Всем привет. Ещё раз. Я уже здесь тусуюсь весь день, но поприветствую всех, кто зашел послушать мой доклад. Соответственно сегодня я расскажу о том, как правильно подойти к Unreal чтобы работать с ним быстро и эффективно и какие требования поставить перед собой и командой чтобы это было ещё и менее больно. Это касается на самом деле не только Unreal это касается также Unity это касается разработки любого софта. В данном случае я буду рассматривать Unreal Engine как близкий мне инструмент я буду рассматривать его как любимый мой инструмент работы и как тяжелый инструмент в плане технологий. То есть всё-таки это очень тяжелая технология это C++ со всеми его вытекающими.

Немножко обо мне раз меня попросили подготовить такое слайд. Кто вообще я такой и чем занимаюсь. На работе висят странные вот такие вот штуки будто бы я торгую шаурмой. Нет это неправда хотя шаурму периодически ем. На самом деле я работаю с Unreal Engine то ли с 2010 то ли 2011 года. У меня выпущен ряд проектов. На текущий момент я являюсь пээмом и техническим лидом Пушкин Studio. Это часть Mail.ru Group. И один из наших таких известных проектов с моим участием это Armored Warfare Assault. Мобильная армата так называемая. То есть ее можно найти в сторах можно посмотреть поиграть и понять вообще, как бы уровень проекта. Соответственно я работал также и с третьим движком. Работал и с четвёртым движком и продолжаю этим заниматься. Кроме того, у меня также есть опыт Unity и в принципе я могу их сравнивать с точки зрения пользователя, а теоретика.

Я начну с интересной части которую я обозначил как мазохизм это постановка требований в первую очередь к себе то есть это локальная работа за компьютером работа вас как творческой единицы. Пока мы говорим не про команду. Основная проблема с чего все начинается и часто этот вопрос мне задают когда в группах, в Community и прочее вот мы начали работать с Unreal когда-то мы колупали Unity что-то на нём делали а вот взялся Unreal. Он тяжелый. Мы запускаем. Нам надо писать что-то на C++. Оно долго компилируется. Долго стартует. И это вызывает у многих вопросы. Что с этим делать? Как это обойти? Как вообще жить в таких условиях? Более того даже я менеджер. У меня есть команда с точки зрения могу подойти к любому своему разработчику я увижу точно такой же экран. Для меня он хоть что-то будет говорить. Для не технических товарищей это будет просто цветные циферки. Оно компилируется. Отличная отмазка. И как ни странно многие современные технологии и это круто они компилируется в отличие от C++ в отличие от тяжелых движков моментально. Соответственно они приучают к очень простому такому интерактивному пэйплайну. Я что-то написал попробовал. О, собралось, замечательно. Оно работает. Не собралось - надо попытаться что-то с этим сделать. Что с этим надо сделать на Unreal что бы вообще всё было побыстрее? Вот здесь было несколько слайдов но они стали неактуальными я их вырезал по причине того что вышел 4.22 буквально там неделю или 2 назад и он сильно ускорился. Работа с кодом стала быстрее. Все генерации стали работать тоже быстрее. В целом не знаю они пишут типа на 30%. Я бы сказал на все 50 стало шустрее с этим работать и эффективно. Более того наконец завезли такую штуку как Life++. Это современный аналог Hoat Reload для Unreal Engine который как ни странно работает. Но для любителей Mac и Linux работает только на Windows. У нас 2/3 команды на Маках поэтому они горько плачут слезами. Смотрят на то, как нам сейчас хорошо работать. Как я по нажатию кнопочки перезагружаю код моментально и жалеют о том что они на маках. Хахахаха.

Это описан стандартный флоу работы на Unreal Engine. На самом деле неважно Unreal Engine это. Третья ли это версия. Первая ли версия. CryEngine это. С точки зрения Unity конечно есть поправки но всё выглядит так. Ты пишешь код в редакторе. Ты запускаешь редактор после того как ты написал код. После того ты проверяешь фичу которую опять же запуск редактора открытия карты нажатия кнопки Play подгрузка всех классов ассетов. Что что-то проверил, убедился. Работает - замечательно но не так как ты ожидал. Что ты делаешь дальше? Ты закрываешь редактор. Какое-то время проходит и снова пишешь код. Я это называю путём мыслителя по одной простой причине. Чтобы этим пользоваться надо думать какой код ты пишешь. Вот это вот время от 1 до 5 шага, к сожалению, после экспорта из google docs здесь немножко сбилась нумерация. Это время одной итерации. Почему именно её называют подход интерактивным? Именно так обозначены мои лекции. Потому что чем короче итерация, чем быстрее итерация тем больше мы можем сделать за один период времени. Ну так нам приходится жить о чём я также написал. В третьем движке было еще все веселее. Писать игровую логику на C++ можно было только если вас заставили сам лично я никогда подобное более такого делать не хотел.

