Роман Сакутин

Разработка игр для начинающих

ОГЛАВЛЕНИЕ:

ОБ АВТОРЕ

Здравствуйте дорогие друзья! Меня зовут Роман Сакутин. Я разработчик игр с опытом работы в три года. Успел помотаться по разным студиям и поучаствовать в разработке не одного проекта. В этой книге я хотел бы описать базовые момент разработки игр, рассказать о Unity3D и программирование. Конечно же я не смог бы уделить достаточное время для описания всех моментов, к сожалению, я должен работать что бы кормить себя, да и очень много я рассказывал в своих видеокурсах. Я часто буду делать отсылки к ним, некоторые платные, а некоторые нет. Ссылки на них всех будут приведены в конце.

Моя профессиональная карьера достаточно обильна и многогранна. И в офисе штаны просиживал и на дому работал. Делал и игры, и сайты. В 13 лет я начал программировать на языке программирования C++ - это было весьма трудно, учитывая, что интернета у меня не было и учился я по книге скачанной у друга. Просить о помощи тоже было некого, но я как-то смог преодолеть эти трудности, надеюсь и вам, читатели, удастся перебороть всё что вам мешает и осуществить свои затеи!

Вы в любой момент можете написать мне на почту – roman@sakutin.ru

ВВЕДЕНИЕ

Вы всегда хотели создать свою игру? В ваших фантазиях игра может быть разной: цветастой, как крайзис но с лошадьми и т.д и т.п. Предупрежу сразу, все ваши мысли будет почти невозможно реализовать! Почему? На этот вопрос вы ответите сами, когда узнаете о геймдизайне и углубитесь в эту сферу деятельности. Но не буду вас расстраивать и рушить планы, ведь не для этого написана книга.

Сейчас я хочу ввести читателя в базовые понятия геймдева, и дать некое направление для дальнейшего изучения.

Вас будет часто преследовать чувство будто вы не справляетесь – наберитесь силы воли и вгрызайтесь с ещё большим усердием! Разработка игр – это смесь технологий и искусства. Не стоит с головой уходить в творчество и забывать о технических аспектах, также не стоит задумываться только о коде и смотреть на игру как на способ добычи денег. Вероятней всего с таким подходом вы заработаете не много.

Эту небольшую книгу мне помогли написать ребята с нашего сообщества http://IndieCamp.ru/dev/

КТО СОЗДАЁТ ИГРЫ?

Люди создающие игры делятся на множества типов и иногда один выполняет специальность другого. Описывать каждую роль глупо и не разумно, так-как зачастую в «боевых» условиях, многие люди комбинируют в себе множества специалистов. Я проведу обобщение и выделю три основные категории:

- **Контент мейкеры** — Художники, 3D моделлеры, звукорежиссёры, композиторы. В общем люди которые делают контент для игры.
- **Геймдизайнеры и ко. -** Ребята занимающиеся проектированием такой части игры, которая очень сложно осязаемая но составляет основу любой игры. Разрабатывают концепцию взаимодействия пользователя с миром, делают этот самый мир на бумаге, составляют уровни из результатов работы контент мейкеров. Сценаристы тоже входят в эту категорию.
- **Технари** — Специалисты которые делают так чтобы то что сделали предыдущие дарования работало. Занимаются разработкой с технической стороны, программируют и создают инструменты для работы геймдизайнеров например. В современных реалиях, при выборе уже готового движка, программисты\скриптеры иногда пишут к нему полезные утилиты которые бы облегчили труд левел дизайнеров.

ЧТО ТАКОЕ ДВИЖКИ?

Движок — это по сути программы, или целый программный комплекс благодаря которому процесс создание игры сильно упрощается. В нём программируют, в нём делают уровни из контента зачастую созданного в других программах и т.д. Для каждого специалиста понятие движка будет разное, но в общем оно такое.

Для того что бы лучше понять, что такое движок**,** лучше изучить какой-нибудь и попробовать сделать на нём что-нибудь простое. У многих тех, кто только начал разрабатывать игры, встаёт резонный вопрос. А какой движок всё-таки выбрать? На данный момент самые актуальные и заслуживающие внимания движки, следующие:

- **Unity3D**
- **Unreal Engine [3-4]**
- **CryEngine**
- **NeoAxis**

Я настоятельно рекомендую изучать Unity3D, для начинающих подойдёт мой курс.

ГЕЙМДИЗАЙН

Создание игр – это многоступенчатый, довольно длительный, и оттого не менее интересный процесс. Мы строим свою реальность, свой мир, в котором мы являемся «богами». Но быть «богом» не так-то просто, именно по этому в этой статье мы рассмотрим, через что проходит идея одного или нескольких человек, прежде чем стать полноценным продуктом который не стыдно показывать.

Миссия «Придумать игру»

«Что же такого сложного в том чтобы придумать игру?» — спросите вы, абсолютно ничего. Придумать игру под силу каждому человеку, гораздо сложнее продумать игру. Вы с лёгкостью вытащите из своей головы задумку шутера про динозавров из будущего, которые подружились с разумными роботами и решили истребить всех карасей на планете Земля. Но позже, вы поймете, что абсолютно не понимаете, как в это можно будет играть. И поэтому, профессионалы, которые придумывают игры (геймдизайнеры) не просто придумывают какую-то идею, из которой можно сделать игру. Их работу можно сравнить с тем, как художник рисует картину.

Художник рисует фон (геймдизайнер придумывает завязку игры, основной геймплей). Художник рисует природу (геймдизайнер обдумывает место, где будет происходить основное действие игры, будь то какая-то выдуманная реальность или же какое-то реально существующее не земле место). Художник наносит на полотно людей (геймдизайнер придумывает персонажей, главного героя игры, кто он или что он, и какими характеристиками он обладает).

Стоит отметить то, что геймдизайнер должен быть игроком. И именно потому, что, человек, который не играет в другие игры, и не следит за тем, что сейчас происходит в сфере игровой индустрии, просто не в состоянии придумать что-то интересное. И хотя бы потому, что он не будет знать, что уже было придумано до него, и любая гениальная идея может просто превратиться в копию такой же фичи, но в другой игре. Но зачастую геймдизайнеры делают одну большую ошибку, в том, что они хотят скопировать какую-либо игру на свой лад. Это целиком и полностью убивает проект, поскольку, кому например нужен ещё один Battlefield от другого разработчика, но со своими идеями. Геймдизайнер может считать, что та или иная игра не получилась, но он мог бы сделать ещё лучше. Этого не следует делать, хотя бы, потому что все ошибки проекта лежат на плечах основных разработчиков, и их задача в своих последующих проектах избежать их. И не факт, что у геймдизайнера получится обойти эти ошибки, и в данном случае проект будет просто обречен. И, конечно же, если геймдизайнер не игрок, он просто не сможет узнать, что

же сейчас нравиться игроку, и что имеет смысл выпускать, а что следует отложить до лучших времён. Конечно, контроль рынка с целью выпуска продукта направленного на большую аудиторию, это больше процесс заработка, ведь игры, которые понравятся большим людям, разойдутся большим тиражом и принесут больше прибыли (стоит отметить что для того чтобы сделать продукт который понадобится аудитории, стоит иметь аналитические способности, без них у геймдизайнера не получится целостно оценить рынок), но порой геймдизайнерам не стоит бояться, и можно рискнуть. Ведь что-то новое в жанре вполне в состоянии изменить индустрию, и принести разработчикам не только деньги, но и известность.

