ВИДЕОИГРЫ
ГЛУБОКОЕ ПОГРУЖЕНИЕ

СЛАВА ГРИС

СДЕЛАЙ ВИДЕОИГРУ ОДИН

И НЕ СВИХНИСЬ!

Слава Грис

Сделай видеоигру один и не свихнись

Серия «Видеоигры: Глубокое погружение»

Текст предоставлен правообладателем
http://www.litres.ru/pages/biblio_book/?art=68745309
Сделай видеоигру один и не свихнись / Слава Грис: АСТ; Москва; 2023
ISBN 978-5-17-151319-1

Аннотация

Всегда хотели делать игры, но не знали с чего начать? Много идей и все хочется реализовать? Страшно браться за что-то новое с нуля? Мысли о программировании вводят в панический ужас?

Тогда эта книга именно для вас. Она поможет понять, как правильно начать работать, где искать ответы на вопросы и самое главное – как не сдаться и довести дело до конца, став настоящим разработчиком видеоигр.

Слава Грис – разработчик-одиночка. На его счету уже три видеоигры, вышедших как на ПК, так и на консолях современного поколения. В своей книге он расскажет, как научиться делать видеоигры одному и начать карьеру разработчика, не имея ни стартовых знаний, ни опыта, ни специального образования!

Эта книга станет вашим трамплином к успеху!
В формате PDF A4 сохранён издательский дизайн.

Содержание

Слава Грис
Сделай видеоигру один и не свихнись

…Он был полон великих идей, но не работал в своей жизни ни дня.
***Divinity: Original Sin*, надпись на надгробном камне**

В оформлении издания использованы работы автора

0. Экран загрузки

Я знал множество людей, создающих видеоигры без поддержки команды. Они посещали сходки разработчиков видеоигр; кое-кто из них появлялся на конференциях, а иногда их можно было встретить на необъятных просторах Сети. Их очень много. Я и сам – разработчик-одиночка.

Я нередко участвовал в пылких спорах о том, возможно ли создать в одиночку нечто столь комплексное, как видеоигра? Она же состоит из графики, музыки, сценария, кода, анимаций, игрового дизайна! Способен ли всего один человек справиться с таким неизмеримым количеством работы?

Многие приходили в своих рассуждениях к положительному ответу: существуют примеры, когда старания соло-разработчиков порождали чудесные и целостные игровые миры. Stardew Valley, Undertale, Cave Story – за каждым из этих проектов стоит всего по одному человеку. Их пример вселяет в людей твердую уверенность, что сделать хорошую игру в одиночку – возможно. И, оглядываясь на эти примеры, люди воодушевленно берутся воплощать свои идеи в реальность, параллельно обучаясь всему и сразу.

Но спустя время знакомые мне разработчики-одиночки начинали пропускать мероприятия, конференции и сходки. Их социальные сети, где они гордо делились своими успехами, покрывались электронной пылью. И вот уже последней

записи на странице их проекта исполняется год, а сами авторы удаляют меня из друзей, потому что я перестал входить в их круг общения. Некогда мечтательных и уверенных в себе разработчиков игр уносили с собой суета будних дней и совсем уже другая работа. Мечта выпустить идеальную игру так и осталась мечтой. Друзья-разработчики больше им не нужны.

Сформулировать программу действий, которая позволит вам освоить столь необычное ремесло, как создание игр в одиночку, не очень-то и сложно: мы займемся чем-то подобным на страницах этой книги. Но, наметив себе путь, начинающие разработчики неистово хватаются за рисование, написание кода, создание музыки, а спустя какое-то время, подобно старой лампочке, мигают пару раз и больше не светят в сторону видеоигр.

Ключевая ошибка всех, кто бросил свой проект, едва начав, заключается в том, что в самом начале пути был поставлен неправильный вопрос и велась работа лишь на одном фронте. Утвердительный ответ на «возможно ли разработать игру в одиночку?» побуждает новичков погружаться в техническую часть создания видеоигр, в то время как моральная и психическая часть остается нетронутой.

Правильный вопрос перед вступлением на красочный путь разработчика звучит так: «возможно ли создать игру в одиночку и не свихнуться?» Чтобы довести проект до ума, нам недостаточно накопить технических знаний и художе-

ственных навыков.

Нам нужно оставаться мотивированными. Нам нужно избегать эмоционального выгорания. Нам нужно запастись терпением, выдержкой и знать все о самодисциплине и силе воли. Работа по внезапному наитию едва ли будет доведена до конца, а если и будет, то этот процесс займет у вас немыслимо длительный период.

Инструмент для разработки видеоигр – это не только компьютер, но и наши нервы, голова, психика, установки и наше окружение. Именно они помогут не пасть духом; именно в них скрыты наши силы; именно там мы найдем способ не сойти с долгого и увлекательного пути.

Пути к игре своей мечты.

1. Раньше было лучше (?)

