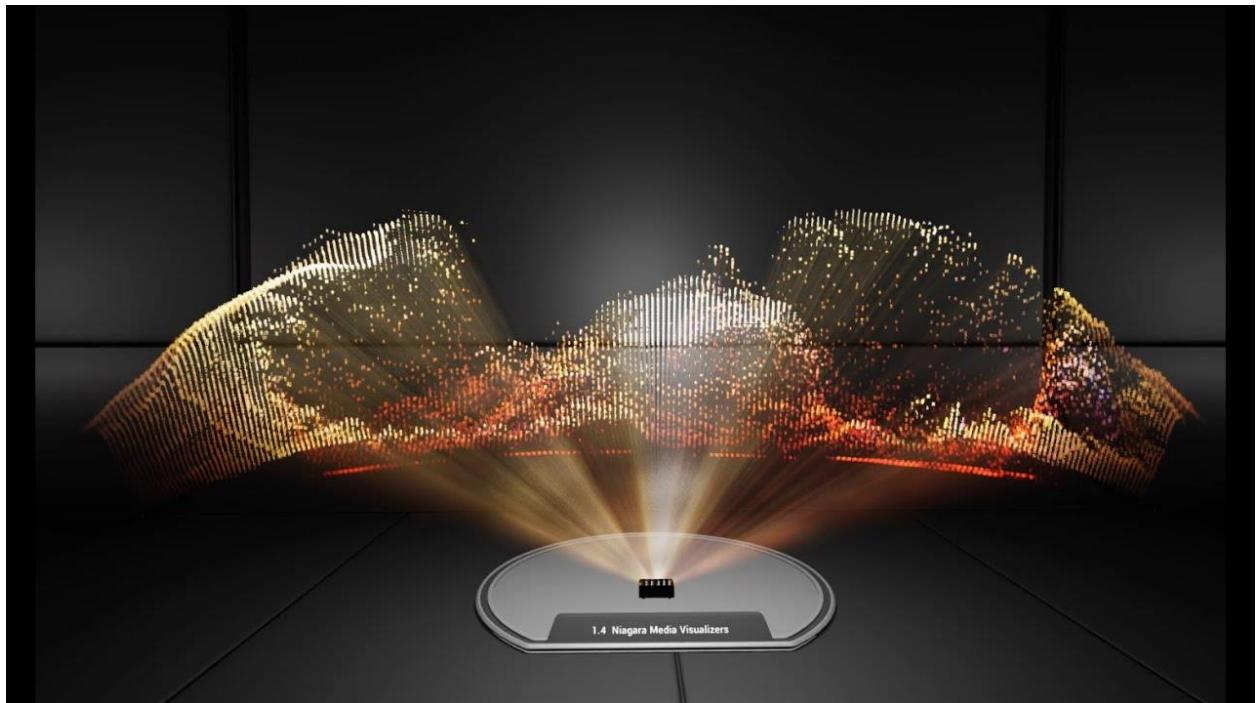
Anriil позволяет создавать очень красивые материалы и эффекты (например, голограммы, реалистичные лужи и так далее).

Если вы инди-разработчик и научитесь создавать красивые материалы, то это будет вам большим плюсом.



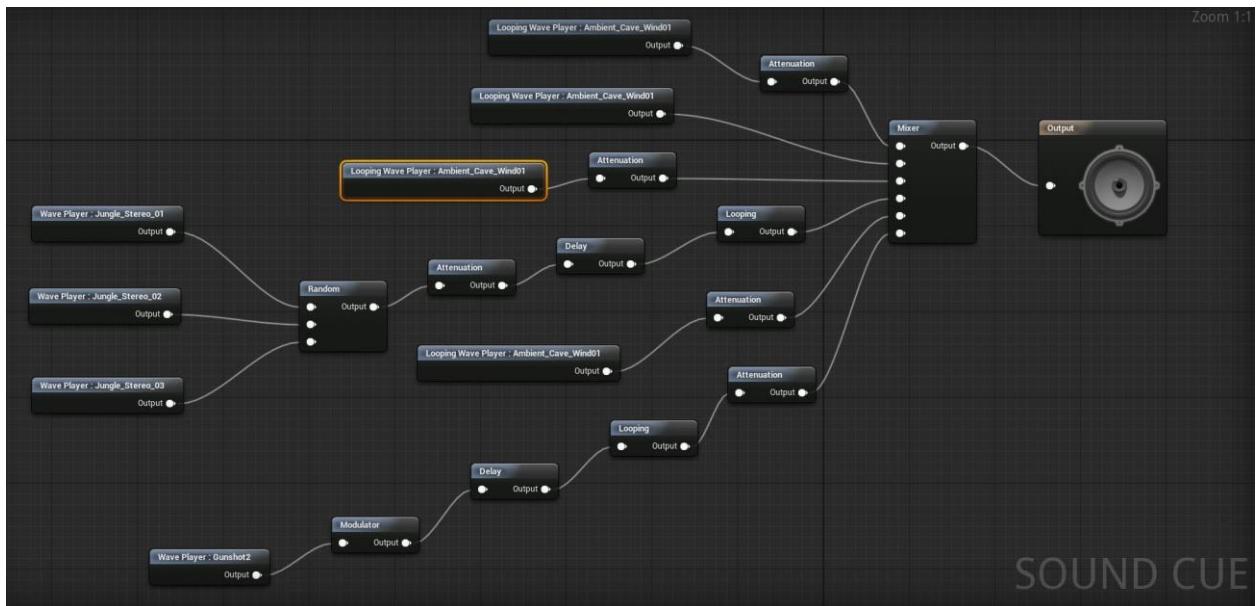
Но если вы будете работать в команде и в компании, то, скорее всего, будет отдельный человек, который будет отвечать за VFX эффекты с использованием материалов.

