Глава 1

В наше время электронно-вычислительные устройства сильно уменьшаются в размерах. Мобильные телефоны и планшеты сейчас - это самые настоящие компьютеры. При этом их возможности, наоборот, возрастают очень быстрыми темпами.

Вычислительная мощность современного мобильного телефона серьезно превосходит общую вычислительную мощность компьютеров, в которых рассчитывали первые космические полеты. Это дает возможность использовать их в том качестве, о котором раньше и не думали. Электронные устройства, в том числе мобильные, давно перестали быть только «вычислительными». Удивительно, что одно устройство весом значительно меньше килограмма является и видеоплеером, и игровой приставкой, и навигатором и, конечно, средством беспроводной коммуникации.

Для того, чтобы увидеть результат программы, пусть даже самой простой нужно:

1) Написать программу. Для этого нужно владеть языком программирования, причем знать не только синтаксис, но и возможности так называемых библиотек, уметь пользоваться текстовым редактором, при необходимости установить его, сохранить файл на диске

2) Скомпилировать. Процессор понимает только машинные команды, поэтому нужно перевести текст программы в числовые коды. Это не всегда происходит сразу после написания кода. Во многих языках программирования окончательная компиляция происходит уже в процессе выполнения программы.

3) В случае, если программа разрабатывается для другого устройства, загрузить откомпилированный (или частично откомпилированный) код на устройство. Для этого нужно связаться с устройством, использовать протоколы передачи информации.

4) Запустить программу на выполнение.

Прежде всего, главный инструмент разработчика это среда программирования (еще называют “среда разработки”). Для эффективной работы необходимо хорошо представлять ее возможности и постоянно пользоваться ими. Обычно среда разработки содержит несколько компонент. Редактор кода. Главное преимущество редактора среды программирования перед обычным редактором, например, Блокнотом в Windows – в упорядоченности и подсветке синтаксиса – ключевые слова и символы выделяются цветом.

Компилятор. Среда может устанавливаться и без компилятора и работать с различными компиляторами, установленными на компьютере, но в любом случае она делает процесс компиляции «прозрачным» для программиста.

Отладчик. Для того, чтобы разобраться в том, почему же программа работает не так, как задумывал разработчик, можно не только пристально смотреть в код, но и выполнить программу по шагам, чтобы посмотреть промежуточные результаты работы и понять, правильно ли ведет себя программа. Все это делает отладчик. Без отладчика (как, в прочем и без среды программирования) в принципе можно обойтись, но грамотное его использование серьезно облегчает работу.

Как компьютер, который работает только с машинными командами, понимает текст, написанный программистом?

Очевидно, что эту задачу можно решить различными способами. Принципиальное отличие между этими способами в том, где происходит процесс преобразования программы в исполняемый код: на стороне программиста или на устройстве пользователя программы.

1 способ. Программист с помощью компилятора формирует на своём компьютере исполняемый (машинный) код, понятный вычислительному устройству. Это характерно для программирования на языках С, С++, Pascal. Пользователь запускает готовый файл, например формата .exe, и устройство его исполняет. Этот способ хорош тем, что вся подготовительная работа, требующая вычислительных ресурсов уже выполнена, поэтому на устройстве не нужно иметь дополнительно никаких специальных программ, наша программа выполняется с высокой производительностью. Однако есть и недостатки. скомпилированный машинный код сильно зависит от операционной системы и типа устройства. Это значит, что, например, для Windows, Linux или Android нужно использовать разные компиляторы и иногда даже писать программы по-разному, учитывая особенности этих ОС.

2 способ. На устройстве пользователя специальная программа интерпретатор “на ходу” преобразовывает программу в команды для процессора вычислительного устройства. Таким образом, например, выполняются программы на языках Python, PHP, JavaScript. Недостатки и достоинства этого способа прямо противоположны первому: программы исполняются медленно, но зато они кроссплатформенные (т.е. не зависят от ОС и типа устройства)

3 способ. Исходный текст программы преобразуется компилятором в некоторый байт код (промежуточный код), состоящий из стандартных команд для специальной программы - виртуальной машины на устройстве пользователя. Такой способ реализован для Java. Байт-код является машинно-независимым и поэтому программист пишет код один раз. Так как байт-код - это уже частично преобразованный код, то от устройства требуется меньше вычислительных ресурсов, чтобы перевести его в машинный код. Понятно, что для каждой операционной системы и конкретной архитектуры процессора необходима своя виртуальная машина, но она пишется разработчиками тоже только один раз. Существует множество различных виртуальных машин (VM): к примеру для программ Java на большинстве устройств используется Java VM (JVM) от Oracle, а для мобильных устройств на Android - это Dalvik от Google.