Когда в компаниях, где меня никто не знает, я говорю, что зарабатываю на жизнь разработкой игр в одиночку, я обязательно сталкиваюсь с убеждением, что моя деятельность почти фантастична, а сам я или лжец, или пришелец с другой планеты. Уверенность в том, что для разработки современной игры нужна огромная команда, слишком уж плотно укоренилась в массовом сознании. Студии по разработке видеоигр нанимают бессчетное количество людей. Титры в играх становятся все длиннее. Команды разработчиков перестают помещаться на одном фотоснимке. В игровой индустрии теперь очень много народа, и это почти никого не удивляет.

А меня – удивляет. И вас, надеюсь, это тоже удивит, когда мы познакомимся с некоторыми аргументами в пользу того, что «один в поле – вполне себе разработчик», и нечего удивляться, когда в компании кто-то говорит, что делает игры один. Я не занимаюсь волшебством.

За время существования такого искусства, как «создание видеоигр», было разработано множество великолепных проектов. И если о культовом статусе некоторых современных произведений множество людей пылко ругаются на форумах, то почетный статус таких вещей, как Mario и Final Fantasy, – неоспорим.

Давайте на короткое время вернемся в конец восьмидеся-

тых, чтобы после такого путешествия свежим взглядом окинуть современную игровую индустрию и подумать о нашем месте в ней.

На заре становления индустрии видеоигр все платформеры или разворачивались на одном экране, или же вынуждали персонажа двигаться вверх. Но тут свершается революция – рождается такая легенда, как **Super Mario BROS**. Миллионы игроков пускаются в оригинальное приключение «слева направо». Революционность Марио на NES (в России продавали клон этой приставки под названием Dendy) заключалась не только в направлении движения, но и во многих оригинальных дизайнерских решениях, которые требуют куда более детального анализа, чем можно уместить на этих страницах. Сейчас же я просто к великому Марио добавлю еще один пример.

Жанр RPG зародился в мире настольных игр. На экраны мониторов он перекочевал в лице Wizardry и Ultima, но это были серьезные игры для взрослых высоколобых ребят. Идеи этих сложных и уникальных проектов, а также опыт коллег, создавших Dragon Quest, легли в основу самой первой части **Final Fantasy**. Именно FF популяризовала жанр JRPG на консолях и доказала, что японские игры могут быть интересны и западной аудитории всех возрастов, а жанр RPG не является нишевым и малопонятным развлечением.

Нужно целиком и полностью абстрагироваться от массовой культуры, чтобы ни разу в жизни не слышать имени Ма-

рио или названия Final Fantasy. Такие титаны не рождались уже очень давно. И можно было бы парировать тем аргументом, что в восьмидесятые создавалось не так много игр и выстоять в конкурентной борьбе с коллегами было проще, а значит, было проще и выделиться, но стоит иметь в виду, что до выхода Марио и Final Fantasy игровую индустрию уже хоронили: на рынке был переизбыток видеоигр. Тиражи нашумевшей E.T. для Atari утилизировали путем буквального захоронения в земле, а в успех приставки NES на западном рынке до конца не верил никто, так как Запад на тот момент уже «устал» от видеоигр после наплыва низкокачественных продуктов на приставки предыдущего поколения.

Позже, перед самым выходом консоли Nintendo64, президент компании Nintendo Хироси Ямаути говорил, что «рынок уже сейчас зашел в тупик. Он завален новыми играми, 70 % которых не находят своих поклонников». Это высказывание было сделано в 1996 году. Игровую индустрию никогда не переставали считать перегруженной новыми проектами и мантру об избыточном количестве игр всю дорогу повторяют в средствах массовой информации и в наши дни.

Подобные беспочвенные утверждения не являются поводом обесценивать заслуги разработчиков прошлого, как и не должны стать причиной опускать руки перед выходом на «перенасыщенный» рынок в наши дни.

Рынок видеоигр абсолютно всегда называли перенасыщенным, но это ничего не значит. Здесь всегда есть место

новым именам. В том числе и вашему.

Условия, в которых выходили на Западе Final Fantasy и Super Mario, были, напротив, гораздо катастрофичнее, чем условия для выпуска игр в наши дни: игровое сообщество не отличалось многочисленностью; количество проданных приставок не составляло десятков миллионов; игру приходилось продавать строго на физических носителях, в то время как сейчас вы можете распространять любое свое творение через Интернет, не заморачиваясь с поставками, логистикой и производством картриджей. В наши дни вы можете даже выпустить игру с ошибками, глюками и багами, а через какое-то время залатать все недоработки «патчем», скачивание которого многие пользователи даже не заметят. В былые же времена обнаружение серьезного бага в игре могло привести к отмене поставки картриджей, а значит, к колоссальным финансовым потерям.

Не только игровой рынок был негостеприимным для творчества: сами инструменты для разработки и жесткие требования к разработчикам создавали воистину спартанские условия для творцов.