Миссия «Продумать и описать игру»

Следующим этапом в работе геймдизайнера, как правило, является перенос своей идеи на бумагу, для того, чтобы в понятной, письменной форме донести до остальной команды весь смысл игры и описание каждого её элемента. По сути, дизайн-документ – это библия всей команды, поскольку именно ему они должны следовать, чтобы создать именно то, что планировалось, а не то что сначала родилось в голове у одного из программистов, а затем было переформулировано моделлером.

Команда может помочь геймдизайнеру советом со стороны для того, чтобы он оценив его и проанализировав, ввёл в игру или нет. Но в основном геймдизайнеру очень сильно помогает сценарист. Это именно тот человек, без которого геймдизайнер не сможет правильно написать дизайн-документ. Хотя бы потому, что одна даже самая маленькая идея, может кардинально не вписаться в сценарий игры. Именно поэтому сценарист придумывает игру именно с геймдизайнером. Иногда геймдизайнер сам придумывает весь сюжет, но в таком случае, сценарист всё равно остаётся при работе. В его обязанностях остаётся написание диалогов, цельной истории и, конечно же, исправление каких-то не состыковок. Ведь каждый человек занимается именно своей работой, и если геймдизайнер самостоятельно начнёт писать и дизайн-документ и сценарий, то уйдёт достаточно много времени, и можно будет с уверенностью сказать, что у сценариста, основной работой которого остаётся именно сюжет игры, получится написать его гораздо лучше.

Стоит также сказать то, что геймдизайнер, как человек, который именно придумывает, продумывает, и описывает игру, должен обладать таким качествами как грамотность и умение излагать свои мысли устно, и в письменном виде.

Всей команде разработчиков будет гораздо проще понять основную задумку игры, её геймплей и то, что от них требуется, если геймдизайнер в доступной и понятной форме сможет это им изложить и в общении и в дизайн-документе. Ведь если тот написан правильно, то можно с уверенностью сказать, что процент качества проекта возрастает. Команда знает, что от них требуется и делает это, а геймдизайнер в свою очередь помогает им советом.

Миссия «Благополучно дождаться релиза»

Каждому геймдизайнеру во время основной разработки игры, зачастую приходят идеи о том, что если изменить ту фичу на эту, то получится в сто раз лучше, и он вновь садится за дизайн-документ и изменяет в нём что-либо. Это очень большая ошибка. Если пытаться дорабатывать дизайн-документ с каждым днём сильнее и сильнее, то может получиться очень плохая ситуация. Зачастую, одна небольшая идея может изменить весь проект, и программист, который так долго писал код для того, чтобы этот персонаж мог сделать вот так, а потом, узнав, что это уже не нужно, и пусть лучше он делает вот так, потеряет в пустую своё время и силы. Что очень сильно сказывается как на дате выхода проекта, так и на команде, которой просто надоест по сто раз переделывать что-либо. Именно поэтому, талантлив тот геймдизайнер, который смог продумать всё ещё изначально, чтобы после не добавлять безумное количество идей в дизайн-документ.

Конечно, это не значит, что уже во время разработки игры нельзя прикасаться к дизайн-документу, это даже нужно делать. Вы могли пропустить какую-то маловажную деталь, которая может убить тот или иной элемент проекта. Вы могли ошибиться в описаниях каких-либо элементов, и вы могли понять, что именно вот эта фича может сильно повлиять на игру, причём в худшую сторону. Поэтому геймдизайнер должен перечитать свой дизайн-документ ещё многое количество раз, чтобы окончательно убедиться в том, что он закончен, в том, что все идеи описаны корректно, и можно смело браться за разработку.

Советы для геймдизайнера

Геймдизайнеру стоит учесть одну вещь. Он работает в команде, а это значит, что помимо его, игру разрабатывают другие люди, обязанности которых отличаются, и для них, главным остаётся их работа. И поэтому, задание тем же моделлерам, должно быть дано в правильной форме. Допустим, вы решили, что в игре должен фигурировать военный джип, и вы даёте задание моделлеру: «Сделай джип». Он кивает. И на следующий день перед вами модель обычного внедорожника. Но результат был бы другой, если бы задание звучало так: «Необходимо сделать модель военного джипа (а затем дополнительные пункты задания, если такие имеются)». В качестве примера, вы можете показать моделлеру фотографию джипа, найденную в интернете, которая оставит у него в голове визуальный образ той модели, которая требуется.

Это относится не только к моделлерам, но и к программистам, людям для которых задание должно быть чётко поставленным и включать в себя всю необходимую информацию. Мало сказать: «Сделай мне управление машиной». Лучше сказать, так: «Я хочу, чтобы ты сделал возможность управления вот этой машиной. Необходимо, чтобы имелась возможность войти в неё в качестве водителя, возможность переключения передач, и, конечно же, возможность её уничтожения как при расстреле из огнестрельного оружия, так и при падении с высоты или столкновении с объектами окружения». Задание сразу же становится более ясным. А это значит, что какие-то промахи, которые могли быть, будут полностью исключены, и вы получите именно ту игру, которой вы её задумывали.

Также, геймдизайнеру стоит помнить, то, что в случае с его работой, краткость – не сестра таланта, а наоборот убийца. Поскольку детальное описание любого элемента игры, будет гораздо лучше пары строчек, которые лишь дают понятие о том, что это такое. Так, геймдизайнер может полностью описать каких-либо вражеских NPC и дать им все необходимые характеристики: количество жизней, скорость передвижения, их тип, расу, и т.д. Но если он опишет их так: «Этот NPC является противником главного героя, он может убить его из автомата», то это будет плохо. Конечно, это не означает, что геймдизайнер должен писать целые романы, на несколько сотен страниц с описание пары персонажей. Это лишь означает, что многие элементы игры необходимо описывать целостно.

Заключение

В заключении хотелось бы сказать, что работа геймдизайнера – это не просто процесс от идеи игры до её описания. Это очень творческая работа, требующая терпеливого и вдумчивого подхода. И если вы решили стать геймдизайнером, то стоит очень много практиковаться, ведь, как известно опыт приходит со временем. И точно также, к геймдизайнеру со временем приходит формула его «идеальной» игры.

ДИЗАЙН УРОВНЕЙ

Представим на миг игру, действие которой происходит, а абсолютной пустоте. Игрок может идти куда угодно, но он не встретит на своём пути ничего интересного. А теперь ответьте на следующий вопрос: «Стали бы вы играть в игру, без игрового мира?».

Большинство игроков ответит, нет, и это будет правильно, поскольку, мы ценим игры не только за геймплей, графику и сюжет, но и за локации в которых происходит всё действие.

Дизайн уровней – это неисключимый этап в разработке любой игры и один из самых тяжёлых, поскольку для того чтобы придумать оригинальную локацию, потрудиться придётся довольно сильно. Конечно, любой игрового мир может быть совершенно разным в отличие от других игр, оно и понятно, сеттинг другой. Но всё же, за все те годы, что существует игровая индустрия, у игроков сложилось чёткое понимание об разных типах игровых миров. Об них вы можете узнать ниже.