46. Niagara System (опционально)



Niagara работает совместно с материалами и позволяет создавать различные VFX эффекты.

47. Audio (опционально)



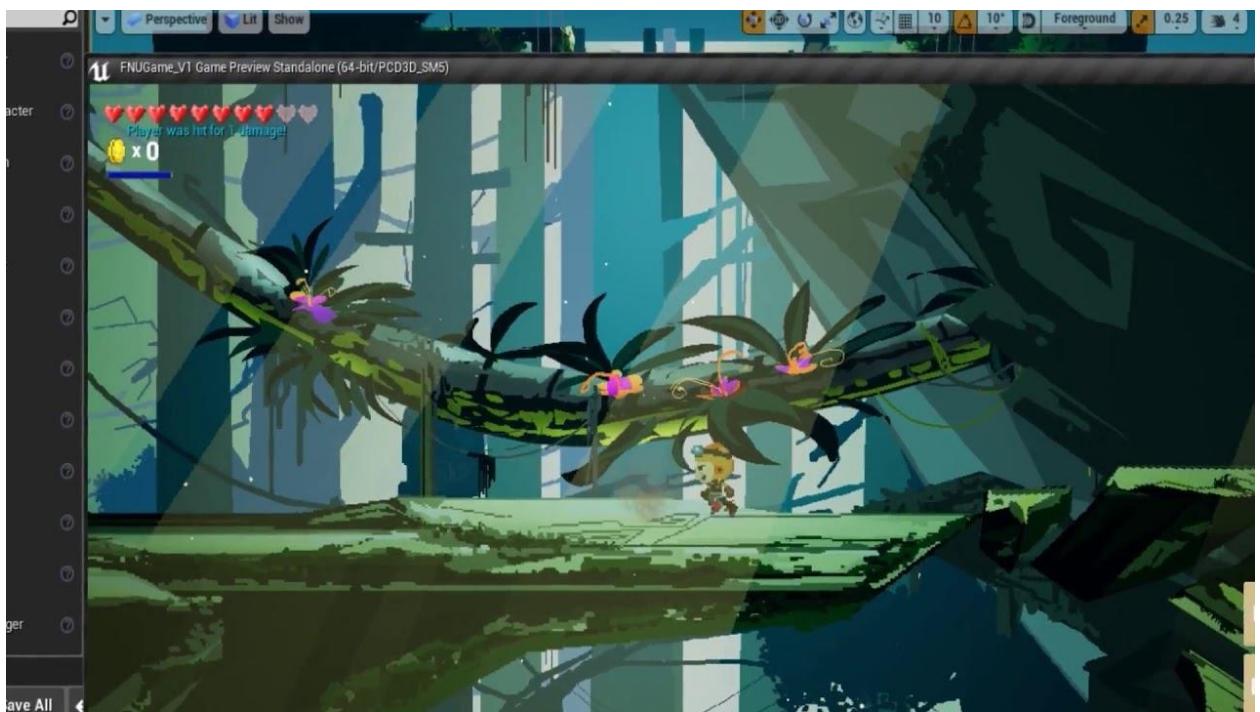
Анрил позволяет работать со звуком. Конечно, лучше использовать специализированные программы именно для обработки звука, но Анрил позволит вам создавать классы звуков, добавлять субтитры и так далее.

48. Cinematics (опционально)



Ну и конечно внутри Анрила вы можете делать синематики, т.е. клипы/ролики/видео и так далее. Если хотите – можете изучить, но, опять же, если будете работать в компании, то синематики будут делать специальные люди, а не программист. Хотя бывают разные ситуации, конечно.