Идем дальше. Что нам делать в этом случае? Во-первых, да мы должны думать над кодом. мы не должны тестировать путем, а давайте попробуем скомпилируется это или нет. Это часто отличает разработчика высокого класса от начинающего разработчика. Начинающий разработчик как только познаёт язык конструкции часто пользуется автокомплиторами. Автокомплиторы в этом случае они мешают развиваться. Автоклмплиторы это крутая штука, когда ты понимаешь, что ты пишешь то же самое касается C++. C++ достаточно сложный язык в плане того что он накладывает свои требования по пониманию того что вы делаете. На многих других языках там будто то Java. Вы можете гораздо вольготнее и свободнее себя чувствовать в вопросах памяти, ассетов, ресурсов и так далее. Это не подходит к C++. На C++ вы стреляете в ногу. Об этом знают все. Но это не значит, что это какой-то сложный язык. Нет надо просто принимать над тем что вы пишете. Надо знать синтаксис того что вы пишете. Как только узнали этот синтаксис вы можете им оперировать. Написать лямбду например. Написать лямбу по памяти я иногда не могу это не значит что я не работаю с этими лямдами. Но я знаю как это работает и я могу по коду понять что я написал правильно или нет. Только потом я собираю и удивляюсь что и где у меня не работает. Это не значит, что я на каждый чих свой жму компиляцию. К сожалению, такой подход жать компиляцию на каждый чих и проверить синтаксис я встречаю вокруг да около у тех, кто только начал заниматься программированием. И пришёл из таких языков как Java, JavaScript и прочее. Где-то это работает моментально. Это не работает в тяжелых языках.

Дальше. Аккуратные хеддеры и инклюды. Forward Declaration и все дела. Это главный бич Unreal, когда он только вышел. Опять же это все к тому, что вы должны понимать технологию, на которой пишете. Если вы не понимаете технологию на которой пишите Вы можете делать такие зависимости где изменения в одном файле порождает каскадные изменения по всей системе. Естественно если вы пишете проект таким образом или не знаю у вас там есть какой-нибудь базовый тип который находится в одном хеддере которой зэнклюжен во все остальные файлы вашего проекта вы поменяли что-то и попытаетесь скомпилировать. Такая компеляция займет у вас максимум времени. Чтобы было понятно компиляция крупного проекта. Мы работаем чистого может занять так минут пять например. То есть это на достаточно мощной машинке просто из-за того что я неправильно заложил архитектуру. Если вы также посмотрите как менялся Unreal Engine за последние годы и как ускорилась компиляция именно к таким же идеям они пришли сами. Вместо огромных хеддеров которые инклювятся везде например любимое мною там Engine или World они используют теперь кусочки. Это работает. Об этом надо знать. Об этом надо помнить это всё по технологии.

Дальше. Unity сборка. Это не тот Unity который движок это-тот Unity который подход к компиляции. Об этом опять же как только вы подгружаетесь движок стоит почитать. Очень интересные доклады есть от ребят которые делали в World of Tank Blitz. О том как они прикручивали Unity сборку. Суть очень простая. На самом деле в Unreal вы даже об этом не грузитесь. Главное ее не отключать когда вы работаете сам над кодом. По умолчанию всё включено. Почему я об этом говорю? Через несколько слайдов. не отключайте это когда работаете сами но в определенных случаях надо буде ее отключать и знать что такая штучка вообще в Unreal есть. Хотя это Unity, и он есть в Unreal.

Дальше. Использования роботов. Что я поднимаю под использованием роботов? Под использование роботов понимается в первую очередь простейший билд сервера. Да это конечно удобно Вы работаете на одной машине прямо здесь тестируйте всё хорошо. Это не скажем так не покрывает все задачи тестирования и контроля. Гораздо проще, когда другие роботы правильно настроенный билд машины будут делать за вас тестирования определённые сборки и проверять что-то что работает не только у вас локально, а ситуация я думаю известна всем, кто работал с кодом что у вас оно работает, но не работает на машине рядом. Такую ситуацию избегать лучше в младенчестве. Пусть машина рядом работает с самого начала это настраивается очень легко это сегодняшний день организации разработки. Ну и уважение к игровым ресурсам это такая штучка интересная. Потому что, когда я вот на предыдущем слайде рассказывал вот эти шаги. Есть шаг запустил редактор и если вы хоть раз открывали Unreal Вы можете не то что можете вы наверняка видели что первый раз Unreal стартует очень долго. Последующие разы он стартует быстрее но тем не менее иногда можно казалось бы чистый проект стартует моментально ваш проект стартует очень долго. Чаще всего разница будет именно в ресурсах. Если вы ресурсы подключили к игре таким образом что они у вас на самом старте загружаются все. А они могут быть тяжёлыми они могут быть тестовыми, могут быть лишними. Да мы получаем ситуацию когда для того чтобы проверить как изменилось поведение шапочки на персонаже вам надо подгрузить все игровые ресурсы в которых там не знаю 20 машин, 35 танков, 45 персонажей 100 мечей, это всё загружается сразу а вам нужна шапочка. Вот этот шаг забывает огромное количество людей я сам наступал на него не раз. То есть я не раз оптимизировал проекты и советовал консультировал в которых вот этот шаг с ресурсами был пропущен.

Далее совет номер один который я устал уже говорить на форумах и комьюнити. На дворе 2019 год купите SSD. Не надо работать на HDD. Не надо. HDD - это хранилище данных. Это не скажем так железо для работы. Просто купите SSD. Вы сохраните себе огромное количество времени количество нервов количество сил. Это окупится они сейчас стоят уже небольших денег. Особенно можно начать со стартовых SSD. Все говорят Unreal весит дофига надо купить SSD. Я не знаю SSD ну на 250 Гб она стоит столько же сколько терабайтный HDD хватит на здоровый проект. Понятно если вы там ещё не знаю какой-нибудь Апекс поставите, пару думов, фар край, то да конечно вам места не хватит. Но разделяйте рабочее и личное. Лучше игрушки на HDD, а на SSD работу держать. И все свои тестовые проекты. Естественно, что скажем так одной SSD сыт не будешь, да и в современном мире надо подкупить процессор там получше и память, но это всё мелочи. Когда собирают сейчас компьютер тестовый для работы, но берут без SSD либо SSD только для системы. У меня вопрос, а причём здесь технологии. Технологии HDD по скорости доступа не растут много лет. Как было 10 лет назад, так и сейчас, то есть она будет стартовать медленно. Просто купите задушите жабу. Простейший совет и он работает.