Игровые миры:

1. **Открытый мир** – этот мир более всего привлекает свободой перемещение. Слово «открытый» подразумевает под собой ничто иное как «иду куда хочу». Применить данный тип мира удастся не в каждой игре, но оно и понятно, зачем аркадной гонке открытая трасса. Зачастую данный тип применяется при создании игр жанра RPG, а осуществляется это, потому что данный жанр подрузамевет собой как раз ту свободу. Играя в тот же Skyrim, мы строим судьбу своего персонажа сами, как бы являясь его реальным прототипом. И именно это уже подразумевает под собой определённую свободу действий, данную игроку. А как эта свобода будет выглядеть в коридорном уровне с одной, двумя развилками? Именно поэтому, игры, которые хотят дать почувствовать игроку своё нахождение не только у экрана, но и внутри его, дают нам открытый мир, который может дать нам чувство исследователя или же искателя приключений.
2. **Коридорный уровень** – данный тип игровых локаций знаком всем игрокам в шутеры и многие другие жанры. Мы не можем зайти в ту или иную дверь, порой мы не можем даже вернуться назад, если забыли там взять патронов. Но всё же, порой мы этого не замечаем и просто рвёмся вперёд к намеченной цели. Во многом, ощущение мира в котором происходит игра зависит не только от того как он выстроен, но и от тех ощущений которые у нас вызывает эта игра. Мы можем бессмысленно бродить по пустым коридорам в поисках какого-либо персонажа, если

перед этим получили довольно положительный заряд эмоций, например от какой-нибудь сюжетной вставки, которая как раз и разбавила пресное нахождение внутри уровня. Но всё же, даже закрытый тип локации можно выстроить таким образом, что игрок будет увлечённо проходить его. Обилие деталей, качественно поставленное освещение и внимание к мелочам смогут дать просто потрясающий эффект, достигнуть которого можно лишь хорошо потрудившись.

3. **Генерируемый мир** – интерес к этому миру уже показал Minecraft. Играя в эту игру, мы можем пойти куда угодно, и наткнуться на что угодно. Бескрайние пещеры, леса, пляжи, луга, озёра и реки, всё это лишь часть того что подарил нам Minecraft своим уникальным генератором миров. Здесь мы могли почувствовать ту свободу, которую нам не дарила прежде ни одна игра. И более того, мы могли сами творить свой мир. Именно пример Minecraft можно приводить тем играм, которые хотят сделать свой генерируемый мир. Даже при относительно небольшом количестве текстур и все различных объектов, мы получали очень красивую картинку при кубизме игры, и могли часами исследовать пещеры и леса в поисках новых приключений. Но всё же, если задуматься, то можно конечно кое что понять. Генерируемый мир – это не такая простая штука. Играя в тот же LEGO в детстве, мы строили свой мир, но достигалось это примитивной формой деталей. Ведь даже из кубов можно собрать шар. А что тут говорить о лесах и горах. Чем легче конструктор, тем красивее вещи из него можно построить. Что же произойдет, если вместо кубов взять high-poly модели? Здесь придётся задуматься над тем, как та или иная модель будет смотреться по соседству с другой. Допустим мы сгенерировали мир из кучи различных моделей разных типов и форм. Конечная картинка может нас не порадовать. Вон там швы не сошлись, а тут вообще невесть что получилось, это дерево перекрывает соседнее и так далее. Так что всё-таки если вы планируете делать мир с самогенерируемым миром, вам придётся провести очень много работы над моделями и генератором, для того чтобы добиться безупречной картинки. Но всё же, лучше генератор миров будет смотреться в игре, сделанной из кубов, или же имеющей 2D графику.

Начало работы

Что же, с типами игровых миров мы разобрались, теперь же самое время перейти к тому, что называется началом работы. Каждый хороший дизайнер уровней не будет начинать делать локацию для игры, не продумав её перед этим.

Залог успеха в создании идеального уровня лежит не в хороших моделях и текстурах (Хотя это тоже неотъемлемый элемент) но и в грамотной постройке игрового окружения. И первым что стоит сделать, для того чтобы добиться этого, конечно же является чертёж или набросок уровня.

Профессиональные дизайнеры уровней, работающие в крупных игровых студиях, всегда имеют в своём распоряжение профессиональное чертёжное

оборудование, или хотя бы чертёжный графический редактор. Ведь создать гениально построенный уровень из головы просто не удастся, мысли постоянно путаются, и полноценная картинка, которая до этого казалось безупречной, просто меркнет. Именно поэтому перед тем как готовить уровень из моделей и текстур, прежде всего, стоит сделать его чертёж или хотя бы набросок.

Вы должны знать, где будет находиться, какой объект, для этого всегда можно делать пометки в виде каких-либо цифр или значков, ведь если вы не сориентируетесь в своём чертеже, то, что же у вас получится, если вы поставите в том месте, где планировался диван, к примеру, стиральную машину? Правильно построенный чертёж — это уже залог успеха.

Стоит отметить то, что если вы собрались строить профессиональный чертёж своего будущего уровня, то без системы координат вам просто не обойтись. И вот тут в силу вступает геометрия. Вас придётся строить все внутри игровые объекты именно тех размеров, каких они должны быть. Впрочем, геометрия итак является неотъемлемым элементов в создании уровня. Ведь если у вас главный герой просто не пролезет в дверь, это будет провал.

После постройки чертежа, как правило, наступает его редактура. Вот ту допустим вазу с цветами забыли указать, а здесь дверь. Не ленитесь редактировать свой чертёж, без него вы просто никуда. Перепроверьте его лишний раз, исправьте недочёты, и если надо, то перечертите.

Построение уровня

Следующим, не менее важным этапом является постройка уровня. Как правило, данное действие осуществляется в специальном редакторе игрового движка. Но порой, уровни приходится делать в 3D редакторе, с последующим экспортом в игровой движок.

В создании уровня уже используются готовые трёхмерные объекты окружения, и зачастую, дизайнерам уровней самим приходиться заниматься созданием некоторых моделей которые были не учтены. Но всё же, в профессиональных студиях, как правило, всеми моделями занимается 3D моделлер.

Помимо грамотной расстановки всех моделей по чертежу, важную роль играет и такой дополнительный атмосферный элемент как освещение. Он добавит вашей игре красок и придаст дополнительной атмосферы. Впрочем, настройка освещения, как может показаться на первый взгляд, является довольно трудоёмким процессом. Тип освещения, цвет и тени. Всё это должно быть настроено правильно, для того чтобы придать картине должный шарм.

Но другое дело, если в игре используется динамическое освещение, например, смена дня и ночи. Как правило, в таком случае приходиться заботиться не только о свете, но и об дополнительных его источниках. Например, чтобы вон та лампа, смогла осветить комнату ночью, или чтобы луна не светила так же

ярко как солнце. Освещение это довольно сложный и многогранный этап настройки игрового окружения.

Также, порой на плечи дизайнера уровней ложится и такой ответственный этап как запекание текстур (Объединение всех текстур в одну). Данный процесс, как правило, лежит на 3D моделлера, но может случиться так, что дизайнеру уровней самостоятельно придётся провести запекание теней и текстур. Как правило, запекание текстур очень сильно помогает при работе. Моделлеру не приходится работать с множество материалов, и он может очень просто настроить объекты.