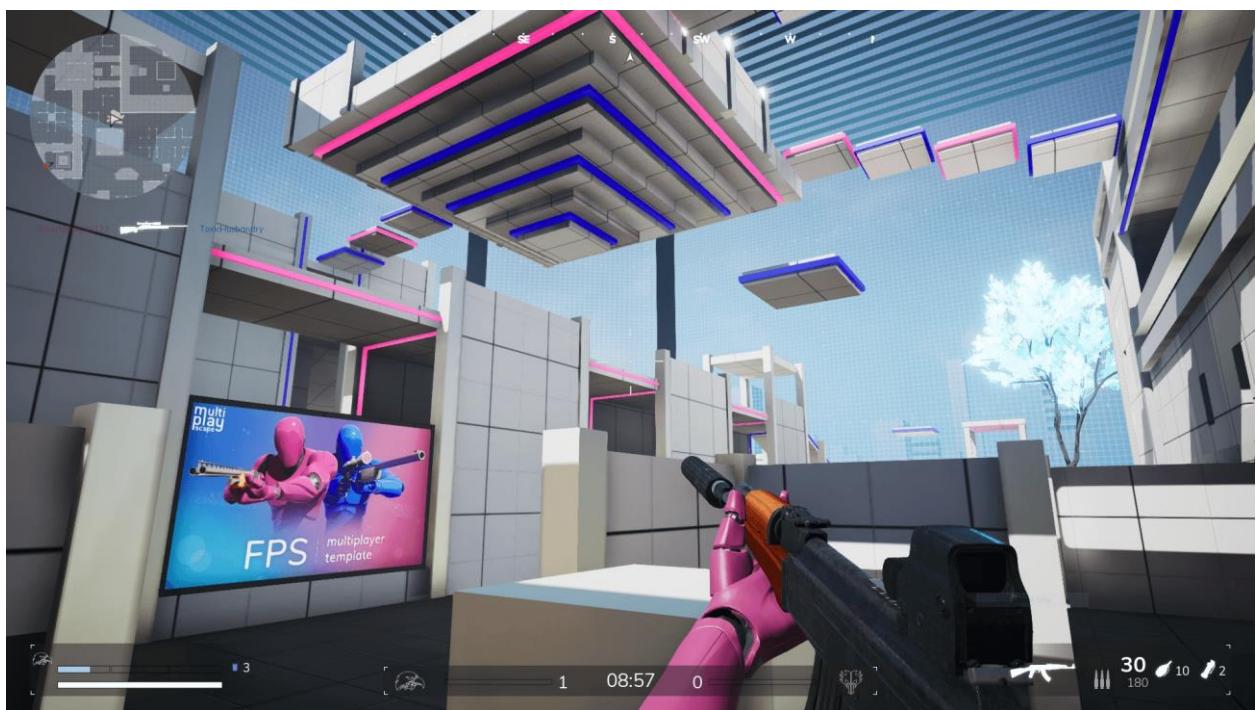
49. Paper2D (опционально)



Анрил позволяет вам создавать 2Д игры по типу *blasphemous* с помощью FlipBook, Sprite и так далее. Если вы хотите создавать такие игры – то «опционально» меняется для вас на «обязательно». Работать с Paper2D в Unreal очень удобно.

50. Multiplayer и репликация

А вот тут мы уже слово «опционально» убираем. Потому что сейчас на рынке выходит очень много онлайн игр и зная мультиплер вы увеличиваете свой шанс на участие в интересном проекте.



Но почему я мультилеер поставил именно на 6 уровень, так далеко от начала? Потому что я не рекомендую сразу пытаться изучать мультилеер, не зная предыдущих вещей (базы). Потому что если вы не знаете базу, то помимо базы вы будете ещё параллельно изучать мультилеер, который имеет множество особенностей, включая сложных, поскольку при мультилеере нужно учитывать репликацию (связь сервер-клиент и т.д.)

СЕДЬМОЙ УРОВЕНЬ

(Вещи, которые знают опытные пользователи)

51. Game Instance

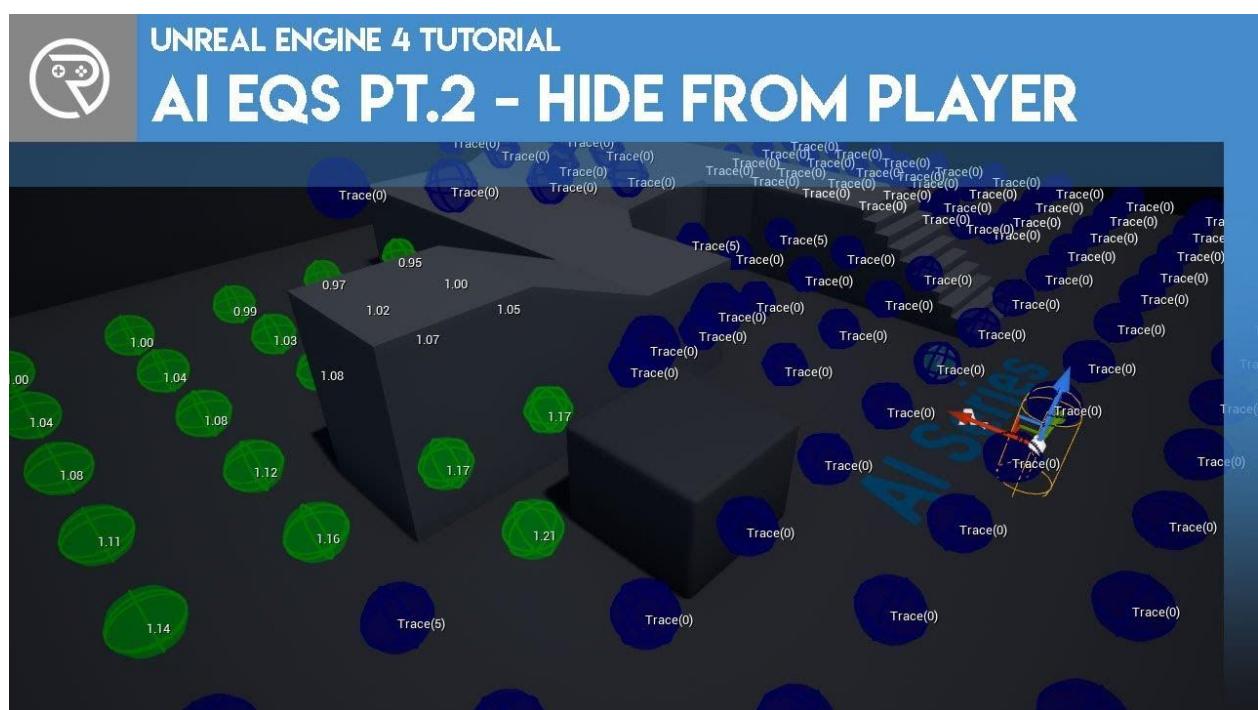
Только в момент, когда вы способны создавать сложную логику, уровни, то вы должны знать Game Instance. Это база. Видео – [здесь](#). Без знания GameInstance вы не сможете сделать нормальный SaveGame и переходы между уровнями.