Узнаваемый стиль Super Mario создавался с использованием 64 цветов; битвы с ордой монстров в Final Fantasy умещались на экране 256 × 240 пикселей. Ограничения в количестве возможных для отображения цветов вынуждали разработчиков FF использовать всего лишь три цвета на каждого отдельного персонажа, а часть монстров была раскраше-

на с использованием одинаковой палитры – только так NES могла отобразить их на одном поле битвы.

Отдельные двухмерные объекты на экране называются «спрайтами». В понятие «**спрайт**» входили и интерактивные объекты, и персонаж, и враги, и элементы окружения вроде кустов и деревьев. Любая отдельная двухмерная картинка у вас на экране – это спрайт. Спрайт может быть как анимированным, так и статичным. Если вы нарисуете в какой-нибудь программе дерево на прозрачном фоне и вставите это дерево в игру, то дерево смело можно назвать «спрайтом».

Чтобы понять, в каких суровых ограничениях работали Миямото и его команда, можно взглянуть на всю графику в Super Mario *(рис. 1)*.

Рис. 1

Вся графика из игры Super Mario Bros., NES, 1985 год.

Почему же мы не видим здесь отдельного спрайта для Марио, отдельного спрайта для боссов и для окружения? Где облака, где кусты? Что это за каша такая? А дело в том, что на NES крупные спрайты состояли из множества квадратиков 8 × 8 пикселей, каждый из которых имел несколько вариаций, чтобы создать анимацию. Спрайт Марио складывался из двух квадратиков, последовательная смена которых создавала иллюзию движения.

Художники не могли просто нарисовать Марио во весь рост и отправить его в игру в таком виде.

Раскрашивались спрайты программно, потому в игру они вносились в черно-белом варианте. Это позволяло использовать один и тот же спрайт, например, для облаков и кустов достаточно было облако покрасить в белый цвет, а куст – в зеленый.

Модуль обработки изображений NES отрисовывал всего два слоя. На верхнем из них располагались спрайты. Отдельный спрайт не мог быть больше, чем квадратик 8 × 8 пикселей, и для достижения такой цели, как «создать крупный спрайт персонажа», художникам приходилось разбивать изображение на 4, 6 или еще большее количество квадратиков. Анимации для каждого квадратика создавались отдельно.

Не менее сложный подход распространялся и на дизайн уровней. Второй из слоев, отображаемых модулем обработки изображений, был фоном. Фон состоял из 960 плиток разрешением 8 × 8 пикселей, но лишь 256 из них могли быть уникальными, в то время как остальные 704 обязаны были быть их копией. Меньше одной четверти экрана могло быть занято уникальными объектами!

Хоть приставка и могла обрабатывать 256 цветов, фону конкретно в Final Fantasy отдавалось 13 цветов, один из которых и без того был занят базовой заливкой. Оставшиеся 12 цветов разбивались на четыре палитры, но и на этом ограничения не заканчивались: система объединяла каждые 4 плитки 8 × 8 в один блок, и использовать в этом блоке можно было только одну из заготовленных палитр.

Файл со всей графикой в Марио занимал 8 Кб памяти. Чтобы так умело распределить объекты и разбить все на кубики 8 × 8, художник был обязан мыслить так, как мыслит программист. Здесь не шло речи о «свободе творчества» и непринужденных мазках кистью. Для создания подобной графики не хватит одних лишь академических навыков классического рисования. Чтобы сделать картинку выразительной, читаемой, узнаваемой и красивой, требовалось множество умений и знаний из областей, не относящихся к изобразительному искусству.

Массивное количество суровых ограничений превращало работу над визуальным стилем в изнурительную игру в неве-

роятно усложненную версию шахмат. Разработка игр в таких сковывающих условиях в наши дни кажется абсурдной и невыполнимой задачей *(рис. 2)*.