Дополнительные советы

1. Сцена должна быть живой, если вы хотите того чтобы игрок поверил в то что это место действительно похоже на реальный мир, мало просто сделать модели домов и наложить на них текстуры. Стоит также позаботиться и об дополнительных факторах, которые придают картинке жизни – освещение, стены, мелкие детали, дополнительные элементы.

2. Вы можете добавить на локацию сколько угодно мелких деталей для того чтобы придать ей реализма, но конечная картинка может вас не порадовать. Как правило, большое количество деталей влияет на производительность. Но всё же стоит помнить и то, что мелкие детали в виде разбросанного мусора, пыли, бумаги и прочего, можно изобразить на текстуре. Этим вы не только улучшите производительность, но и не, сколько не проиграете в картинке.

3. Дайте игроку почувствовать хоть немного свободы перемещений. Даже если у вас открытый мир на несколько километров, то даже возможность войти в какое-нибудь простое здание даст игроку дополнительные ощущения. Если же у вас коридорный шутер, то не стоит пренебрегать свободой, дайте её в виде возможности обойти противника с фланга пробежав каким-нибудь скрытым путём, или же открывайте побольше дверей, чтобы игрок всегда мог спрятаться в каком-нибудь надёжном укрытии и восстановить здоровье.

4. Представьте, как игрок сможет взаимодействовать с тем или иным элементом окружения. Допустим, он сможет спрятаться за этим ящиком, и отстреливать оттуда врагов, но они при этом ему ничего не смогут сделать. Тогда стоит сделать разрушаемый ящик. Чтобы игрок чувствовал себя на месте главного героя, и после разрушения укрытия искал другое. Взаимодействие с игровым окружением может придать игре дополнительно шарма. Та же возможность включить или выключить где-то лампочку даст игроку почувствовать какие-то элементы реализма при взаимодействии с окружением.

5. Делайте окружение интерактивным. Игрок всегда может подойти к любому объекту и попробовать с ним взаимодействовать. Будь то лампа или дверь, дайте игроку почувствовать, что в этом мире он может хоть что-то кроме выполнения заданий. Ведь если он войдёт в уже открытую дверь это не придаст ему ощущений, а если он предварительно откроет её, и всё это будет

сопровождаться реалистичной анимацией, то это придаст игре интерактивности в плане взаимодействия с игроком. Не бойтесь экспериментировать, делайте объекты динамичней. Если это укрытие, то оно может сломаться, а если это механизм, то его можно включить.

6. Если вы создаёте игру с открытым миром, то важно придать игроку не только чувство свободы, но и чувство реализма. Если действие происходит в лесу, добавьте в него пение птиц и животных. Если это городок, то добавьте туда жителей, живущих своей жизнью вне зависимости от игрока. Открытый мир подразумевает под собой, некую копию реального, и этого можно добиться множеством путей, от расширенного взаимодействия с пространство до мелочей вроде правильно расставленных звуков.

7. Представьте себя на месте игрока, и подумайте, как бы вы прошли данный уровень. Куда бы вы пошли, где бы вы что-то сделали и как бы провели взаимодействие с тем или иным объектом. Ощутить себя игроком это очень важный элемент, который должен знать каждый дизайнер уровней.

Заключение

В заключение главы хотелось бы сказать то, что дизайн уровней – это довольно сложный и многогранный процесс, требующий должного влияния, и, конечно же, довольно творческий процесс, который может принести удовольствие игрокам от прохождение красиво построенной локации.

ПРОГРАММИРОВАНИЕ

Программирование – одна из самых сложных дисциплин в разработке игр. Для начала вам стоит выбрать язык программирования. Если вы последуете моему совету и будете изучать Unity3D, то вам нужно будет изучить язык программирования C#.

Учиться этому ремеслу нужно в обязательном порядке! Без хотя бы базовых знаний программирования – вам не обойтись. Подберите язык под движок, и по больше практикуйтесь. Напишите для начала хотя бы калькулятор.

Если вы всё-таки остановили свой выбор на C# то советую следующий учебный материал:

1. Для начала - C# 4.0. Полное руководство.
2. И для оттачивания мастерства - C#. Программирование для профессионалов

Также для начала подойдёт книга “C# для вундеркиндов”.

Не забываю и себя пропиарить. Дело в том, что я регулярно собираю группы для занятий C#. Купить записи этих занятий можно здесь - http://indiecamp.ru/dev/wppage/обучение-c-курс/

Сейчас я хочу ввести вас в основы программирования.

ЧТО ТАКОЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ И ИХ ОТЛИЧИЯ

Язык программирования – это по сути какая-то совокупность команд, понимаемая компьютером. Точнее компьютер не понимаем язык программирования как таковой. Он лишь промежуточная часть, эдакий компромисс, при котором удобно и программисту программировать и компьютеру понимать, что этот программист там накалякал.

Исходный код, написанный на каком-нибудь языке программирования компилируется в понимаемые компьютером команды, которые уже впоследствии и исполняются. Эти команды очень сложно программировать в ручную.

Языки делятся на множество типов, подробный разбор всего этого дела я приводил в 50 минутном занятие из своего курса по C#. Сейчас лишь расскажу о двух основных:

- Компилируемые – это языки которые компилируются в программу, которую запускают у себя пользователи (exe файл в системах Windows, например). Пример: C++, C#, Си, Java, Pascal, Delphi.
- Скриптовые (интерпретируемые) – языки которые не компилируются, они исполняются в особенной программе называемой интерпретатором. Пользователь работает с исходными текстами которые самостоятельно запускает в интерпретаторе. Программы на таких языках писать проще, но есть проблема с производительностью и то, что исходные тексты доступны всем. Последнее иногда оборачивается плюсом так, например, в среде системных администраторов распространены так называемые скрипты, исходные коды которых они могут посмотреть и убедиться в том, что они не нанесут вред компьютеру. Или банально изменить их под себя. Пример: Python, JS, PHP.

Помимо этих двух типов, применяется так называемые промежуточный тип, принцип которого изменяется от реализации к реализации. Дело в том, что, что бы наша программа одновременно работала на куче платформ (как

интерпретируемые языки) и при этом сохраняла все достоинства компилируемых. Её компилируют в промежуточный так называемый байт-код, которые в последствие исполняется в специальном интерпретаторе (виртуальная машина). Наглядным примером является C# и Java. В C# используется CLR (Common Language Runtime) которая исполняет промежуточный байт-код CIL. Из-за этого получается, что все .Net языки (C#, VB.Net, F# и т.д) компилируются в один и тот же код который в последствие исполняется в CLR.

В результате этого добавляется гибкость разработки и не привязанность к системе как таковой, но к сожалению, Microsoft свою реализацию CLR сделала только под Windows и прочие свои платформы обойдя стороной другие ОС. Из этого нашли выход, сообщество языка сделало Open Source (открытый исходный код) реализацию это исполняемой машины которая называется Mono, о которой я говорил чуть выше.

В Unity3D, например, также используется Mono в качестве исполняемой среды, в результате чего мы можем использовать скрипты, написанные на C# не только в играх под ОС Windows.

