

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ по дисциплине
«Программирование в бизнес-системах»
направление подготовки 38.03.05 «Бизнес-информатика»
направленность (профиль) «Информационная бизнес-аналитика и
цифровые технологии»

Ставрополь 2026 г

Содержание

3 семестр

Лабораторная работа №1.....	3
Лабораторная работа №2.....	13
Лабораторная работа №3.....	18
Лабораторная работа №4.....	25
Лабораторная работа №5.....	38
Лабораторная работа №6.....	45
Лабораторная работа №7.....	46
Лабораторная работа №8.....	49
Лабораторная работа №9.....	54
Лабораторная работа №10.....	57
Лабораторная работа №11.....	65
Лабораторная работа №12.....	68
Лабораторная работа №13.....	73
Лабораторная работа №14.....	77
Лабораторная работа №15.....	81
Лабораторная работа №16.....	86
Лабораторная работа №17.....	89
Лабораторная работа №18.....	90

4 семестр

Лабораторная работа №1.....	98
Лабораторная работа №2.....	100
Лабораторная работа №3.....	104
Лабораторная работа №4.....	106
Лабораторная работа №5.....	111
Лабораторная работа №6.....	114
Лабораторная работа №7.....	116
Лабораторная работа №8.....	121
Лабораторная работа №9.....	125
Лабораторная работа №10.....	128
Лабораторная работа №11.....	129
Лабораторная работа №12.....	131
Лабораторная работа №13.....	134
Лабораторная работа №14.....	136
Лабораторная работа №15.....	140
Лабораторная работа №16.....	144

Семестр 3

Лабораторная работа №1. Подготовка к разработке, установка и настройка среды разработки Android Studio.

Android - бесплатная операционная система, основанная на Linux с интерфейсом программирования Java.

Android поддерживает 2D и 3D-графику, используя библиотеки OpenGL, а также хранение данных в базе данных SQLite.

Каждое Android-приложение запускается в своем собственном процессе. Поэтому приложение изолировано от других запущенных приложений, и неправильно работающее приложение не может беспрепятственно навредить другим приложениям.

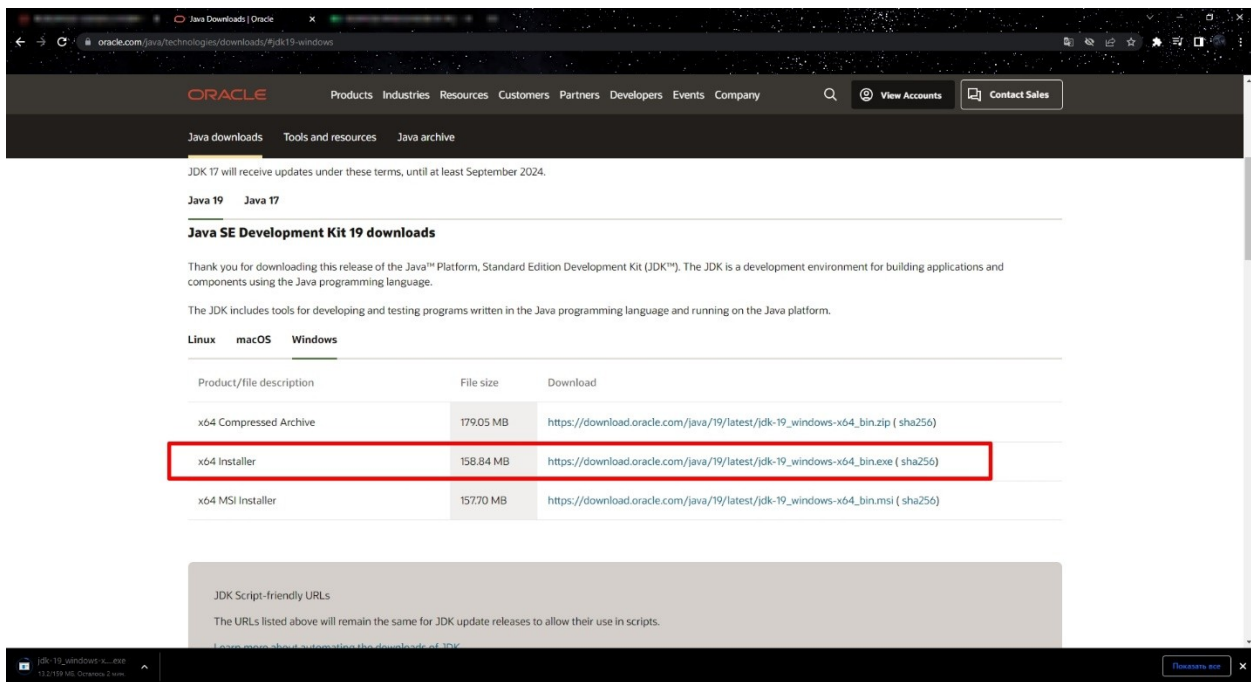
В качестве языка программирования для Android используется Java. Для создания пользовательского интерфейса используется XML.

Основные компоненты Android:

- Activity (Активность, Деятельность) — представляет собой экран пользовательского интерфейса, который представлен классом Activity и разметкой в виде XML-файла (разметку можно создать и программно, но теперь так не делают). Android-приложение способно состоять из нескольких форм (Activity) и может переключаться между ними во время выполнения приложения
- Intents (Намерения) — асинхронные сообщения, которые позволяют приложению запросить функции из других служб или действий. Приложение может делать прямые запросы службе или действию (явное намерение) или запросить у Android зарегистрированные службы и приложения (неявное намерение). Для примера, приложение может запросить через Intent приложение Контакты для получения данных или запустить браузер при нажатии ссылки в тексте вашего приложения.
- Views (Представление, Вид, Элемент управления, Компонент) — пользовательский интерфейс, создаваемый виджетами классов. Это знакомые нам элементы управления: кнопки, текстовые поля, флажки, переключатели и т.п.
- Services (Службы) — выполняют фоновые задачи без предоставления пользовательского интерфейса (например, проигрывание музыки). Они могут уведомлять пользователя через систему уведомлений Android.
- Content Provider (Контент-провайдер) — предоставляет данные приложениям. С помощью контент-провайдера ваше приложение может обмениваться данными с другими приложениями.
- Broadcast Receiver (Приемник широковещательных сообщений/запросов) — принимает системные сообщения и неявные намерения, может использоваться для реагирования на изменение состояния системы. Приложение может регистрироваться как приемник определенных событий и может быть запущено, если такое событие произойдет.

Установка Android Studio

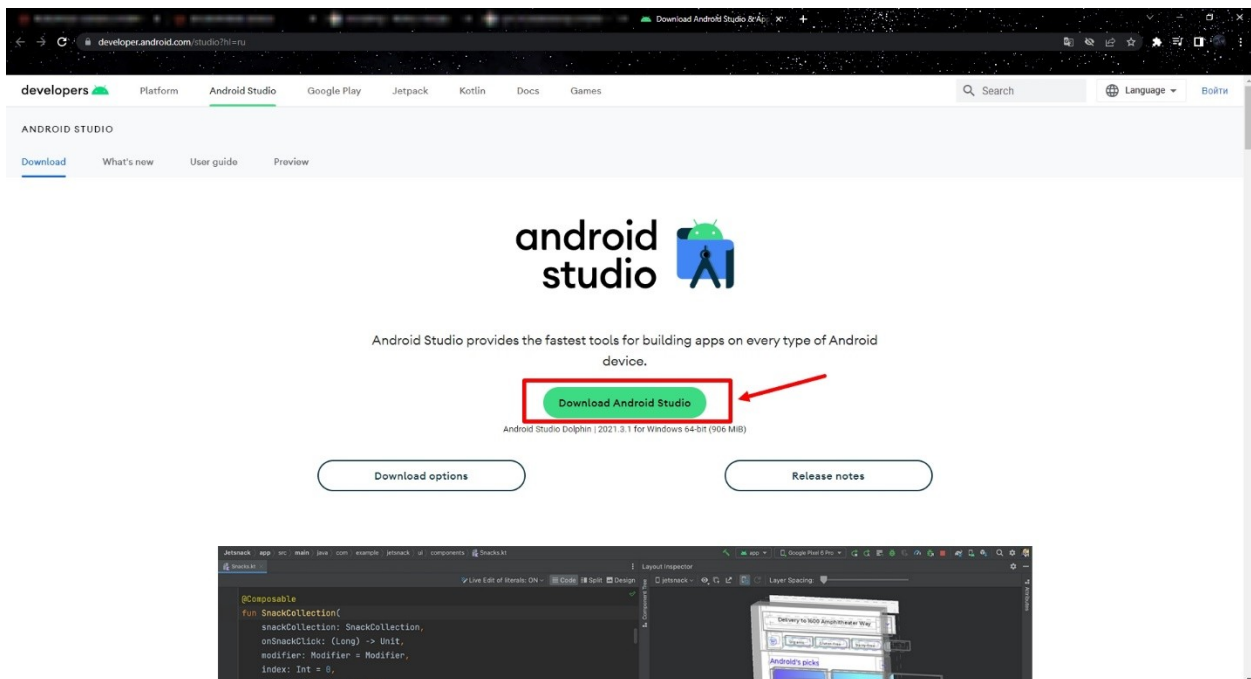
Для работы Android Studio необходимо прежде установить Java SE Development Kit - комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java. Скачать ее можно по этой [ссылке](#). Выберите необходимой пакет для x64 или x86 системы. С установкой проблем не должно возникнуть, используйте настройки по умолчанию.



После того как скачивание завершилось, необходимо открыть программу инсталлятор и выбрать путь к папке для установки данного ПО, дождаться завершения процесса и нажать кнопку “Close”.



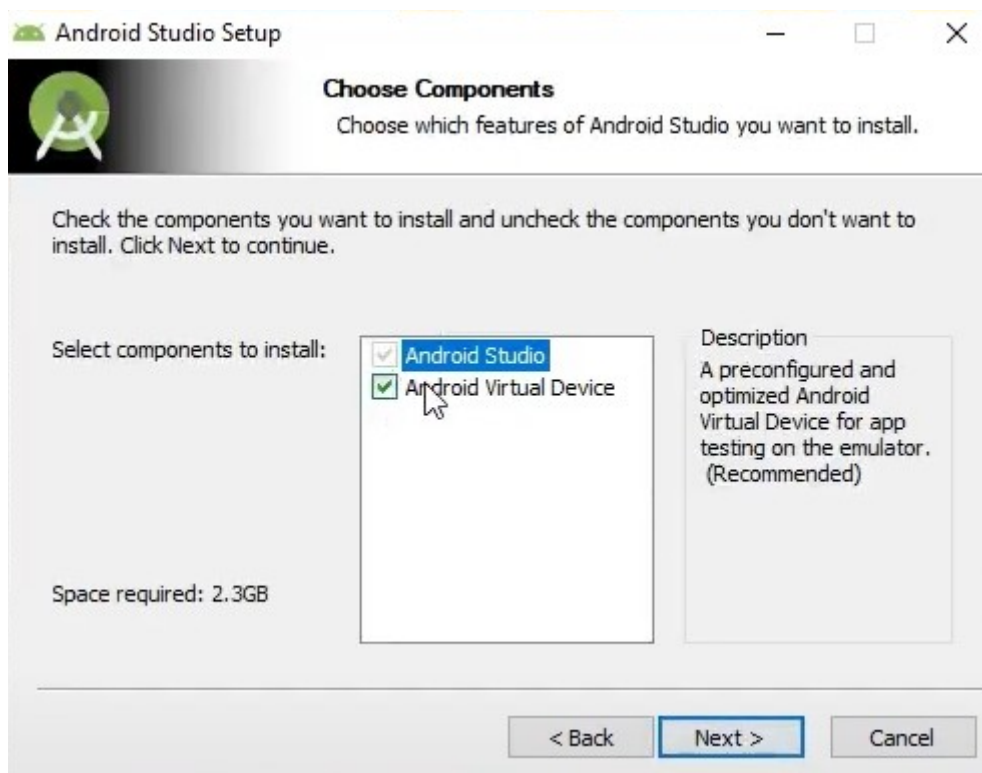
Теперь можно перейти к установке самой Android Studio. Ее скачать можно по следующей [ссылке](#).



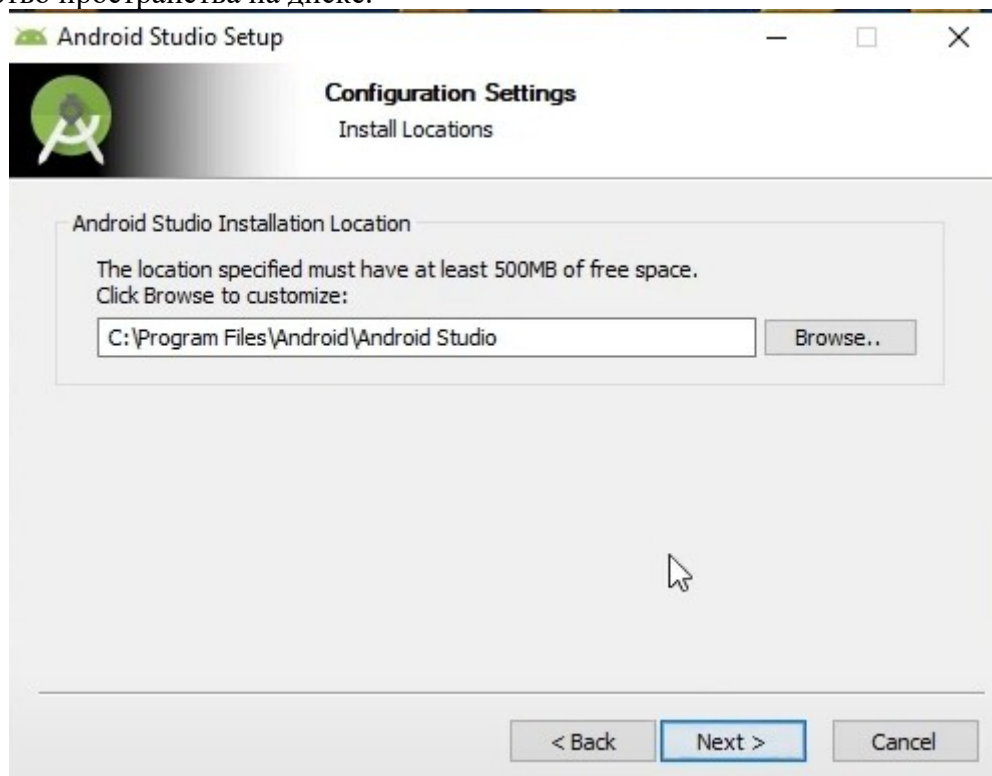
Далее, открываем установщик и нажимаем **Next**.



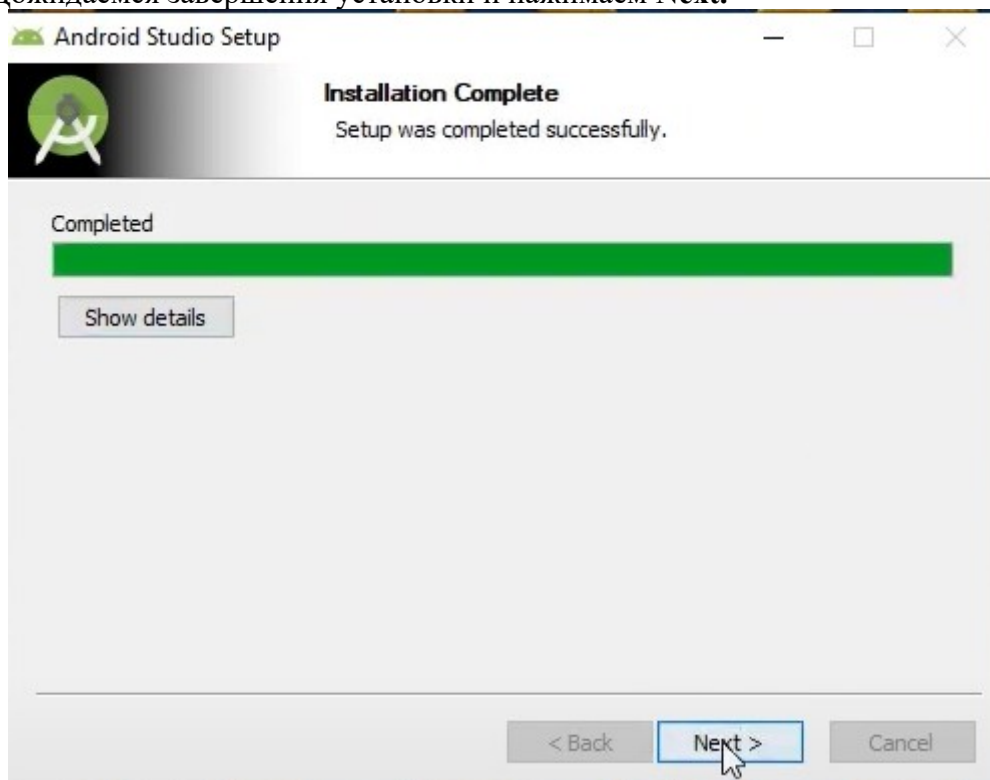
На этапе выбора компонентов для установки, оставляем всё без изменений.



Далее, установщик предлагает указать путь для установки необходимых компонентов. Если места на диске **C:** более, чем 30 гб., оставляйте без изменений. В ином случае, укажите диск с большим количеством свободного пространства, так как компоненты, которые программа скачает после первого запуска займут значительное количество пространства на диске.



Дожидаемся завершения установки и нажимаем **Next**.

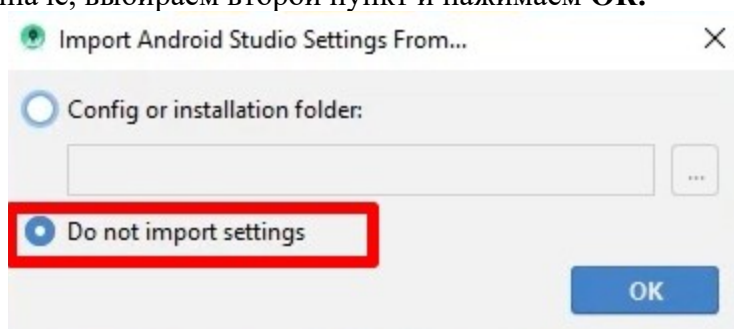


Оставляем галочку **Start Android Studio** и нажимаем **Finish**.

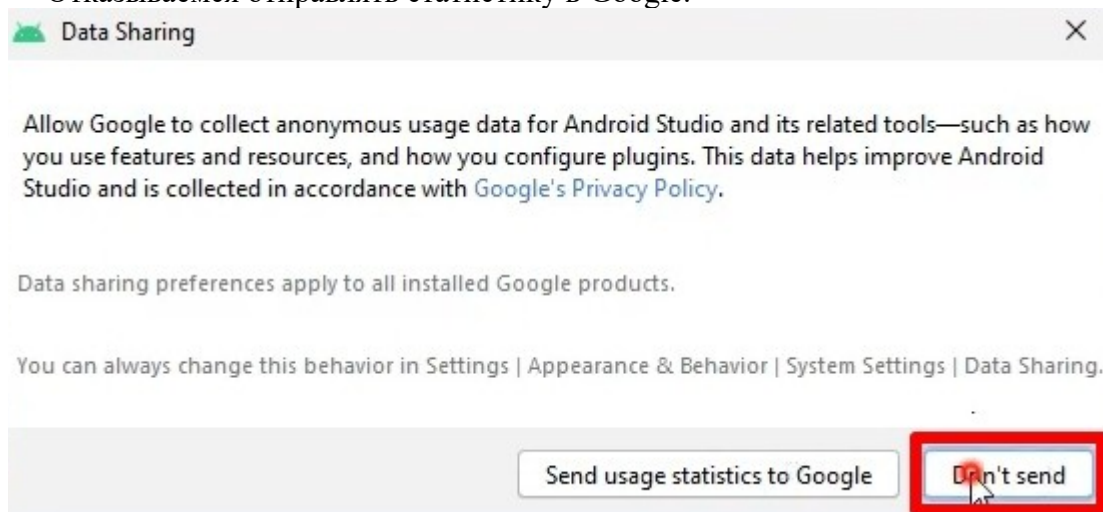


Процесс установки завершен, перейдем к настройке.

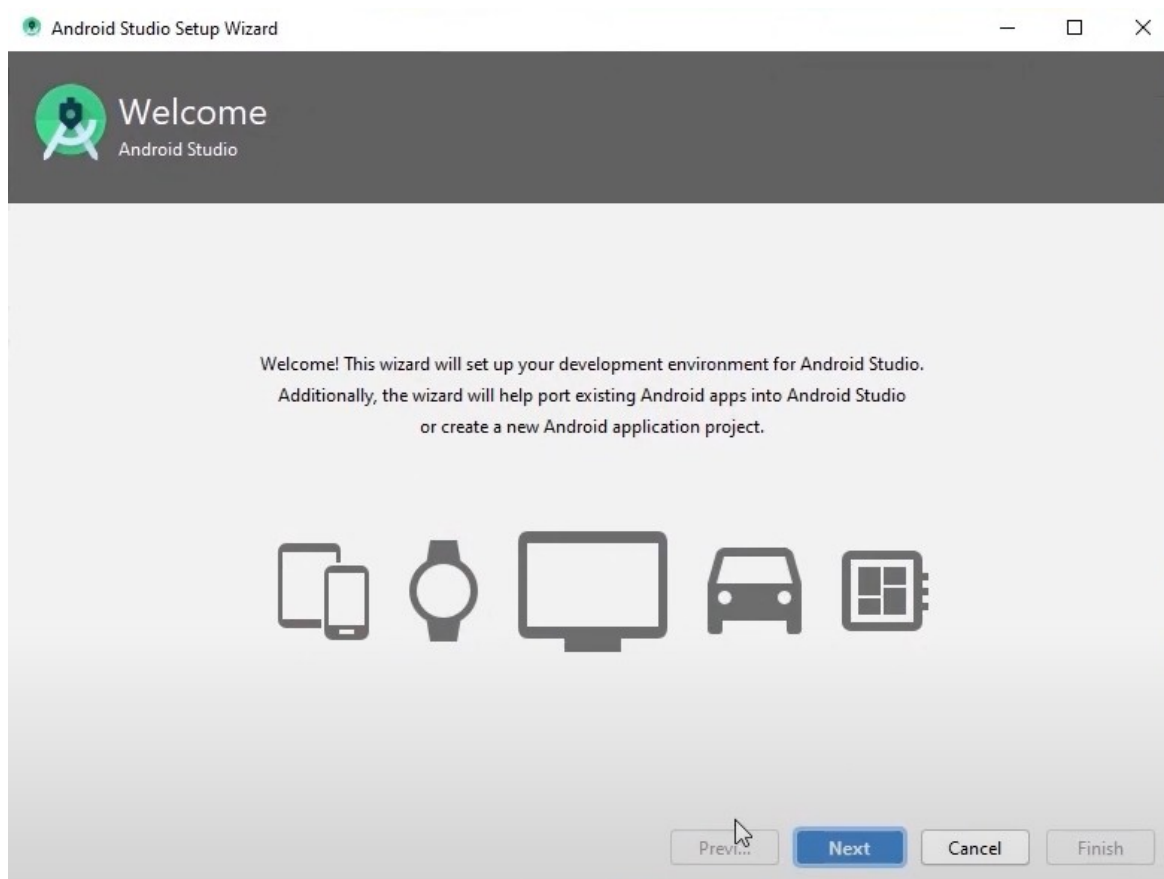
Если ранее, у вас был установлен Android Studio, а потом вы его удалили, программа автоматически найдет файлы для импорта настроек и укажет его в поле первого пункта, иначе, выбираем второй пункт и нажимаем **ОК**.



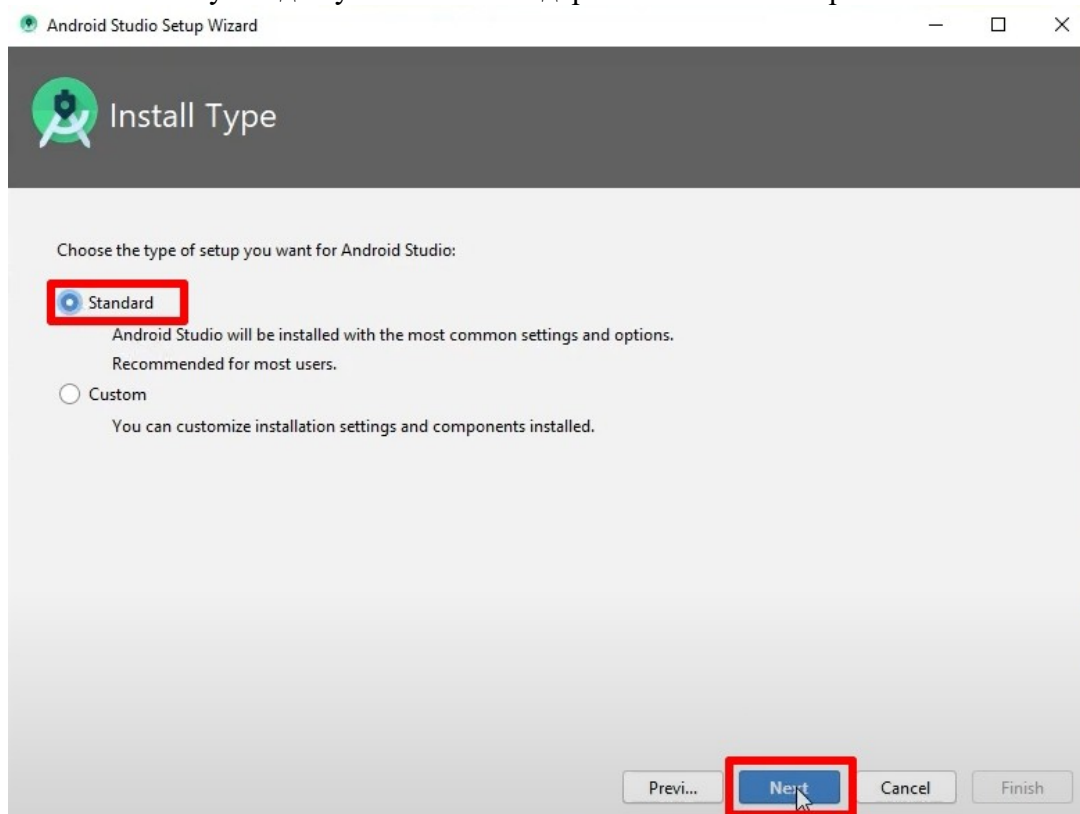
Отказываемся отправлять статистику в Google.