Вот это как раз моя любимая тема про ассеты. Я даже статью на эту тему писал. Там ссылочка снизу прикреплена. Естественно этой статье уже 2 или 3 года и там есть данные которые стоит перепроверить они уже называется не так. Классы называются не так потому что всё развивается но рекомендую посмотреть если кто-то с Unreal работает чуть более чем. Суть какая? Первая - ты начинаешь работать с новой технологии. На Unity одна система с ассетами связана на Unreal другая система работы с ассетами. Особенно со ссылками между ассетами. В том числе между классами но в первую очередь мы говорим об ассетах. Стоит это изучить. Совет номер 0 до того, как купить SSD изучаем матчасть. Он весьма такой конечно капитанский но возьмите документацию, почитайте. Ничего сложного в этом нету все расписано. Соответственно если вы знаете как работает система рефенсов хотя бы как проверить что у вас грузится на старте. И посмотреть, а вот эту штучку я загружаю что от нее там дальше идет. Бесценно сохраняет часы вашей жизни проведённые просто за пустыми запусками. Соответственно минимизируйте на старте приложения. Чем меньше у вас разных ассетов грузится на старте тем лучше. Как я уже говорил вы запускаете редактор вы запускаете потом тестовую игру. Вы всегда тестируйте какие-то фичи. Вы не тестируете игру целиком чаще всего. Игра целиком это скажем так отдельная задача тестирования — это не задача запуска редактора. Поэтому вы должны четко понимать что у вас стартует, что загружается. Какие шейдера, какие текстуры полезли в память. Какие объекты. Какие таблицы данных полезли в память это всё лучше всего изучить на старте никакой там особой магией нету - это просто набор правил работы с ассетами. Вы изучаете на старте закладываете в архитектуру и дальше что вы получаете? Моментально стартующий редактор быстро запасающаяся сессия. Всё остальное происходит асинхронно. Оно происходит по запросу и позволяет вам это тестировать. Это касается как игровых проектов, так и VR проектов. И особенно мобильных проектов. Оптимизировать этот кусочек. Более того для мобилок это вообще дико важно. Например, у вас есть таблица, в которой забито 100 персонажей. Не знаю шутер там какой-нибудь делаете. Ну не 100, а 20. И игрок выбирает всегда одного из них. Зачем вам все 20 персонажей в памяти, а если они в таблице, и вы обратились к таблице не зная, что это подргузит за собой все ассеты. У вас будет 20 персонажей в памяти. Опасно. TsoftObjektPTR посмотрите статью. Все ссылки я думаю будут опубликованы или можно найти там ссылку на меня и там все эти статьи есть.

Дальше вот такой интересный пункт – следи за художниками. В первую очередь я обращаюсь к аудитории как к техническим специалистом и либо тем кто хочет как-то сильнее разобраться с технологией. Иногда, казалось бы, невинная штучка, новый материал с выставленной галочкой генерировать автоматически новые шейдеры, загруженные текстуры или ещё что-то положено в какую-нибудь левую папочку. Но за счёт опять же наличие ссылку на неё не прибитую где-то не почищенную из блюпринтов приводит к тому, что у вас внезапно так оп и добавилась там не знаю мегабайт 200 потребление памяти и всё это загружается а вы даже об этом не знаете. Вам это не надо. Надо за этим следить. Надо знать. Надо знать эти галочки как генерятся материалы. Опять же это всё даже несколько про Unreal Engine. Это касается также Unity. В том же Unity можно со ссылками на ассетах всякого наделать что потом сиди и разгребай. Свои минусы и плюсы. Ну и один так скажем так у меня все пункты они немножко разные по глубине. То есть вещи которые из разряда капитанские. Изучить просто как работает, а есть немножко уход в глубь для тех, кто, наверное, в этом чуть больше разбирается. Если вы правильно упакуете все в паки так называемые файлы. Вы получите опять же очень быструю скорость загрузки в том числе уже собранного проекта. Об этом мало кто знает об этом почему-то не пишут везде Epic, но так и есть.