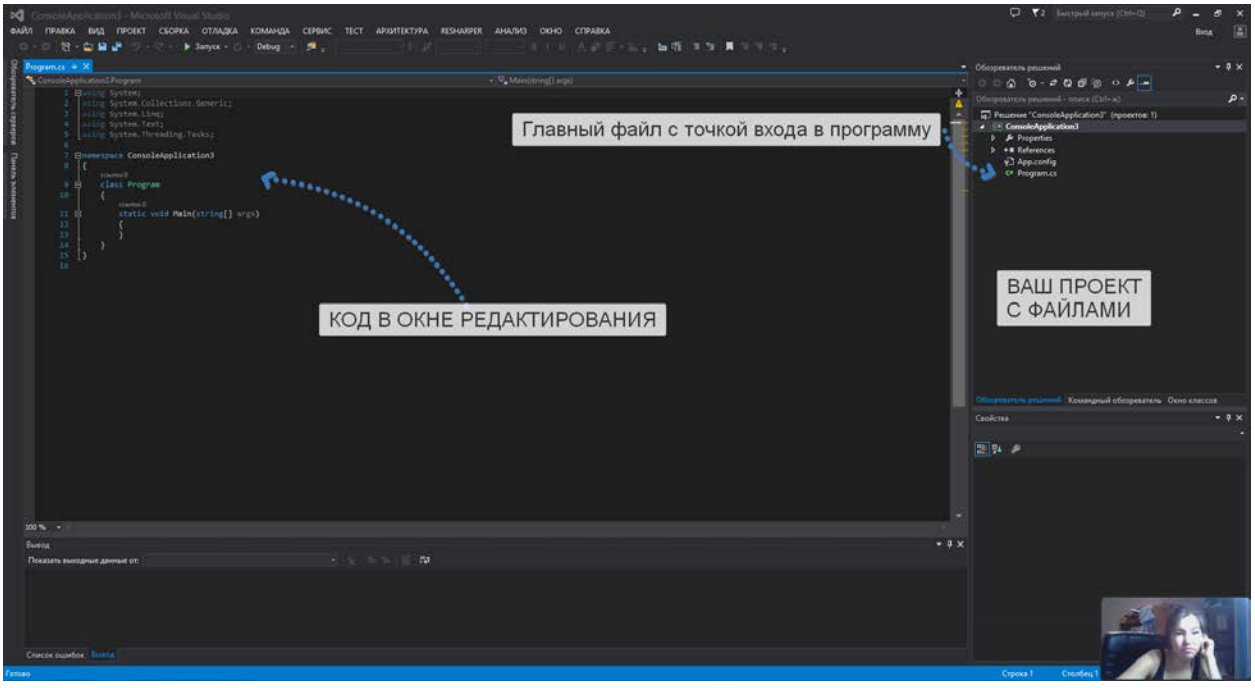
ВВЕДЕНИЕ В C#

C# - ~~это нота~~ это ОО (Объектно Ориентированый) язык со статической типизацией. Он является компилируемым, и работает только в системах семейства Windows. Но из этого есть исключение – Mono, разработка с открытым исходным кодом которая позволяет программам, написанным на C# работать и на других платформах. В том числе и на Android, Linux, Mac OS и т.д.

Разработка на этом языке в основном ведётся в Visual Studio. В VS есть сразу всё, и компилятор и текстовый редактор кода, и прочие приятные вещи которые упрощают разработку. Есть несколько версий VS в том числе и бесплатная. Её можно скачать с официального сайта Microsoft и называется она Visual Studio Express.

На C# можно разрабатывать не только классические программы, но и веб-сайты, с помощью Фреймворка ASP.Net. Также используя Unity3D можно писать игры, к слову C# ещё используется в движке NeoAxis.

Скачав и установив VS вы должны создать новый проект. У вас на выбор будет много проектов, но для начала хватит Win32 Console Application (Консольное Приложение). После создания этого проекта у вас будет файл Program



В главном файле вы моете найти каркас программы на C#. Давайте разберём её структуру.

- **Using** – это пространства имён. По сути это библиотеки которые позволяют использовать некоторые дополнительные возможности которые могут быть не всегда востребованы. Допустим в пространстве имён System.Colliction.Generic находятся классы для реализации динамических массивов.
- **Namespace** – здесь мы указываем что код заключённый в этот блок кода (фигурные скобки) являются частью пространство имён, имя которого мы указываем сразу после этого ключевого слова.
- **Class** – объявление класса. Так, как C# ОО язык, то всё состоит из классов из которых в последствие конструируются объекты. Класс также является типом данных, который мы определяем сами.
- **static void Main(string[] args)** – здесь мы объявляем метод, он является базовым и из него начинается исполнятся программа. Main – это точка входа. Void означает что метод не возвращает значение а то, что в круглых скобках – это аргументы которые следует передать при вызове этого метода. В данном случае система сама позаботится об аргументах.

Это базовый разбор, замечу что здесь я намеренно не упрощал повествование. Для совсем новичков есть мой курс где мы прям пошагово разбирали каждую возможность от самых простых к самым сложным. В книге, без контакта с читателем это сделать достаточно сложно, надеюсь вы меня поймёте.

Как вы заметили весь код группируется в блоки с помощью фигурных скобок ({, }) . Весь код делится на эту блоки в которых находятся инструкции, которые должны заканчиваться точкой с запятой (**;**). Если мы добавим в блок кода метода Main следующую инструкцию, после запуска программы у нас выводиться сообщение на экран.

Здесь вы также можете заметить инструкцию **Console.ReadLine();** которая нужна что бы программа не закрылась после вывода сообщения. Это происходит из-за того, что если у нас закончатся инструкции в Main то программа прекратит своё исполнение. Добавив **Console.ReadLine();** мы указываем что надо подождать что бы пользователь нажал на клавишу Enter.

Поздравляю, вы сделали свою первую программу! Нажмите на зелёную стрелочку в верхней части VS и сможете насладиться результатом своей работы.

C# Является регистрозависмым языком – это означает что регистр букв имеет значение. Так переменная AppleCount и appleCount совершенно разные переменные. Также C# не поймёт, что делать с оператором IF который он не знает, а вот if знает.

ПЕРЕМЕННЫЕ

У императивных языков программирования есть такое понятие как переменные. Любая программа хранит какие-то данные и как-то ими оперирует. В компьютере для работы с памятью программы предназначена оперативная память, но программисту работать с ней напрямую достаточно проблематично. Для облегчения жизни программиста введено такое понятие как переменная.

Переменная – это по сути ящик, в котором мы можем хранить какое-либо значение определённого типа. Когда мы можем хранить значение только одного типа, и должны об этом говорить заранее (в момент объявления переменной), в таком случае язык называется статически типизированным.

Типичными типами являются:

- **Integer (int)** – целое число (-254, -2, 0, 1, 5, 543).
- **Float, Double** – дробное число (0.15, -5.42, 243.524).
- **String** – строка (“Hello World”, “User24”).
- **Char** – символ, строка состоит из символов (‘f’, ‘r’, ‘2’).

Подробней о переменных и конкретных их реализациях читайте в руководстве по выбранному вами языку (или смотрите в моём курсе по C# ☺).

Вот как объявляются базовые переменные в C#.

Тип Имя = НачальноеЗначение.

Начальное значение можно иногда не указывать, тогда переменная будет неинициализированная. Присвоение значения переменной при её объявления называется инициализация.