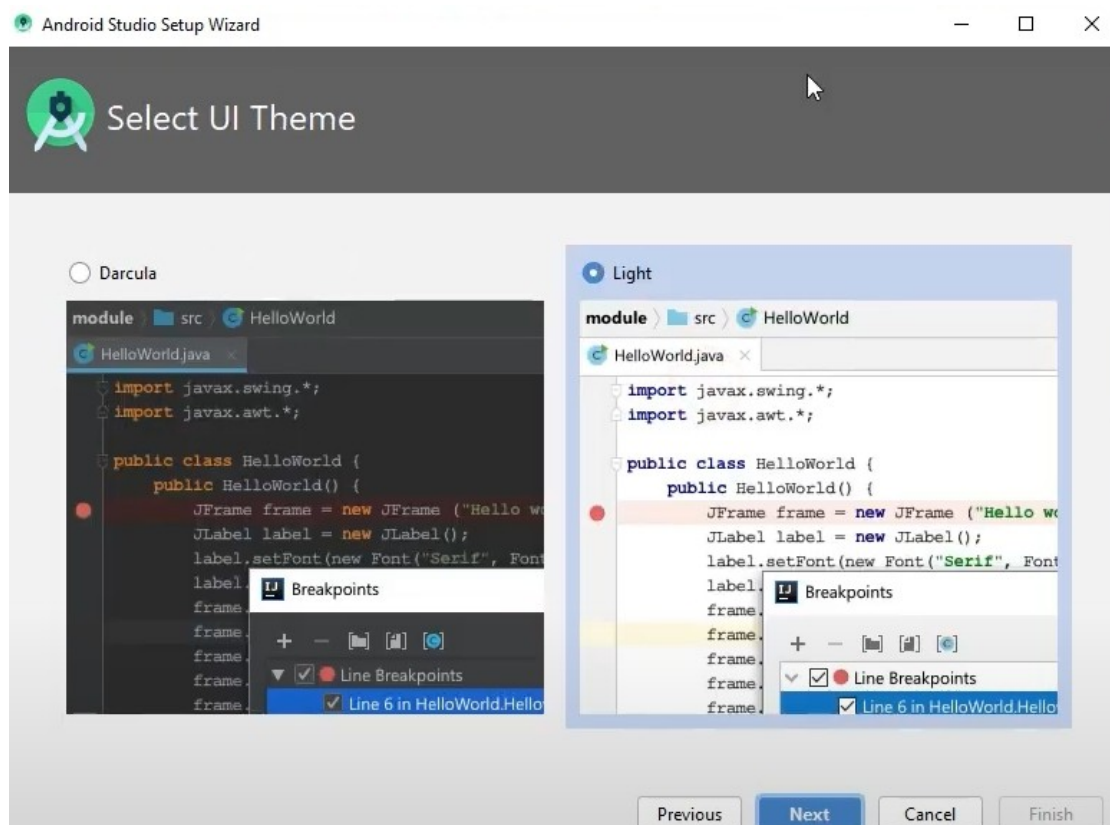
Далее, нажимаем **Next**.



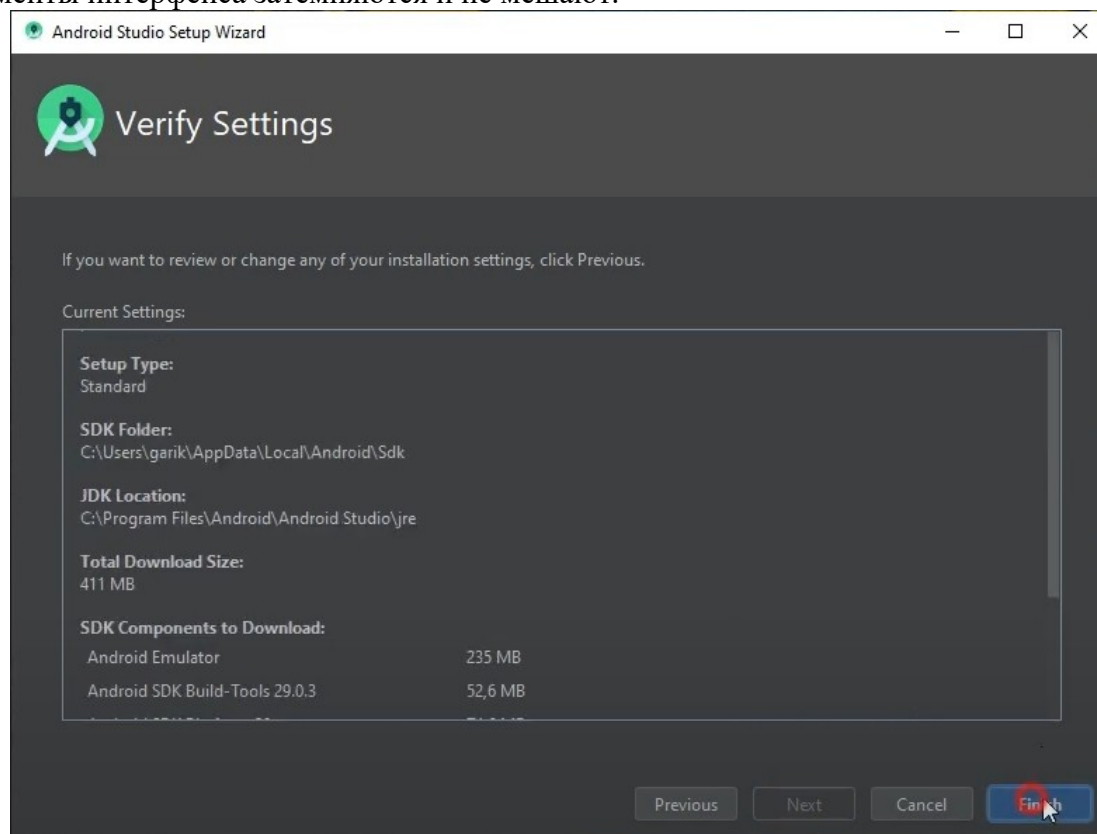
Указываем пункт для установки стандартного пакета настроек.



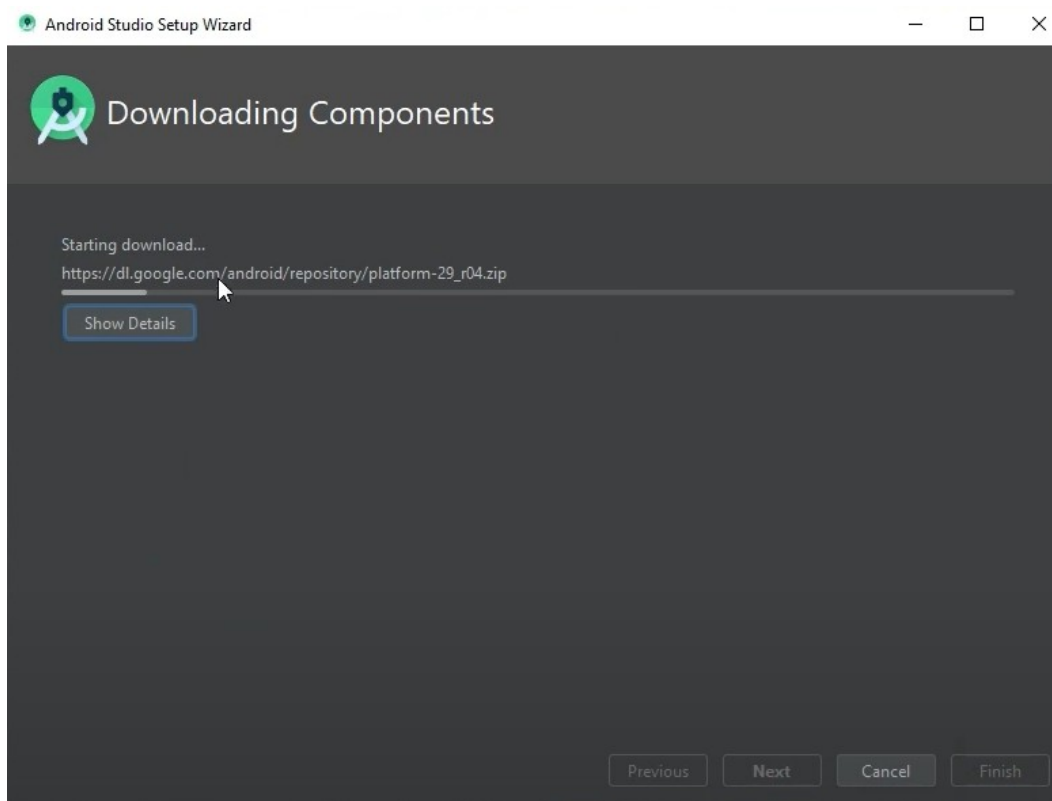
Выбираем внешний вид интерфейса программы, в которой будем работать. И нажимаем **Next**.



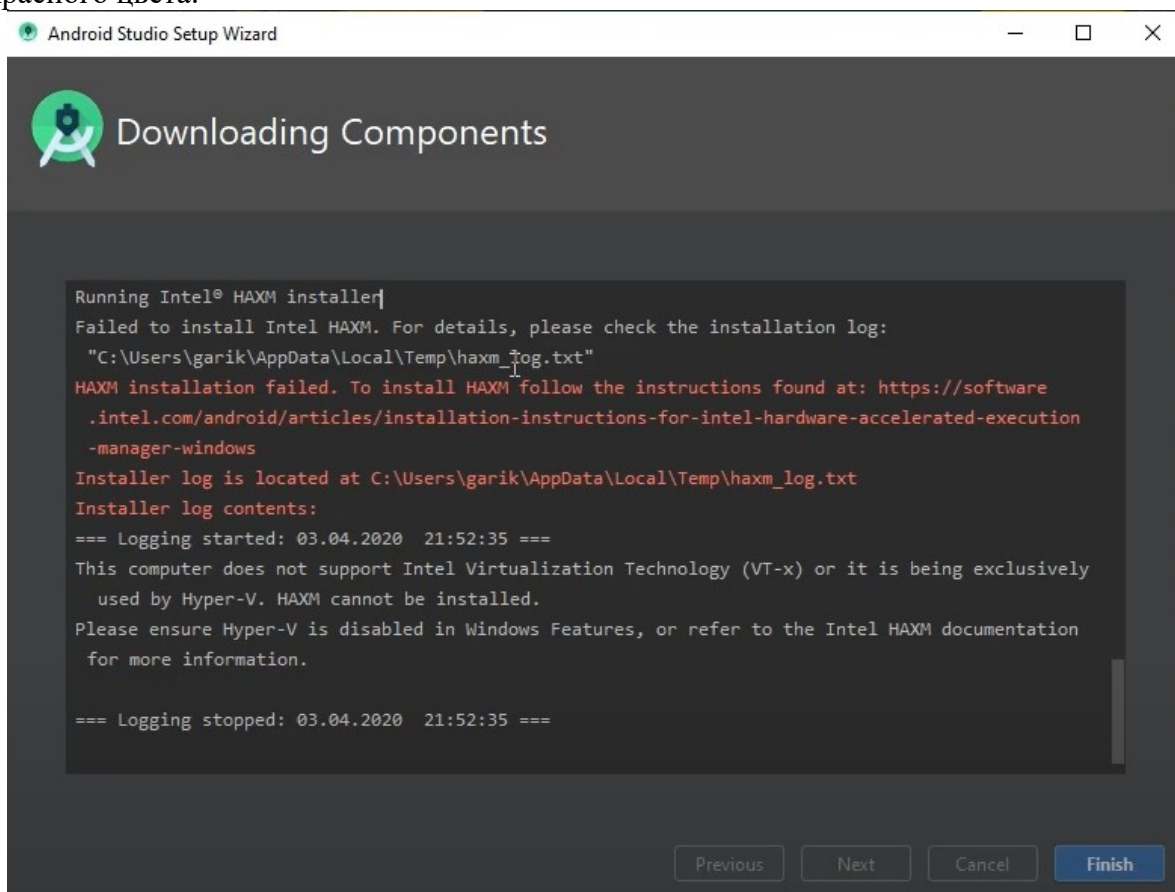
Из личного опыта могу сказать, что в темной версии работать более удобно. Она помогает сосредоточиться при работе с текстом. Нужный контент выделяется, а элементы интерфейса затемняются и не мешают.



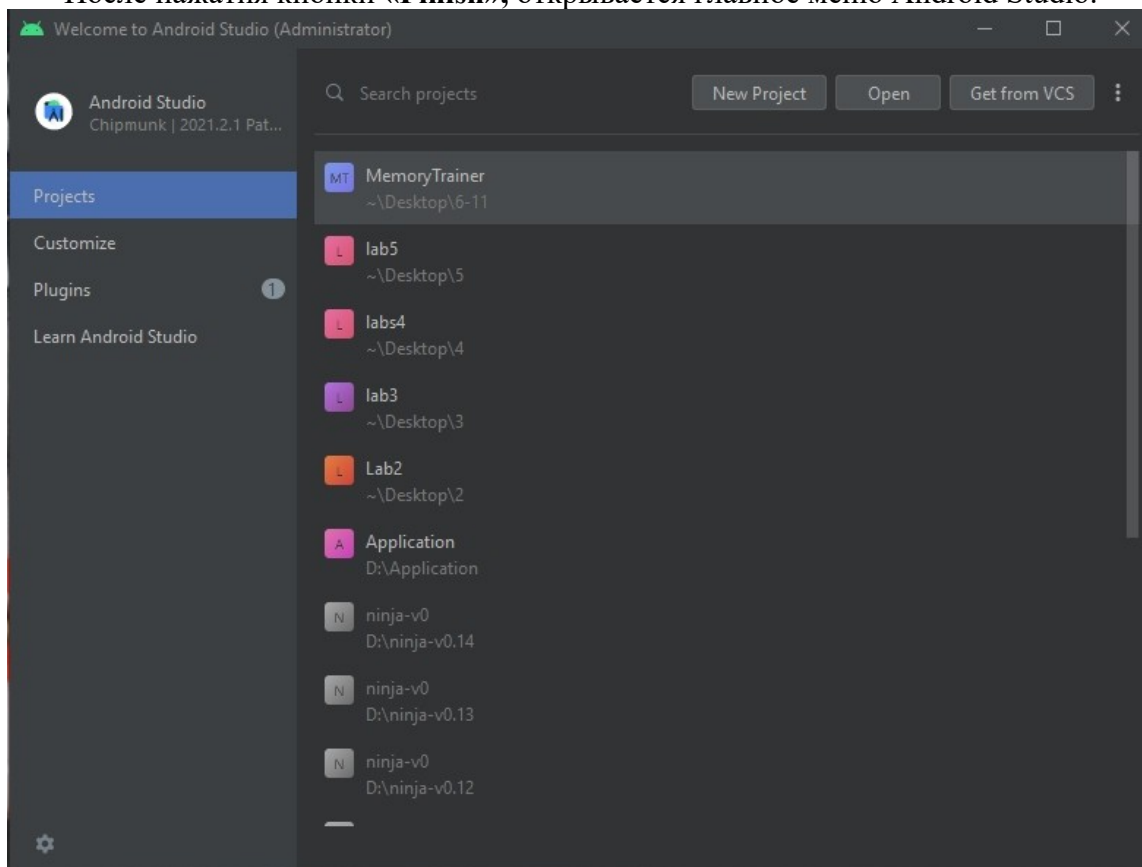
Нажимаем **Finish** и ждем скачивания необходимых компонентов.



Процесс установки окончен. Если у вас видеокарта от компании «Intel», расширение «Intel HAXM» для запуска Эмулятора (ПО для создания виртуальных устройств Android на ПК) в «Android Studio» установится у вас без проблем. Если у вас видеокарта от компании «AMD» или другого производителя вы увидите подобный текст красного цвета.

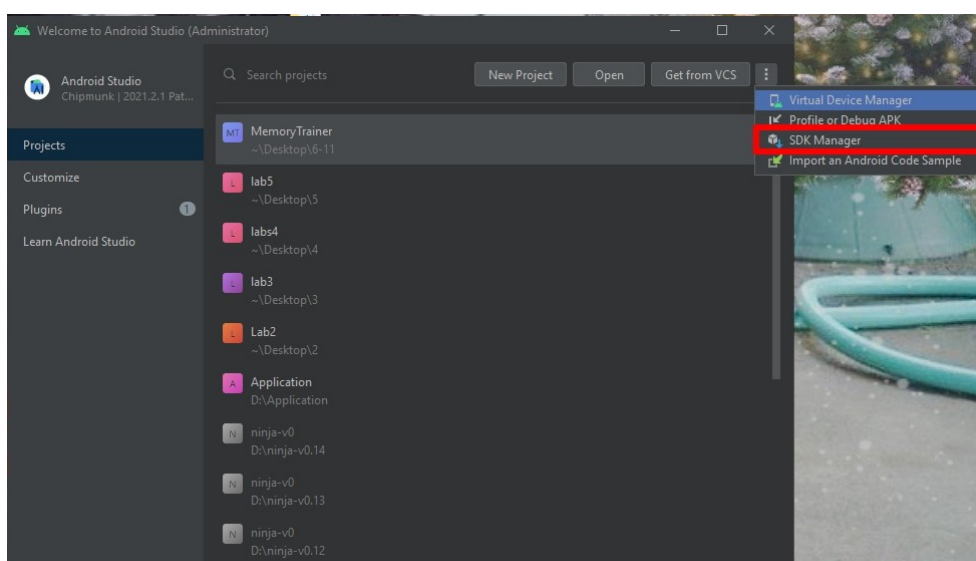


Нажимаем «**Finish**». На этом процесс установки и настройки окончен.
После нажатия кнопки «**Finish**», открывается главное меню Android Studio.



(Внешний вид главного меню у вас может отличаться, в зависимости от версии программы, которую вы устанавливаете)

Установочный пакет включает в себя необходимый минимум. При первом запуске откройте **Configure | SDK Manager**, где необходимо выбрать нужные версии платформы Android. Рекомендуется указать версии от 4.4.2 (рис. 1), но если вы обладаете Android устройством, то можете проверить установленную версию Android на нем и также указать ее.



Это нужно для того, чтобы вы имели возможность тестировать разработанное приложение на своем устройстве. Не рекомендуется указывать все версии платформы, так как это будет отягощать проект.

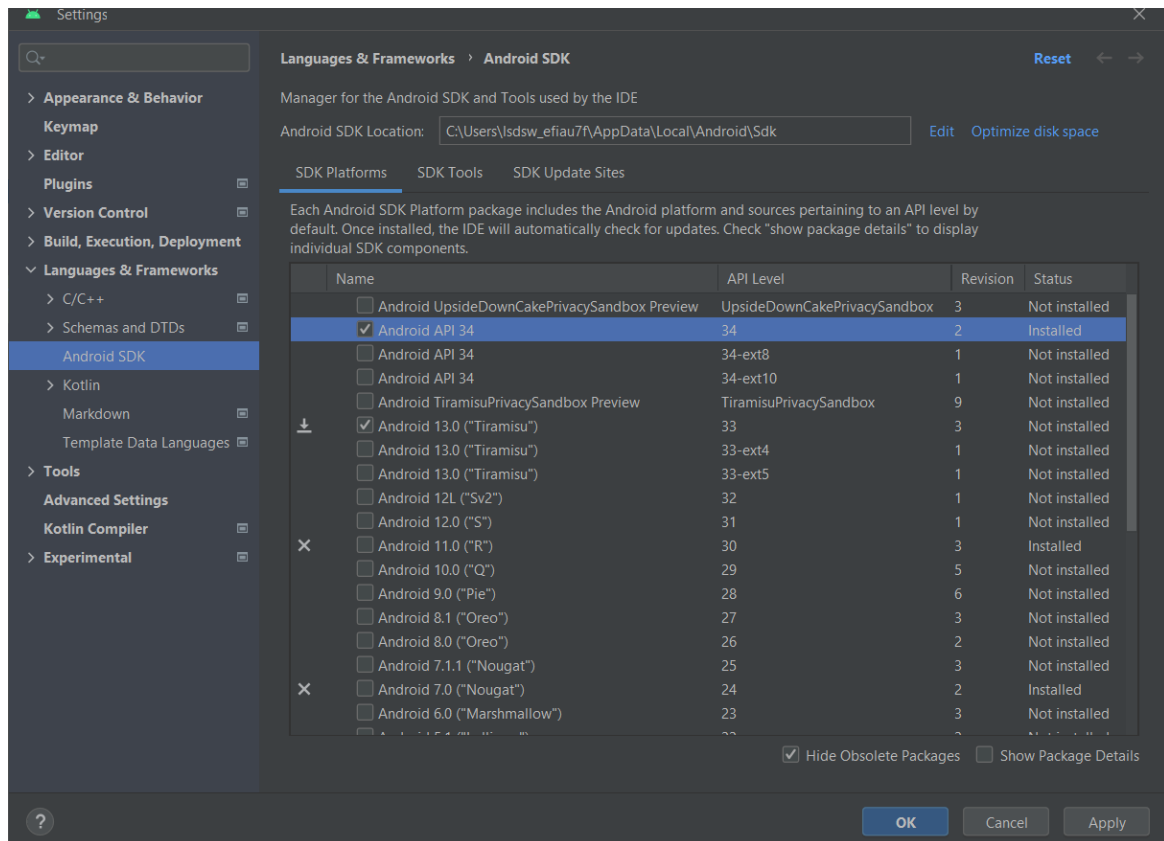


Рисунок 1 - Выбор версий платформы.

Для выполнения лабораторных работ, нам подойдет любая из предложенных версия, выше 4.4.2. Скачаем Android 33 или 34 версию платформы. Это нужно, потому что в новых версиях лучше всего поддерживаются последние версии платформы. Затем нажмите «Ок», после чего произойдет загрузка и установка выбранных платформ.

Лабораторная работа №2. Создание эмуляторов

Для отладки приложений можно воспользоваться двумя способами:

- эмулятор телефона - виртуальная машина, на которой будет запускаться разрабатываемое приложение;
- реальное устройство.

Чтобы создать эмулятор телефона, выбираем в меню **Configure | Virtual Device Manager**. (В зависимости от версии Android Studio, расположение элементов может отличаться)

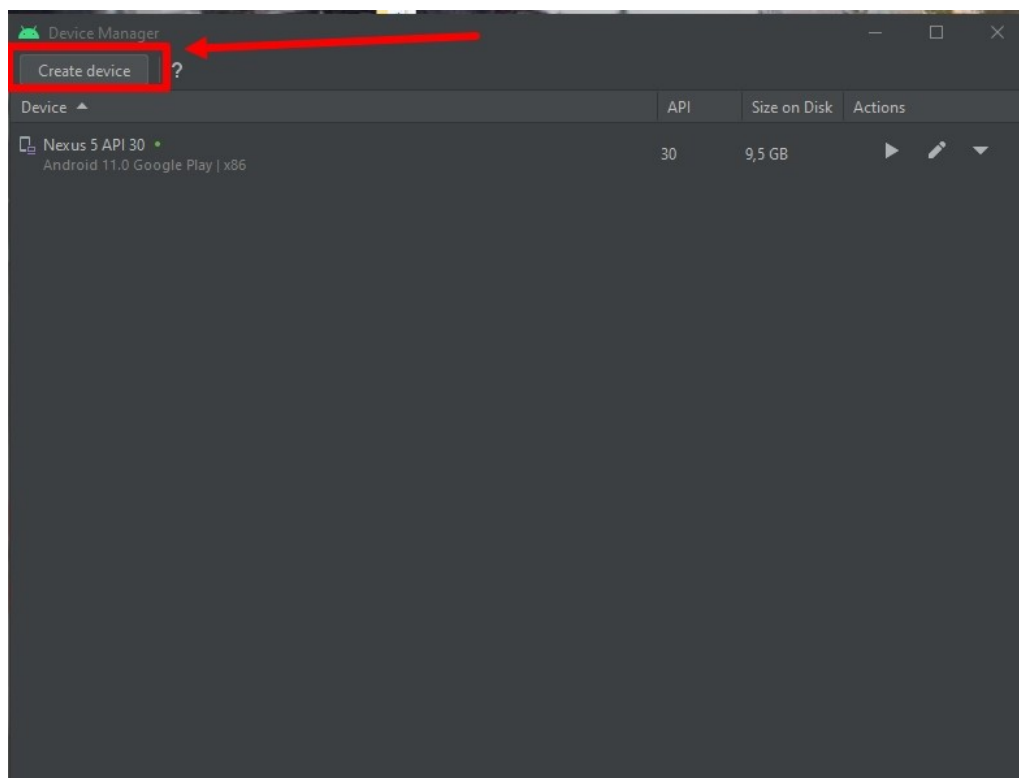
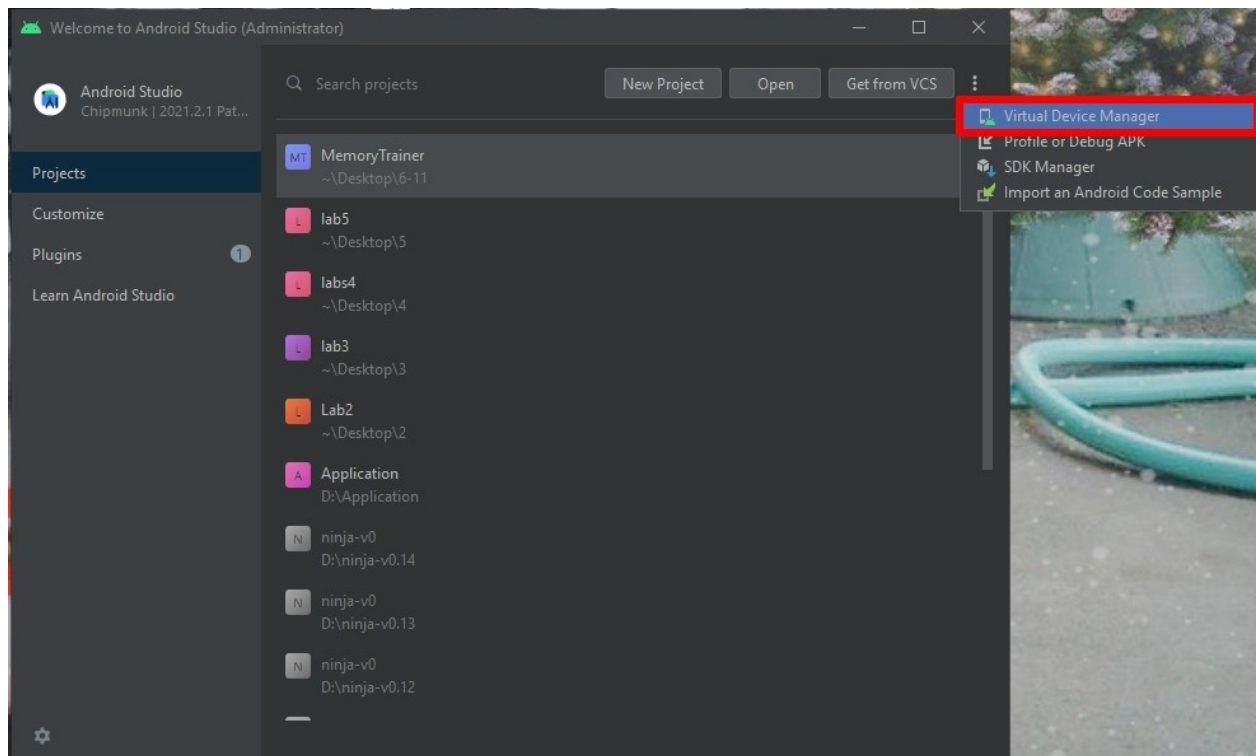


Рисунок 2 - Мастер создания эмуляторов устройств.