52. Save Game

В игре должна быть функция сохранения и загрузки. И делается эта логика не самым простым способом. Тем не менее, класс Save Game обязательно знать для опытного разработчика. Самое плохое, что можно сделать – это забагованный сейв, когда игрок потратил несколько часов на игру, а его сейв при следующем заходе в игру сломался. Тут вам рефанд обеспечен.

53. Environment Query System (EQS)

Если вы хотите иметь сильный искусственный интеллект у ваших ботов в игре, то EQS – важная вещь. К сожалению, простых уроков по этой системе нет, поэтому рекомендую самостоятельно найти уроки по EQS.



54. World Partition (оциально)

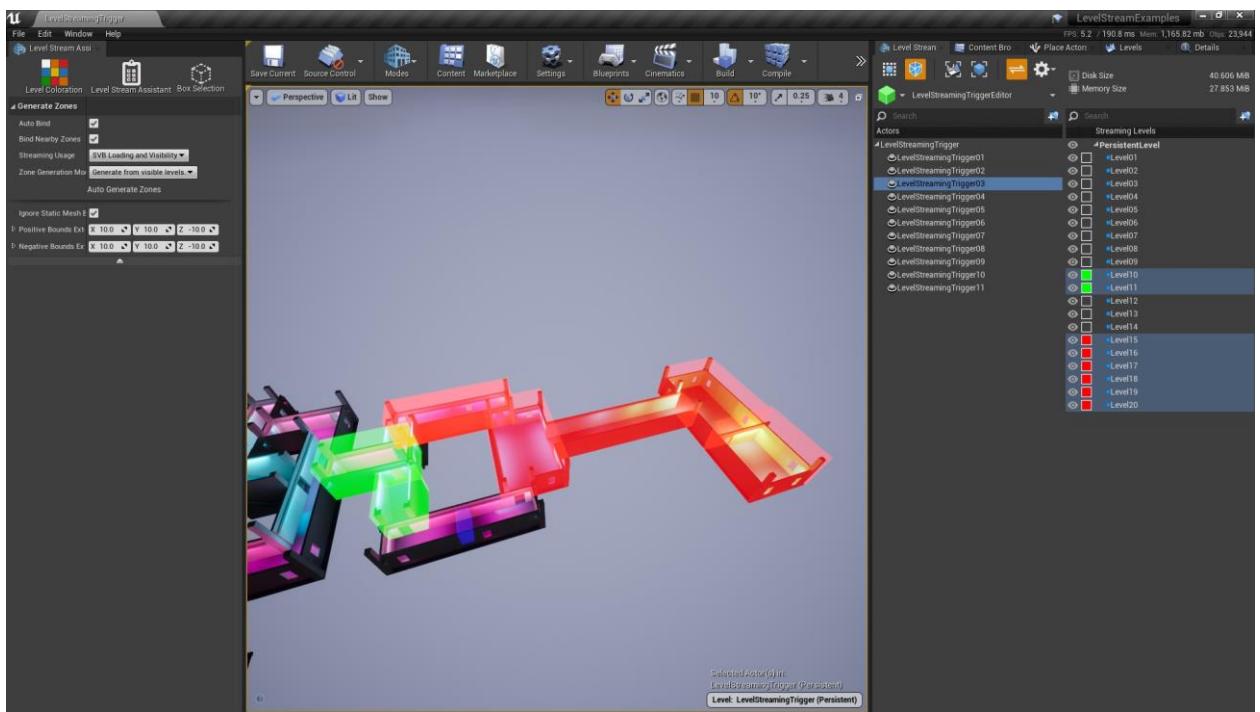
Позволяет делить большие миры на «кусочки». Обычно этим занимаются дизайнеры уровней. Почему это относится к седьмому уровню? Потому что делить

большие уровни на маленькие нужно ещё уметь правильно, иначе – баги, просадки FPS и так далее.