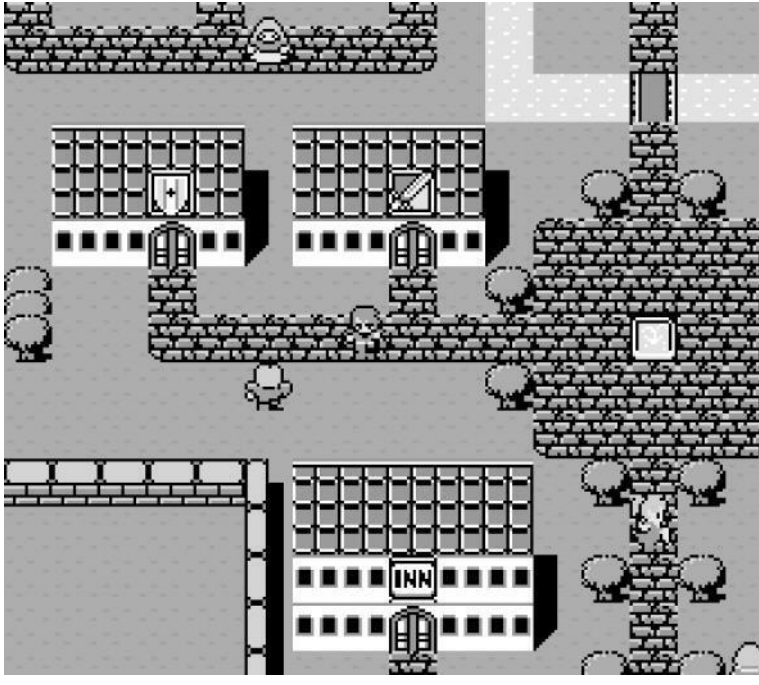

Рис. 2
Final Fantasy, NES, 1997 год

Можно углубиться еще дальше в системные ограничения

NES и ужаснуться тому, что весь код игры приходилось писать на низкоуровневом языке **Assembler**, внешний вид которого способен ввести в состояние ужаса любого современного программиста. Командам Миямото (Super Mario) и Сакагути (Final Fantasy) приходилось писать свой собственный рендер; качество графики в их играх зависело не только от художников, но и от программистов; ребята не могли даже поворачивать спрайты! В NES не было такой функции! Если вы видели в играх крутящийся спрайт, то такой эффект был достигнут рядом изобретательных и оригинальных решений, а не простой современной командой rotate to angle. А про всего лишь пять голосовых потоков, которые были доступны авторам музыки для NES, можно написать отдельную книгу – и она будет достойна внимания, ибо в эпоху NES начало зарождаться такое направление, как chiptune – «восьмибитная» музыка, у которой все еще есть поклонники и авторы.

Команды и Миямото, и Сакагути состояли всего из четырех человек. Четыре человека, находясь во внушительных ограничениях, обладая, казалось бы, неимоверно скудными возможностями, смогли создать нечто столь грандиозное и революционное, как Super Mario и Final Fantasy.

В восьмидесятые не было Youtube с его колоссальным количеством уроков. Не было книг и курсов. Не существовало понятия pixel-artist (тот, кто рисует изображения с помощью «пикселей» – самых крошечных объектов на экране), а ведь рисовать на уровне пикселей – это отнюдь не то же самое,

что заниматься классическим изобразительным искусством. Разработчики не могли перенять опыт своих коллег, потому что со своими революционными идеями смелые и умные ребята выходили на абсолютно не истоптанную тропу, где они оставались со своими уникальными проблемами один на один.

Но что самое интересное – у них получилось создать нечто волшебное, удивительное, положившее начало целым культурам и вырастившее не одно поколение игроков. У них не было и десятой части тех инструментов, что есть сейчас у нас. Наши возможности – гораздо шире, чем были у разработчиков в конце восьмидесятых, и сделать мы можем в разы больше. Свою Final Fantasy у вас сейчас получится создать играючи.

> Уделите внимание ретро-играм эпохи NES, Snes и Sega Mega Drive и сравните их с современными независимыми играми. Не собраны ли, на ваш взгляд, современные игры из точно тех же элементов, из которых создавали хиты прошлых лет?

Давайте поговорим о том, что именно мы имеем на текущий момент.

2. Идеальное время для одиночек: движки

Начнем с хороших новостей: вам больше не нужно изучать Assembler, чтобы сделать игру. Более того, можно вообще не знать никаких языков программирования и не получать образования программиста. Я, например, так и не удосужился этого сделать. По образованию я психолог.

Для разработки видеоигр в наши дни используются «игровые движки», коих развелось великое множество, но упоминания достойны единицы.

Игровой движок представляет собой программное обеспечение, запуск которого откроет вам такие возможности, которые не снились ни авторам Super Mario, ни авторам Final Fantasy. Представляете – в наши дни с помощью абсолютно любого современного движка можно крутить спрайты! Разработчики игр на NES и Sega MD о таком и мечтать не могли! А знаете, в каких современных движках сохранилось ограничение на вывод всего лишь 64 цветов на экран? Ни в каких! Такого ограничения больше не существует! Вы способны использовать все цвета, которые могут отобразить современные мониторы, а также сопроводить все это дело любыми звуками, которые вам приглянутся, – достаточно просто закинуть. wav или. mp3 файл в игру и не заморачиваться

больше о пяти доступных каналах. Вам не нужно собирать спрайты из квадратиков 8 × 8 – вы можете нарисовать спрайт абсолютно любого размера и в любом соотношении сторон.

Перечисление таких возможностей в качестве «удивительных» может звучать саркастично и глумливо, но я отнюдь не ставлю перед собой цель высмеять инструменты прошлого. Я лишь хочу подчеркнуть, насколько доступнее и проще стала разработка видеоигр. Final Fantasy была создана, как уже говорилось, усилиями четырех человек, 3,5 из которых занимались тем, что за вас сейчас сделают движки и программы. Больше не нужно писать рендеры, больше не нужно пытаться уместить всю графику игры в смехотворные 8 Кб памяти. Шансов создать великолепное произведение у нас сейчас гораздо больше, чем было в свое время у Миямото и Сакагучи. Пока эти ребята двигались на ржавых телегах, мы сейчас будем выбирать свой истребитель среди десятка игровых движков.