Нажимаем кнопку «**Create device**» и в новом окне видим набор возможных эмуляторов, в том числе и для часов (рис. 3).

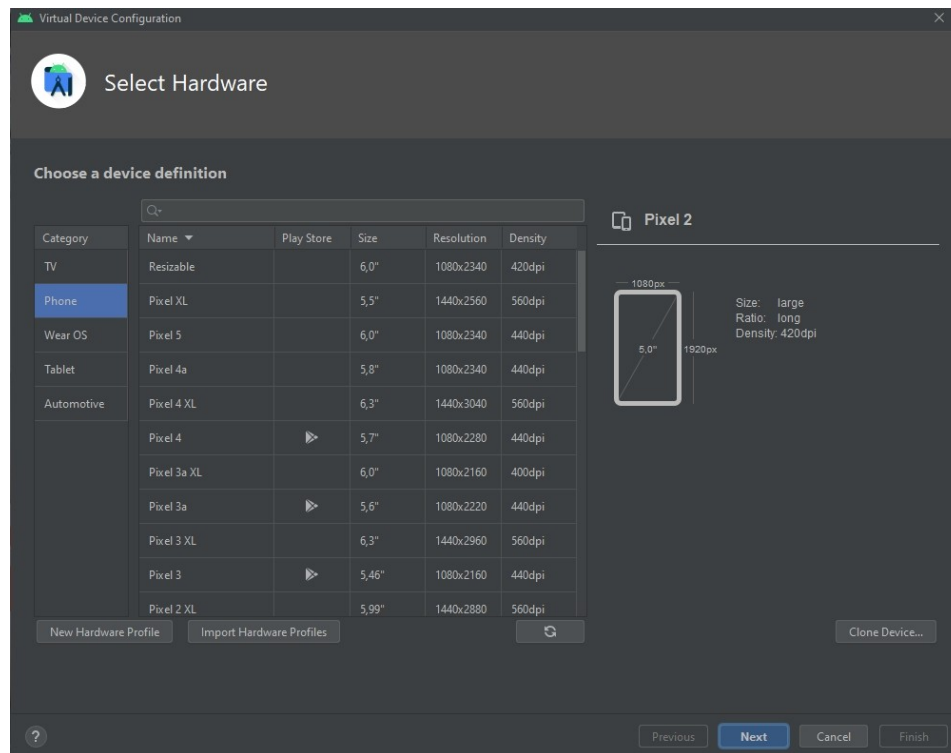


Рисунок 3 - Выбор необходимого устройства.

Скачайте необходимые эмуляторы (рис. 4). Для начала вполне подойдёт один эмулятор. Например «Nexus 5». Нажмем «Next» и выберем ранее установленную версию платформы Android.

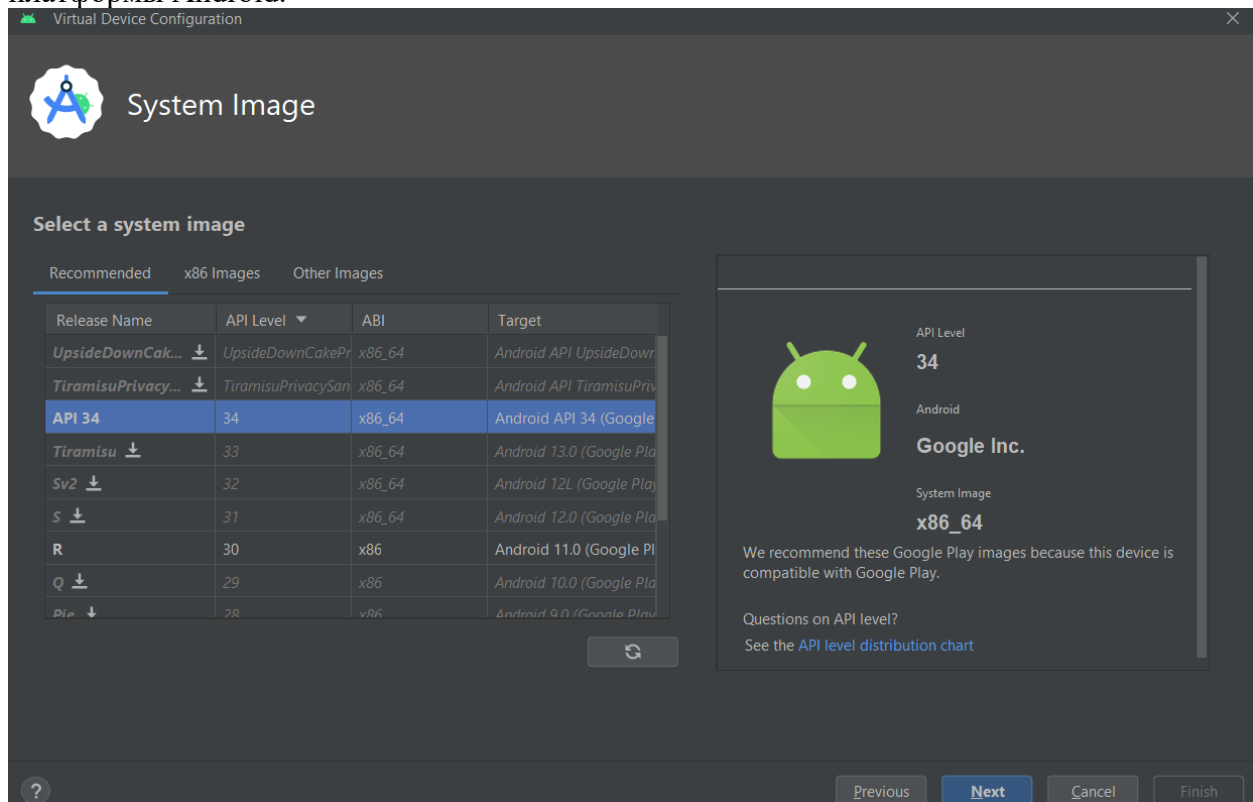


Рисунок 4 - Загрузка необходимого эмулятора.

Выбираем нужное расположение экрана и нажимаем «Finish»

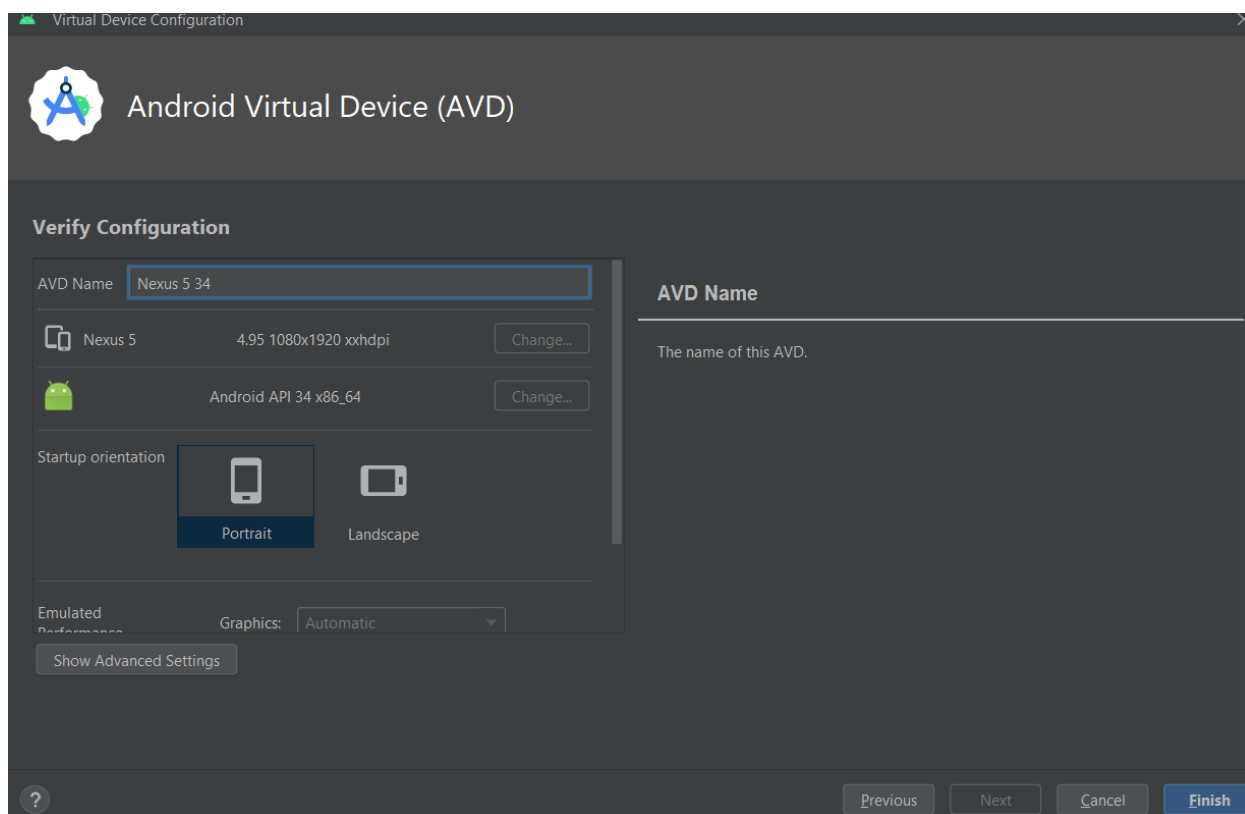


Рисунок 5 - Завершающий этап создания эмулятора.

При необходимости можно создать эмуляторы для каждой версии ОС и проверять программу на работоспособность. Остальные настройки можно оставить без изменений. Вы всегда можете вернуться к настройкам и отредактировать снова. Часто рекомендуют использовать опцию «**Use Host GPU**», чтобы задействовать возможности графического процессора. Это даёт прирост скорости эмулятора. Нажимаем кнопку «**Finish**».

Добавленные эмуляторы будут храниться в менеджере эмуляторов (рис. 6).

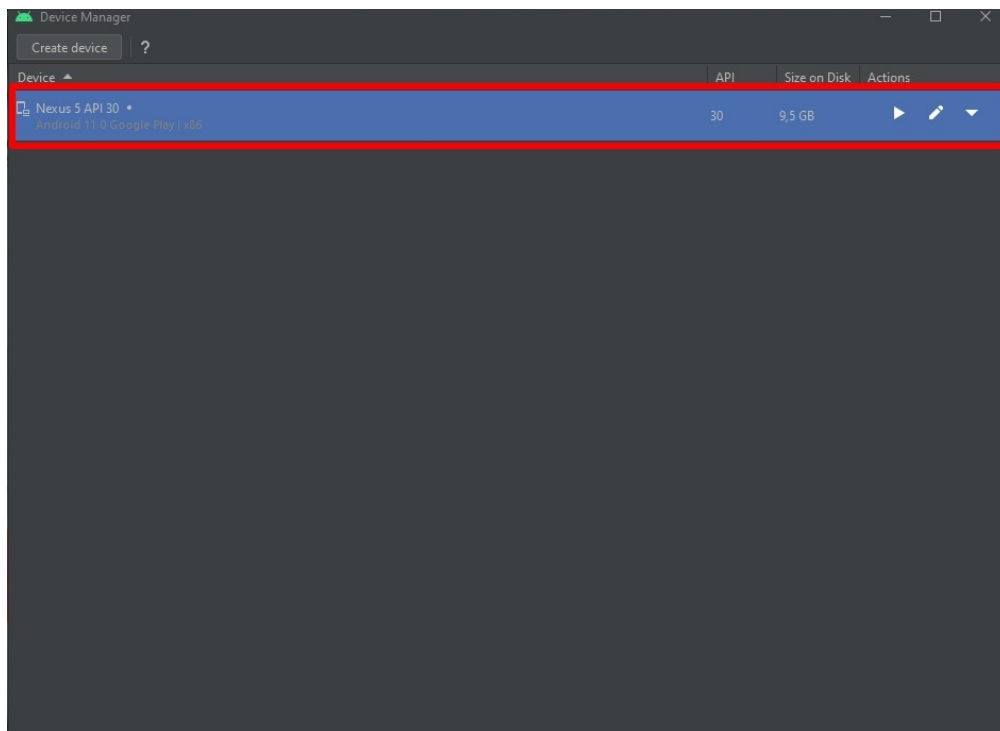


Рисунок 6 - Список созданных эмуляторов.

Если вы создали несколько эмуляторов, то выделите нужный и нажмите кнопку с зелёным треугольником для запуска эмулятора. Значок карандаша позволяет отредактировать настройки. Но обычно эмулятор не запускают отдельно. Когда вы будете запускать приложение, то студия сама предложит запустить эмулятор.

Помните, что виртуальные машины по умолчанию сохраняются в папке пользователя, и пути к папке не должны содержать русские символы во избежание проблем.

Также можно подправить ini-файл и прописать путь к виртуальному устройству таким образом, чтобы в пути не встречались русские буквы (соответственно, сам файл *.avd также нужно переместить в другое место).

В зависимости от мощности компьютера нужно немного подождать, чтобы загрузился эмулятор. Эмулятор от Intel загружается на порядок быстрее, но его установка немного сложнее. Информацию можно поискать у них на сайте.

К сожалению, бывает так, что при первом запуске эмулятор загружается, а приложение в нём не запускается. Это происходит от того, что во время загрузки эмулятора ваш компьютер теряет связь с ним. В этом случае вам нужно перезагрузить сервисы. Для этого необходимо открыть **Device manager**, выбрать наше устройство, открыть меню дополнительных функций и выбрать **Wipe Data**.

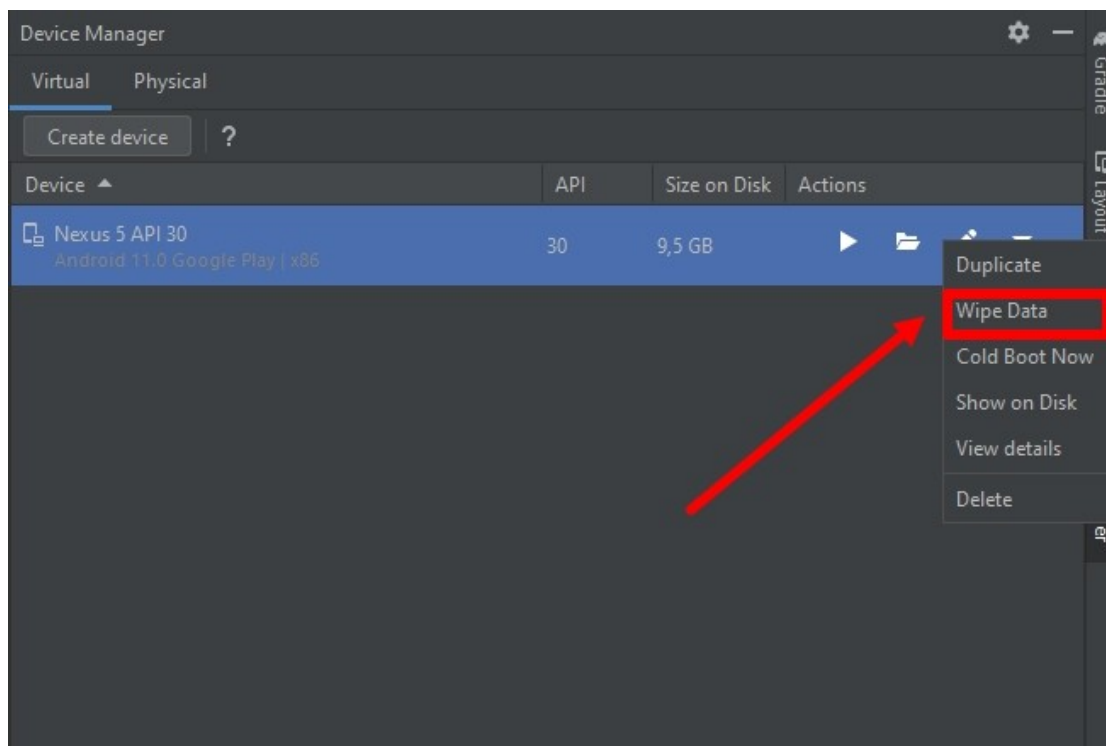


Рисунок 7 - Android Device Monitor.

Иногда помогает повторный запуск приложения. Если эмулятор не потерял связь с компьютером, то при повторной попытке приложение загрузится без проблем. Повторяю, что эта проблема возникает при первой загрузке эмулятора. Далее, во время работы не закрывайте эмулятор.

Тестируем на реальном устройстве

Если у вас есть Android устройство, то удобнее будет проводить отладку на нем, да и окончательную версию приложения желательно проверять на настоящем устройстве. Начиная с Android 4.4, на устройстве (планшет, телефон, часы) нужно активировать режим разработчика. Для этого нужно зайти в Настройки, там открываем страницу "О телефоне" и щёлкаем семь раз на строчке «Номер сборки». После этого в настройках появится новый пункт «Для разработчиков». Открыв страницу для разработчиков, вам нужно включить отладку через USB (рис. 8). Также для удобства можно включить опцию «Не выключать экран». Для некоторых устройств требуется установить отдельный драйвер, ищите информацию на сайте у производителей.

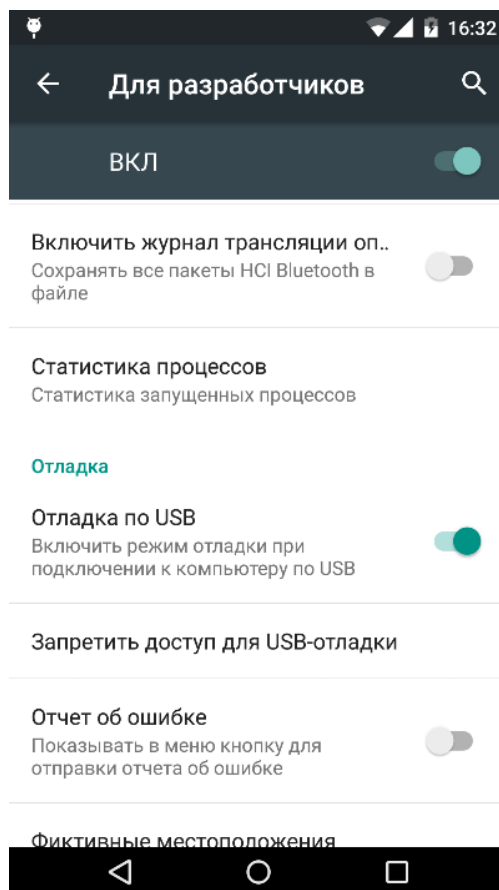


Рисунок 8 - Страница "Для разработчиков".

Лабораторная работа №3 Знакомство со средой разработки. Создание первого проекта.

После установки Android Studio (далее Студия) можно приступить к созданию своей первой программы.

В качестве языка программирования для Android используется Java. Для создания пользовательского интерфейса используется XML.

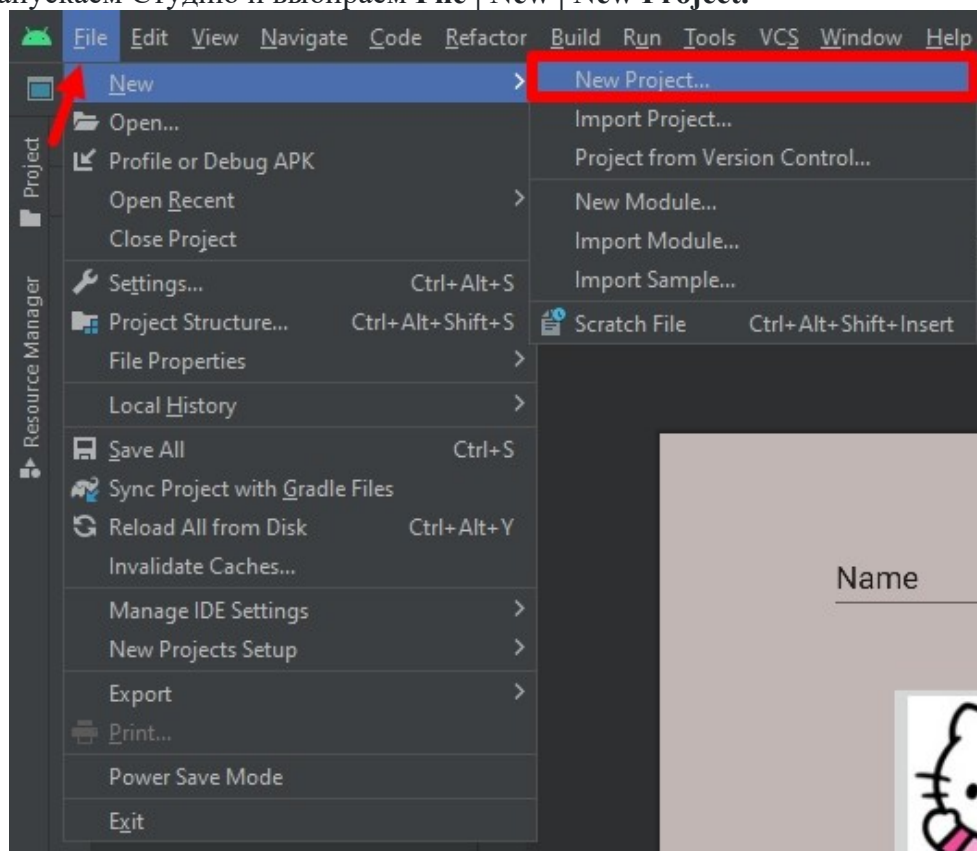
Здесь следует сделать небольшое отступление. В Android Studio 3.0 добавлена полноценная поддержка нового языка Kotlin, разработанная котанами. Google объявила о своих планах сделать новый "кошачий" язык основным. Но вы должны понимать, что за предыдущие годы было написано огромное количество примеров на Java. Если вы новичок в программировании, то лучше в первый период обучения полностью сосредоточиться на Java, вам будет проще находить ответы на вопросы. Kotlin от вас никуда не денется, перейти потом на него будет проще, а вот обратный процесс будет проходить тяжелее. Когда немного освоитесь в Java, то можете параллельно изучать примеры на Kotlin. Google сейчас активно переписывает документацию под Kotlin, но до полного перехода ещё далеко, даже меньше 50%.

По традиции, заложенной в прошлом веке, каждый программист должен был написать «Hello World!» (Здравствуй, Мир!) в качестве первой программы. Времена меняются, и программа «Hello World!» уже встроена в среду разработки под Android в целях совместимости, а современные программисты должны писать программу **Hello Kitty!** (Привет, киска!). Согласитесь, что здороваться с котёнком имеет больше здравого смысла, чем с каким-то миром.

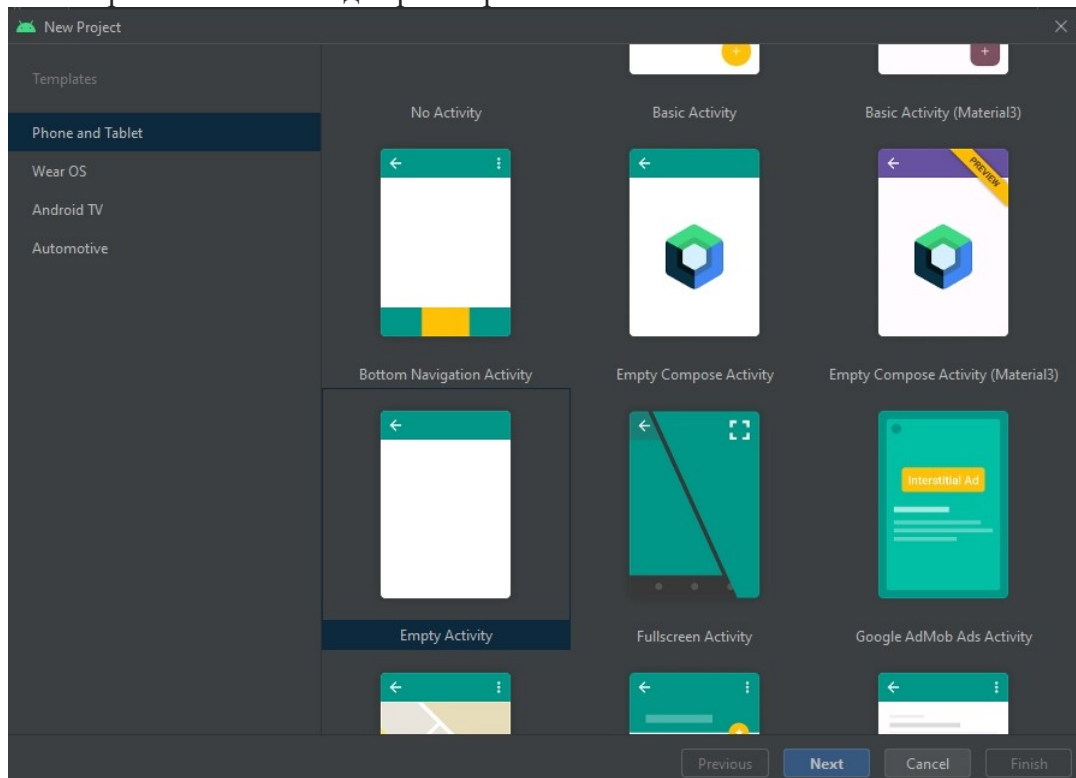
Поэтому разобьём задачу на две части. Сначала запустим готовую программу **Hello World!** без написания кода, чтобы убедиться, что весь инструментарий корректно установился, и мы можем создавать и отлаживать программы. А потом уже напишем свою первую программу.

Создание нового проекта

Запускаем Студию и выбираем **File | New | New Project**.



Выбираем внешний вид экрана приложения.



Предложенные шаблоны позволяют сэкономить время на написание стандартного кода для типичных ситуаций. Опытный разработчик может вручную написать любой из предложенных вариантов, используя вариант **No Activity**, где никаких заготовок не будет.

Несколько лет назад был только один шаблон. Список шаблонов постоянно меняется (добавляют и удаляют) и теперь их уже 13. Перечислю часть из них.