Вопрос что делать будет задам потом. Первый вопрос - кто виноват? Виноваты в этом естественно Epic Games. Можно сказать что не только Epic Games виноваты и всё такое. В этом виноват лично тоже Галёнкин это мы тоже можем сказать. Но суть простая. У нас есть выбор либо мы пишем на тяжелом C++, либо мы пишем на блюпринтах. Вот они две стороны. Темная сторона и светлая сторона. Темная сторона, наверное, кода, а блюпринты светлая. На самом деле нет нифига не наоборот. К чему мы пришли? Прототипирование на блюпринтах. Это быстро. Это круто. Это не требует перезапускать редактор. Все интерфейсы на блюпринтах но реализация конечных фичей, реализация конечного функционала в коде это легче контролировать это легче отлаживать это легче за этим следить и работать в команде. Понятно, когда вы делаете инди проект какая разница какой код. Я за свою жизнь сделал немало прототипов. Я вот так делаю прототипы и мини-проекты. Я не делаю их в коде. Зачем? Это излишне. Я работаю один. Максимум там два человека. Я не буду писать для этого код. Вот это быстрее сделать. Всегда разделяйте. Если вы пишете крупный проект вы должны понимать инди который делает ММО это немножко бред. Инди проект который делает что-то поменьше с минимум функционала это не бред это немножко другое. Средний проект они уже начинают балансировать. Соответственно что мы сейчас ждём? Epic Games купили такую компанию как Шукум Скрипт. Шукум Скрипт закрылся. На reddit очень клёвое обсуждение было. Мы ждем промежуточный слой для игровой логики. Идея какая? Техническую обвязку мы пишем на плюсах. Для этого нужны мощные специалисты и понимание того что вы делаете. То есть геймплейную логику условно геймплей игры чаще всего состоит из управления персонажем, надеванием шапочек, написанием кучи интерфейса. Взаимодействия этого всего. Дата-модель. Персонаж бегает на него надели шапочку у него поменялись параметры. От этих параметров он ускорился. Поменялась анимация и так далее и тому подобное. Для этого не нужен C++ код. Для этого нужны скрипты. Для этого нужно то, что я не знаю вот я просто поменял и оба шапочка стала другой. Я хочу это видеть. Я не хочу перезапускать редактор поэтому мы все этого ждем. То есть ребята которые давно работают на Unreal нет они не защищают ++ как-то быстро язык для игровой логики. Нет нифига. Это не удобно. Но это тоже работает. Поэтому мы ждем скриптовой язык и тогда наступит вселенское счастье.

Перейду ко второй части. Вторая часть это немножко садизма. Это уже то, как мы работаем в команде. И как мы используем несчастных роботов что бы они трудились за нас. Потому что всё что было для нас первая часть это из разряда собери себе компьютер и изучи матчасть немножко. Буквально можно было описать так не раскрывая детали. Вторая часть это уже автоматизация определенных процессов. Это вот скриншот с нашего проекта как реальный то есть мы используем бамбу. С такого расстояние не видно. Наши роботы провели 130000 тестов. Было собрано там 8000 билдов. Это не то, что ты собираешь руками. Это то, что делают роботы. Это то, что они сами тестируют. И прийти к некоторым вещам, в которых роботы помогают вот эти билд машины надо начав это делать. С чего можно начинать? для начала научиться запекать игровой клиент. О чём идёт речь? Вы собираете игру. Вы должны быть уверены, что каждый раз вот ваша ветка основная репозиторий, что ваш проект текущий или что-то он собирается и всё что у вас находится в репозитории не знаю как правильней сказать хорошего качества. То есть у вас там не закомиченны личное. У вас не закомичены странные названия файлов из разряда давайте мы назовем ассеты на русском языке. Или особенно мой любимый случай это когда ассет назван Charekter но с буквой С Русской. Потому что без таких вот автотестах попробуйте это найти. То есть вы ещё должны знать что искать или умляуты какие-нибудь. Мы с этим сталкивались. То есть я с этим сталкивался тоже не раз и да все, по-моему, кто более-менее работал с кучей ассетов с этим сталкивался. Аутсерсеры тебе присылают файлик, а половина букв русские половина английские. Просто, потому что он начал набирать видимо на русском, недостер и так далее. Это делают роботы.

Дальше. Кукинг ассетов. Как раз-таки для этого и нужен. Вы сделали какую-то текстуру не знаю решили использовать на мобилках она разрешение там 323 на 2 пикселя почему-то. На десктопе у вас всё будет работать. Десктоп умеет. Десктоп хороший. Не знаю на мобилки вы такое соберете, а оно у вас будет сразу ругаться. Почему не степень двойки? Если вы используете такие классические подходы. То же самое например сборка. Вы работаете локально на виндоусе. Пишите кроссплатформенную игру. Пусть у вас по расписанию эта игра собирается. Пусть она поверит что она собирается. Вы там нигде ничего не нарушили что все у вас блюпринты не знаю вы отредактировали кусочек кода это поломал опять блюпринтов но вы с ними сталкивались их не открывали. Вы даже не знаете что они не сломаны пока игра не будет собираться. Вы не можете локально у себя на каждый чих начинать собирать игру просто не можете, потому что вы будете работать. Ну не знаю оно компилируется а Вы чем заняты в это время? То есть нельзя. Билды по расписанию. Ночной билд например просто проверяет то что у вас сейчас рабочая версия. Тоже самое вы работаете над мобилами? Пусть роботы собреут и всё заоплодят там в центре, хопиаптом, отпрявят в Гугл плей стор альфу. Или там в тест-лайт загрузят. Куда угодно. Хотите обновить движок чтобы вот это всё собиралась? Тоже вы не должны подходить к билд серверу ручками всё это устанавливать. Запустили план он всё собрал и сделали. А главное, что ещё вам пригодится, когда мы говорим о ревизии игры. Ладно сделать игру это полдела. Вы выпустили игру. Вы запустили на пользователей, у пользователей что-то крашится. По умолчанию краш выглядит так какие-то странные циферки 0x и ничего не понятно. Если вы не будете хранить отладочные символы Вы не узнаете на чём игра закрашилась.