«Лучшего» движка не существует. Каждый из них хорош по-своему. Потому сначала стоит хотя бы в общих чертах определиться с тем, что собой будет представлять ваш будущий проект. Будете ли вы использовать двухмерную или трехмерную графику? Планируете ли вы делать шутер, платформер или визуальную новеллу? Говорят, что ответов на эти вопросы хватит, чтобы выбрать себе игровой движок. На самом деле их хватит, чтобы лишь присмотреться к движкам, а самый главный вопрос мы зададим себе, когда пой-

мем, какие движки вообще существуют и чем они отличаются друг от друга.

Первый актуальный движок, на который, возможно, упадет ваш взор, – это Unity. С помощью этого инструмента были созданы такие игры, как Genshin Impact, Rust и Cuphead. Он бесплатен для независимых разработчиков и, что важно, подходит для создания как двухмерных, так и трехмерных игр. Качество графики, которое вы получите, полностью зависит от вашего упорства – Unity ничем не уступает другим движкам по возможностям выдавать детализированное изображение. Но достоинства движка играют не такую важную роль, как ответ на вопрос «а может ли в нем разобраться разработчик, не обремененный лишними знаниями о программировании?».

Unity поддерживает C#, и знание C# – огромный плюс, который ускорит обучение этому изумительному инструменту. По сравнению с Assembler, на котором создавались шедевры ушедших эпох, C# является весьма высокоуровневым языком, особенно с учетом того, что сам движок постоянно пытается угадать, что мы хотим написать, и пестрит красочными подчеркиваниями, автоисправлениями и предложениями по «правописанию». Тем не менее в наши дни даже C# кажется суровым инструментом высоколобых программистов, в то время как те, кто не разбирается в C#, могут глянуть в сторону «визуального программирования» *(рис. 3)*.

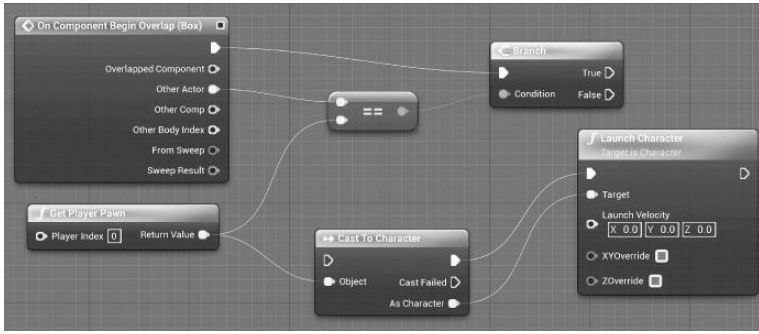

Рис. 3
Система Blue Prints в Unreal Engine

«Визуальное программирование» представляет собой не тот изнурительный процесс, в ходе которого вы пишете сложный код, используя выражения по памяти, а потом ищете, где вы пропустили закрывающую скобку или очередную маленькую запятую. «Визуальное программирование», несмотря на красивое и сложное название, являет собой перетягивание окошек и натягивание стрелочек. Для освоения «визуального программирования» не нужны знания, которые придется получать несколько лет в университете: чтобы совладать с этим «макаронным монстром», достаточно научиться понимать логику движка. Используя свойственный движку «метод мышления», вы будете «объяснять» ему, как должна работать ваша игра. Даже запоминать никаких выражений не придется: при создании «события» вы всегда буде-

те видеть список возможных условий.

Я – не программист. C+, C# и Pyton для меня являются трудноотличимыми друг от друга иероглифами. Тем не менее моим основным заработком является разработка видеоигр, и три выпущенные игры сейчас полностью обеспечивают мое существование. Удалось мне это благодаря «визуальному программированию», которое легко освоить, даже не обладая техническим складом ума.

Мне часто приходится сталкиваться с критикой моего убеждения, что «визуальное программирование» – это легко и просто, а мои слова принижают таланты людей, которые создают игры, используя тот же инструмент, что использую я. Все, что я могу порекомендовать критикам, – это почитать еще раз о том, в каких условиях и какими инструментами создавалась Final Fantasy, и тогда сразу станет понятно, чья работа представляла собой вечный поиск оригинальных решений, а чья – больше напоминает прогулку по парку в теплый летний день.

Не нужно выстраивать вокруг себя барьер из убеждений, что вы занимаетесь сложной и изматывающей работой. Напротив, чем сильнее ваша «работа» будет ассоциироваться у вас с чем-то легким и приятным, тем больше вероятность, что и утомляться вы будете меньше. Если вы научились получать от работы в движке удовольствие – возрастет тяга к тому, чтобы сесть наконец-то за рабочее место и начать делать игры.