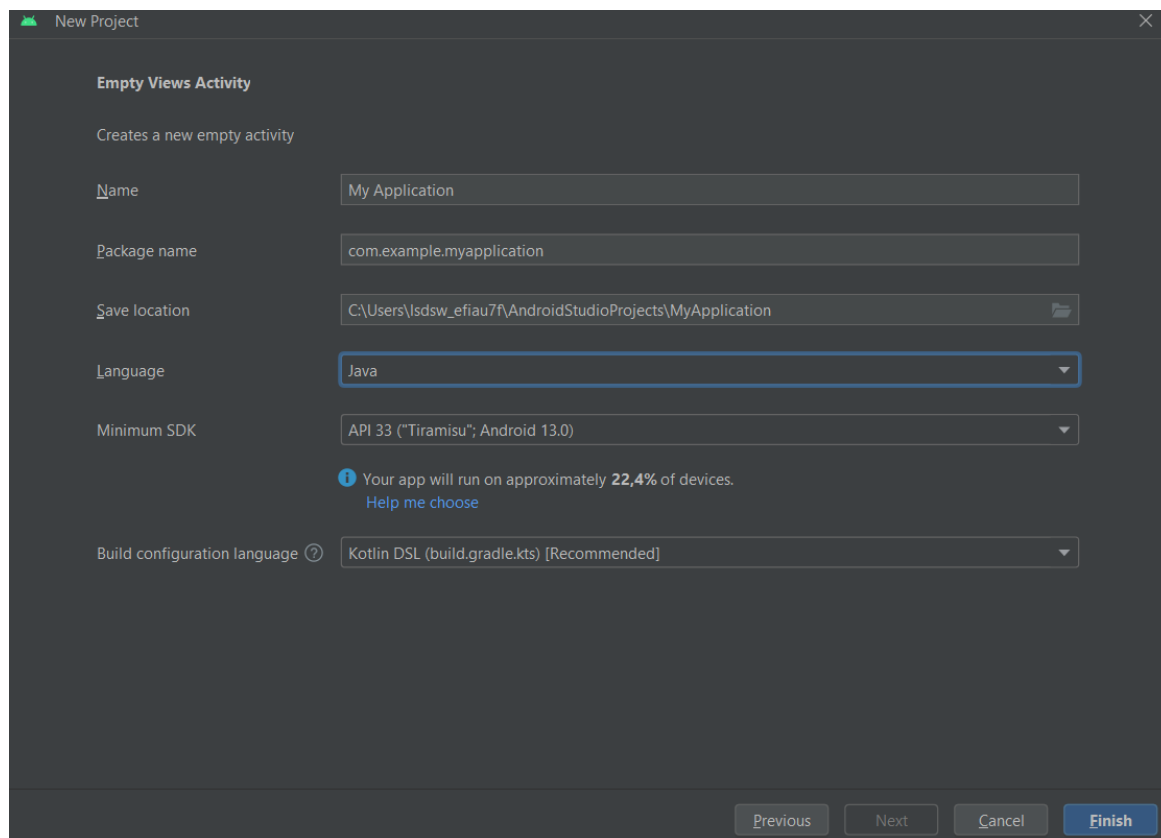
- **Basic Activity**
- **Bottom Navigation Activity**
- **Empty Activity**
- **Fullscreen Activity**
- **Google Maps Activity**
- **Google AdMob Ads Activity**
- **Login Activity**
- **Primary/Detail Flow**
- **Navigation Drawer Activity**
- **Scrolling Activity**

Шаблон **Empty Activity** предназначен для обычных телефонов. На картинке над названием шаблона вы видите приблизительный вид приложения с использованием данной заготовки. Для учебных программ в 99% подойдёт этот вариант. Практически все примеры на сайте написаны с помощью данного шаблона.

Шаблон **Primary/Detail Flow** предназначен для планшетов с реализацией двухпанельного режима. Шаблон **Fullscreen Activity** можно использовать для игр, когда требуется дополнительное пространство без лишних деталей. Другие шаблоны нужны для создания приложений с гуглокартами или сервисами Google Play.

В последних версиях Android Studio разработчики продвигают язык Kotlin, поэтому шаблон **Empty Activity** не поддерживает Java. Поэтому мы будем использовать **Empty Views Activity** во всех лабораторных.

Итак, мы выбрали вариант **Empty Views Activity** и переходим к следующему окну.



Поле **Name** - понятное имя для приложения, которое будет отображаться в заголовке приложения. По умолчанию у вас уже может быть **My Application**. Заменяем на **Hello World**. В принципе вы могли написать здесь и **Здравствуй, мир!**, но у Android есть замечательная возможность выводить нужные строки на телефонах с разными языками. Скажем, у американца на телефоне появится надпись на английском, а у русского - на русском. Поэтому в первоначальных настройках всегда используются английские варианты, а локализованные строки подготовите позже. Необходимо сразу вырабатывать привычку к правильному коду.

Поле **Save location** позволяет выбрать место на диске для создаваемого проекта. Вы можете создать на своём диске отдельную папку для своих проектов и хранить свои программы в ней. Студия запоминает последнюю папку и будет автоматически предлагать сохранение в ней. В случае необходимости вы можете задать другое местоположение для отдельного проекта через кнопку с изображением папки.

Поле **Package name** формирует специальный Java-пакет на основе вашего имени из предыдущего поля. В Java используется перевёрнутый вариант для наименования пакетов, поэтому сначала идёт **ru**, а потом уже название сайта. Пакет служит для уникальной идентификации вашего приложения, когда вы будете его распространять. Если сто человек напишет сто приложений с названием "Cat", то будет непонятно, где приложение, написанное разработчиком Василием Котовым. А приложение с именем пакета **ru.vaskakotov.cat** проще найти. Обратите внимание, что Гугл в своей документации использует пакет **com.example** в демонстрационных целях. Если вы будете просто копировать примеры из документации и в таком виде попытаетесь выложить в Google Play, то у вас ничего не выйдет - это название зарезервировано и запрещено к использованию в магазине приложений. Подготовленный вариант можно отредактировать. Например, вы пишете приложение на заказ и вам нужно использовать имя пакета, утверждённое заказчиком, а не ваш вариант по умолчанию.

Ниже представлены два варианта для написания программ на C++ и Kotlin. Эти варианты мы пока не рассматриваем. Когда вы будете писать на Kotlin, то ставьте

соответствующий флажок. Впрочем, вы можете сконвертировать проект с Java на Kotlin и позже средствами студии.

Поле **Minimum SDK** позволяет выбрать минимальную версию системы, под которую будет работать приложение. Оставьте тот который предлагает система.

ОЧЕНЬ ВАЖНО!! При установке Андроид Студии мы выбирали API 33 и 34, но при желании вы можете попробовать установить еще несколько версий, например, если у вашего телефона уровень платформы ниже. Это можно сделать через **Tools | SDK Manager** на панели сверху. Желательно, чтобы уровень платформы совпадал с уровнем проекта, который мы выбираем при создании нового проекта и с уровнем эмулятора/телефона, в противном случае, высока вероятность того, что приложение не запустится

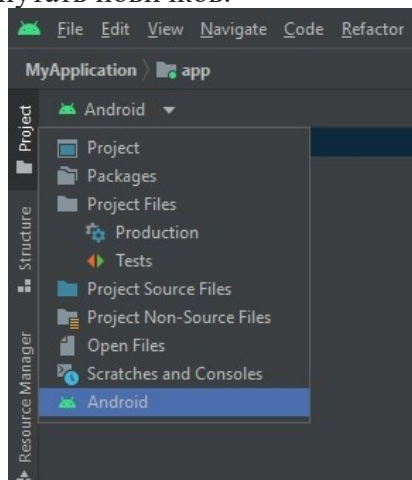
Если щёлкнуть по ссылке [Help me choose](#), то откроется окно с графиком. Если вам интересно, можете посмотреть, но котиков там нет.

Мы закончили с первоначальной настройкой. Нажимаем кнопку **Finish** и смотрим, что дальше будет.

А дальше студия формирует проект и создаёт необходимую структуру из различных файлов и папок. Поначалу глаза разбегаются. Давайте разбираться.

Боковая левая часть студии имеет несколько вертикальных вкладок. Скорее всего у вас будет активна первая вкладка **1: Project**. Вкладка **Structure** используется гораздо реже.

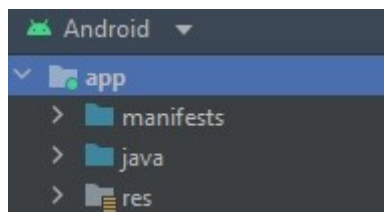
В левой части среды разработки на вкладке **Android** появится иерархический список из папок, которые относятся к проекту. В некоторых случаях желательно переключиться на режим **Project**, который показывает истинное расположение файлов. Но на первых порах удобнее использовать именно вид **Android**, который прячет служебные файлы, чтобы не путать новичков.



Содержание проекта

Вкладка **Android** содержит две основные папки: **app** и **Gradle Scripts**. Первая папка **app** является отдельным модулем для приложения и содержит все необходимые файлы приложения - код, ресурсы картинок и т.п. Вторая папка служит для различных настроек, управления проектом и многих других вещей.

Сейчас нас должна интересовать папка **app**. Раскройте её. В ней находятся три папки: **manifest**, **java**, **res**.



manifest

Папка **manifest** содержит единственный файл манифеста **AndroidManifest.xml**. В этом файле должны быть объявлены все активности, службы, приёмники и контент-провайдеры приложения. Также он должен содержать требуемые приложению разрешения. Например, если приложению требуется доступ к сети, это должно быть определено здесь. «AndroidManifest.xml» можно рассматривать, как описание для развертывания Android-приложения.

java

Папка **java** содержит три подпапки - рабочую и для тестов. Рабочая папка имеет название вашего пакета и содержит файлы классов. Сейчас там один класс **MainActivity**. Папки для тестов можете не трогать. Если вы знаете, как работают пакеты в Java, то можете создавать новые папки и подпапки.

res

Папка **res** содержит файлы ресурсов, разбитых на отдельные подпапки.

- **drawable** — в этих папках хранят графические ресурсы - картинки и xml-файлы, описывающие цвет и фигуры.
- **layout** — в данной папке содержатся xml-файлы, описывающие внешний вид форм и различных элементов форм. После создания проекта там уже имеется файл **activity_main.xml**, который отвечает за внешний вид главного окна приложения.
- **mipmap** - здесь хранят значки приложения под разные разрешения экрана
- **values** — тут размещаются строковые ресурсы, ресурсы цветов, тем, стилей и измерений, которые мы можем использовать в нашем проекте. Здесь вы можете видеть файлы **colors.xml**, **strings.xml**, **styles.xml**. В старых проектах был ещё файл **dimens.xml**, сейчас от него отказались

Со временем вы будет свободно ориентироваться в этих папках, пока не забивайте себе голову.

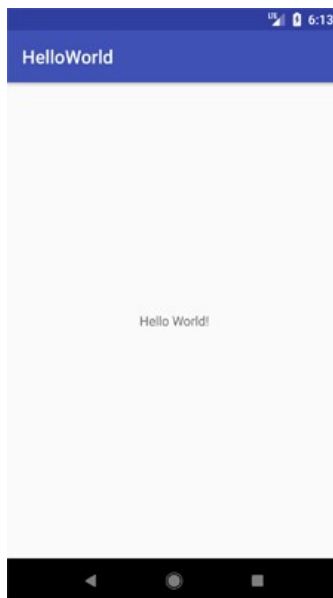
Работа с проектом - Здравствуй, Мир!

Как уже говорилось, программа **Hello, World!** уже встроена в любой новый проект, поэтому вам даже не нужно ничего писать. Просто нужно запустить проект и получить готовую программу!

Для изучения вам нужно открыть два файла - **MainActivity** (скорее всего он уже открыт) и **activity_main.xml (res/layout)** в центральной части Студии. Если файлы не открыты, то откройте их самостоятельно двойным щелчком для редактирования (или просмотра). Таким способом вы можете открыть любой нужный вам файл.

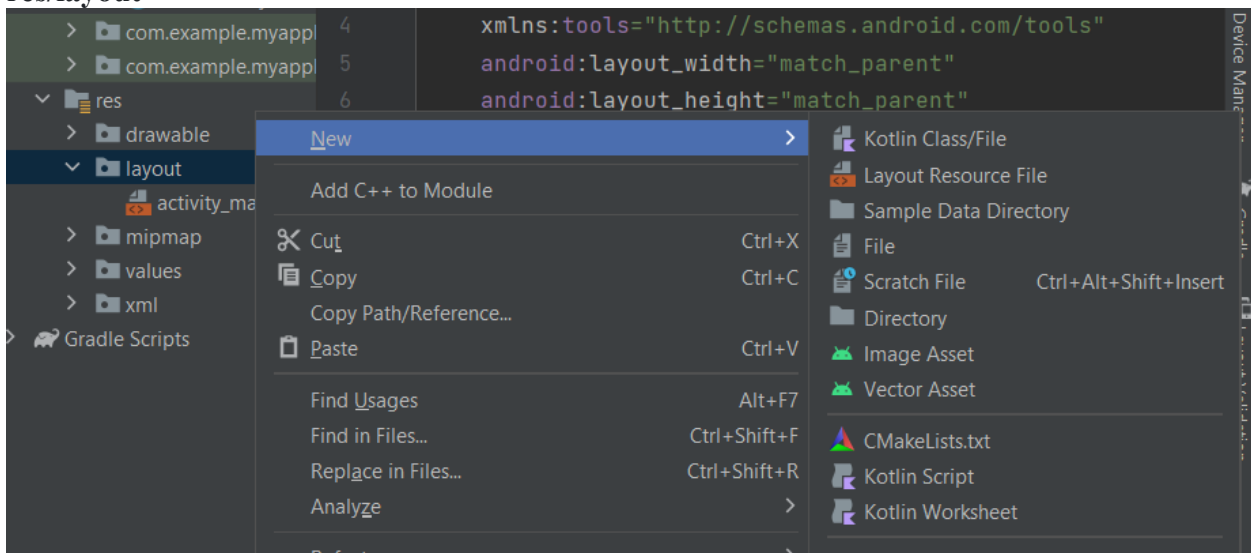
Не будем пока изучать код, а просто нажмём на зелёный треугольник **Run (Shift+F10)** на панели инструментов в верхней части студии для запуска приложения.

Если вы не настроили эмулятор, значит вы не читали предыдущий урок. Настройте сначала эмулятор и запускайте проект снова. Либо подключайте реальное устройство. Если всё сделали правильно, то в эмуляторе или на устройстве загрузится ваша программа. Поздравляю! Итак, если программа запустилась, то увидите окно приложения с надписью **Hello World**. Заголовок у программы будет также **Hello World**. Все эти строки можно найти в файле **res/values/strings.xml** и отредактировать при желании.



Теперь посмотрим на код. Сначала изучим **activity_main.xml**.

Если у вас его нет, что бывает очень часто, создайте Layout Resource File в **res/layout**



Лабораторная работа №4. Создание кода

Смотреть код можно в двух режимах - **Design** и **Code**.

Откройте в режиме **Code**.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="ru.alexanderklimov.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

```

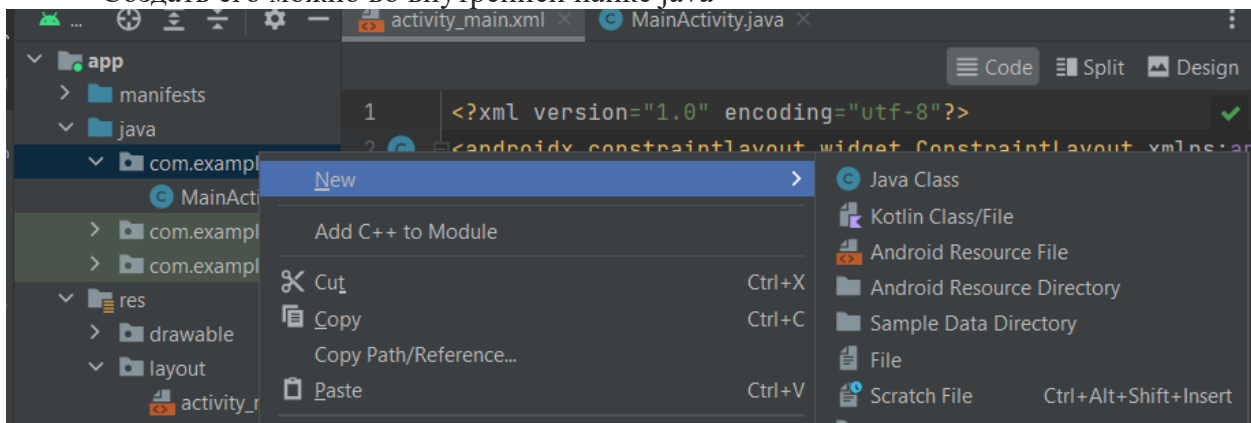
```
</android.support.constraint.ConstraintLayout>
```

Это новый код шаблона, который появился в Android Studio 2.3 в марте 2017 года. Раньше использовался другой код с **RelativeLayout** (а ещё раньше и другой код с **LinearLayout**). Если вам будут попадаться старые примеры, то в студии есть контекстное меню, которое поможет сконвертировать старый код в новый.

Немного о XML-коде. Имеется специальный контейнер **ConstraintLayout**, в котором размещён компонент **TextView**, предназначенный для вывода текста.

Теперь посмотрим на Java-код (**MainActivity.java**)

Создать его можно во внутренней папке java



```

package com.example.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Перед вами открыт файл класса, где имя класса **MainActivity** совпадает с именем файла с расширением **java** (это правило, установленное языком Java). В первой строке идет название пакета - его мы задавали при создании проекта (**Package Name**). Далее идут строки импорта необходимых классов для проекта. Для экономии места они свёрнуты в одну группу. Разверните её. Если однажды вы увидите, что имена классов выводятся серым цветом, значит они не используются в проекте (подсказка **Unused import statement**) и можете спокойно удалить лишние строки. Также они могут удаляться автоматически (настраивается).

Далее идёт объявление самого класса, который наследуется (**extends**) от абстрактного класса **Activity**. Это базовый класс для всех экранов приложения. Не

исключено, что у вас будет **AppCompatActivity**, если при создании проекта вы оставили поддержку старых устройств (флажок **Backwards Compatibility (App Compat)**). В старых версиях не было плюшек, которые появились после Android 4, поэтому для них была создана специальная библиотека совместимости, которая позволяет использовать новинки от новых версий Android в старых программах. Класс **AppCompatActivity** как раз и относится к библиотеке совместимости. Считайте её бедным родственником базовой **Activity**. У неё есть все нужные методы и вспомогательные классы, но названия могут немного различаться. И смешивать названия нельзя. Если уж используете класс из библиотеки совместимости, то методы берите соответствующие.

На разных этапах использовались разные названия класса активности, которые могут вам встретиться в старых проектах. Например, сначала использовался **FragmentActivity**, затем **ActionBarActivity**, а 22 апреля 2015 года вышла новая версия библиотеки совместимости и на данный момент используется новый класс **AppCompatActivity**.

В самом классе мы видим метод **onCreate()** – он вызывается, когда приложение создаёт и отображает разметку активности. Метод помечен как **protected** и сопровождается аннотацией **@Override** (переопределён из базового класса). Аннотация может пригодиться вам. Если вы сделаете опечатку в имени метода, то компилятор сможет предупредить вас, сообщив об отсутствии такого метода у родительского класса **Activity**.

Разберём код метода.

Строка **super.onCreate(savedInstanceState);** – это конструктор родительского класса, выполняющий необходимые операции для работы активности. Эту строчку вам не придётся трогать, оставляйте без изменений.

Вторая строчка **setContentView(R.layout.activity_main);** представляет большой интерес. Метод **setContentView(int)** подключает содержимое из файла разметки. В качестве аргумента мы указываем имя файла без расширения из папки **res/layout**. По умолчанию проект создаёт в нём файл **activity_main.xml**. Вы можете переименовать файл или создать свой файл с именем **cat.xml** и подключить его к своей активности. Тогда код будет выглядеть так:

```
setContentView(R.layout.cat);
```

Чтобы ваш код был аккуратным, старайтесь придерживаться стандартов. Если вы создаёте разметку для активности, то используйте префикс **activity_** для имени файла. Например, разметка для второй активности может иметь имя **activity_second.xml**.

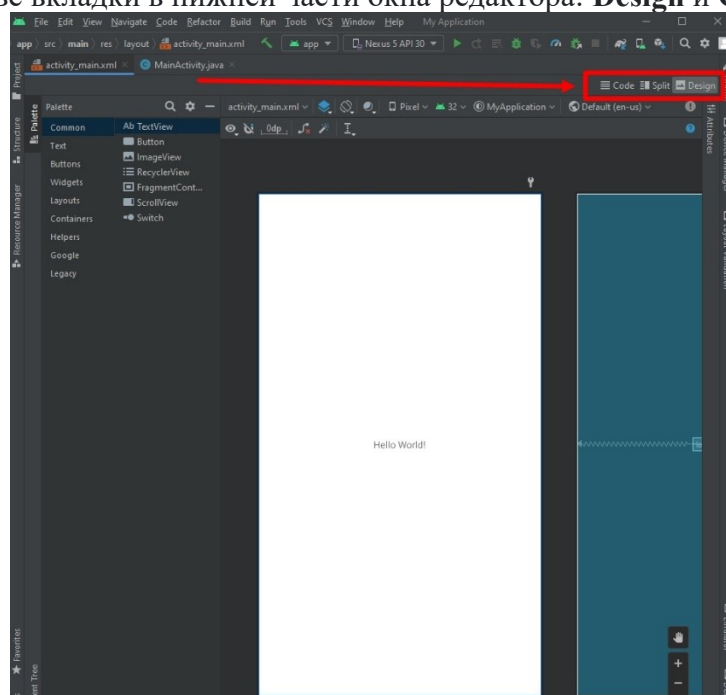
Hello Kitty!

Вы создали новую программу, но это ещё не повод считать себя программистом, так как вы не написали не единой строчки кода. Настало время набраться смелости и создать программу "Hello Kitty!". На данный момент наша программа слишком проста. Представьте себе, что у вас на экране должны располагаться несколько кнопок, текстовых полей, картинок. Каждому объекту нужно задать размеры, координаты, цвет, текст и так далее. Android поддерживает способ, основанный на XML-разметке, который будет напоминать разметку веб-страницы. Начинающие программисты могут использовать визуальный способ перетаскивания объектов с помощью мыши. Более продвинутые могут писать код вручную. Чаще используется комбинированный подход.

Файлы XML-разметки находятся в папке **res/layout** вашего проекта. Слово "res" является сокращением от слова "resources" (ресурсы). Папка содержит ресурсы, не связанные с кодом. Кроме разметки, там же содержатся изображения, звуки, строки для локализации и т.д.

Раскройте слева в структуре проектов папки **res/layout** и дважды щелкните на файле **activity_main.xml**, если он у вас закрыт. Обратите внимание, что XML-файлы

можно просматривать в двух режимах: текстовом и визуальном. Для этого предназначены две вкладки в нижней части окна редактора: **Design** и **Code**.

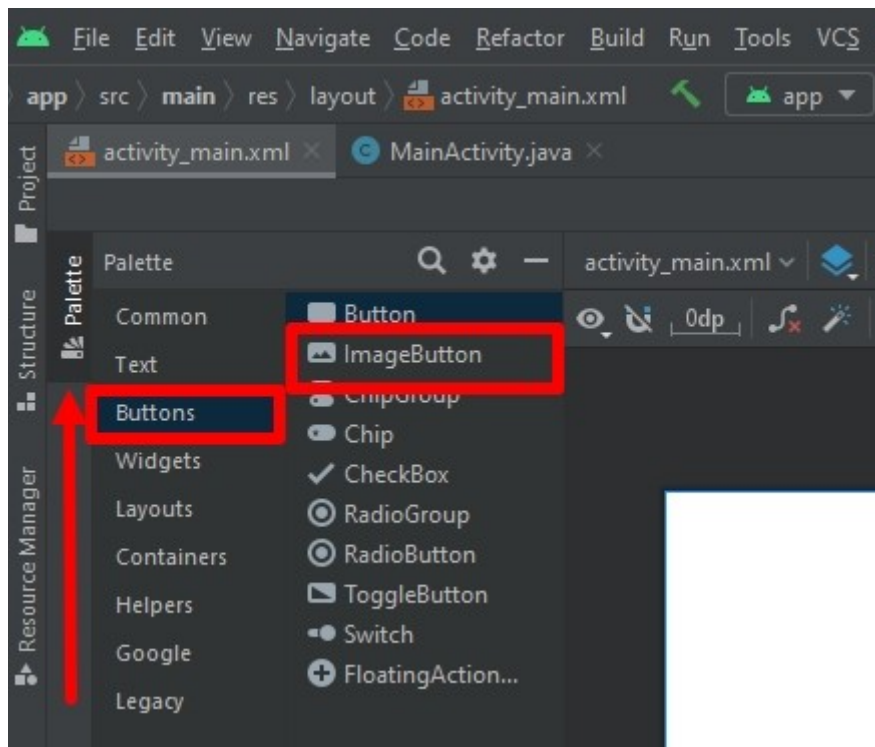


Переключитесь в режим **Code**.

Структура XML-файла достаточно проста - стандартное дерево XML-элементов, где каждый узел является именем класса **View** (**TextView** - один из элементов **View**). Вы можете создать интерфейс программы, используя структуру и синтаксис XML. Подобный подход позволяет разделить код программы и визуальное представление.

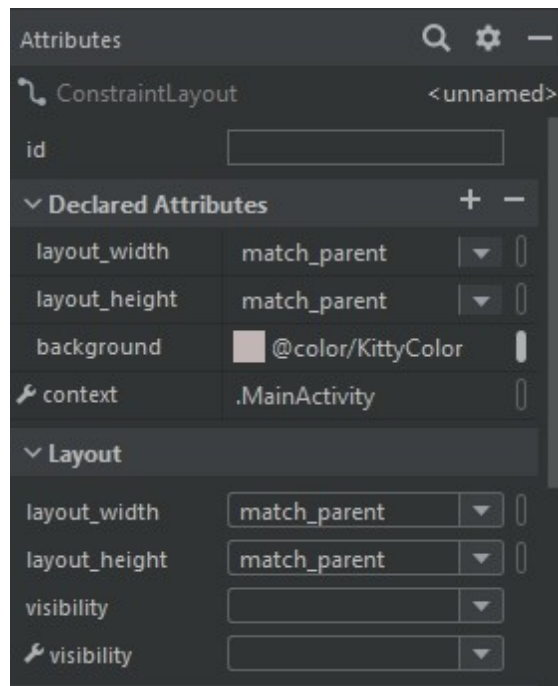
Можно продолжить работу над открытым проектом **Hello World** и модифицировать её под новые задачи. Для приобретения твёрдых навыков можете снова создать новый проект, не подглядывая в описание. Назовите проект **Hello Kitty** и повторите все предыдущие шаги.