Идём дальше. Что ещё могут делать эти несчастные роботы? Это просто роботизированное рабство. У нас 8 билд серверов работают ежедневно без простоя чтобы было понятно но у нас достаточно крупный проект. На Проектах поменьше столько не требуется но тем не менее роботы постоянно работают. Роботы постоянно проверяет всё это. Вы не можете это удержать в голове. Вы не можете постоянно там не знаю всю неделю пытаться это проверить ручными какими-то тестами. Это автоматизация тестирования. Проверить что там нету на русском языке. Проверить что художники не закомитили ассеты какими-то непонятными именами. Что все патроны соблюдены запущены боты на сервере которые сами пытаются что-то поиграть. Например мы делаем танки. Мы делаем танки что мы делаем? Мы запускаем ботов которые просто ежедневно на каждую сборку играет в эту сборку. Если у ботов не получилось закончить бой, если не было нанесено урона или ещё подобные условия их на самом деле там несколько десятков мы чётко сразу же увидим какая-то проблема. Геймдизайнер случайно сделал пулю которая убивает с первого выстрела. Мы видим бой, в котором 30 убийств или он завершился за полминуты. Мы сразу говорим это неправильно. Мы об этом могли не увидеть до тех пор потому что мы запускаем бета-тесты с различными комбинациями. Это всё гоняется, гоняется и гоняется. Не какой-то кватом инженер, никакой разработчик не будет тестировать все сценарии какие вы можете составить с помощью ботов. Тоже самое клиентские боты. О чём уже мой коллега сегодня на первой лекции говорил. Запускается бот который не знаю проходит миссии. Он потыкивает экранчики и определяется он смог это сделать или нет. Не смог? У нас проблема. И это день за днем это может быть по несколько раз за день либо по команде либо автоматически. Тоже самое валидация модели данных. У вас в игре есть какая то модель данных. Внезапно вы вместо здания прописали туда меч. Вы можете об этом не узнать пока пользователь не знаю там в пятой главе не попытается построить здание которое на самом деле меч если не будет этого шага. Почему это не сделать роботом? Сразу же на месте не надо вручную тестировать. Ну я уж молчу о том, что выкладку релизов и рестарт серверов тоже делать таким образом очень удобно. То есть представьте у вас мультиплеерная и вам надо это развернуть на 10-ти серверах. Вы можете да собрать сами артефакт, заоплоодить, подключиться к каждому серверу настроить поля авторизации. Нет это так не делается. Это точно так же делают билд-сервера.

Это мой любимый скажем так пример. Злодейство. Злодейство с точки зрения менеджмента. Долгое время мы жили без этого и внезапно мы то же к этому пришли. То есть такие менеджерские решения. И код стал гораздо чище. Есть такая штука как кленк формат -автоматическое форматирование кода. Во многих языках не знаю там в Яве с этим попроще. правда в Яве есть, точнее в JavaScript мне очень понравилась можно сказать ну написать код отдать его кленк-формату и он формулирует свод правил которые вы используете. Отлично. Замечательно. Я могу писать очень странный код теперь и это будет мне прощено. Нет. Мы принимаем определенные правила оформления кода. Мы делаем так чтобы роботы опять же роботы проверяли что это ветку можно замешить develop. Всё будет проверено и если это проверка не проходит мы не можем мешать develop. Тоже самое разработчик работает над чем-то он работает свои ветки он не может просто так взять и закометить develop. Работа в команде, проверка, повышение общего качества, обязательно связка с трекером жира чтобы любые комиты были к чему-то привязаны. Не должно быть комитов которые просто так. То есть это просто базисы такого как бы качествм работы в команде. Это уважение к другому разработчику. Потому что если пишешь коды иначе, если ты коды я не знаю там три дня поработал непонятно с чем и хитренько это закидываешь там в develop, а это может поломать коды остальных это очень сложно отследить. Такие подходы позволяют это отслеживать. Позволяют превентивно решать проблемы Legacy, оформление, поиска багов и всего-всего прочего.

И немножко советов уже конкретно касающихся Unreal. Зачем нужны Dll в репозитории? Опять же кто касался с Unreal тот наверное меня поймет. Мы работаем с какой-то версией Unreal. Я вот программист я легко возьму и соберу себе любые Dll. Любой проект который мне дадут. Ну я посмотрю версию, соберу его и радостно запущу. Для меня это не проблема. Но большая часть команды это не программисты. Это геймдизайнеры, художники по звуку, левел-дизайнеры, другие там 3D модельеры. Кто угодно. Concept Artist. Это не программисты. У не должно быть вообще возникать вопросы как мне проект собрать. У них первый вопрос вот дай мне репозиторий. Научи меня этим пользоваться системой контроля версий, и я хочу работать. Я не хочу собирать и знать как это происходит. Мы просто храним develop репозиторий. Для этих вещей есть клевая штука – Git LFS. Работает как надо. Dll живут там. Художник просто берёт выкачивает себе всё и получает сразу проект с которым можно работать. Роботы собирают эти Dll на каждый чих. То есть я закончил новый код. Робот такой ха-ха код пришёл. Я собрал Dll закометил его и все. Художник получил изменения. И пару нюансов как обязательный шаг к сборке. Можно собирать только Dll но лучше еще проверять кукинг. Даже просто для себя. Если кто-то какой-то ассет побил, не правильно заимпортил, что-то косяк – робот сразу скажет warning, надо его написать сразу же в слаг. Мы так же используем слаг. Придет уведомление я его сразу же увижу и подойду к ответственному и спрошу, а зачем ты сделал вот так. Отлично. Я такие вещи люблю.