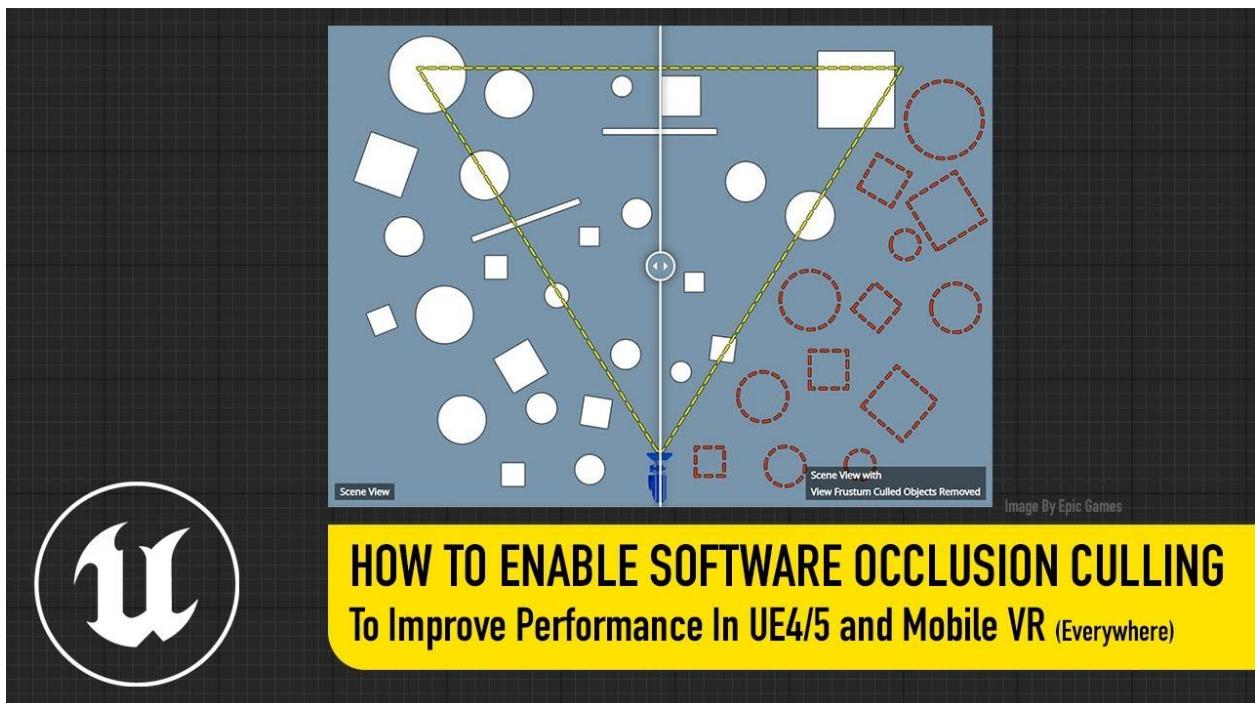
55. Level Streaming (опционально)

Тоже в основе своей занимаются этим дизайнеры уровней. Позволяет прогружать уровни в определенные моменты (например, когда персонаж ползёт по вентиляции).