Когда разметка открыта в графическом представлении, то слева от основной части редактора кода можно увидеть панель инструментов (**Palette**), в которой сгруппированы различные элементы по группам **Widgets**, **Texts**, **Layouts** и так далее.

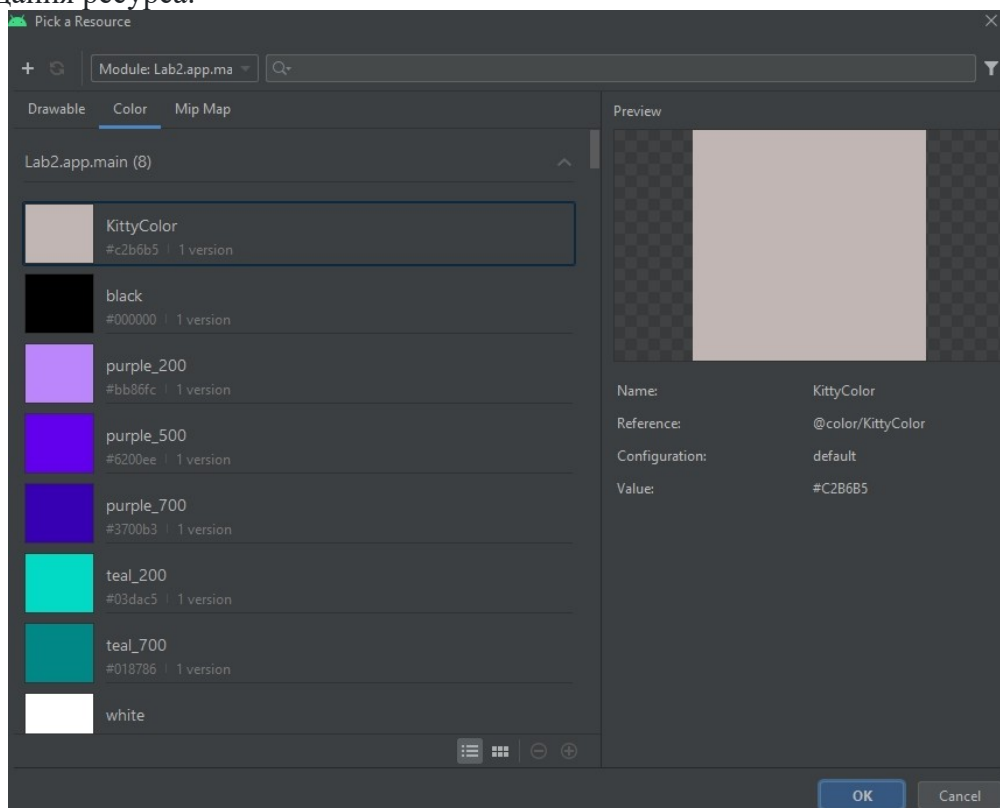


В группе **Buttons** найдите элемент **ImageButton**, перетащите её на форму и отпустите. Точное расположение нас не интересует, поэтому не заморачивайтесь по этому поводу, постарайтесь разместить компонент в центре экрана активности. У вас появится диалоговое окно с просьбой выбрать изображение для кнопки. Пока выбирайте любую, например, первую. Потом заменим.

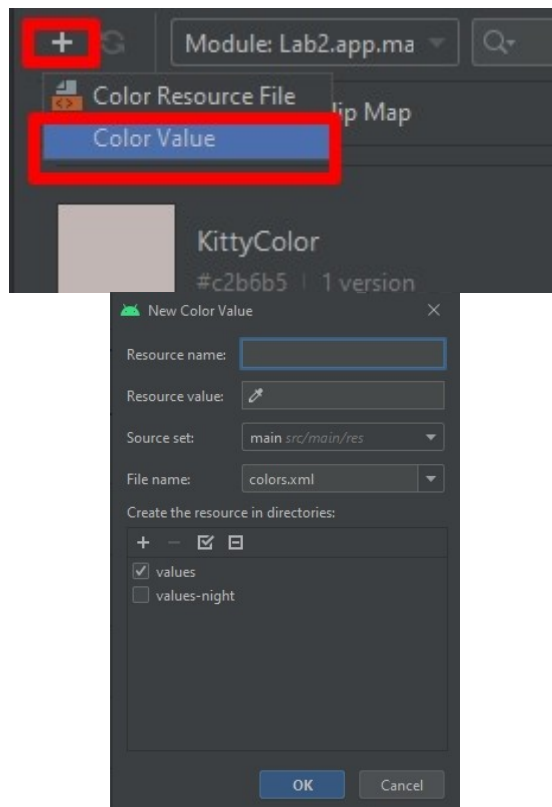
Теперь научимся менять фон для экрана приложения. Сейчас у нас экран белого цвета. Возвращаемся в файл разметки **activity_main.xml**. Справа найдите вкладку **Attributes**, в которой отображаются свойства для выбранного элемента. А слева есть вкладка **Component Tree**, который отображает структуру компонентов на экране. Вам нужно выделить какой-нибудь компонент, что на вкладке свойств увидеть все доступные свойства компонента. Новички часто путаются на первых порах и начинают менять свойства не у тех элементов, которые им были нужны. Сейчас у вас есть окно активности, графическая кнопка **ImageButton** и текстовая метка **TextView** с надписью **Hello World!**. Пощёлкайте по этим элементами, чтобы увидеть, как меняется содержание свойств в панели свойств. Так как мы собираемся работать с фоном экрана приложения, то щёлкните на **ConstraintLayout**. В панели свойств отобразятся самые употребительные свойства выбранного компонента. К ним относятся идентификатор, ширина и высота.



Найдите свойство **background**. Щелкните рядом с этим словом во второй колонке, где нужно прописывать значения. Появится текстовое поле, в которое можно ввести значение вручную, и кнопка с тремя точками, которая запустит диалоговое окно для создания ресурса.



Переходим на вкладку **Color** и выбираем цвет. Сверху в разделе **Lab.app.main** показаны цвета, определённые в проекте в стиле Material Design. В разделе **Android** показаны цвета, существующие в ресурсах системы. При желании можно установить свой цвет. Давайте создадим свой цвет и назовем его KittyColor



Экран окрасится в розовый цвет.

Если переключиться в текстовый режим, то увидим, что у элемента **ConstraintLayout** добавилась строчка:

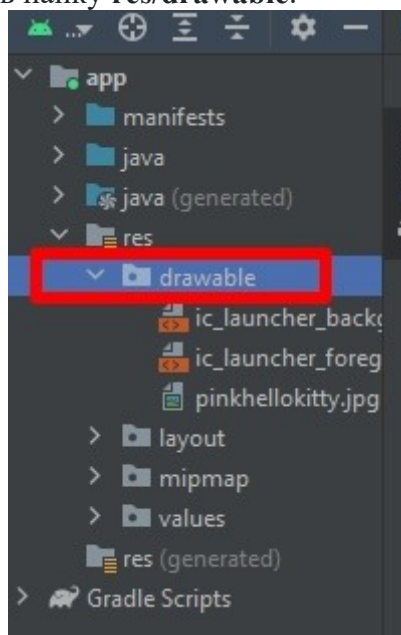
```
android:background="@color/colorAccent"
```

Мы связали фон экран с именем цветового ресурса. Существует неправильный подход, когда можно сразу напрямую указать значение цвета.

```
android:background="#ffffc0cb"
```

Старайтесь так не делать, всегда используйте ресурсы.

Далее поменяем картинку для графической кнопки. Находим подходящее изображение и копируем его в папку **res/drawable**.



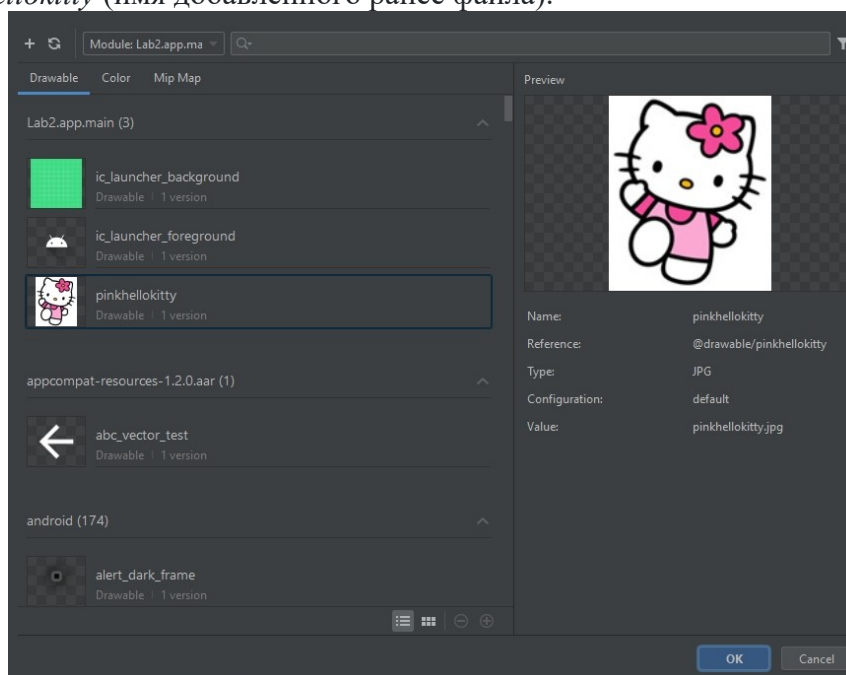
(Картинку можно сохранить из данного Word документа)



Простое перетаскивание из проводника в папку студии не работает. Поэтому лучше скопировать картинку в буфер, затем щёлкнуть правой кнопкой мыши на папке **drawable** в студии, выбрать команду "Вставить".

Когда вы поместите графический файл в указанную папку, то студия автоматически создаёт ресурс типа **Drawable** с именем файла без расширения, к которому можно обращаться программно.

Выделяем элемент **ImageButton** на форме и в панели свойств откроем только важные свойства, выбираем свойство **srcCompat**. Снова щёлкаем на кнопке с тремя точками и выбираем ресурс в категории **Drawable** - вы там должны увидеть ресурс *pinkhelloworldkitty* (имя добавленного ранее файла).



Запомните, что имена ресурсов должны начинаться с буквы и могут содержать буквы и цифры, а также знак нижнего подчеркивания. Другие символы типа тире, решётки и т.д. использовать нельзя.

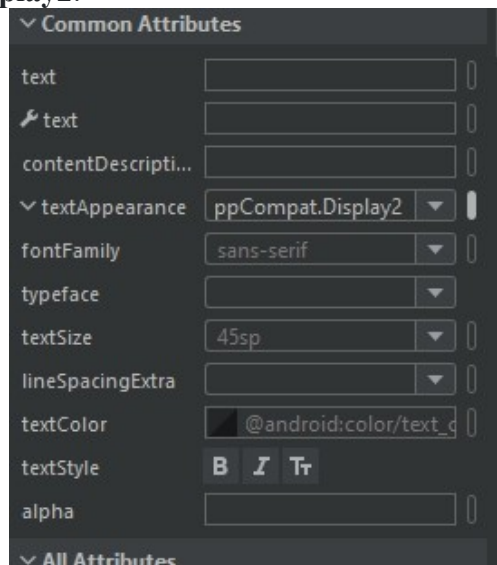
Там же в окне свойств находим свойство **onClick** и вручную прописываем **onClick** - это будет именем метода для обработки нажатия на кнопку. Вы можете придумать и другое имя, например, **onButtonPressed**, но я привык к такому подходу.

Мы закончили работу с графическим интерфейсом приложения. Напоследок, выделите элемент **TextView** с надписью **Hello, World!** в окне свойств посмотрите на его идентификатор (ID). Если там пусто, то удалите его, он не оправдал наших надежд. В разделе **Text** найдите компонент **TextView** и перетащите его на форму приложения. Постарайтесь разместить его под графической кнопкой с котёнком.

У этого компонента точно будет что-то написано в свойстве **id**. Скорее всего, это будет **textView**. Запомните его. Впрочем, мы могли не удалять первый компонент, а

прописать идентификатор вручную. Но мне пришлось бы объяснять лишние детали, а так сразу получили результат. Вот я не удалял его и у меня экран выглядит так. А у вас будет текст **TextView**. Ничего страшного.

Если текст вам кажется мелковатым, то у свойства **textAppearance** установите значение **AppCompat.Display2**.



Теперь перейдем в режим **Code** и посмотрим. У меня получилось следующее:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/KittyColor"
    android:foregroundGravity="bottom">

    <ImageButton
        android:id="@+id/imageButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="71dp"
        android:layout_marginTop="173dp"
        android:layout_marginEnd="71dp"
        android:layout_marginBottom="4dp"
        android:contentDescription="@android:string/unknownName"
        android:src="@drawable/pinkhelloworldkitty"
        app:layout_constraintBottom_toTopOf="@+id/textView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
```

```

        android:layout_marginBottom="156dp"
        android:fontFamily="sans-serif-light"
        android:text="@string/app_name"
        android:textAlignment="center"
        android:textAppearance="@style/TextAppearance.AppCompat.Display1"
        android:textColor="@color/black"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageButton" />

```

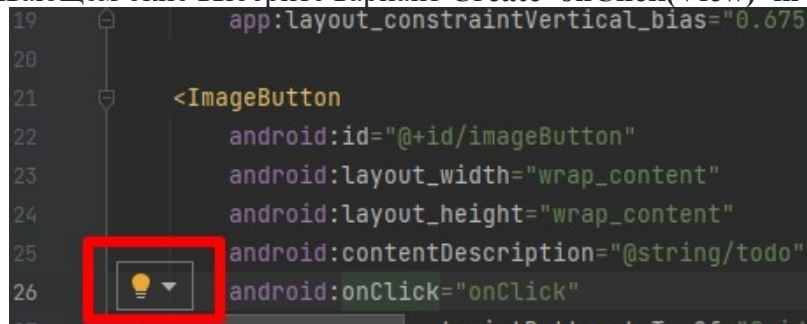
```
</android.support.constraint.ConstraintLayout>
```

Здесь очень много кода, который генерирует **ConstraintLayout** при размещении и выравнивании компонентов. Это новый контейнер, к которому нужно привыкать.

Установите курсор мыши внутри текста **"onClick"** у кнопки и нажмите комбинацию **Alt+Enter**



В всплывающем окне выберите вариант **Create 'onClick(View)' in 'MainActivity'**.



В коде класса **MainActivity** появится заготовка для обработки щелчка кнопки.

Раз уж у нас теперь открыт файл **MainActivity.java**, то продолжим теперь работу в нём. Так как мы собираемся менять текст в текстовой метке, необходимо прописать данный элемент в коде. До метода **onCreate()** наберите строчку:

```
private TextView mHelloTextView;
```

Мы объявили переменную типа **TextView** под именем **mHelloTextView**.

Если вы набирали вручную и при подсказках использовали клавишу Enter, то класс **TextView** автоматически импортируется и запись появится в секции **import**. Если вы просто копируете текст с сайта, то студия подчеркнёт название класса **TextView** и предложит импортировать его вручную.

Далее внутри метода **onCreate()** после вызова метода **setContentView()** добавьте строку:

```
mHelloTextView = (TextView)findViewById(R.id.textView); // помните, я просил  
запомнить идентификатор?
```

Избегайте соблазна скопировать строку с сайта и вставить в код, пишите самостоятельно и активно используйте автозавершение (Ctrl+Пробел) при наборе слов. Студия часто сама активно помогает подсказками. Теперь система знает о существовании элемента **TextView**, и мы можем к нему обращаться для изменения различных свойств, например, поменять текст.

Если вы используете компилятор **compileSdkVersion 26**, то скобки с названием компонента можно опустить, так как изменилась сигнатура метода.

```
// было  
public View findViewById(int id);  
  
// стало  
public <T extends View> T findViewById(int id);  
Пишите теперь так.
```

```
mHelloTextView = findViewById(R.id.textView);
```

Эта возможность появилась совсем недавно, поэтому во всех примерах используется старый код. Можете сразу привыкать к новому способу, студия будет подсказывать.

Переходим к заготовке для щелчка кнопки.

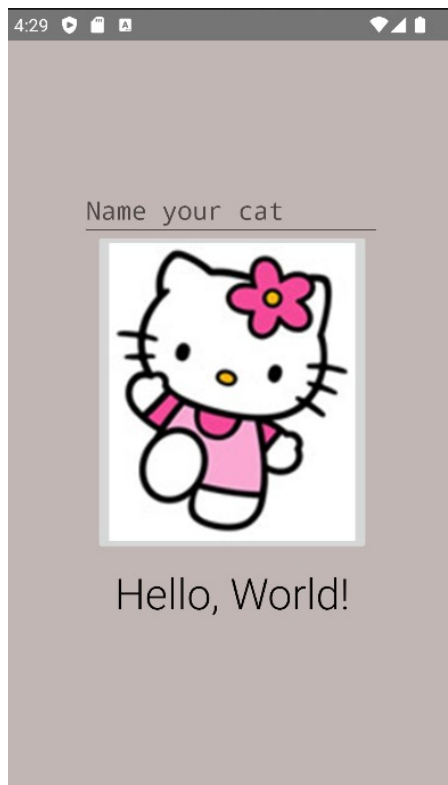
```
public void onClick(View view) {  
}
```

В следующих занятиях мы подробнее разберём работу с данным методом, пока просто пишем код между фигурными скобками:

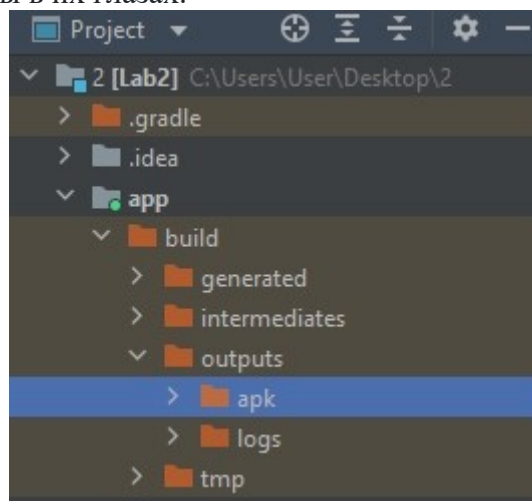
```
mHelloTextView.setText("Hello Kitty!");
```

Мы обращаемся к элементу **mHelloTextView** и через его метод **setText()** программно меняем текст на нужные слова.

Запускаем программу и нажимаем на кнопку с изображением котёнка. Если всё сделали правильно, то отобразится замечательная фраза. С этого момента можете считать себя настоящим программистом - вы научились создавать цветовые и графические ресурсы, менять фон у приложения через XML-разметку, обрабатывать нажатия кнопки и выводить текстовые сообщения.



В папке **app\build\outputs\apk** проекта можно найти готовый APK-файл, который вы можете выложить у себя на сайте и дать скачать знакомым девушкам и парням (в телефоне должно быть разрешение на установку неподписанных приложений), вы станете невероятно круты в их глазах.



Забегая вперёд, скажу одну вещь. Каждому приложению выделяется определённый объем памяти под картинки. В новых устройствах чуть больше, в старых поменьше. Но в любом случае не нужно пытаться загрузить в этом примере фотографию своего любимого кота объёмом в несколько десятков мегабайт, иначе можете получить ошибку в приложении. Позже вы узнаете, как лучше использовать большие картинки.

Здороваемся с вашим котом

Исходный код:

```
package ru.alexanderklimov.hellokitty;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
```

```

import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private TextView mHelloTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mHelloTextView = (TextView) findViewById(R.id.textview);
    }

    public void onClick(View view) {
        mHelloTextView.setText("Hello Kitty!");
    }
}

```

Программа получилась замечательная, но у неё есть недостаток. Она показывает одну и ту же фразу "Hello Kitty!". Вряд ли ваш кот знает английский, да и здороваться лучше по имени. Не пытайтесь с котом мяукать, иначе разговор выглядит следующим образом.

Поздороваемся с котом по человечески. Найдите в разделе **Text** компонент **Plain Text** и перетащите его на экран активности, разместив где-то над картинкой. Оставляем все свойства без изменений, разве только в свойстве **hint** можно добавить строчку-подсказку, которая будет исчезать при вводе текста.

```

<EditText
    android:id="@+id/editText"
    android:inputType="textPersonName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="74dp"
    android:layout_marginTop="122dp"
    android:layout_marginEnd="76dp"
    android:layout_marginBottom="4dp"
    android:ems="10"
    android:fontFamily="monospace"
    android:hint="@string/cat_name"
    android:textSize="20sp"
    app:layout_constraintBottom_toTopOf="@+id/imageButton"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

Переходим в класс **MainActivity** и добавляем новую переменную рядом с переменной **mHelloTextView**:

```
private EditText mNameEditText;
```

Свяжем созданную переменную с компонентом в методе **onCreate()**:

```
mNameEditText = (EditText) findViewById(R.id.editText);
```

Поменяем код для щелчка кнопки.

```
public void onClick(View view) {  
    if (mNameEditText.getText().length() == 0) {  
        mHelloTextView.setText("Hello Kitty!");  
    } else {  
        mHelloTextView.setText("Привет, " + mNameEditText.getText());  
    }  
}
```

Мы внесли небольшую проверку. Если в текстовом поле пустой текст, то длина текста составляет ноль символов, и мы по-прежнему выводим надпись "Hello Kitty!". Если пользователь введёт имя своего кота, то приложение поздоровается с ним.



Лабораторная работа №5. Обработка щелчка кнопки.

Щелчок кнопки/Счётчик ворон

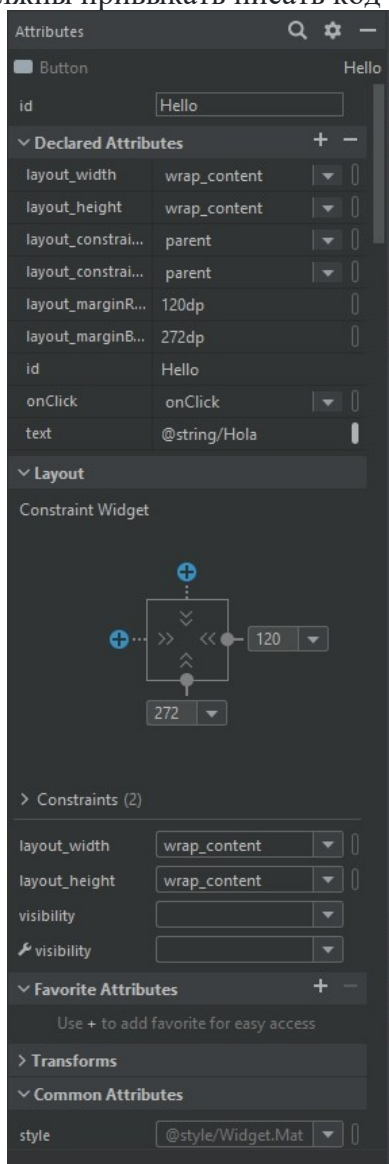


На предыдущем занятии мы немного поработали с кнопкой. Сегодня рассмотрим работу с обработкой щелчков подробнее.

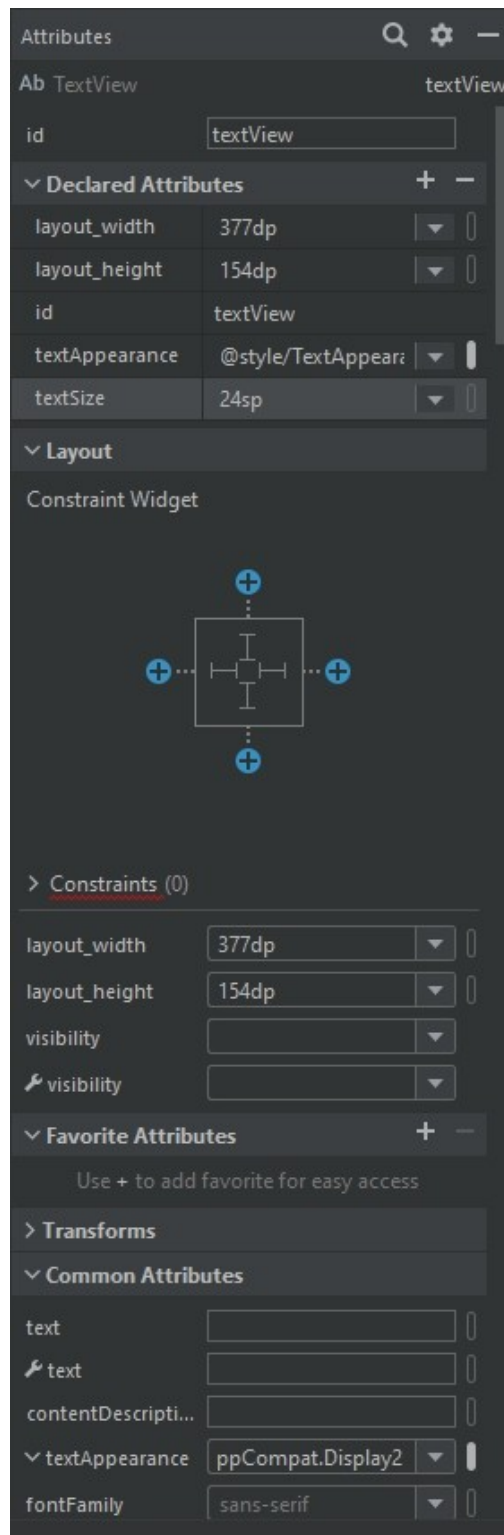
Можете использовать старый проект или создать новый проект с стандартными настройками. Для разнообразия в последнем окне мастера снимите флажок **Use legacy android.support.libraries** Мы не будем использовать никаких компонентов из

библиотеки совместимости, поэтому можем обойтись без обратной совместимости. Это позволит нам существенно снизить размер установочного файла.

В прошлый раз мы использовали **ImageButton**. Сегодня будем работать с обычной кнопкой. В режиме **Design** подведите курсор к компоненту **Button** и перетащите его на форму. В результате ваших действий на форме появится стандартная кнопка с настройками по умолчанию. В правой части экрана у вас имеется окно **Attributes**, в котором вы можете настроить необходимые свойства для кнопки. По умолчанию студия присвоит значение **button** атрибуту **ID**, который идёт самым первым. Если временно переключиться в режим **Text**, то увидите, что на самом деле атрибут имеет значение **@+id/button**. Если в вашем приложении будет много кнопок, то студия будет генерировать имена **button2**, **button3** и т.д. Не используйте подобные идентификаторы, а сразу давайте им говорящие имена - **button_name**, **button_food**. Атрибуту **Text** присвойте текст **Поздороваться**. Будет отлично, если вы добавите этот текст через ресурсы. В учебных примерах я буду использовать текст прямо в свойствах для экономии места, но вы должны привыкать писать код правильно.