К психологическим барьерам мы еще вернемся – других же барьеров перед нами на самом деле не выстроено. Время снова поговорить про Unity, который гордо выставлял грудь вперед на протяжении нескольких абзацев текста, а сейчас делает огромный шаг назад. Хоть инструменты для визуального программирования на Unity и существуют (один из них называется Bolt), они не являют собой основной способ взаимодействия с движком и развиваются скорее «параллельно ему». Помимо Bolt есть множество других надстроек на Unity, с которыми можно ознакомиться здесь по ссылке на *рис. 4*.

https://assetstore.unity.com/tools/visual-scripting
Рис. 4

Каждый из них формирует внутри движка свою собственную среду для создания игр определенного плана: платформеров, новелл, шутеров. Помимо удобного инструментария эти надстройки «награждают» разработчика неповоротливо-

стью и неспособностью использовать функционал Unity на полную мощь, вынуждая творца все-таки прибегнуть к изучению хотя бы каких-то аспектов C# в случае, когда его воображение выйдет за рамки, которые создают подобные надстройки.

Вторым титаном среди игровых движков выступает **Unreal Engine**, блистающий своей технологичностью и всегда словно опережающий свое время. На нем были созданы Fortnite, PUBG и Hellblade. Распространяется этот мощный и серьезный движок бесплатно, но в случае если ваши доходы превысят определенное значение, команда Epic Games попросит вас оплатить Royalty – процент со своего дохода. Этим «определенным значением» сейчас выступает миллион долларов, и я искренне буду рад за тех, кому придется платить Royalty, – ведь это означает, что ваша игра стала чрезвычайно успешна.

Я не побоюсь утверждать, что именно Unreal Engine стоит за популяризацией визуального программирования: его система Blueprints уже интегрирована в движок и доступна для понимания каждому, вне зависимости от навыков и склада ума. Epic Games щедро предоставляет своим пользователям доступ к куче исходников, которые можно использовать как обучающие материалы. Например, если вы собрались делать многопользовательскую игру, то исходник **Lyra** для Unreal Engine 5 сможет стать неплохой основой для воплощения в жизнь такой, казалось бы, трудной задачи.

Кругом витает убеждение, что использование любой технологии может стать ошибкой, если технология эта применена не в той отрасли. Так, например, создание 2D-игры на Unreal Engine для многих не выглядит разумным решением: этот могучий инструмент абсолютно не предназначен для работы с плоскими картинками. Вы же не будете использовать самолет для того, чтобы добраться в «магазин-через-дорогу», верно? Ровно как и использовать велосипед для того, чтобы перебраться на другой континент, вроде бы неразумно, но именно так для многих выглядит попытка создать нечто высокотехнологичное на движках **RPGMaker, Construct, Gamemaker** или **Ren-py**. Все они предназначены для работы только с 2D-графикой, а Ren-py так и вовсе разработан только для создания визуальных новелл.

3. Движки и ваши личностные особенности

Я только что рассказал про два могущественных движка, которые позволяют воплотить в жизнь любые по сложности идеи, но тем не менее на Unity и Unreal Engine создается лишь около половины игр, выпускаемых в Steam. Некоторые крупные AAA-студии продолжают использовать свои собственные закрытые движки, чтобы меньше зависеть от сторонних компаний, и к этим ребятам вопросов у нас не имеется. Но что же движет небольшими студиями и соло-разработчиками, которые выбирают себе в качестве основного инструмента другие, куда менее популярные и продвинутые продукты?

Unreal Engine и Unity, по сути, не сложнее в освоении, чем другие «мелкие» и «неповоротливые» движки. С одной стороны, научиться на них работать даже проще: у Unity есть официальный бесплатный курс для начинающих разработчиков, а уж количество обучающих материалов по Unreal Engine зашкаливает, в то время как найти толковый обучающий курс по какому-нибудь Construct – задача непосильная.

Люди, выбравшие инструментом для воплощения своих идей весьма примитивные программы, на мой взгляд, опровергают популярное убеждение, упомянутое мною выше:

якобы самое важное – чтобы инструмент использовался по назначению.

Но главное в выборе движка – это ваши личностные особенности. От них-то и надо отталкиваться.

Если вам будет некомфортно работать на выбранном движке, то вы не будете на нем работать. При разработке игры в одиночку нам ничто не мешает опустить руки и бросить свои светлые начинания. Нам важно создать условия, в которых вероятность устать от создания игр будет минимальной.

> Ключевым нюансом в выборе движка является то, насколько вам приятно и удобно с ним работать. Уделите по одному вечеру каждому из движков, выполните в них по одному уроку и определитесь с тем, где вам было приятнее работать.

Даже если вы собрались делать 2D-игру, но вас привлекают эстетика «высокотехнологичности», сияющая новизна и чувство причастности к чему-то очень продвинутому – выбирайте Unreal Engine. Если же вам неприятно работать в Unreal Engine и он кажется вам излишне перегруженным, то какой бы проект вы ни делали с помощью этого инструмента – он навсегда останется недоделанным.

Если у вас есть планы найти работу в среде разработчиков, делать мобильные приложения или устроиться в игровую студию, то Unity на данный момент является движком, навык владения которым быстрее всех поможет вам найти работу. Если же вы все время спотыкаетесь о C#, чувствуете,

как написание кода вас тормозит и мешает воплощать идеи в жизнь, а использование 3D для имитации 2D кажется вам глупостью, то у вас не получится раз за разом возвращаться к интерфейсу Unity и создавать свой продукт.