Вот пример объявления переменной типа **int** которая хранит количество яблок.

```
int  AppleCount = 25;
```

Дальше с этой переменной можно производит арифметические действие. Например

```
AppleCount = AppleCount + 40;
```

После этих действие значение переменной будет 65. Вот как это выглядит в коде.

```
ссылок 0
static void Main(string[] args)
{
    int AppleCount = 25;
    AppleCount = AppleCount + 40;

    Console.WriteLine(AppleCount);

    Console.ReadLine();
}
```

Здесь вы также видите, что мы выводим значение переменной через вызов метода **WriteLine**. Отдельного слова заслуживают литералы, это просто значения определённого базового типа. Так, например, если в коде встретится число 25, то оно будет обычным литералом. С помощью литералов мы можем задавать начальные значения для переменных или использовать их в арифметических выражениях, как это показано выше.

У нас в распоряжение есть литералы строк, они помечаются двойными кавычками. Пример: **“Hello World”**.

Особенно хочу отметить имена переменных. Они не могут начинается с чисел и некоторых спец символов. Но они могут содержать числа в название.

Пример разрешённых имён: **Time, Apple2, _view**.

Пример запрещённых имён: **2Apple, %Time**.

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

В программирование мы очень часто работаем с числами, а работать с ними без арифметических операций было бы весьма затруднительно. C# как и любой другой язык предоставляет нам эту возможность. Вот список доступных нам операций: **+**, **-**, *****, **** (разделить), **%** (разделить по модулю), **++** (декремент, увеличение числа на 1) и **--**(инкремент, уменьшение числа на 1).

Если с первыми всё более-менее ясно, то вот некоторые требуют более детального разбора.

- **%** - Деление по модулю, оно же деление с остатком. Результатом такого выражения будет остаток. Пример: **13 % 5 = 3**.
- **++** и **--** - Декремент и инкремент. Это обычное увеличение или уменьшение числа на 1. Классически мы бы записали так **N = N + 1**. Но с применением данного оператора мы можем поступить более кратко **N++**.

На языке C# это выглядит следующим образом, допустим у нас есть переменная типа **int** с названием **AppleCount**, тогда мы можем сделать следующее:

```
AppleCount = 25 * 5;
```

В данном случае мы присвоили переменной **AppleCount** значение выражения **25 * 5**.

Арифметические операции можно применять не только с литералами, но и с другими переменными.

```
AppleCount = AppleCount + NewAppleCount;
```

В этом примере мы сложили количество яблок, которые у нас имеются с количеством новых яблок и присвоили значение выражения в нашу переменную.

БУЛЕВЫ ТИПЫ ДАННЫХ

Помимо строк и чисел, у нас есть булевы типы данных, для их обозначения используется ключевое слово **bool**. Спектр значений, принимаемых таким типов, ограничивается всего двумя, это true и false.

Исходя из этого можно совершать множество различных действий. Но прежде чем мы перейдём к конкретным примерам их применения, я хотел бы рассказать об операторах работы с таким типом так, как здесь мы не ограничиваемся только арифметическими операциями.

Первые операторы называются **Операторы Отношения,** которые оперируют различными типами, но результат их работы тип **bool**.

Вот их список:

- **==** - Проверяет два значения на равенство. **(10 == 10) = true**, **(5 == 10) = false**.
- **!=** - Проверяет два числа на не равенство.
- **<, > -** Сравнивает два числа, больше и меньше.
- **>=, <= -** Помимо сравнение на больше и меньше, также учитывает равенство чисел. То есть, результатом **true** при операторе >= будет если числа равны или число слева больше.

Помимо этого, в C# предусмотрена **Логические Операторы**. Вот их список, подробно на них останавливаться не будем и рассмотрим их применение в следующей главе.

- **&** - И
- **|** - Или
- **^** - Исключающее ИЛИ
- **&&** - Укороченное И
- **||** - Укороченное ИЛИ
- **! –** Не

УСЛОВНЫЕ ОПЕРАТОРЫ: IF, ELSE

Условные операторы нужны для того что бы задавать условия в вашем коде и тем самым ветвить его – выполняя некоторые действия только по мере надобности. Условные операторы тесно связаны с булевыми типами данных, и работают на прямую с ними.

Рассмотрим оператор if, его синтаксис следующий

```
if(булевоевыражение){
 код которые выполнится только если выражение равно true

}else{
код которые выполнится только если выражение равно false

}
```

Под булевым выражением подразумевается любое выражение результат, которого **true** или **false**. Для оперирования числами в таком выражение нам следует использовать операторы отношения, а для оперирования несколькими логическими выражением логические операторы.

Рассмотрим следующий код что бы вам было понятней

```
int userAge = 15;

if(userAge >= 18)
{
        Console.WriteLine("Ваш возраст подходит");
}
```

Здесь мы задали условие, если возраст пользователя 18 или больше, то тогда мы выведем сообщение. Здесь мы можем также применить логические операторы, например оператор И.

```
int userAge = 15;

if(userAge >= 18 && userAge <= 50)
{
```

```
        Console.WriteLine(“Ваш возраст подходит”);
}
```

Теперь мы проверяем, если возраст пользователя больше 18 и меньше 50. Помимо, И мы также можем воспользоваться ИЛИ **||**. Хочу отдельно остановится на операторе НЕ !, который инвертирует значение. Так, если справой от него стороны будет true, то результатом выражения будет false и на оборот.

```
if(!(10 == 10)){
Console.WriteLine(“Это сообщение не выведится”);
}
```

СОЗДАНИЕ КОНТЕНТА

И так вам понадобился контент, что делать? Правильно, сделать его! В основном вам понадобится контент двух типов:

- 3D модели
- 2D спрайты

Если у вас будет 2D игра, то вероятней всего вам не понадобятся 3D модели, но если у вас будет 3D игра, то вам понадобятся 2D спрайты для пользовательских интерфейсов.

Для создания 3D моделей вам понадобится знание пакетов для 3D моделирования. Самый подходящий на данный момент: Blender. Также есть очень распространённый 3D Max. На YouTube много полных курсов по каждой из этих программ!

С 2D немного посложней, нужно уметь рисовать! Ещё желательно владеть Photoshop.

В интернете полно и бесплатных текстур, и бесплатных моделей. Не надейтесь, что накидав контента из интернета у вас всё будет выглядеть шикарно. Когда вы наберётесь опыта делать это всё самостоятельно, вы узнаете о общей стилистике, цветовой гамме и прочем. Нельзя соблюдать базовые принципы из скаченных ресурсов.

Также не нужно пытаться быть мастером на все руки, если вы научитесь хорошо программировать, то с лёгкостью сможете найти в команду 3D моделлера.

UNITY3D

Unity — это мультиплатформенный инструмент для разработки 2-х и 3-х мерных приложений и игр, работающий под операционными системами Windows и OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Android, Apple iOS, а также на игровых приставках Wii, PlayStation 3 и XBox 360.

Есть возможность создавать интернет-приложения с помощью специального подключаемого модуля к браузеру Unity, а также с помощью экспериментальной реализации в рамках модуля Adobe Flash Player. Приложения, созданные с помощью Unity, поддерживают DirectX и OpenGL. Так же хотелось бы добавить, что Unity имеет очень гибкую документацию и на нем можно сделать абсолютно любую игру, любого жанра. Поддерживает 3 языка программирования: JS, C#, Boo.