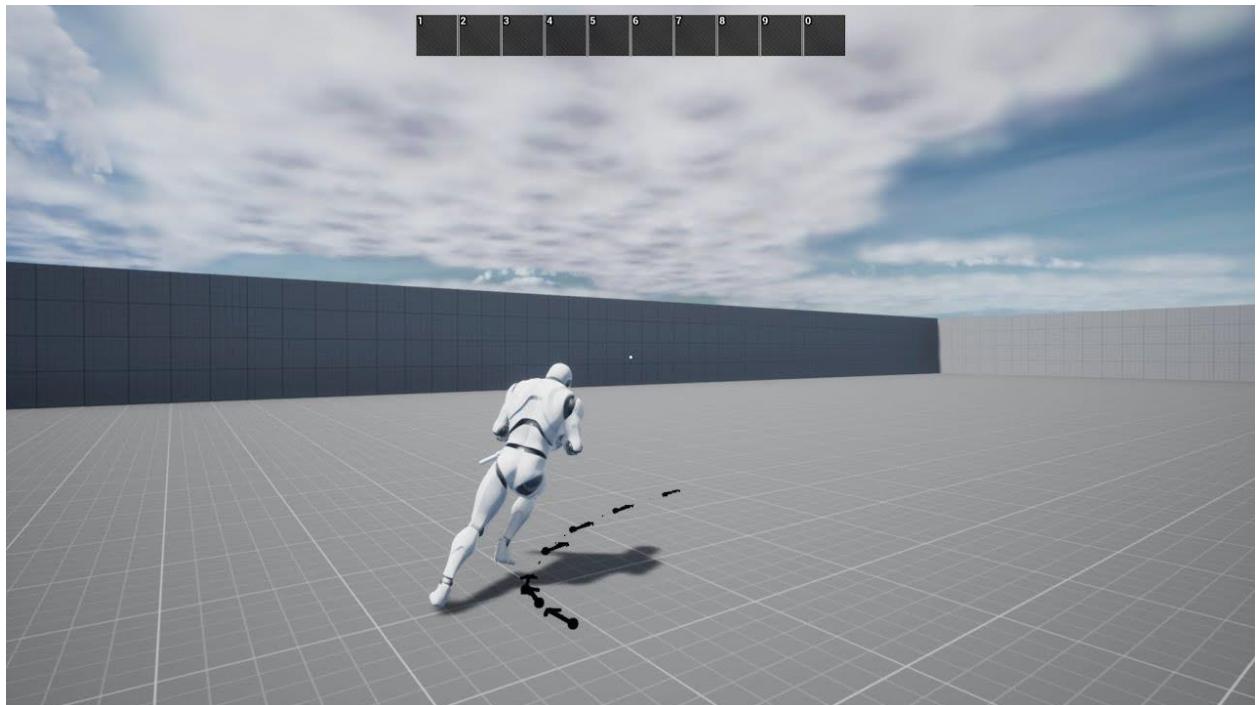
56. Culling

Есть несколько типов Culling («Выборка»). Т.е. эта система позволяет скрывать объекты, которые находятся вне зоны видимости камеры, не прогружать объекты за стеной и так далее.



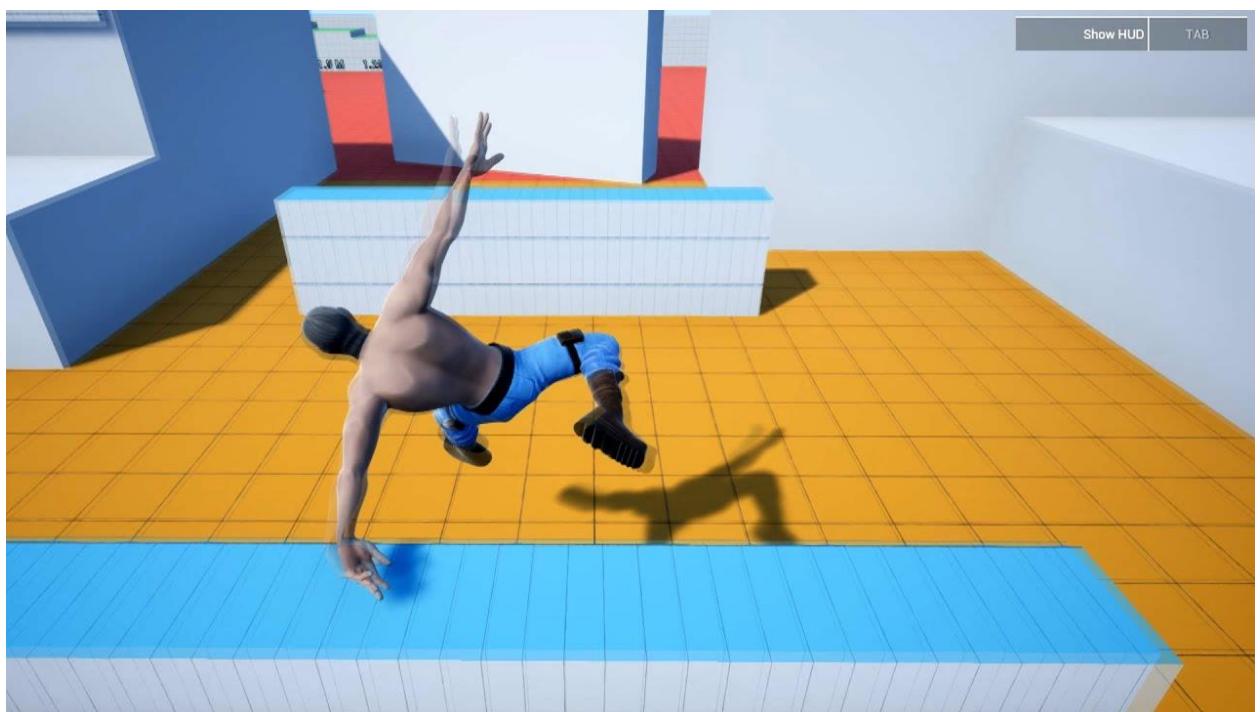
57. Motion Matching

Система, которая позволяет создавать реалистичные анимации.