Компонент **TextView** является текстовой меткой для вывода текста, который нельзя редактировать. В метке будем выводить приветствие после щелчка кнопки. В окне свойств удаляем текст из свойства **Text**, чтобы в текстовой метке ничего не было. Проследите, чтобы у него был идентификатор. Если в окне свойств ничего нет, то добавьте свой идентификатор, например, **textView**(в текстовом режиме **@+id/textView**).



Настройте взаимное расположение компонентов по своему желанию. Это обязательно для Constraint Layout, иначе проект показывает ошибку. Будем считать, что интерфейс программы готов - у нас есть кнопка для нажатия и текстовая метка для вывода сообщений.

Теперь нужно научиться писать код для щелчка кнопки. Переключитесь с режима **Design** на режим **Code** и найдите тег **<Button>**. Добавьте к нему еще параметров:

<Button

```

        android:id="@+id/hello_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="134dp"
        android:layout_marginTop="20dp"
        android:layout_marginEnd="134dp"
        android:layout_marginBottom="337dp"
        android:text="@string/action1"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2" /> />

```

На самом деле это можно было сделать и через графическую модель. Переключитесь обратно в режим **Design** и найдите в свойствах кнопки пункт **OnClick**. Там будет прописано имя метода, которое мы задали через XML (смотри рисунок выше). Пользуйтесь любым удобным вам способом.

Мы задали событие **onClick** для кнопки (щелчок) и теперь осталось написать обработчик события. Вы уже знаете, как в Android Studio быстро создать заготовку. В текстовом режиме устанавливаем курсор на строчке **android:onClick="onClick"** и нажимаем комбинацию **Alt+Enter** - студия сгенерирует шаблон метода в классе **MainActivity**.

```

public void onClick(View view) {

}

```

Вставим код.

```

public void onClick(View view){
    TextView helloTextView = (TextView)findViewById(R.id.textView);
    helloTextView.setText("Шатаут СПб!");
}

```

Этот код вам уже знаком по предыдущему занятию. Есть небольшое отличие. Текстовую метку мы объявляем и присваиваем ссылку на нужный компонент сразу в методе щелчка. Приложение у нас простое и доступ к текстовой метке больше нигде не осуществляется. Поэтому нет нужды объявлять переменную на уровне класса и инициализировать её в методе **onCreate()**. Если вы набирали текст самостоятельно, то у вас всё получится. Если просто скопировали и получили ошибки, то разбирайтесь сами. Я лентяям не помогаю.

Запускаем проект и нажимаем на кнопку, чтобы увидеть результат. Можете изменить текст по своему желанию, например, на *Ты кто такой? Давай, до свидания!*, но это сообщение будет выглядеть мягко говоря, странным.

Полный текст кода.

```

package ru.alexanderklimov.hellokitty;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

```

```

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        TextView helloTextView = findViewById(R.id.textview);
        helloTextView.setText("Шатаут СПб!");
    }
}

```

Ещё раз обратите внимание на строчку **TextView helloTextView = findViewById(R.id.textview);**.

Метод **findViewById()** переводится как **НайдиКомпонентПоИдентификатору** и применяет тот же подход - вы указываете в параметре идентификатор ресурса и программа ищет подходящий компонент. В дальнейшем вы постоянно будете использовать данный код в своих программах.



Новички на первых порах часто делают элементарную ошибку - помещают строчку инициализации объекта до метода **setContentView()**. Запомните, сначала нужно вывести шаблон (**R.layout.activity_main**), а только потом кнопки, текстовые поля, переключатели и т.д. В нашем случае используется отдельный метод для кнопки, который формируется после метода **setContentView()**. Позже, в других проектах вы увидите стандартные приёмы инициализации объектов.

Другой способ обработки щелчка кнопки

На самом деле, способ обработки щелчка кнопки, который я вам показал, является относительно новым для разработчиков. Google рекомендует использовать данный способ как удобный, требующий меньше кода и понятный для чтения. Но тем не менее вам придётся сталкиваться со старым способом, который берет свои корни от Java. Поэтому необходимо изучить и второй способ, чтобы понимать другие примеры.

Добавьте в проект новую кнопку и разместите под первой кнопкой. Подправьте несколько атрибутов, например, так.

```

<Button
    android:id="@+id/button_counter"
    ... другие атрибуты
    android:text="Счетчик" />

```

Я поменял для кнопки текст и идентификатор, а также некоторые параметры, отвечающие за положение кнопки на экране. Переключитесь в графический режим и посмотрите, как выглядит кнопка. Обратите внимание, что атрибут **android:onClick** у второй кнопки мы уже не используем.

Снова открываем файл **MainActivity.java** и пишем следующий код сразу после объявления класса и до метода **onCreate()**:

```
private Button mCounterButton;
```

Во время набора студия будет пытаться угадать, что вы хотите ввести. Активно используйте подсказки. Например, уже при первом вводе символа **B** студия предложит несколько вариантов на эту букву. Если нужное слово находится первым в списке, то нажимайте клавишу Enter, иначе выберите нужное слово из списка и потом уже нажимайте Enter. Это удобно при наборе длинных имён классов и переменных.

Google разработал целое руководство по наименованию переменных. Например, закрытая переменная на уровне класса должна начинаться с символа **m** (member), а далее идёт понятное название с заглавной буквы. Давайте попробуем придерживаться этого стиля.

После строчки **setContentView(R.layout.activity_main);** пишем:

```
mCounterButton = findViewById(R.id.button_counter);
```

Ещё раз напоминаю, не копируйте текст со страницы, а вводите текст вручную. Так вы быстрее освоите программирование, кроме того научитесь пользоваться системой подсказок и уменьшите количество опечаток.

Переходим к самому важному - обработчику щелчка кнопки. Нам понадобится дополнительная переменная-счётчик **mCount**, которая будет содержать число подсчитанных ворон (её необходимо разместить выше метода **onCreate()** рядом с переменной **mCrowsCounterButton**).

```
private int mCount = 0;
```

Мы создали переменную и сразу её инициализировали.

Теперь пишем обработчик щелчка кнопки, активно используя всплывающие подсказки для быстрого набора. Код добавляется в методе **onCreate()** после других строк, написанных ранее в этом методе.

Попробую описать, как использовать автодополнение. Сначала начинаем вводим первые символы слова **mCrowsCounterButton** (можно маленькими буквами) и нажимаем **Enter**, если видим, что нужная подсказка появилась. После этого слова ставим точку и должны появиться опять подсказки, которые относятся к данной переменной. Начинаем вводим первые буквы слова **setOnClickListener**. Здесь тоже проблем обычно не возникает. На данный момент у нас получилась строка **mCrowsCounterButton.setOnClickListener();**. Ставим курсор внутри круглых скобок и набираем **new OnClickListener**. Здесь важно набрать символ **O** в верхнем регистре. Тогда у вас появится нужная подсказка типа **View.OnClickListener{...}** (**android.view.View**). Нажимаем **Enter** и получаем нужную заготовку, внутри которой вставляем код:

```
mCrowsCounterButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        mCount += 1;  
        helloTextView.setText("Форсаж " + mCount);  
    }  
})
```

```
});
```

Ранее мы объявили переменную **helloTextView** внутри метода **onClick()**, из которого она недоступна в других методах. Поэтому поступим с ней так же, как с кнопкой - объявим текстовую метку на уровне класса и инициализируем её в методе **onCreate()**. Сделайте это самостоятельно, создав переменную с именем **mInfoTextView**.

Запускаем приложение и начинаем щёлкать по кнопке. При каждом щелчке счётчик **mCount** будет увеличиваться на единицу и эта информация будет отображаться на экране.

Нет смысла дублировать код для одной текстовой метки. Поэтому для первой кнопки, которая здоровалась, код можно сократить.

```
public void onClick(View view) {  
    mInfoTextView.setText("Hello Kitty!");  
}
```

Мы познакомились с новым способом обработки щелчка кнопки. Какой способ вы предпочтёте - зависит от ваших привычек и вкуса. Более того, есть ещё один способ, о котором можно прочитать в отдельной статье о кнопках.

Вдобавок, если подключить к проекту [Java 8](#), то вместо кода с подключением слушателя, можно использовать лямбда-выражение. Студия сама предложит заменить код.

```
mCounterButton.setOnClickListener(v ->  
    mInfoTextView.setText("Форсаж " + mCount));
```



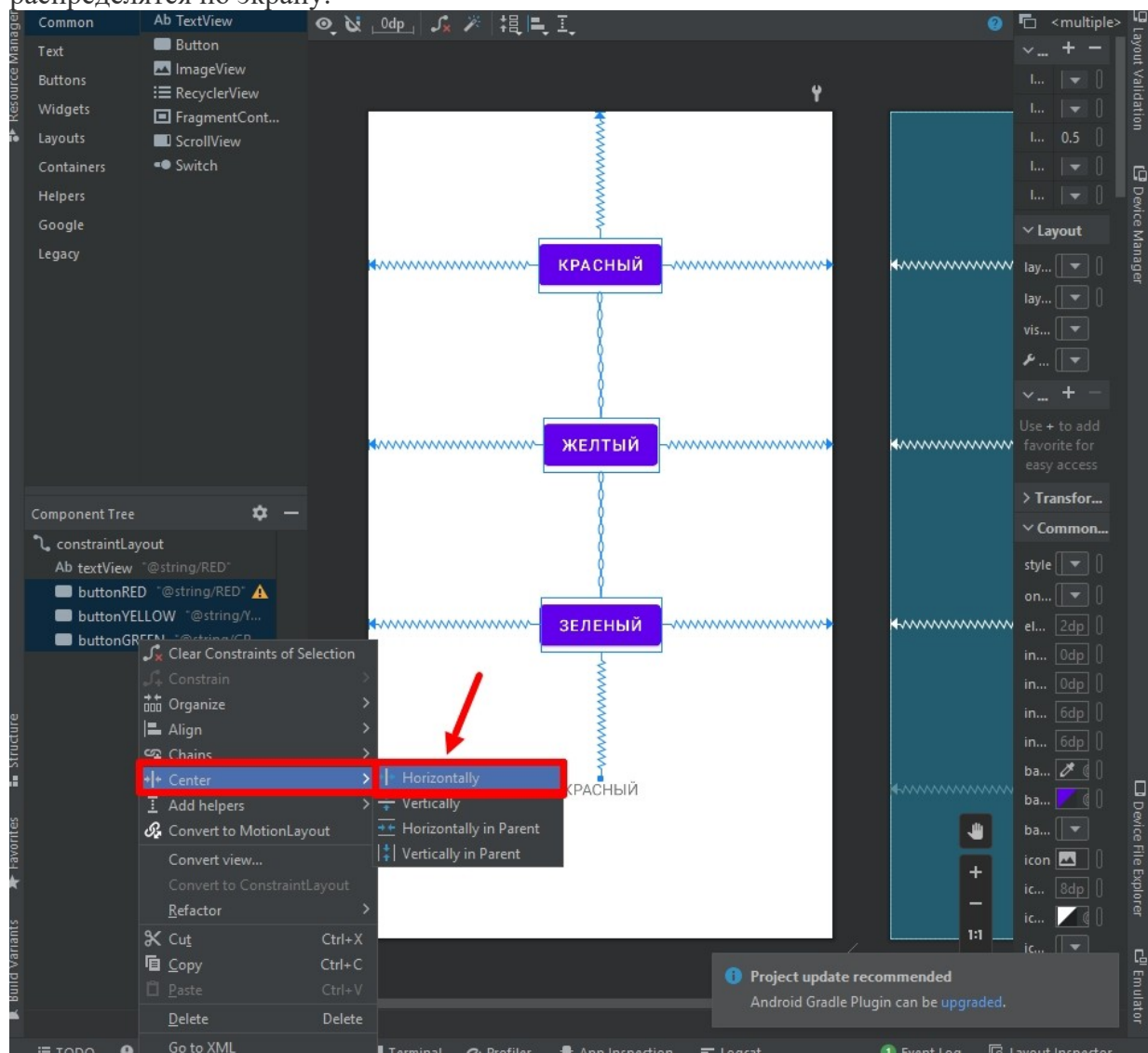
Лабораторная работа №6. Светофор.

Для закрепления навыков создадим приложение чуть посложнее предыдущего, а также познакомимся с другими приёмами программирования.

В качестве примера напишем программу под условным названием «Светофор». Интерфейс программы будет выглядеть следующим образом. На красном экране расположены три кнопки и одна текстовая надпись. При нажатии кнопок фон программы будет меняться на соответствующий цвет, который закреплён за определённой кнопкой. Я попробую вам показать решение задачи с разных сторон, чтобы вы почувствовали себя увереннее.

Первые шаги вполне очевидны. Создаём новый проект на основе "Hello, World" и перетаскиваем с панели инструментов три кнопки. Когда будете размещать кнопку, попробуйте потаскать её по разным позициям внутри экрана активности. Вы будете замечать различные всплывающие линии, подсказывающие о стандартных отступах от края или о центральной оси по вертикали и горизонтали. Это удобно, когда вы точно знаете, где нужно разместить один компонент.

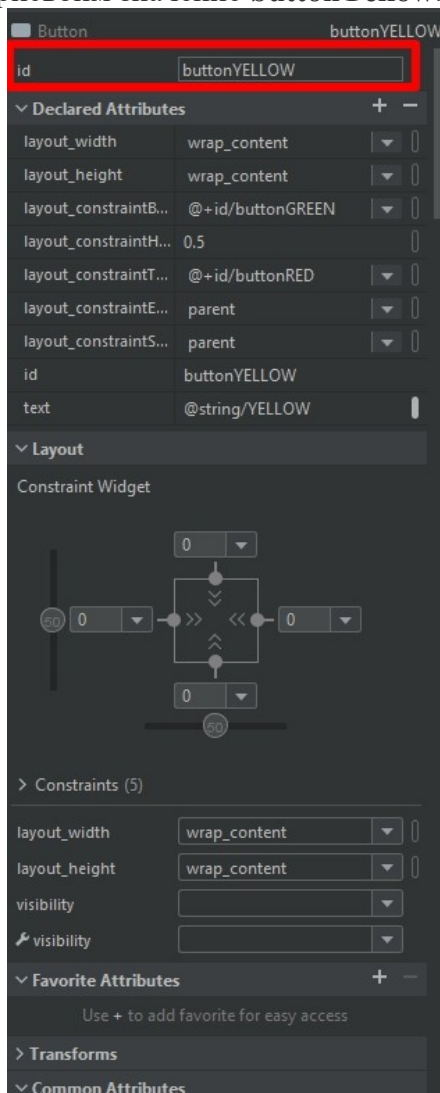
В нашем случае мы поступим иначе. Сначала просто перетащим три кнопки на экран, не думая о позиционировании. Далее в окне **Component Tree** выделяем три кнопки, удерживая клавишу **Shift**, вызываем контекстное меню и выбираем команды **Center - Horizontally** и потом **Center -Vertically**. Кнопки равномерно распределятся по экрану.



Контекстное меню можно вызвать и напрямую у кнопок на экране. Достаточно указателем мыши начертить ограничивающий прямоугольник вокруг всех кнопок и щёлкнуть правой кнопкой. Но у разметки **ConstraintLayout** наблюдается неприятное поведение - после выравнивания кнопок последующий щелчок кнопки сдвигает их. Если у вас такого эффекта не происходит, то пользуйтесь.

В окне **Component Tree** выделите строку **button**. У вас должно появиться окно свойств **Properties**. Давайте избавимся от стандартных идентификаторов, а будем сразу приучаться давать осмысленные имена. Например, для первой кнопки присвоим свойству **id** значение **buttonRed** вместо стандартного **@+id/button**

Для второй кнопки присвоим значение **buttonYellow**.

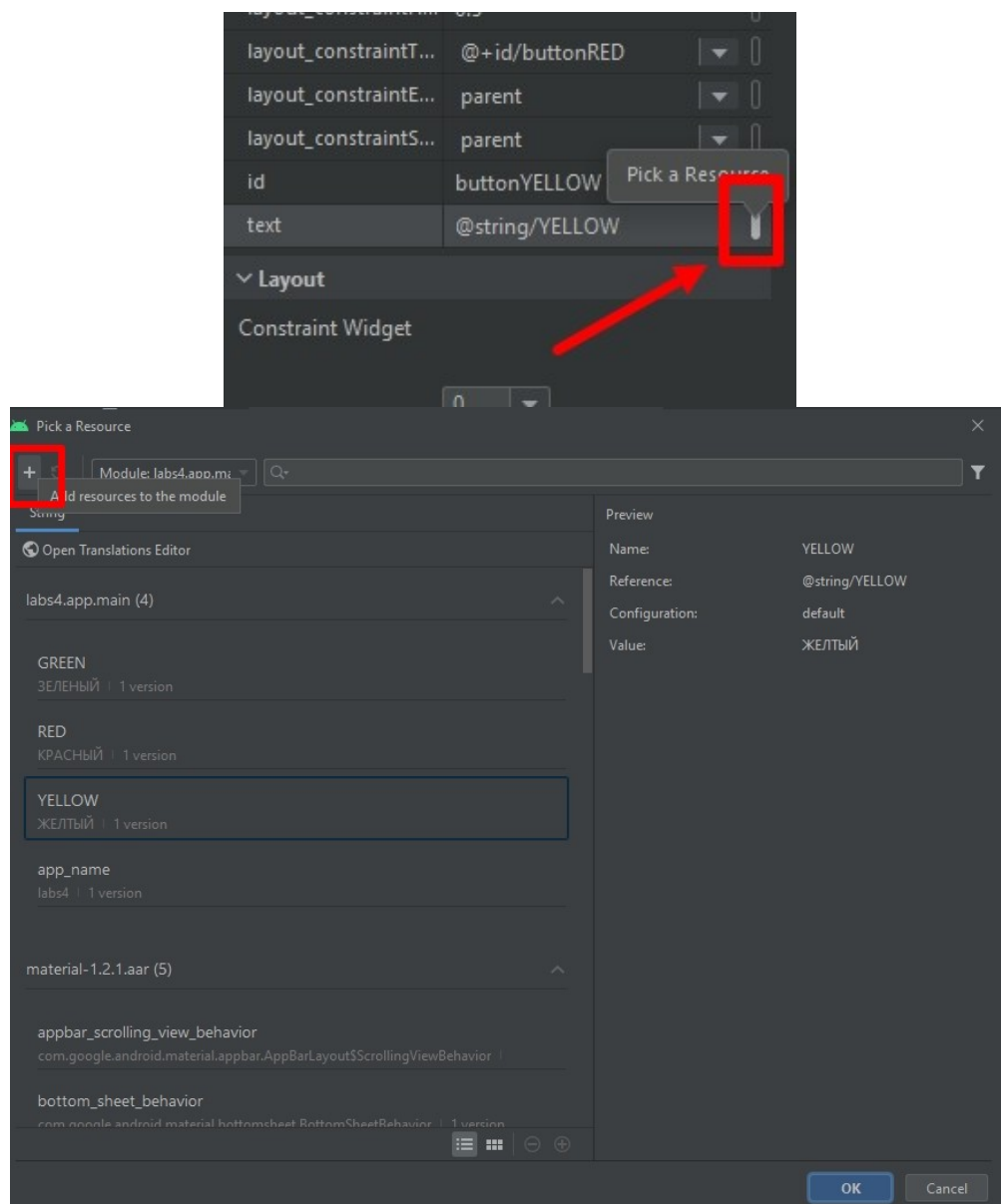


Аналогично настраиваем третью кнопку **buttonGreen**.

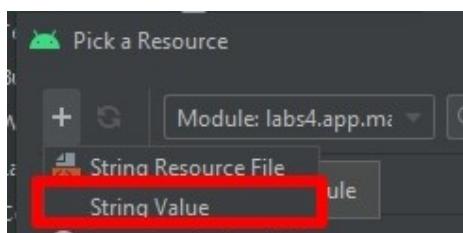
Лабораторная работа №7. Светофор. Строковые ресурсы.

Строковые ресурсы

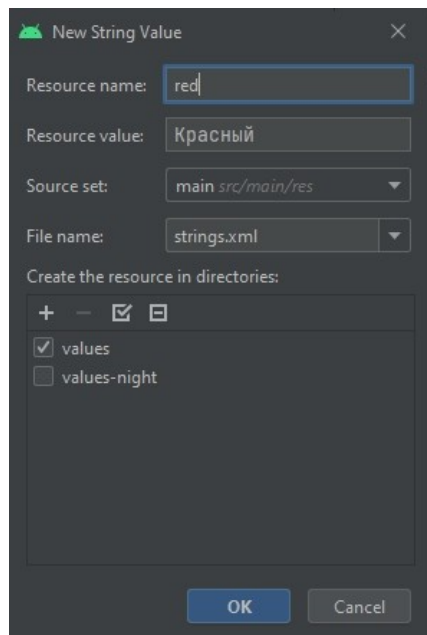
Теперь нам нужно заменить текст на кнопках на слова *Красный*, *Жёлтый* и *Зелёный*. На прошлом уроке мы просто присвоили свойству **Text** нужную строку. Но на самом деле это неправильный подход и даже среда разработки выводит предупреждающие значки у кнопок, если вы переключитесь в режим **Text**. По правилам, строки нужно хранить в строковых ресурсах. Подобный подход даёт разработчику множество преимуществ, в частности, быструю локализацию приложения. Считайте это стандартом, которого нужно придерживаться.



Нажмите на кнопку с изображением “+” для создания нового строкового ресурса и выберите **String Value**.



В новом окне **New String Value Resource** в первом поле **Resource Name** введите название ресурса, например, **red**, а во втором поле **Resource Value** введите текст для кнопки (напр. Красный). Остальные поля не трогаем. Аналогичным образом поступите с другими двумя кнопками (Жёлтый и Зелёный).



Программно можно добиться такого же результата, отредактировав файл **strings.xml**, который находится в папке **res/values** вашего проекта. Сейчас он может выглядеть так.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TrafficLight</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="red">Красный</string>
    <string name="yellow">Жёлтый</string>
    <string name="green">Зелёный</string>
</resources>
```

Мы совсем забыли про элемент **TextView**. Исправим упущение. Разместим компонент под кнопками и снова сделаем выравнивание.

Пусть на нём выводится текст, извещающий о текущем цвете фона приложения. Так как в ресурсах у нас уже есть слова *Красный*, *Жёлтый* и *Зелёный*, изначально предназначенные для кнопок, то мы не будем создавать новые строковые ресурсы, а воспользуемся готовыми наработками. По умолчанию у нас используется красный цвет.

В окне свойств выбираем свойство **text** для **TextView** и нажимаем кнопку  для вызова знакомого диалогового окна. На этот раз мы не будем щёлкать на кнопке **New Resource**, а сразу выберем строку **red**, которая, как мы помним, содержит текст **Красный** и щёлкнем кнопку **OK** (можно сделать сразу двойной щелчок на строке).

Заодно расскажу о другой настройке под ней с изображением гаечного ключа и таким же названием **text**. Представьте себе, что в реальной программе изначально у **TextView** нет никакого текста, он будет сформирован позже программно. Но чтобы разработчик видел, как будет выглядеть дизайн экрана, ему нужно видеть текст на этапе проектирования. Для этих целей и предназначена вторая настройка для текста. Подобное вы можете увидеть и у других свойства компонента.

С текстом разобрались. Для нахождения следующего, необходимого свойства, в окне свойств **Attributes** воспользуйтесь поиском. Я помню, что для размера шрифта используется что-то со словом **size**. Вспомнил, это **textSize**.

Выбираем нужное значение из выпадающего списка. Студия предлагает варианты, которые рекомендованы по правилам Material Design. Если вы хотите установить своё значение, то переключитесь в режим **Text** и вручную поменяйте значение. Если этого не сделать, а попытаться отредактировать значение в режиме **Design**, то студия не будет реагировать.

Рекомендую постоянно переключаться в режим **Text** и смотреть, что происходит в коде. Это позволит вам увереннее разбираться в коде и читать чужой код. Как правило, новички предпочитают работать через визуальные инструменты, а программисты с опытом самостоятельно пишут практически весь код. Нужно найти разумный баланс между двумя подходами. Всё придёт со временем.

Лабораторная работа №8. Цвет фона программы

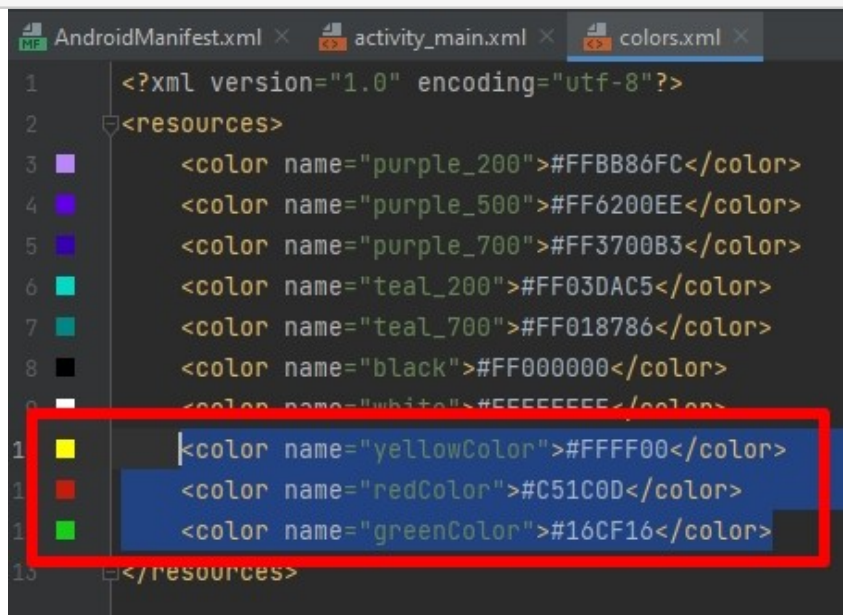
Со строками вроде разобрались. Давайте теперь в ресурсах зададим цвета для фона программы. Ресурсы для цветов принято хранить в отдельном файле **res/values/colors.xml**, хотя технически никто не запрещает хранить их в том же файле **strings.xml**.

Откроем указанный файл и добавим ресурс жёлтого цвета между тегами **resources**:

```
<color name="yellowColor">#FFFF00</color>
```

Слева появится жёлтый квадрат, по нему легко видеть цвет заданного ресурса. По такому же принципу добавьте зелёный цвет.

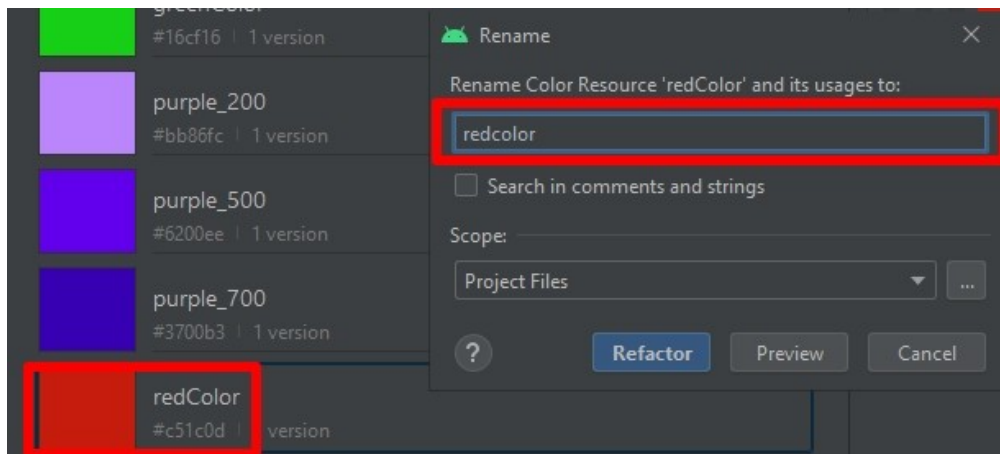
```
<color name="greenColor">#FF00FF00</color>
```



Самостоятельно добавьте ресурс для красного цвета под именем **redColor**.

Если вам знаком такой формат цвета, то трудностей добавление новых цветов подобным способом вам не доставит. Если нужно выбрать более сложный цвет, то проще воспользоваться мастером, как мы это делали в уроке "Hello Kitty", когда выбирали розовый цвет, а затем полученный цвет скопировать в ресурсы.

Определив в ресурсах все необходимые цвета, можно сразу присвоить красный цвет для контейнера **ConstraintLayout**. В окне свойств находим для данного элемента свойство **background**. Снова нажимаем на данную кнопку , чтобы открыть диалоговое окно. В окне выбираем раздел **Color** и ищем свой ресурс **redColor**.



Если посмотреть в текстовом режиме, то увидите строку **android:background="@color/redColor"** для тега **ConstraintLayout**.

Так как мы будем менять фон у **ConstraintLayout**, то присвоим ему идентификатор.

Общий каркас приложения завершен. У нас есть три кнопки с соответствующими текстами, текстовая надпись со словом *Красный*, и красный фон, используемый в контейнере **ConstraintLayout**. Пора приступить к программной логике программы. А пока можно запустить приложение, чтобы убедиться, что мы не сделали ошибок в разметке.

Код для программы

Наша задача - обработать щелчки трёх кнопок и менять цвет фона приложения, а также текст в **TextView**. На прошлом занятии мы уже познакомились с удобным способом обработки события **onClick**. Давайте закрепим пройденный материал и повторим тот же код для первой кнопки. Пропишем ручную событие **onClick** в теге **Button**:

```
android:onClick="onRedButtonClick"
```

Вспоминаем - в режиме **Text** помещаем курсор на названии метода и нажимаем комбинацию **Alt+Enter**, чтобы создать заготовку щелчка первой кнопки в классе **MainActivity**.

Объявим переменные в классе и получим к ним доступ в методе **onCreate()**:

```
// до метода onCreate()
private ConstraintLayout mConstraintLayout;
private TextView mInfoTextView;

// в методе onCreate()
mConstraintLayout = (ConstraintLayout) findViewById(R.id.constraintLayout);
mInfoTextView = (TextView) findViewById(R.id.textViewInfo);
```

Напоминаю, что код нужно писать вручную, а не копировать с сайта. Тогда многие вопросы отпадут и вам не придётся бежать на форум с криками, что ничего не работает.

Кстати, этот код вызвал трудности у новичков. Меня завалили письмами и устроили вой на форуме. Раньше в студии по умолчанию использовался компонент **RelativeLayout** и пример был написан под него. Новички размещали кнопки в **ConstraintLayout**, а в код вставляли строчки:

```
private RelativeLayout mRelativeLayout;
mRelativeLayout = (RelativeLayout) findViewById(R.id.relativeLayout);
```

Для не слишком догадливых программистов подчёркиваю - **ConstraintLayout** и **RelativeLayout** - это разные вещи. Вы же не путаете кота с собакой, хотя у них есть и хвосты и лапы. Не нужно писать код с использованием несуществующих компонентов, точнее не надо копировать и вставлять мой код. Иначе никогда не научитесь писать программы.

Пишем код для щелчка кнопки с надписью "Красный":

```
public void onRedButtonClick(View view) {
    mInfoTextView.setText(R.string.red);
    mConstraintLayout.setBackgroundColor(ContextCompat.getColor(this,
R.color.redColor));
}
```

Мы обращаемся к созданным ресурсам через специальный класс **R** и через точку указываем тип ресурсов, а затем имя ресурса.

Для кнопки "Зелёный" напишите код самостоятельно, добавив метод **onGreenButtonClick()**.

Для кнопки "Жёлтый" напомним код в традиционной манере через слушателя **OnClickListener**. В методе **onCreate()** добавляем:

```
Button yellowButton = (Button) findViewById(R.id.buttonYellow);
yellowButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mInfoTextView.setText(R.string.yellow);
        mConstraintLayout.setBackgroundColor(ContextCompat
            .getColor(MainActivity.this, R.color.yellowColor));
    }
});
```

Раньше для изменения кода использовался код:

```
mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.redColor));
```

В Android 6 (API 23) метод **getColor(int id)** объявили устаревшим и студия теперь подчёркивает данный метод. Можно заменить на один из двух вариантов:

```
// у второго параметра используем значение null
mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.yellowColor,
null));
// используем метод из библиотеки совместимости
mRelativeLayout.setBackgroundColor(ContextCompat.getColor(MainActivity.this,
R.color.yellowColor));
```

Первый вариант подойдёт, если вы не поддерживаете старые устройства, а сразу пишете приложение для телефонов с API 23. Второй способ более универсальный. Его я и использовал.

Запускаем приложение и щёлкаем по кнопкам - текст в надписи и фон в приложении должны меняться в соответствии с нажатой кнопкой.



Полный текст кода будет выглядеть следующим образом:

```
// Если этот код работает, его написал Александр Климов,  
// а если нет, то не знаю, кто его писал.  
package ru.alexanderklimov.trafficlight;  
import android.os.Bundle;  
import android.support.constraint.ConstraintLayout;  
import android.support.v4.content.ContextCompat;  
import android.support.v7.app.AppCompatActivity;  
import android.view.View;  
import android.widget.Button;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
  
    private ConstraintLayout mConstraintLayout;  
    private TextView mInfoTextView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mConstraintLayout = (ConstraintLayout) findViewById(R.id.constraintLayout);  
        mInfoTextView = (TextView) findViewById(R.id.textViewInfo);  
  
        Button yellowButton = (Button) findViewById(R.id.buttonYellow);  
        yellowButton.setOnClickListener(new View.OnClickListener() {  
            @Override
```

```

        public void onClick(View v) {
            mInfoTextView.setText(R.string.yellow);
            mConstraintLayout.setBackgroundColor(ContextCompat
                .getColor(MainActivity.this, R.color.yellowColor));
        }
    });
}

    public void onRedButtonClick(View view) {
        mInfoTextView.setText(R.string.red);
        mConstraintLayout.setBackgroundColor(ContextCompat.getColor(this,
R.color.redColor));
    }

    public void onGreenButtonClick(View view) {
        mInfoTextView.setText(R.string.green);
        mConstraintLayout.setBackgroundColor(ContextCompat.getColor(this,
R.color.greenColor));
    }
}

```

В качестве домашнего задания упростите код для трёх кнопок, создав для них общий метод **onClick()** (почитайте статью [про кнопки](#)).

Приведённая ниже информация немного устарела. В Android 5.0 используется другой стиль оформления заголовка программы и значка там теперь нет (хотя его можно туда поместить). Но значки всё равно нужны для отображения вашей программы на домашнем экране или в лаунчерах.

Поговорим о значках. По умолчанию студия использует изображение зелёного робота в качестве значка для вашей программы. Откройте в студии папку **res/mipmap**. Я уже рассказывал, что эта папка является виртуальной и в действительности существуют папки **res/mipmap-hdpi**, **res/mipmap-mdpi**, **res/mipmap-xhdpi**, **res/mipmap-xxhdpi**. В каждой из этих папок есть файл с одинаковым именем **ic_launcher.png** и недавно добавили ещё **ic_launcher_round.png** для Android 7. Вся разница между этими файлами заключается в размерах. В зависимости от разрешения экрана на устройстве система выбирает наиболее подходящий по размеру изображение и выводит его в качестве значка в заголовке приложения и на домашнем экране. Самый простой вариант заменить стандартное изображение на своё - создать своё изображение и заменить им имеющийся значок. Рекомендуется создавать под каждое разрешение свой значок. Причем здесь указаны не все варианты. В таком случае вам нужно создать самостоятельно папку, например, **mipmap-xxxhdpi** и разместить там картинку требуемого размера. Если вы пропустите какой-то размер, то система попытается взять какой-нибудь значок с этим именем из другой папки и масштабировать его. Но лучше так не делать.

Если вы не хотите менять существующие стандартные значки, а хотите использовать значки под другим именем, то в этом случае подготовьте все необходимые размеры, разместите их во всех папках **mipmap-** под своим именем, а затем в манифесте замените строчку у атрибута **android:icon**:

```

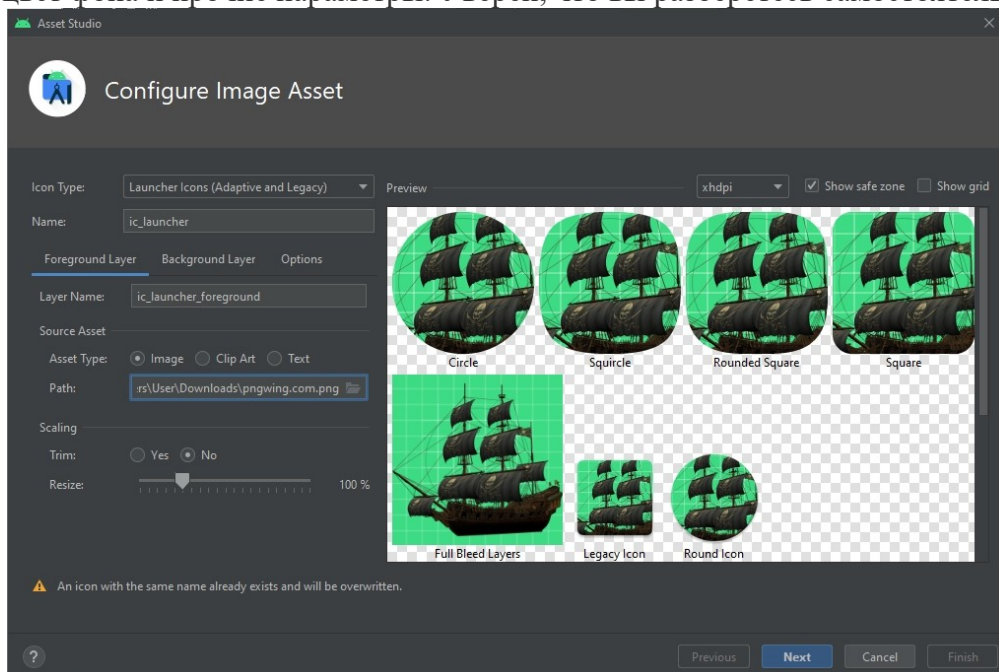
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher_cat"
    android:label="@string/app_name"

```

```
android:theme="@style/AppTheme" >
```

В состав студии входит набор предопределённых значков и генератор собственных значков. Щёлкните правой кнопкой на папке **drawable** или **mipmap** и выберите в меню **New | Image Asset**.

Откроется диалоговое окно, где вы можете указать в качестве источника файл на компьютере, вариант из клип-арта или набор символов. Также вы можете задать форму значка, цвет фона и прочие параметры. Уверен, что вы разберётесь самостоятельно.



Кроме значков для различных разрешений, генератор создаст дополнительный файл с суффиксом **-web**, который будет скопирован в папку **main**. Этот файл используется для Google Play, когда будете размещать приложение в магазине приложений и давать описание к нему.

Лабораторная работа №9. Переключение между экранами приложения. Ориентация экрана.

Простое переключение на другой экран.

Приложение не всегда состоит из одного экрана. Например, мы создали очень полезную программу и пользователю хочется узнать, кто же её автор. Он нажимает на кнопку «О программе» и попадает на новый экран, где находится полезная информация о версии программы, авторе, адресе сайта и т.д. Воспринимайте экран активности как веб-страницу с ссылкой на другую страницу. Если вы посмотрите на код в файле **MainActivity.java** из прошлых лабораторных работ, то увидите, что наш класс **MainActivity** тоже относится к **Activity** или, если говорить точнее, наследуется от него.

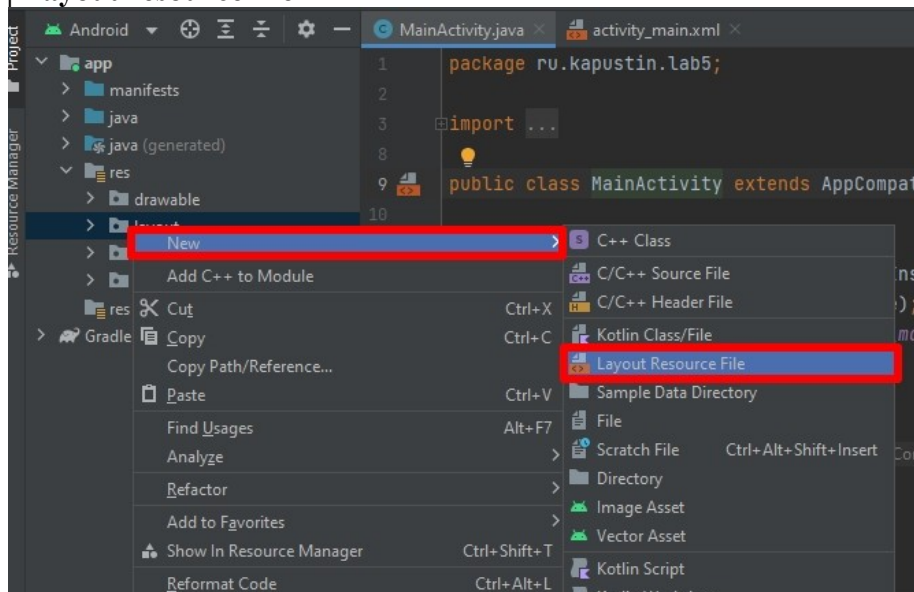
```
public class MainActivity extends AppCompatActivity
```

Для эксперимента создайте новый проект с одной кнопкой на экране. Далее создадим новую форму для отображения полезной информации. Например, покажем пользователю версию программы и имя автора.

Как нетрудно догадаться, нам следует создать новый класс, который может быть похож на **MainActivity** и затем как-то переключиться на него при нажатии кнопки.

Создавать новую активность будем вручную, хотя в Студии есть готовые шаблоны. Но там ничего сложного и для лучшего понимания полезно всё делать руками. Создадим новый XML-файл разметки **activity_about.xml** в папке **res/layout**.

Щёлкните правой кнопкой мыши на папке **layout** и выберите из контекстного меню **New | Layout resource file**.



Появится диалоговое окно. В первом поле вводим имя файла **activity_about**. Во втором нужно ввести корневой элемент. По умолчанию там стоит **ConstraintLayout**. Стираем текст и вводим **ScrollView**. Ввода нескольких символов достаточно, чтобы студия подсказала готовые варианты, можно сразу нажать Enter, не дожидаясь полного ввода слова (рис. 1).

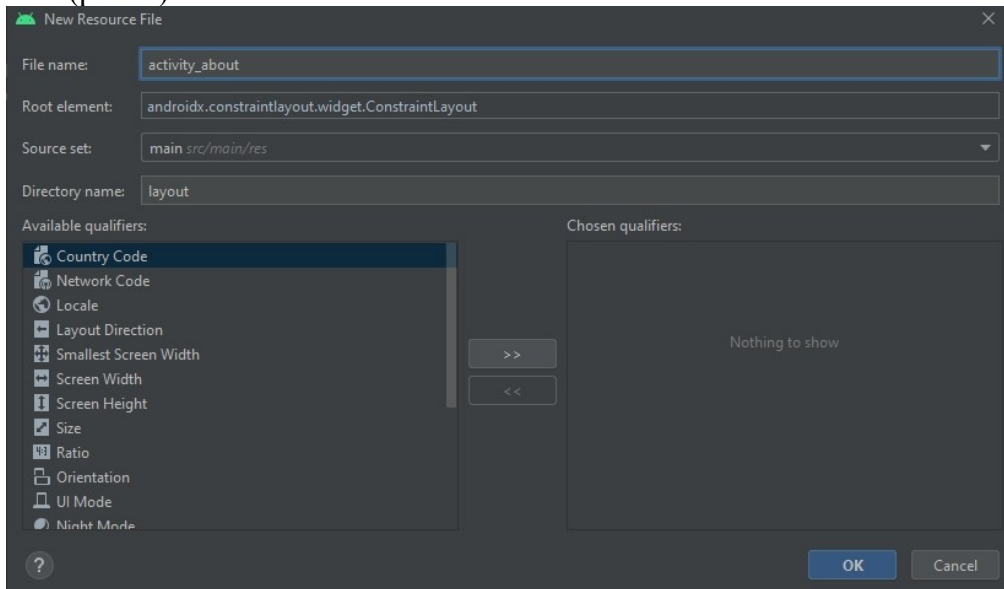


Рисунок 9 - Диалоговое окно добавления новой активности.

Получится соответствующая заготовка, в которую вставим элемент **TextView**.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
<TextView
    android:id="@+id/about_content"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/about_text"/>
```

```
</ScrollView>
```

Информация будет извлекаться из ресурсов, а именно из строкового ресурса **about_text**. Сейчас он подсвечен красным цветом, сигнализируя об отсутствии информации. Можно было нажать **Alt+Enter** и ввести текст в диалоговом окне. Но для нашего примера этот способ не подойдёт, так как наш текст будет многострочным, с использованием управляющих символов. Поэтому поступим по-другому. Откроем файл **res/values/strings.xml** и вводим следующий текст вручную:

```
<string name="about_text">
    Версия программы: <b>1.0</b>;\n
    Дата выхода: <b>10.10.2020</b>;\n
    Автор: <b>Имя Фамилия</b> </string>
```

Мы использовали простейшие HTML-теги форматирования текста типа ****, **<i>**, **<u>**. Для нашего примера достаточно выделить жирным ключевые слова. Для перевода текста на новую строку используйте символы **\n**. Добавим ещё один строковый ресурс для заголовка нового экрана:

```
<string name="about_title">О программе</string>
```

С разметкой разобрались. Далее необходимо создать класс для окна **AboutActivity.java**. Находим в директории **Android** - папку **Java**, разворачиваем ее, нажимаем правой кнопкой мыши на первую папку в списке – **New | Java Class**.

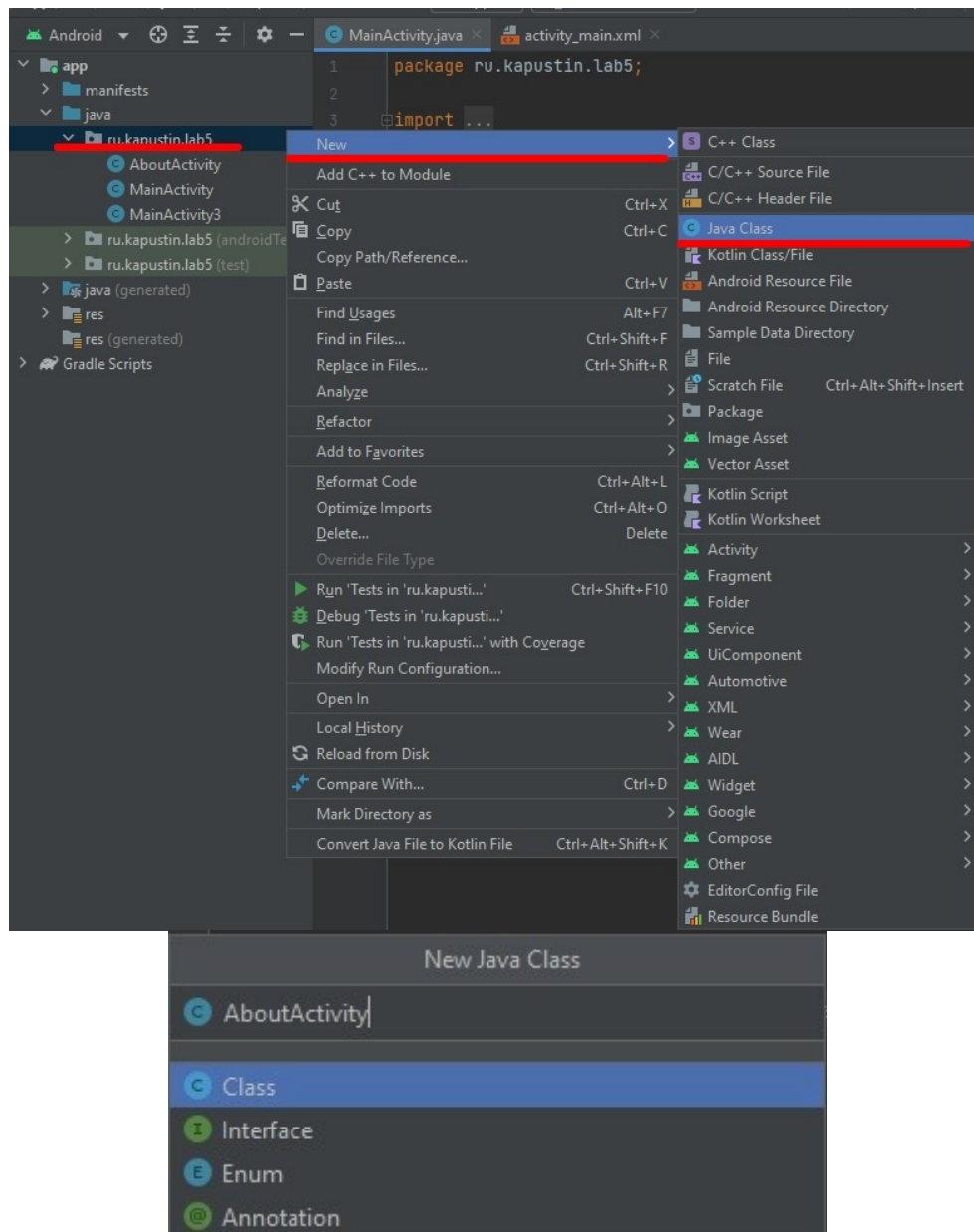


Рисунок 10 - Диалоговое окно создания нового класса.

Получим заготовку. Сейчас класс практически пустой. Добавим код вручную. Класс должен наследоваться от абстрактного класса **Activity** или его родственников типа **FragmentActivity**, **AppCompatActivity** и т.д. Дописываем **extends AppCompatActivity**. У класса активности должен быть метод **onCreate()**. Ставим курсор мыши внутри класса и выбираем в меню **Code | Override Methods** (Ctrl+O). В диалоговом окне ищем нужный класс, можно набирать на клавиатуре первые символы для быстрого поиска. В созданном методе нужно вызвать метод **setContentView()**, который подгрузит на экран подготовленную разметку. У нас получится такой вариант.

```
package com.google.myapplication;
import android.app.Activity;
import android.os.Bundle;

public class AboutActivity extends Activity {

    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_about);
}
}
```

Теперь начинается самое главное. Наша задача - перейти на новый экран по щелчку кнопки на первом экране. Переходим обратно к классу **MainActivity**. Напишем обработчик щелчка кнопки:

```
public void onClick(View view) {
    Intent intent = new Intent(MainActivity.this, AboutActivity.class);
    startActivity(intent);
}
```

Лабораторная работа № 10. Создание экземпляра класса.

Для запуска нового экрана необходимо создать экземпляр класса **Intent** и указать в первом параметре текущий класс, а во втором - класс для перехода, у нас это **AboutActivity**. После этого вызывается метод **startActivity()**, который и запускает новый экран.

Если вы сейчас попытаетесь проверить работу приложения в эмуляторе, то получите сообщение об ошибке. Что мы сделали неправильно? Мы пропустили один важный шаг. Необходимо зарегистрировать новый **Activity** в манифесте **AndroidManifest.xml**. Найдите этот файл в своем проекте и дважды щёлкните на нём. Откроется окно редактирования файла. Добавьте новый тег **<activity>** после закрывающего тега **</activity>** для первой активности. Печатайте самостоятельно и активно используйте подсказки. Получится следующее:

```
<activity android:name=".AboutActivity"
    android:label="@string/about_title">
</activity>
```

Вот и пригодился строковый ресурс **about_title**. Запускаем приложение, щёлкаем на кнопке и получаем окно **О программе**. Таким образом мы научились создавать новое окно и вызывать его по щелчку кнопки.

Ещё раз обращаю внимание, что второй создаваемый класс активности должен наследоваться от класса **Activity** или ему похожих (**ListActivity** и др.), иметь XML-файл разметки (если требуется) и быть прописан в манифесте.

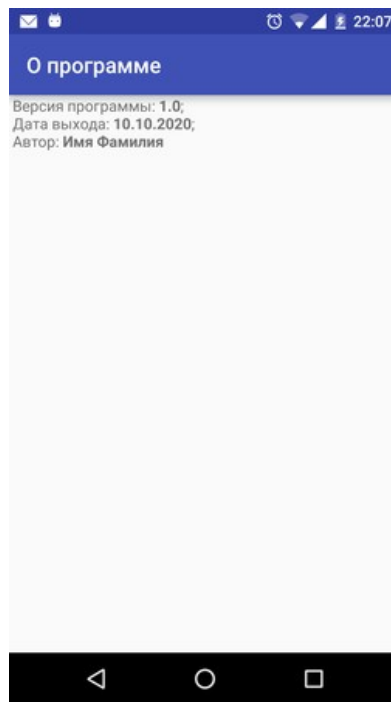
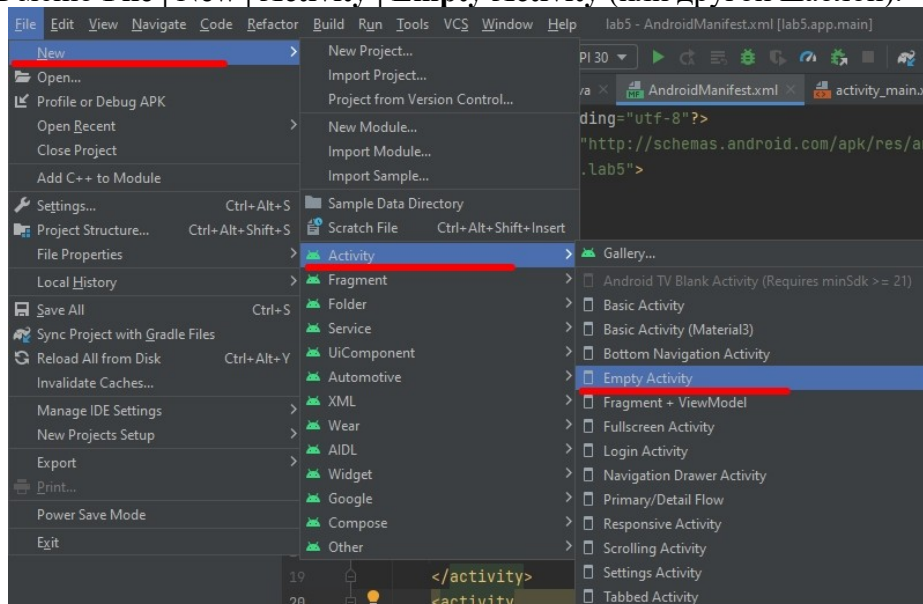


Рисунок 11 - Экран "О программе".