Компьютерные игры – один из самых популярных способов использования компьютеров, смартфонов и планшетов. Первые игры отличались простотой графикой, но со временем игры становились все сложнее. Над их созданием работал уже не один программист, а целый коллектив разработчиков – программисты, дизайнеры, художники, аниматоры, специалисты по звуковым эффектам и люди других профессий. Для обеспечения эффективности разработки, ее ускорения и повышения качества игрового продукта в середине 90-х годов появился новый вид программного обеспечения для разработчиков – игровой движок.

Игровой движок - это центральный программный компонент приложения, который обеспечивает основные технологии и, зачастую, кроссплатформенность. Изначально под игровым движком понимали подсистему обеспечения визуализации двумерной и, затем, трехмерной графики, создания анимации и визуальных эффектов. Позже движки стали обеспечивать поддержку физики игрового мира (физический движок), звука, сетевого взаимодействия, искусственного интеллекта. Какие преимущества дает игровой движок разработчикам? - Прежде всего, игровой движок создает каркас игрового приложения, делая код более организованным и управляемым. Не секрет, что современные игры содержат миллионы строчек кода. Продуманная архитектура игры, которая во многом обеспечивается движком, упрощает командную разработку, управление и поддержку кода из миллионов строк. Движок сокращает рутинные технические моменты реализации игровых процессов. Например, вместо изучения технических аспектов, обеспечивающих создание быстрой и плавной анимации, вызова множества библиотечных функций, можно использовать высокоуровневые функции движка, компактно решающих эту задачу. Движок позволяет сконцентрироваться на разработке игровой логики, позволяет мыслить более высокоуровневыми категориями. Особая ценность движка – переносимость, кроссплатформенность. Это, конечно, справедливо не для всех игровых движков. Но хорошо спроектированный игровой движок упрощает перенос игры на другие платформы. Игровые движки - это сложные программы, которые невозможно освоить “с ходу”. Поэтому и говорят, что это инструмент для профессионалов.

В проектировании игрового приложения можно выделить некоторые этапы:

1) Идея. Прежде всего, разработка игры начинается с идеи. Она может возникнуть случайно или в процессе долгого перебора. Можно отталкиваться от любимого жанра или от жанра, который популярен на данный момент. Идея может быть заимствована у другой игры или, напротив, представляет нечто совершенно новое.

2) Исследование. Иногда для разработки игры нужно выяснить некоторые делали. Если вы разрабатываете симулятор спортивной игры, вам нужно подробно ознакомиться с её правилами. Скорее всего, в процессе найдется что-то новое. Это поможет привнести в игру оригинальные находки в сюжет (если он присутствует) или игровой процесс.

3) Разработка геймплея. На этом этапе начинается проработка игрового процесса. На этом Ээтапе нужно уже полностью представлять как будет выглядеть готовое приложение. Какие там будут противники (если они будут), управление и сюжет(если будет). Проработанный геймплей лучше заинтересовывает игрока. Но нужно не перестараться, например, введя пол сотни видов врагов.

4) Проектирование архитектуры. Подходы к проектированию игровых приложений в целом не отличаются от остальных. Главное на этом этапе определиться с необходимыми классами, рассмотреть возможность применения наследования, абстракции. К примеру, если мир в игре случайно генерируемый, нужно проработать алгоритм генерации, если в игре много уровней, которые создаются вручную, возможно, стоит задуматься о написании редактора уровней. Нет уникального рецепта проектирования архитектуры, многое зависит от конкретного приложения.

5) Реализация, тестирование. Этот этап включает себя написание кода. Также сюда входит рисование графики, подбор звукового сопровождения, проектирование уровней. Другими словами, подготовка или создание игровых ресурсов и контента.