И DisableUnity. Роботы Unity не любят. Точнее они Unity любят, но проблема в том, что Unity который Unreal Unity он работает на всех платформах по-разному. Если у вас собирается на маке с Unity билдом это не значит, что у вас соберется на винде. Или даже на двух рядом стоящих компьютерах. Это то, как работает система. Поэтому роботы должны собирать DisableUnity. Они проверят за вас такое количество различных связей кода что красота. Просто начинается красота и спасает мир. Ещё буквально несколько моментов. Опять же мне часто задают вопрос на каких версиях движка мы работаем. Напомню, что есть две версии движка одна которая устанавливается через Epic Games Launcher, а другая это которую можно скачать с гитхаба и установить у себя. Дак вот мы работаем с бинарными движком это удобно. Это позволяет получать любые версии просто через лаунчер. Вам не надо опять же объяснять не художнику, не самому себе откуда мне взять кастомную версию движка. И уж тем более собирать на всем парке устройств. Нет есть лаунчер. Мы скачали оттуда и работаем. Все наши проекты с ним совместимы. Dll лежат именно к бинарному движку. Есть, конечно, возможности Install Build использовать. Это когда вы кастомный движок специальным образом собираете, распространяйте. Это менее удобно. Это реально менее удобно только скажем так заморачиваться с кастомизацией движка чтобы это, это нужно только если вы не можете без этого обойтись. Если ваш проект настолько сложен или вы уперлись в какой-то баг, который именно для вашего проекта непреодолимым. Вы его решаете и только тогда вы этим пользуюсь. Лучше сидеть на бинарной сборке это гораздо проще, интереснее и полезнее. Более того надо обновлять проект. Не надо сидеть. Я знаю команды, которые до сих пор работают на 16 версии движка 4.16. Сейчас чтобы вы понимали 4.22 вышла. 4.16 устарела морально и физически. Это не lts это не Unity в котором есть lts или это не Ubuntu. Это разработка игр быть здесь скажем так на острие технологий и оптимизации — это хорошо. Да если у вас проект уже в релизе скорее всего этот срок будет отодвигаться но если мы говорим про проект разработки абсолютно нормальная практика брать и обновляться. Не надо сидеть на старых версиях это потом аукнется. Чем дольше старых версии вы сидите тем меньше шансов что вы вообще когда-либо это сделаете. Переходить с 4.16 на 4.22 да нет давай мы уже это на 4.16 проект как нибудь выпустим и пойдём новый делать. Это несколько советов которые также стоит положить на роботов. Если вы делаете любой проект любую игру Вы должны понимать что она будет работать не идеально. Она будет крашить. Она будет бажить. Она будет падать. Она будет где-то тормозить. Собирайте а – статистику, б – если что-то закрашилось Вы должны об этом узнать. Для мобилок всё удобно. Если папа центр. Есть хоки апп. Есть кашлитекс текстов фейбейзовский и так далее. Вы должны как можно раньше это встроить в свою игру в свой проект. У вас падение тестовое вы должны сразу об этом узнать. Не знаю вебхуки каждый раз вот я сижу пока на конференции вот у нас тестовые роботы гоняют и они мне много уже интересного прислали. Просто потому что вчера прежде чем отправиться на сегодняшнюю конференцию я закомител не очень хорошие вещи. Соответственно эти не очень хорошие вещи сегодня проявляются. Я об этом узнал уже сегодня в самолете. Это замечательно, потому что это позволило мне также написать коллегам. Ребята посмотрите я сделал вот то-то надо это отменить. Если бы вот этого всего не было так кто об этом где узнал? Это очень специфический кейс. Не знаю. Мы выкатили на пользователях, а потом бы поняли, что у большинства у них просто не работает. Ну хорошо. Это было бы печально.

Ещё. Есть Unreal и есть такая прекрасная вещь как Gauntlet. Появилась недавно Если вы хоть как то заморачиваетесь с Unreal с тестированием или рядом. Стоит её изучить. По ней практически нет документации кроме как сайт на китайском, но это сегодняшний день. Как ни странно я не мог не сказать об этом в этой презентации это стоит изучать это стоит использовать это то что используют Эпики сами на фортнайте это инструмент фортнайта. К сожалению, документации нету, но надо читать код. Если вы не чувствуете в себе силы читать этот код всегда есть простой путь пишите велосипеды. То есть вся вот эта идея презентации не обязательно знать даже как это сделано на Unreal. Вы можете сделать неправильно главное сделайте это. То есть научитесь скажем так тестировать с помощью роботов. Научитесь тестировать с помощью си. Си позволяет работать параллельно да то есть пока вы работаете над задачей он позволит проверить другие вещи, ассеты, правила кода, краши не знаю там то что вы не сломали что-то рядом. Потому что легко вы можете я не знаю отредактировать какие-то параметры снаряда сломать танк который вы там не запускали 2 месяца потому что он вам не нравится. Чем больше проект, тем сложнее взаимосвязи.

Более того такие подходы позволяют комфортно работать вам в тех условиях и дэешкой с которыми вы привыкли. Например, я виндузятник. Я работаю на винде давно. Я очень люблю Visual studio. При этом, например тайлер у нее не очень хороший. Я предпочитаю в экскор. Но тем не менее в основном я работаю на винде. Отлично. Я работаю на винде. Я собираю билды под iOS с помощью роботов и отправляю в команду они всё собрали мне отдали я всё протестил. Зачем мне Mac для этих целей. Да я могу облачный сервак поднять в конце концов Mac для того чтобы проверить эти сборки. Это стоит дешевле чем я буду покупать себе эту машинку. То есть это в том числе и экономия.