Вполне удобный инструмент, сам работаю с ним уже третий год и всё это время он улучшался. Я и вам предлагаю работать именно с этим движком.

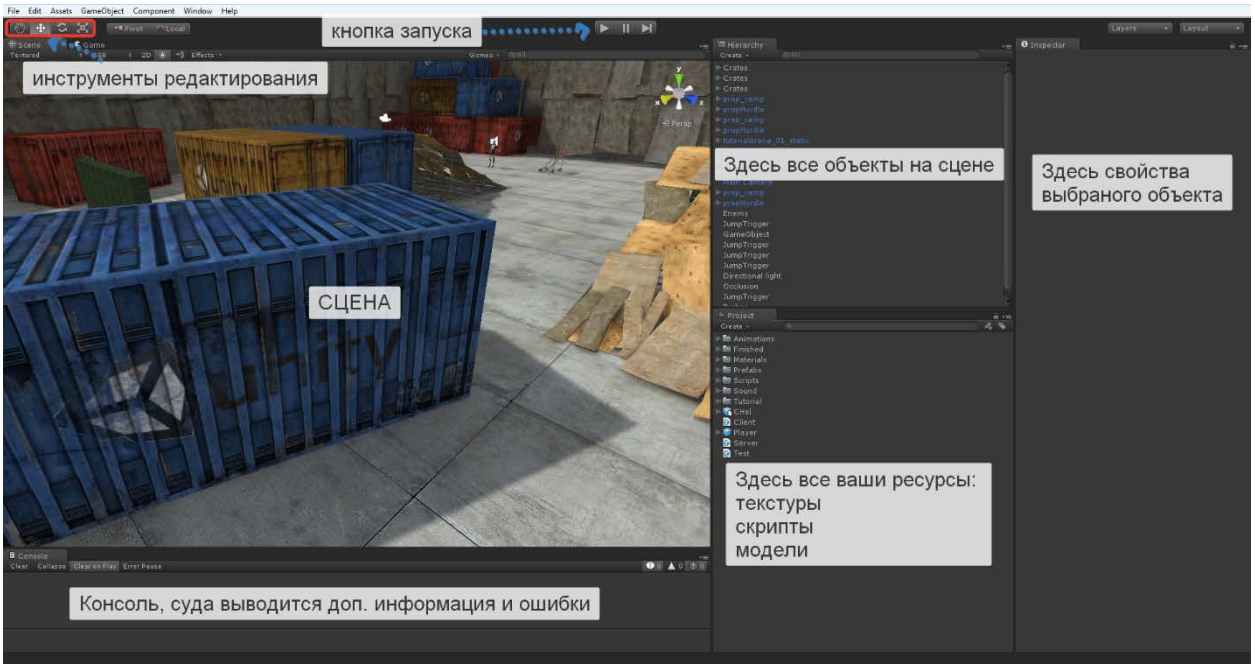
УСТАНОВКА UNITY3D И БАЗОВЫЙ РАЗБОР ИНТЕРФЕЙСА

Сейчас мы установим Unity3D и разберём её интерфейс. И так, по шагам.

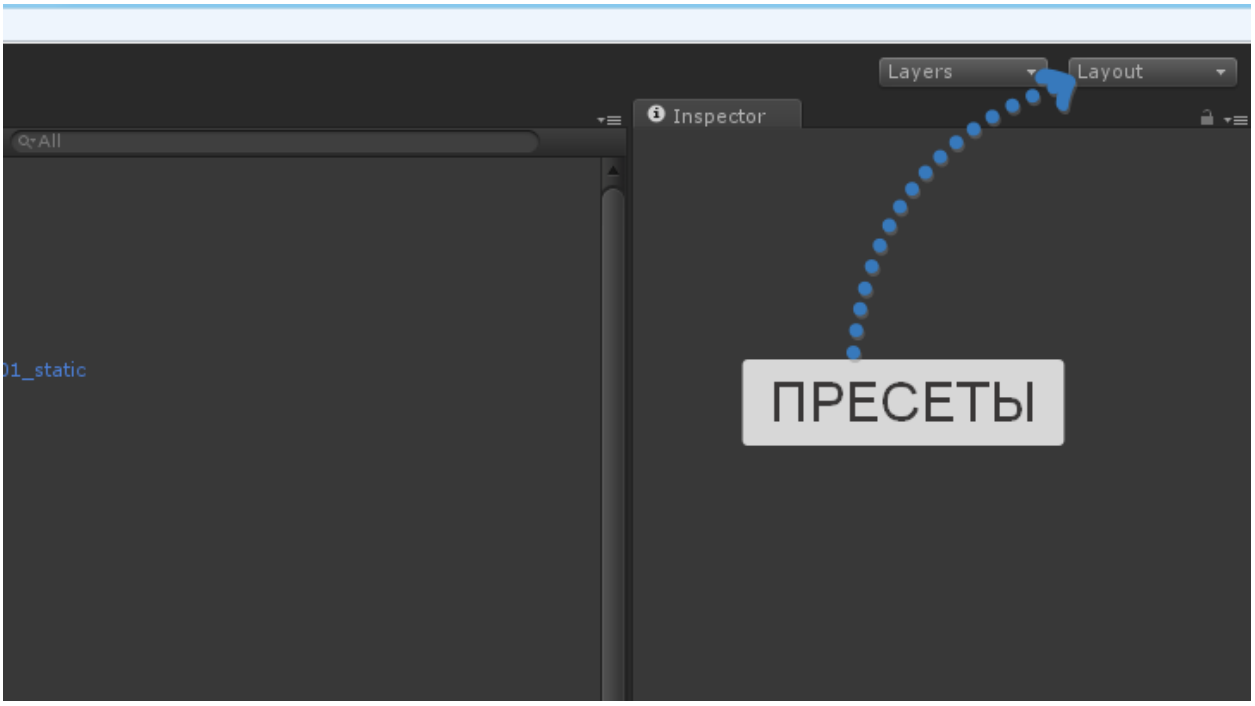
1. Идём на официальный сайт http://unity3d.com/unity/download и качаем оттуда последнюю версию.
2. После загрузки запустите файл.
3. Пройдите типичную процедуру установки.
4. Далее запустите исполняемый файл Unity.
5. При первом запуске вам предложат выбрать формат лицензии. Вы можете активировать полностью бесплатную free версию, или же активировать trial pro версии, он позволяет использовать все возможности 30 дней без покупки лицензии.
6. После всего этого вам предложат создать новый проект. Сверху вбиваете путь и его имя, а чуть ниже указываете стандартные ассеты с которыми хотите работать. Подробней об ассетах вы можете узнать в моём бесплатном видеокурсе http://indiecamp.ru/dev/unity3d-для-начинающих

Вы восхитительны!

После запуска перед вами будет примерно следующее



У меня такое расположение окон, каждое окно можно изменять по своему усмотрению. Также предусмотрены пресеты, то есть вы можете быстро переключится в какой-нибудь набор окон.



Давайте попробуем что-то сделать? Я создам новый проект

File –> New Project

При создании нового проекта, поставлю галочку на **Character Controller**. Это позволит использовать стандартные скрипты для управления персонажа.