Construct – движок для работы только с 2D-графикой. Именно он – мой личный выбор. Construct полностью основан на визуальном программировании и создает среду, в которой работа с неким подобием кода протекает невероятно быстро. Я человек абсолютно не технического склада ума. Меня не пугает, что я не могу заглянуть «под капот» движка и до конца понять все процессы, которые скрываются за аккуратными «блоками» с условиями. Мне это просто неинтересно. Я могу уделить больше внимания спрайтам и визуальной составляющей моей игры. Именно внешний вид создаваемых мной продуктов для меня является важнейшим аспектом: я одержим контролем над каждым спрайтом, над каждой тенью и каждым эффектом, оттого предпочитаю классическую анимацию и полное отсутствие программных спецэффектов и постобработки в своих проектах. Такие игры, как **Hollow Knight, Cuphead** и **Hotline Miami**, не были созданы на Construct2, но могли бы – движок позволяет создать все то, что мы видели в этих играх *(рис. 5)*.

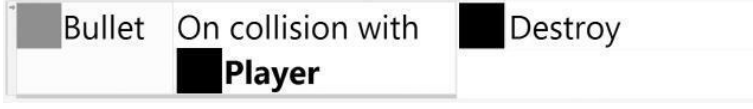

Рис. 5

Логика поведений в Constuct. Слева – условие (пуля коснулась игрока), справа – событие (игрок уничтожен)

Упомянутая Hotline Miami была создана с помощью **Gamemaker**. Точно так же как и Construct, этот движок запирает разработчика в двух плоскостях, лишая его возможности создавать трехмерные игры. В Gamemaker используется собственный язык GML, который имеет и визуальное ответвление. GML отличается от языка Construct своей глубиной и сложностью, что делает Gamemaker инструментом, подходящим для людей с техническим складом ума. Если вас воодушевляет и дает силы работа именно с кодом, но при этом профильного образования у вас пока еще нет, то в создание игр на Gamemaker вполне можно втянуться.

Мне бы очень хотелось составить табличку в духе: если вам от движка нужно вот «это» и «это» – выбирайте «такой-то движок», но, во-первых, такие таблицы уже лежат на просторах Сети, а во-вторых (если вы правильно уловили мою мысль), существование такой таблицы не очень поможет вам выбрать инструмент.

Если при выборе движка отталкиваться только от технических требований к своей будущей игре, то велика вероятность, что вы натолкнетесь на программу, с которой не сможете «подружиться» и в среде, в которой не будете чувство-

вать себя комфортно. Если бы сотни моих вечеров, проведенных за работой в Construct, являли собой борьбу с логикой движка, его интерфейсом, а заодно с моим дискомфортом и ярым нежеланием возвращаться к работе, то я бы не довел до ума ни один из своих проектов. Именно соответствие Construct моим личностным особенностям помогло мне создать на нем три игры и провести в этой программе больше 6000 часов.

Работа с игровым движком напоминает общение между двумя людьми из разных стран. Вы очень плохо знаете язык приезжего, а если и заучите каждое слово, то вам все равно потребуются дополнительные годы, чтобы говорить на уровне носителя языка и понимать все культурные особенности своего товарища.

Каждый вечер вы будете пытаться объяснить этому иностранцу на ломаном подобии его наречия: «Я хочу, чтобы ты сделал “вот такую штуку”» (например, чтобы персонаж атаковал по нажатию на кнопку «X»). Чтобы в понятной форме донести до движка ваши требования, вы должны быть терпеливы и внимательны. Сложно оставаться терпеливым и внимательным к продукту, который вызывает у вас только раздражение и ненависть.

Один мой знакомый был программистом веб-сайтов и неплохо знал такой язык программирования, как Java Script. Желание делать игры привело его к Unity – благодаря многочисленному сообществу и отличной маркетинговой кам-

пании пути многих начинающих разработчиков упираются именно в Unity. В ходе разработки своего проекта с использованием всех предоставляемых Unity благ он начал спотыкаться о реализацию сложных и глубоких задач, вроде создания AI (искусственного интеллекта) или проработки поведения NPC. Знания Java Script едва ли помогали ему в этом деле. Скорее, напротив, они мешали освоить новую логику в новой среде.

Вместо удобного, раскрученного, многофункционального движка мой знакомый в итоге обратился к более низкоуровневому программированию и начал разработку проекта на «сухом» Java Script с использованием react-компонентов (готовой библиотекой некоторых шаблонов кода для реализации конкретных задач) и массово используемым готовым решением для корректно работающей физики под названием phaser.js. Работа над игрой стала для него куда менее утомительной, среда – куда более понятной и знакомой лично ему. Но набор его инструментов выглядит как что-то неповоротливое и сложное! Так почему ему не понравилось работать на Unity?