Ну и очень крутая штука я сам начал ей пользоваться только в этом году к своему стыду. Есть Ubunta на винде называется WSl. Я даже сейчас не расшифрую как эти буковки там называется. В общем есть винда которую можно запустить…Есть Linux который можно запустить внутри винды. Это позволяет вам писать кроссплатформенные скрипты. Если вы написали какой-то скрипт на баше Вы можете его запускать теперь и на Windows легко и на линуксе и на Mac и это всё будет нативное и работать. То есть я этим пользуюсь у меня все серваки так собираются, локальный, я просто счастлив. Но у всех приемов, которые я писал есть, конечно, и своя темная сторона. Во-первых, в какой-то момент сложно остановиться какой уровень валидации тебе нужен. Казалось бы, хочется всё больше и больше и давайте еще линд валидацию какую-нибудь прикрутим и вот это проверим. Это же круто. Это же позволяет нам сократить времени. Нет ничего подобного надо знать где остановиться. Надо уметь этим пользоваться. Это хорошо. Это правильно но знать где остановиться. Для этого всегда требуется человек для обслуживания. То есть в любом случае тот кто настраивает эти системы кто их запускает кто за ними следит. Опять же вы разработчик неважно какой. Вы должны хотя бы на базовом уровне знать об этом это сегодняшний день. если программист не знает как работают там такие вещи бамбур, Continuous или что-то. Ну о'кей. Юниор может не знать он обучится. Я расскажу. А когда вы уже хотя бы мидл и этого не знаете в современном мире это ну это вызывает вопросы. Это корпоративный подход. Минусы такого подхода я знаю разработчиков, которые скажут роботы за меня все проверят. Давайте я закомечу. О тут ошибка. Замечательно. Тоже очень странный подход потому что если разработчик не тестирует то что он пишет это ну не знаю тут мой коллега показывал что с такими разработчиками делают хотя он так не делат. но вообще если программист не тестирует то что он пишет за это надо бить. Просто бить. Не физически, но морально и рассказывать почему так делать нельзя. Если ты не протестировал это сам отдал команде Ну блин команда тратить своё время на то, чтобы искать твои баги — это неправильно. Ну и конечно же можно так сказать что всё отлично. Роботы за тебя работают. Иногда ты ищешь очень локальную проблему, которая заключается в какой-нибудь одной строчке. Тебя надо постоянно пересобирать билды. Вот буквально рассказывал как я вчера весь день просидел и так и не решил эту проблему. У меня где-то Race Condition кроется в одной строчке, и я не знаю как это пока что решить. Я сидел и тестировал за тем, что мне не было с собой Мака мне пришлось просто каждый раз поменял одну строчечку отправил на сервак запустил робота. Я бы гораздо быстрее вот такие тесты сделал у себя локально потому что это изменение буквально одной строчки. Но к слову успешно обошелся и без мака с такой системой. Ещё немножко моментов о чём это я вообще. Связка жиры используется повсеместно в TimCity дженкенс де факты стандарты. Дженкинс бесплатный, TimCity корпоративный стандарт. Gitlab который сейчас прикрутил огромное количество вещей тоже. Хоть что-то из этого надо знать, надо уметь, надо понимать. Это можно сказать ну если вам ничто из этого не знакомо изучите хотя бы хоть что-то. Они все равноправны. Просто что-то в основном корпоративного сегменте что-то в менее корпоративном сегменте. Мы используем связку атласион. Хотя TimCity я люблю гораздо больше. Но у нас бабмбур. Отлично. Я также умею им пользоваться.

В качестве немножко эпилога. Во-первых… Вся система ещё раз повторюсь это сегодняшний день. То есть если там 10 лет назад об этом только слышали и это начинало распространяться то сегодня огромное количество вещей автоматизированы. Вы не работаете как боевая единица в компаниях. Вы не такой условно не корпоративный Самурай. Нет мы работаем в команде. Когда ты работаешь в команде ты скажем так учишься и уважать команду и тестировать какие-то вещи и формулируешь как бы свод правил как к себе, так и команде. Но если не формулируешь ты то за тебя это сформулирует лид. Соответственно для того, чтобы быть сейчас как бы эффективным там программистом, разработчикам и прочим надо учиться пользоваться такими вещами. Я даже не знаю, что ещё добавить, потому что вопрос зачем козе баян. Да все эти вещи они позволяют делать больше итераций. Итерации по контролю качества и итерации проверки кода, итерации сборки. Любые итерации повышают вашу скорость до соответственно скорость говорит о том что вас больше количества опять же этих итераций. Количества в качество ну и блин GameDave вообще для самураев. Это определенные страдания. Если вам даже роботы говорят не знаю там ты здесь неправильно пишешь ну пострадай и сделай роботам приятно. В конце концов. Поэтому вкратце я наверное на этом закончу. Мои контакты указаны здесь если кому-то интересно в том плане посмотреть GitHub в том числе наш корпоративный. Я там забыл на одном слайде сказать огромное количество функционала стоит выносить в плагины вам нужно что-то сделать то, что не имеет привязки к вашему проекту. Сделайте это плагином вы потом себе спасибо Скажите, когда будете апгрейдить версии движка. Если внезапно решили сделать какую-то вещь и не знаю там вот в движке отличное место я сейчас сюда это вставлю. При апгрейде движка возможно это скажем так будет отдельной болью. Не надо ее создавать пишите плагины это отлично. На этом я наверно закончу и готов ответь на вопросы если они возникли.