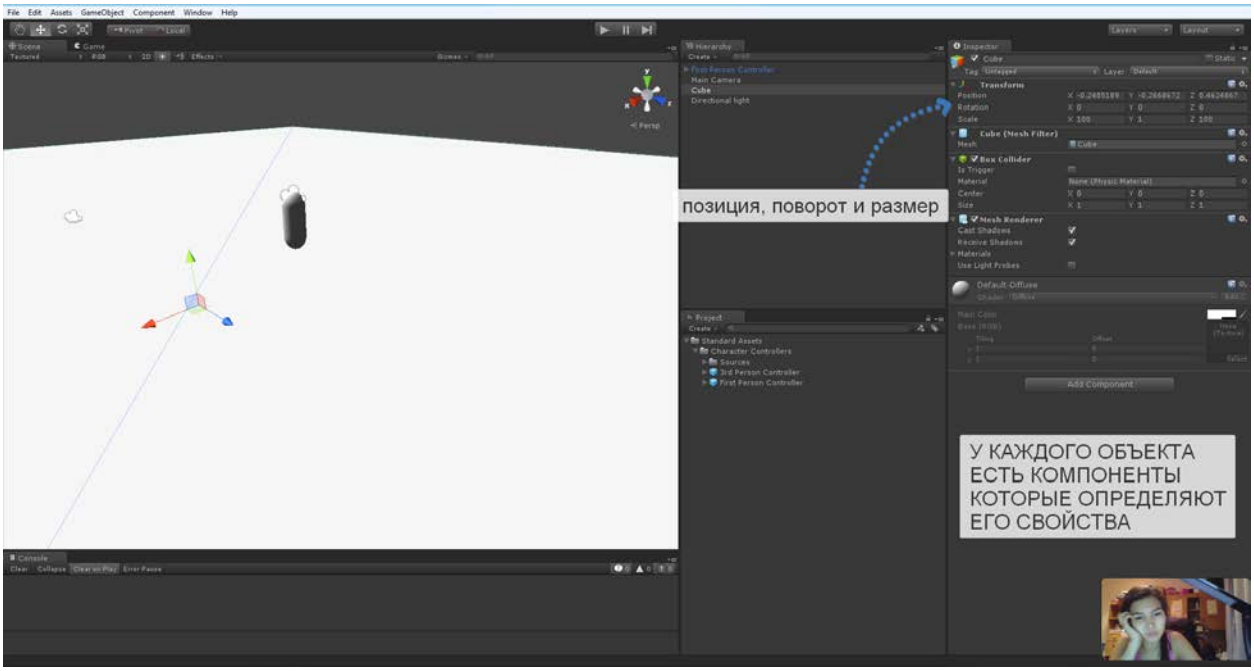
После того как вы создадите новый проект, у вас будет совершенно пустая сцена. Нашему персонажу нужно по чему-нибудь ходить! Для этого перейдите во вкладку **Game Object -> Create Other** и выберите там **Cube**.

У вас создастся в сцене куб, но он слишком маленький что бы по нему можно было ходить. Кликните в окне сцены по нему и в окне Inspector, на компоненте Transform измените его размер (**scale**) с 1 на 100 по оси **x** и **z**.

Назовём это землёй!

В окне **Project** у вас должна быть папочка **Standard Assets**, найдите в ней файл с именем **First Person Controller** и перетащите его на сцену. Чуть выше нашего куба. Только что бы он не пересекался с ним! Можно поставить даже чуть выше. Далее запустите игру нажав на кнопку Play которая находится в верхней части вашей программы. После запуска вы сможете управлять персонажем от первого лица.

В сцене достаточно темно, нам следовало бы добавить источников света. Опять идём в **Game Object -> Create Other**, но теперь выбираем **Direction Light**. Только что мы создали глобальный источник света.



СИСТЕМА СЦЕН

Вся игра у нас состоит из сцен, содержащих объекты которые обладают некоторыми свойствами. При создание нового проекта у вас будет пустая сцена которая даже не сохранена. Нажмите сочетание клавиш **Ctrl + S** или на соответствующую кнопку во вкладке **File -> Save Scene**. После этого вам откроется диалоговое окно с предложением сохранения сцены, сохранённая сцена будет отображаться в окне **Project**.

Для создания новой сцены нужно опять нажать на кнопку во вкладке **File**, но теперь эта кнопка **New Scene**.

Двойное нажатие на сцене в окне **Project** откроет её для редактирования. Перемещение по сценам происходит из кода, с помощью вызова специальных методов. Для того что бы сцена была доступна из кода, её нужно задекларировать, для этого откройте нужную сцену и пройдите в окно **File -> Build Settings** где следует нажать на кнопку **Add Current**.

ПРОГРАММИРОВАНИЕ В UNITY3D

Как я уже сказал ранее, каждый объект обладает каким-либо набором свойств. Но стандартные компоненты не позволяют реализовать всех задуманных возможностей, для того что бы решить эту проблему, мы можем написать себе скрипт (компонент) который бы что-то делал.

Писать скрипты мы можем на трёх языках: JavaScript (Unity Script), C# и Boo. В этой книге и в моих курсах я буду использовать C#.

Для начала нам нужно создать скрипт, для этого кликнете правой кнопкой мыши в окне **Project** и выберете в разделе **Create -> C# Script**. У нас создался новый скрипт, его можно сразу повесить на объект перетянув мышкой из вкладки **Project** в **Inpsector** выбранного объекта.

Далее нам следует открыть скрипт для редактирования, для этого нажмите два раза на него в окне **Project**. У вас должен открыться стандартный редактор скриптов Mono Develop. У меня установлена **Visual Studio**, она у меня и откроется.

```
using UnityEngine;
using System.Collections;

ссылок 0
public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    ссылок 0
    void Start () {

    }

    // Update is called once per frame
    ссылок 0
    void Update () {

    }
}
```

Здесь вы видите два метода **Start** и **Update** код помещённый в которые будет исполнятся в определённый момент. **Start** соответственно при старте игры а **Update** каждый кадр.

Важно заметить, что имя класса (то что после ключевого слова class, должно совпадать с названием скрипта в проекте).

Для того что бы хоть что-то произошло добавьте следующий код в метод **Update.**

transform.Translate(Vector3.up * 0.4);

После этого, объект на котором будет висеть этот скрипт будет лететь вверх.

Помимо **Start** и **Update** у нас есть ещё куча стандартных методов, с полным их списком можно ознакомится здесь в разделе **Message**.

http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

ЧТО ДЕЛАТЬ ДАЛЬШЕ?

Не советую браться за что-нибудь большое. Начните с простого арканоида, во время создания даже очень простой игры вы столкнётесь с множеством трудностей, в процессе преодоления, которых вы сможете подготовиться морально и умственно к разработке чего-нибудь крупного!

Изучайте программирование, это очень полезно. Также попробуйте изучить 3D моделирование, овладев этими двумя дисциплинами вы не будете чувствовать себя ограничено и сможете набираться опыта не завися от других людей. Старайтесь чаще общаться в сообществах разработчиков, и не бойтесь задавать вопросы.

ССЫЛКИ

Курс по C# [ПЛАТНЫЙ]

Курс по созданию игру без программирования [ПЛАТНЫЙ]

Курс по Unity3D [ПЛАТНЫЙ]

Курс по Unity3D [БЕСПЛАТНЫЙ]

Скачать новую версию этой книги всегда можно здесь - http://sakutin.ru/1/gamebook/