Его личный опыт, его склад ума и его знания поспособствовали тому, что работа на чем-то более, казалось бы, сложном и неудобном, протекала быстрее и приносила больше удовольствия.

Надеюсь, никто не забыл предисловия к этой книге, и не ждет, что я четко скажу вам в одном предложении, какой

движок вам выбрать? Я не могу этого сделать – проблема выбора движка лежит на вас, я уже перечислил, на что нужно опираться в своем решении. Но я могу дать еще две подсказки.

Во-первых, никто не запрещает вам попробовать каждый из популярных движков. Зайдите на сайт одного из них, посмотрите на игры, которые созданы с его помощью, взгляните на пару вводных уроков и прислушайтесь не к своему разуму, а к более глубоким ощущениям – цепляет ли вас хоть чем-нибудь эта программа? А продукты, созданные на ней? У некоторых игр на Unity есть некое неуловимое сходство, отличающее их от игр на Unreal. А какое ощущение оставляет беглый просмотр форума сообщества? Вы почувствуете, как что-то внутри в определенный момент щелкнет и даст ответ на вопрос: «А для меня ли эта программа?»

Если прислушаться к себе окажется чрезвычайно трудной задачей, то скачивайте все подряд движки и пытайтесь в каждом из них выполнить хотя бы один урок. Так вы точно поймете, выполнение какого урока принесло вам удовольствие и на каком движке стоит остановиться. Без удовольствия от работы вы ни за что не заставите себя провести месяцы, а может быть, и годы свободного времени за созданием видеоигр.

Второй совет – общайтесь с разработчиками. Никто лучше человека, работающего на Game Maker, не ответит вам на вопрос «а легко ли на нем будет сделать “вот-такую-то-

игру”»

Лучше, конечно, общаться в реальной жизни – так вы уловите больше их личных свойств и характерных манер.

На мой взгляд, существуют некоторые неуловимые качества, объединяющие тех, кто выбрал себе в «коллеги» тот или иной движок. Эти незримые сходства и трудноуловимые качества и должны стать еще одним вашим помощником в выборе движка – кто из разработчиков окажется ближе к вам по своим манерам, взглядам и убеждениям?

4. Мы не одни

Поиск живого общения с будущими коллегами по цеху должен увенчаться прекрасным открытием: во множестве городов России регулярно проводятся как формальные, так и неформальные сходки разработчиков. Если их не бывает в вашем городе, то помните, что поезда все еще ходят и, потряхивая ваши косточки в купе или плацкарте, доставят вас до места встречи разработчиков.

Наши сходки условно можно разделить на три формата.

Первый из них является самым легким и непринужденным и подразумевает встречу в специально арендованном зале какого-нибудь бара, где можно пересаживаться из-за одного столика за другой и коротать время в беседах с людьми из игровой индустрии.

Второй формат – чуть более деловой, и помимо необузданного и хаотичного общения он подразумевает наличие в программе доклада от одного из участников сходки. Во время выступления местный разработчик будет рассказывать про свой опыт, отвечать на вопросы и делиться полезной информацией.

Если вы живете в Петербурге или планируете его посетить, то поиск мероприятий должен привести вас к «**Индикатору**» – площадке для разработчиков видеоигр, где лекции сочетаются с необузданным общением. Чтобы только

послушать доклады, стоит обратить внимание на сообщество игровых разработчиков **Braindie**, а провести время в совсем уж легкой обстановке можно с **GamedevHouse** – сообществом, которое организует сходки еще и в Москве. В поисках площадок, где читают лекции для разработчиков игр в столице, можно присмотреться к мероприятиям от **ВШБИ**, а гостям и жителям других городов – прибегнуть к помощи Интернета и социальных сетей, потому что я, разумеется, не могу перечислить все мероприятия во всех городах.

Третий формат, самый, казалось бы, серьезный из всех, – это конференции. Одно слово «конференция» рисует перед глазами серьезную встречу угрюмых дядечек в пиджаках, которые говорят только о будущих сделках, а за могучими плечами каждого из них стоит как минимум одна AAA-игра.

Я же предпочитаю описывать конференции как «бизнес-праздники», в которых «бизнес» вполне можно отодвинуть на задний план. Чаще всего конференция проходит в красочно обставленном зале, где и крупные компании, и маленькие независимые разработчики вроде нас демонстрируют свои проекты, участвуют в конкурсах, ищут коллег, а главное – общаются, делятся опытом и весело проводят время.

Из крупных российских конференций необходимо упомянуть **White Nights**, организаторы которой регулярно проводят еще и серии лекций как раз для тех, кто пришел на конференцию не как журналист, издатель или разработчик,

а ищет новых знаний для того, чтобы приобщиться к индустрии.

Причины посещать эти мероприятия кроются далеко не в контактах, связях и знакомствах, без которых, честно говоря, разработчик-одиночка теоретически может обойтись. Как и все самое важное, причины приехать в другой город на сходку или конференцию кроются у нас в голове.

Конец ознакомительного фрагмента.