**Вопрос:** Привет ещё раз коллега мы вот с вами мобильные игры делаем. Зачем с плюсами так страдать Unreal когда Unity есть? В чем профит то?

**Ответ:** Оптимизация.

**Вопрос:** чуть поподробнее. В чем профит проявляется? Вы можете на пятом Айфоне пойти там, где остальные на 6 - 7 могут пойти?

**Ответ:** на самом деле примерно да. Примерно скажем так это выбор движка и технологии в целом а не кода. То есть с точки зрения выстраданных, даже не выстраданных паплайнов мы не страдаем мы спокойно пишем. То есть скорее вся речь про страдания для тех кто только хочет войти в тему. Потому что, если ты занимаешься C++ ну тебе без разницы на чем писать. Я честно предпочитаю Unreal по многим скажем так архитектурным… исходя из архитектурных вопросов и вопросов оптимизации с которыми я умею. Работать.

**Вопрос:** Привет Спасибо за доклад. Вопрос такой бывает ли такое что Вы у себя в репозитории используете зависимости от других репозиториев.

**Ответ:** Вопрос хороший мы долго обсуждали, потому что было понятно. У нас проект и в нём порядка 12 плагинов. Из них половину это наши половина это сторонние. Как это выглядит? На самом деле мы просто храним копию коды. Все остальные зависимости репозиториев на репозиториев приводят к проблемам которые связаны во многом со сборкой, с ключами доступа и подобными вещами. Поэтому все наши зависимости чётко упакованы в проект. Один репозиторий в нём есть всё чтобы этот проект собрать и запустить.

**Вопрос:** ну то есть всякими там модами не балуетесь?

**Ответ:** Да. Мы от них полностью отказались. у нас была попытка это попробовать, и мы поняли, что мы ну то есть сабмодуль это конечно удобно в чём-то, но в другом случае доставляет столько неудобств что зачем?

**Вопрос:** собственно вопрос тот же все равно будет. У нас с тобой Game Jim проходили два раза и всё два раза выиграли ребята на Unity а кто на Unreal делал тормозило или не запустилось Вы реально проверили что на Unreal круче делать, быстрее?

**Ответ:** Да. Забегай к нам как ни будь. Я с радостью буду готов тебе показать. Все продемонстрировать. Это немножко на самом деле комментарий к глючило или тормозило. На Unreal сразу скажу выше порог входа и сразу огромное количество плюшек, которые для десктопных проектов. Если говорить про Unreal для мобил. Первое что ты делаешь ты просто отрезаешь к хренам собачьим лишнее. Чисто из-за того, что Unreal в первую очередь позиционируется как десктопный движок. Если говорить про десктоп там работать гораздо проще. Если говорить про мобилы мы просто отрезали и там нужен опыт. Вот и все.

**Вопрос:** Это опять я. Немножко про Unreal и не совсем про Unreal. То есть такой вариант на сколько вы в свое работе используете те же самые подходы. (далее иностранные слова)

**Ответ:** я так скажу все эти подходы это конечно с точки зрения теории звучит прекрасно по факту мы их используем кусочно в тех местах где это надо. То есть да мы это используем. Да иногда там аналоги ЦС используются. И Ди о ди используются. Только мы это такими вещами не называем. Мы это называем оптимизация. Ну то есть иногда очень странно если у тебя не знаю на мобильной игре стоит на сцене там 100 деревьев и каждое дерево само решает что ему делать. Нет будет внешний менеджер который решить что делать каждому дереву просто потому что это гораздо быстрее. Вот и все. Потом это оказывается ди о ди подход.

**Вопрос:** вот еще интересно услышать ваше мнение насчёт того, как сейчас многие разработчики такие продвинутые отказываются от C++ и полностью переходят на C.

**Ответ:** для меня это звучит странно потому что современный C++ особенно там 11-14 Standard. При знании этих стандартов позволяет делать очень крутые вещи. Опять же это просто требует немножко опыта в этих вещах часто как современного C++ очень многие боятся и в него не лезут. Потому что его презентация идёт как это такой низкоуровневый движок для настоящих таких задротов что там не знаю. Красноглазики Linux и все в шаблонах. Нет ничего подобного с точки зрения написания как бы игровой логики и каких-то функционалов ну это крутой клёвый язык. Мы сейчас создали Data Model построенную полностью на шаблонах но которая вкручена в Unreal Engine и проведена с этой рефлексией и там с точки зрения опять же языка всё красиво и удобно и быстро. Ты такой думаешь ну просто как бы потребовался человек который знает плюсы чуть лучше чем Си. Я вот не понимаю зачем писать сейчас на чистом Си для каких целей?

**Вопрос:** они как бы апеллирует к тому что С++ настолько мультипарадигменный язык что это сложно на самом деле для понимания что можно писать вообще там десятью разными стилями а с другой стороны Си он очень простой, топорный и как раз вот в нем сложнее себе в ногу выстрелить.

**Ответ:** В Си сложнее архитектурно. Ты в ногу себе начинаешь стрелять как только ты выходишь за предмет обсуждения каких-то маленьких примеров. Хорошие примеры это, например написанные на чистом си движки quake, движки doom.

**Вопрос:** а Linux?

**Ответ:** не возьмусь отвечать за Linux я с ним не так сильно знаком. Я всё-таки про GameDev. А игровые движки они решают очень много задач и решать очень много задачи в парадигме Си это такая странная боль. И ради чего всё непонятно.