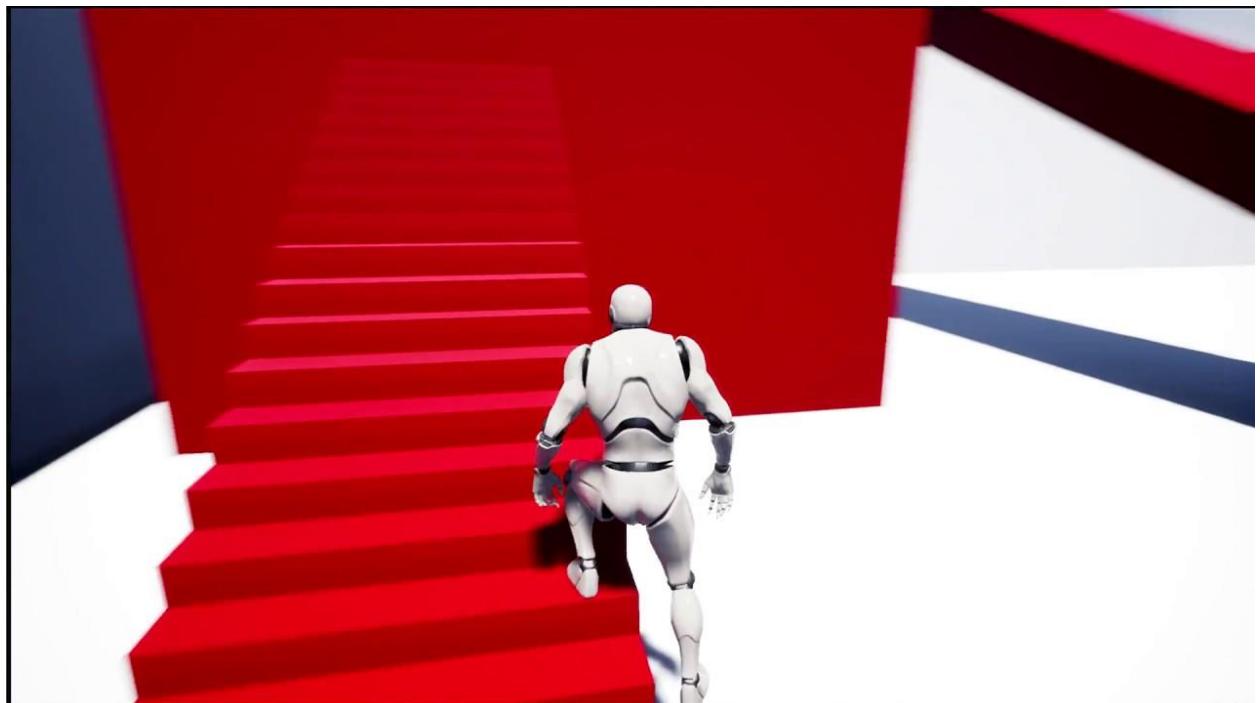
58. Motion Warping

Система, которая позволит «подстраивать» анимации под окружающую среду. Например, прыжок через стену зависит от высоты и толщины стены.



59. ИК

Делать качественную инверсивную кинематику нужно уметь, иначе получится ситуация, что ваш персонаж наполовину будет висеть в воздухе (как в игре Gollum)



ВОСЬМОЙ УРОВЕНЬ

(Вещи, которые знают профессионалы)

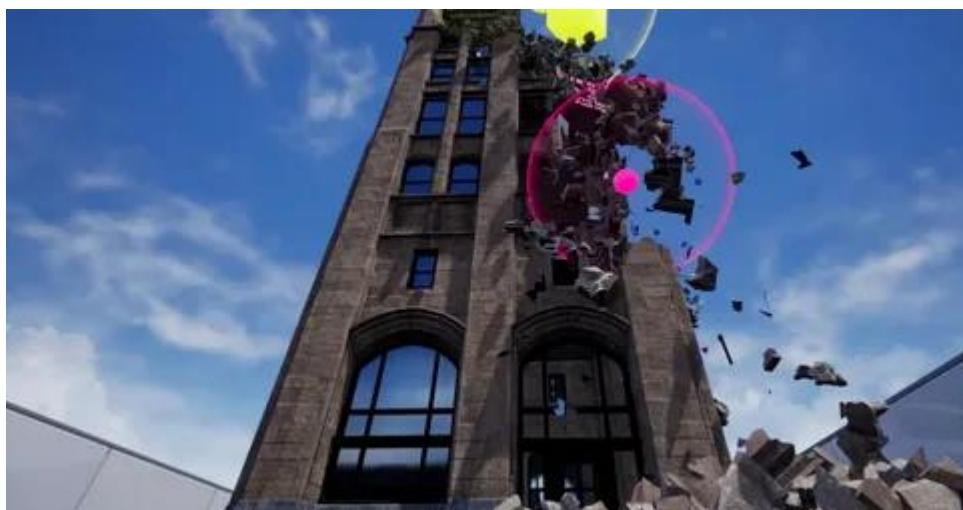
60. Packing (Сборка)

Собирать ваши проекты нужно иметь. Конечно, сборку можно делать и на начальном уровне, но если у вас большая игра (например, AAA), то собирать готовый проект требует большой сноровки.

Например, ваш проект не запакуется, если у вас в проекте будет папка... Apple! Почему? Надеюсь, если вы дочитали до этого момента, то пришлётте мне свой ответ на roveddy@yandex.ru :)

61. Plugins

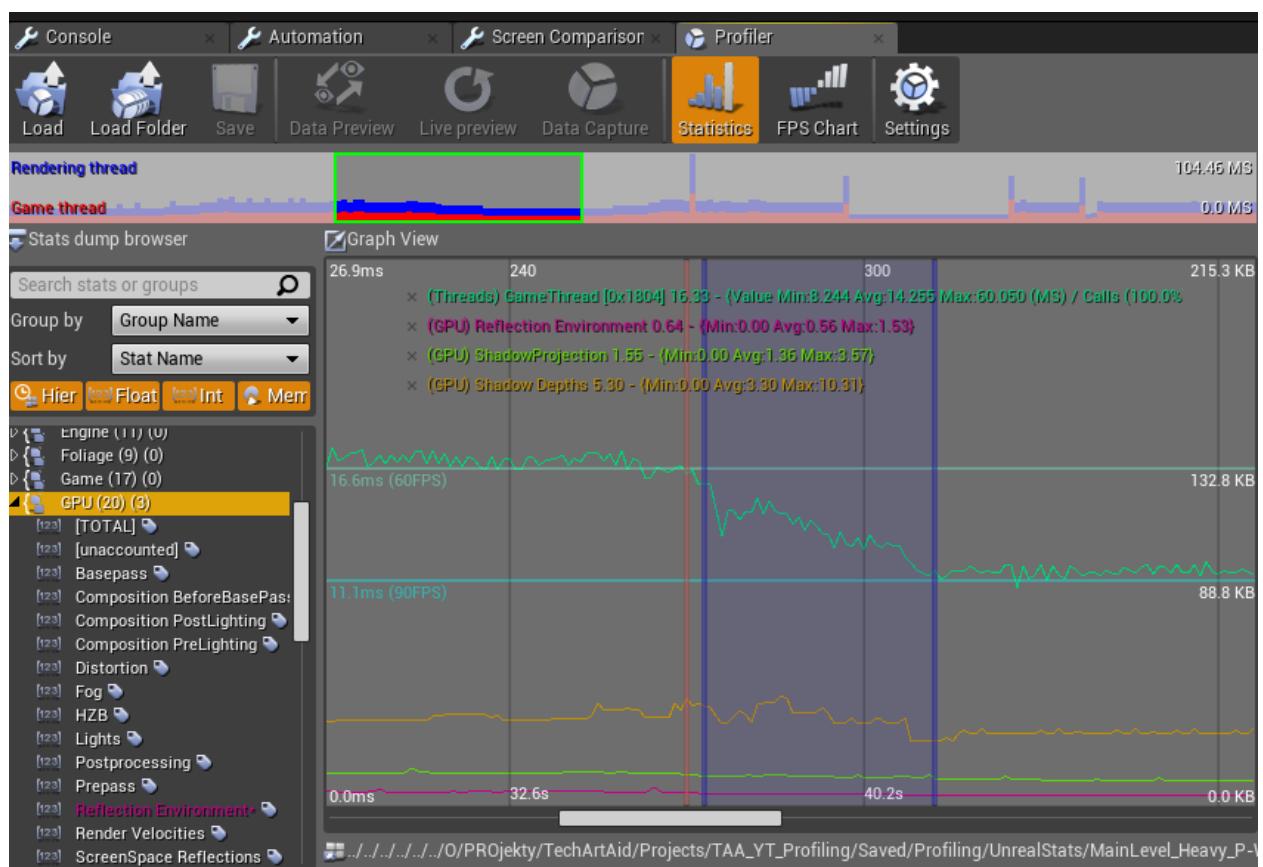
Профессионалы способны создавать свои плагины для Unreal Engine. Почему это относится к последнему уровню? Потому что если вы хотите создавать плагины для Unreal Engine, то вы, во-первых, должны знать как работает движок Unreal Engine (а это нужно залезть ему далеко «во внутренности»), а также вам нужно знать отлично C++.



Конечно, простые плагины можно сделать более-менее легко (что очевидно), но если вы будете уметь делать плагины по типу Chaos Destruction – то это будет высший уровень.

62. Profiling

В анриле есть система профайлинга, которая довольно сложная. Профайлинг позволяет посмотреть чуть ли не сколько памяти ОЗУ или видеопамяти тратится на определенные текстуры/модели/логику и так далее. Такая работа движка с «внутренностями» компьютера требует большой сноровки.



Друзья, на этом всё. Конечно, ваше видение изучения Unreal Engine может отличаться. Я работаю в движке уже

несколько лет, у меня есть закрытый Telegram-чат, где я вижу вопросы и сложности новичков. Ну и мой YouTube канал с комментариями людей. И проанализировав свой опыт и вопросы людей, то я составил такой вот «Step-by-Step» гайд.

Если вы новичок, то пусть этот текст у вас будет просто как ориентир, что вы можете изучать и в какой последовательности. Также я ещё не написал ничего про Lumen и Nanite, потому что если рассматривать каждый компонент (например, Physics Handle), каждый класс (например, Drag and Drop) и другие вещи, то... Я сойду с ума, поскольку я таким образом поставлю себе задачу в одиночку переписать всю документацию Unreal Engine:)

Поэтому здесь – только основа. И уже работая с этой основной вы наверняка зацепите и другие дополнительные интересности.

Подписывайтесь на мой YouTube канал:

<https://www.youtube.com/@makeyourgame2210>

Подписывайтесь на мою группу в ВКонтакте:

<https://vk.com/makeyourgameunreal>

Ну и если вы хотите поддержать меня любой суммой и попасть в закрытый Телеграмм Чат, то можете осуществить донат здесь:

<https://boosty.to/makeyourgame>