После вызова метода **startActivity()** запустится новая активность (в данном случае **AboutActivity**), она станет видимой и переместится на вершину стека, содержащего работающие компоненты. При вызове метода **finish()** из новой активности (или при нажатии аппаратной клавиши возврата) она будет закрыта и удалена из стека. Разработчик также может перемещаться к предыдущей (или к любой другой) активности, используя всё тот же метод **startActivity()**.

Создаём третий экран - способ для ленивых.

Есть более простой способ добавления новых активностей. В этом случае выберите в меню **File | New | Activity | Empty Activity** (или другой шаблон).



Дальше появится знакомое вам окно создания новой активности. Заполняем необходимые поля.

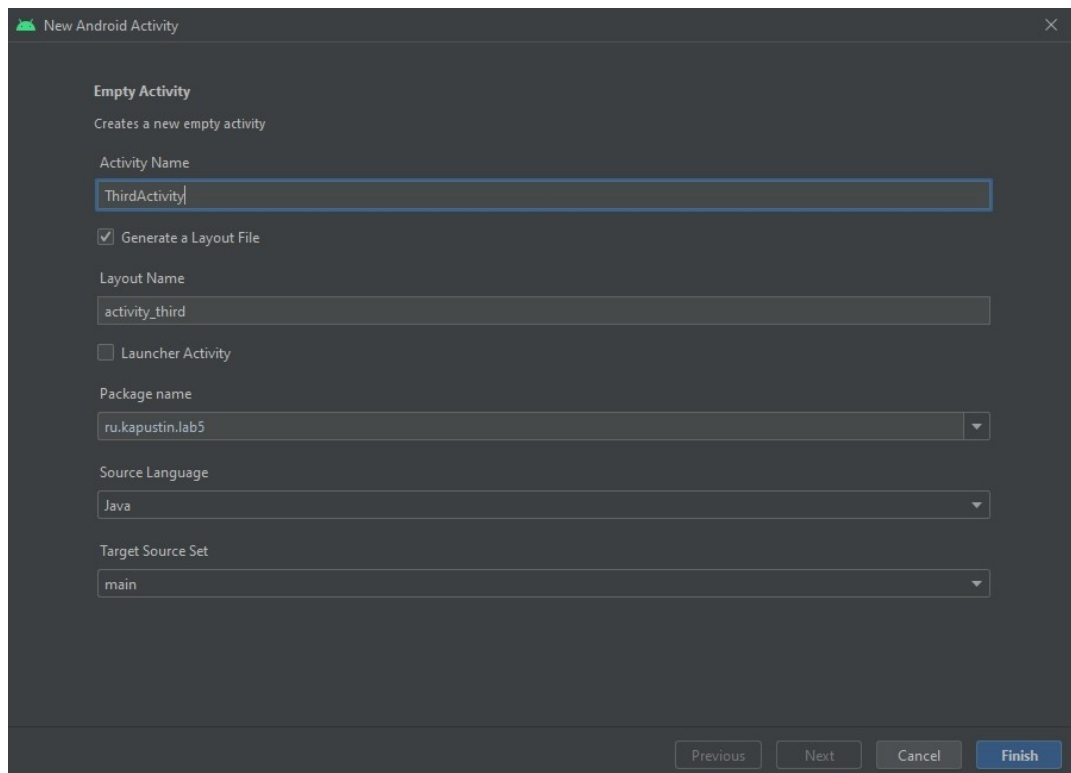


Рисунок 12 - Диалоговое окно создания активности.

Нажимаем на кнопку **Finish** и активность будет готова. Чтобы убедиться в этом, откройте файл манифеста и проверьте наличие новой записи, как и файлы класса и разметки.

Самостоятельно добавьте новую кнопку на экране главной активности и напишите код для перехода на созданную активность.

Ориентация экрана.

Добавьте на созданный экран шесть кнопок, как показано на рисунке 5. В результате файл разметки будет иметь следующий вид.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="30dp"
    android:paddingRight="30dp"
    android:paddingTop="30dp"
    android:paddingBottom="30dp"
    tools:context="com.google.myapplication.ThirdActivity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Первая кнопка"
        android:id="@+id/button1"
        android:layout_marginTop="75dp" />

    <Button
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:text="Вторая кнопка"
        android:id="@+id/button2"
        android:layout_below="@+id/button1" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Третья кнопка"
    android:id="@+id/button3"
    android:layout_below="@+id/button2"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Четвертая кнопка"
    android:id="@+id/button4"
    android:layout_below="@+id/button3"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Пятая кнопка"
    android:id="@+id/button5"
    android:layout_below="@+id/button4"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Шестая кнопка"
    android:id="@+id/button6"
    android:layout_below="@+id/button5" />
</RelativeLayout>

```



Рисунок 13 - Третья активность.

Вроде всё нормально. Но стоит нам повернуть устройство на 90 градусов (для эмулятора нужно нажать комбинацию клавиш Ctrl+F11), как сразу обнаруживаются проблемы. Пятая и шестая кнопки оказались за пределами видимости (рис. 6). Непорядок!

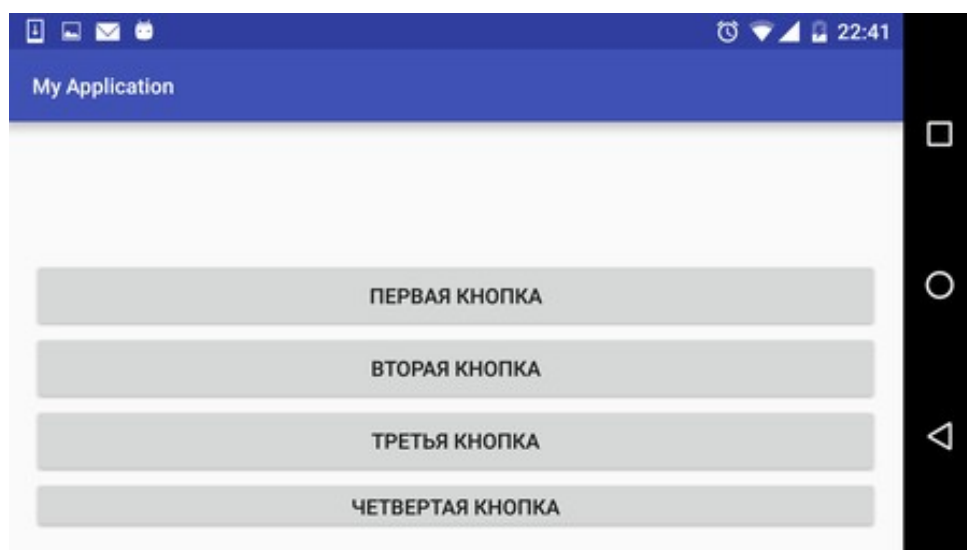
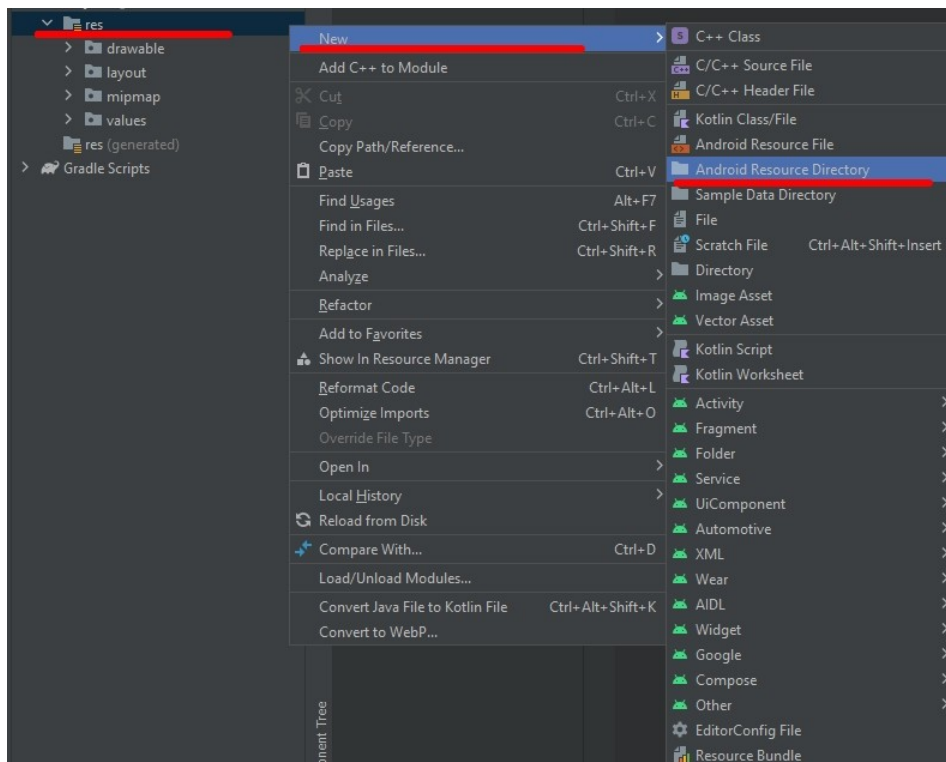


Рисунок 14 - Третья активность в альбомной ориентации.

Чтобы избежать такой проблемы, необходимо как-то по-другому скомпоновать кнопки. Например, расположить их не подряд друг за другом, а разбить на пары. Воспользуемся контейнером **TableLayout**. С его помощью мы можем разбить кнопки на две колонки и поместить их в три ряда.

Для этой операции нам понадобится сделать несколько важных шагов. Сначала нужно создать новую подпапку в папке **res**. Выделяем папку **res**, вызываем из него контекстное меню и последовательно выбираем команды **New | Android resource directory**.



В диалоговом окне из выпадающего списка **Resource type**: выбираем **layout**. В списке **Available qualifiers**: находим элемент **Orientation** и переносим его в правую часть **Chosen qualifiers**: с помощью кнопки с двумя стрелками. По умолчанию у вас появится имя папки **layout-port** в первой строке **Directory Name**: но нам нужен альбомный вариант, поэтому в выпадающем списке **Screen orientation** выбираем **Landscape**. Теперь название папки будет **layout-land**.

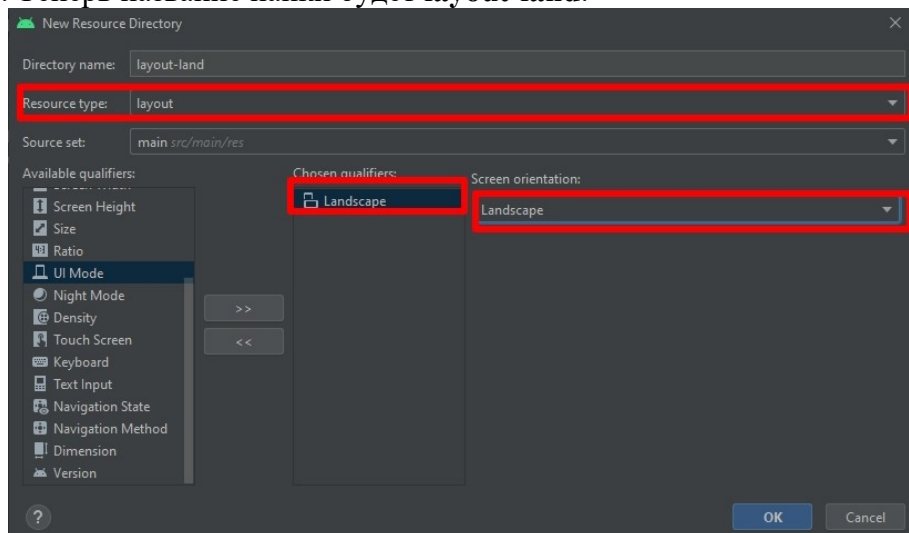


Рисунок 15 - Диалоговое окно создания директории.

Можно обойтись без помощи мастера, создав папку сразу через меню **New | Directory**. Этот способ годится для опытных разработчиков, которые знают, как следует назвать папку. Важно запомнить, что имя даётся не произвольно, а именно в таком виде **layout-land**. По суффиксу **-land** система понимает, что речь идет о новом режиме.

Теперь нам осталось создать новый XML-файл **activity_third.xml**. Выделите в папке **res/layout** активность **activity_third.xml**. Вызываем контекстное меню и выбираем команды **New | Layout Resource File**. В диалоговом окне присваиваем имя **activity_third.xml**, которое должно совпадать с именем существующего файла. Во втором поле вводим **LinearLayout**, по мере ввода появится подсказка, облегчающая

выбор. В списке **Available qualifiers**: находим элемент **Orientation** и переносим его в правую часть **Chosen qualifiers**: с помощью кнопки с двумя стрелками. Выбираем нам нужен альбомный вариант в выпадающем списке **Screen orientation** - **Landscape**.

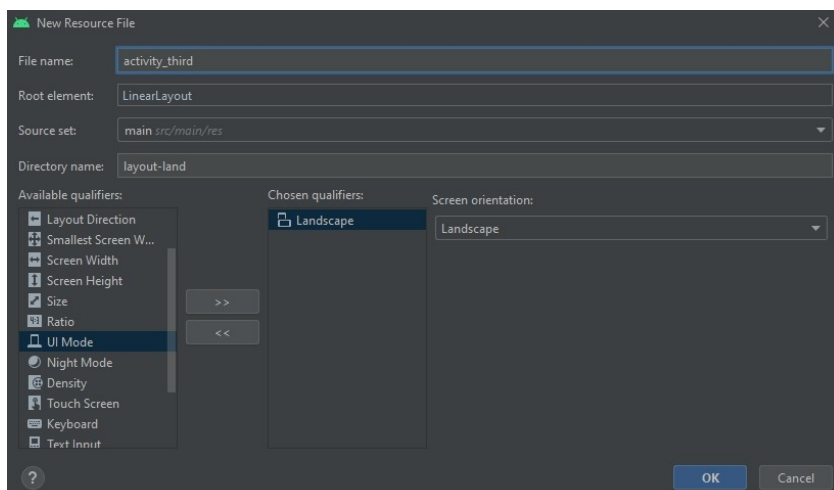


Рисунок 16 - Диалоговое окно создания активности.

Откроем созданный файл и модифицируем его следующим образом.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:orientation="vertical"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:layout_gravity="center"
        android:paddingLeft="20dp"
        android:paddingRight="20dp">

        <TableLayout
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_gravity="center"
            android:stretchColumns="*"
            android:layout_marginTop="50dp">

            <TableRow>

                <Button
                    android:id="@+id/button1"
                    android:layout_height="wrap_content"
                    android:layout_width="match_parent"
                    android:text="Первая кнопка"/>

                <Button
```

```

        android:id="@+id/button2"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="Вторая кнопка"/>
    </TableRow>
    <TableRow>

        <Button
            android:id="@+id/button3"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Третья кнопка"/>

        <Button
            android:id="@+id/button4"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Четвёртая кнопка"/>
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/button5"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Пятая кнопка"/>
        <Button
            android:id="@+id/button6"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="Шестая кнопка"/>
    </TableRow>
</TableLayout>
</LinearLayout>
</LinearLayout>

```

Запускаем приложение и проверяем. Теперь видны все кнопки.

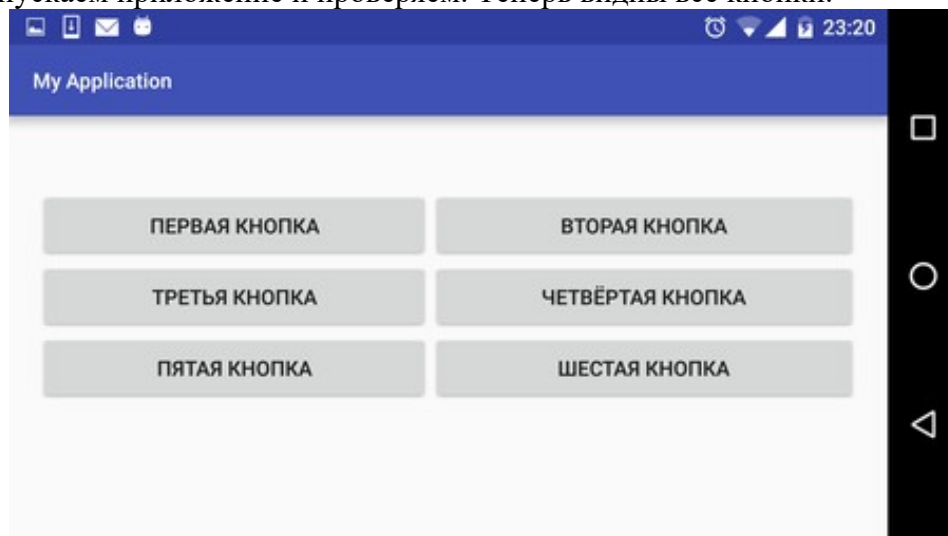


Рисунок 17 - Третья активность в альбомной ориентации после редактирования.

Когда вы создаёте альтернативную разметку, то не забывайте включать все компоненты, к которым будете обращаться программно, иначе получите ошибку. Допустим, вы забыли добавить шестую кнопку. В портретном режиме программа будет работать, а когда пользователь перевернёт экран, то активность будет инициализировать все компоненты для работы, а кнопки-то и нет.

Теперь, когда вы разобрались, как создавать новый файл для альбомной ориентации, можно воспользоваться быстрым способом без участия мастера настройки. Переключитесь в режим дизайна вашей основной разметки. Наверху на панели инструментов есть значок

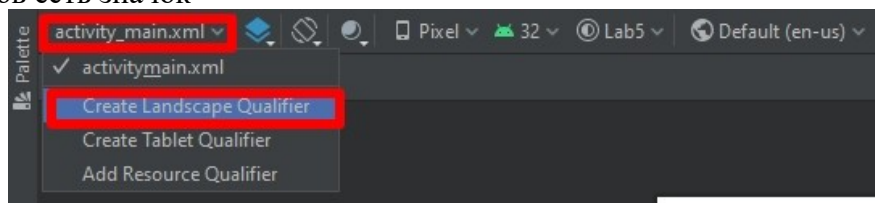


Рисунок 18 - Быстрое создание альбомной ориентации.

Щёлкните на нём и выберите пункт **Create Landscape Qualifier** -появится готовый файл в папке **res/layout-land**. Содержимое файла из портретной ориентации будет скопировано в файл и вы сможете его отредактировать по своему желанию.

Лабораторная работа №11. Разработка игры. Игровое поле.

Вступление.

Разработаем хорошо известную игру для тренировки памяти. На поле будут расположены перевернутые картинки, переворачивая по две необходимо найти среди них одинаковые. Найденная пара исчезает с поля.

По задумке в игре должно быть:

1. Игровое поле 6x6 (9x4 для альбомной ориентации) с картинками.
2. Учет количество ходов (или времени).
3. Начальный экран.
4. Настройки: выбор цвета фона и набора картинок.
5. Просмотр таблицы рекордов.

По этой логике разделим разработку на пять лабораторных работ.

Игровое поле.

Создайте новый проект (назовите его, например, «Мемо»), в качестве **Minimum SDK** выберите Android 4.4 (KitKat), однако, если вы владеете Android устройством с более ранней версией операционной системы, то может выбрать подходящую. В качестве шаблона выберите **Empty Activity**.

Для игрового поля будет использоваться элемент **GridView**. **GridView** - это представление данных (то есть их отображение на экране). А сами данные хранятся и передаются в **View** в адаптере (**BaseAdapter**), который связан с этим **GridView**. Стандартный адаптер нам не подойдет, поэтому напомним свой. Им будет класс **GridAdapter** унаследованный от стандартного класса адаптеров **BaseAdapter**.

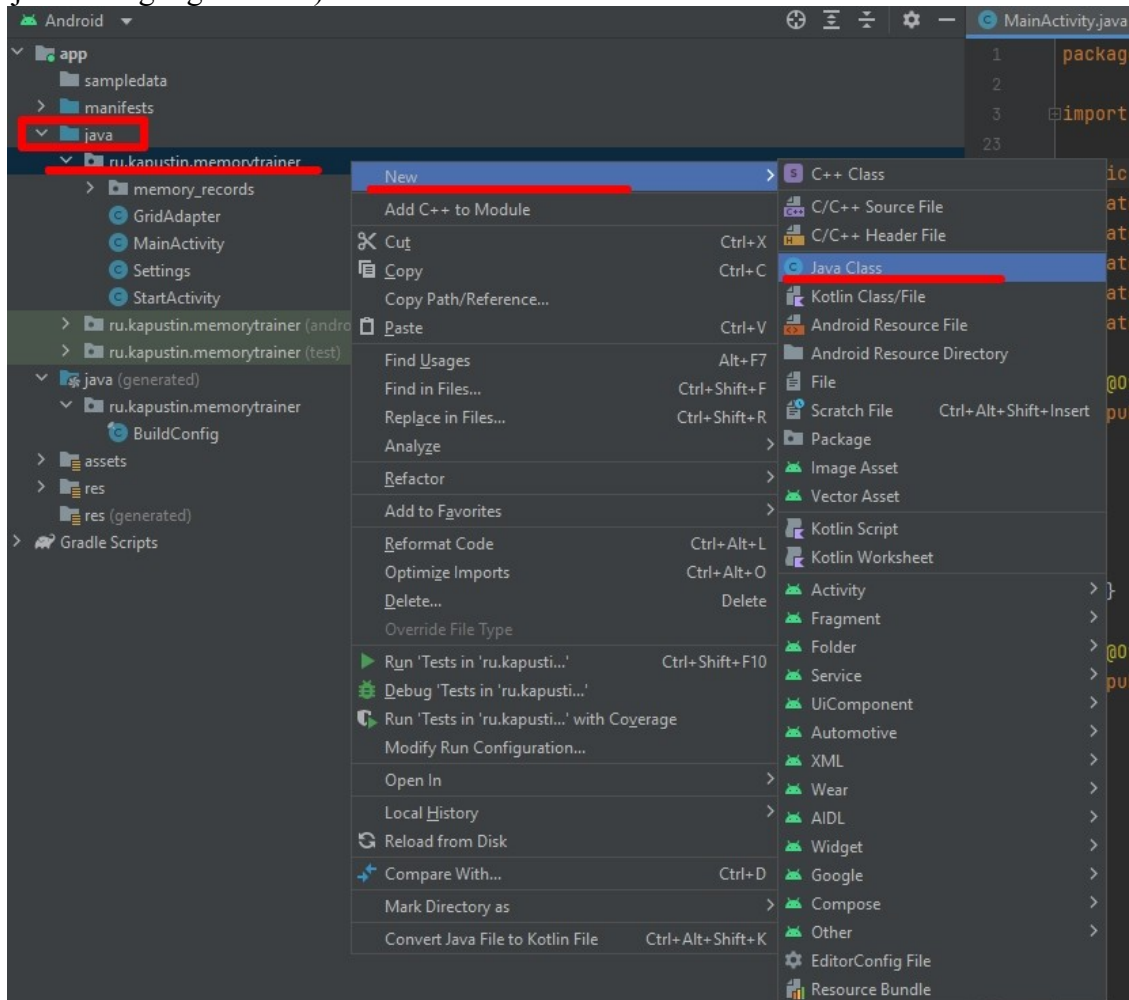
Сначала в **activity_main.xml** добавляем **GridView**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

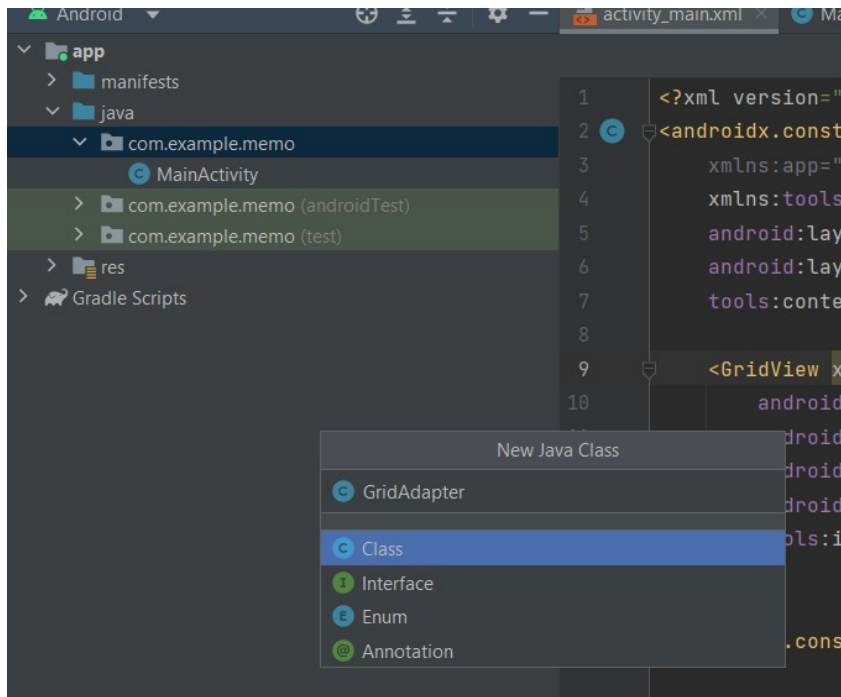
    <GridView xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:id="@+id/field"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:gravity="center" />
</LinearLayout>
```

Создайте в проекте класс **GridAdapter.java** (в директории `app/java/com.google.memo`).



Выберите Class и нажмите Enter



Унаследованные от **BaseAdapter** классы должны определять четыре метода:

int getCount() - возвращает количество элементов в **GridView**.

Object getItem(int position) - возвращает объект, хранящийся под номером **position**.

long getItemId(int position) - возвращает идентификатор элемента, хранящегося в под номером **position**.

View getView(int position, View convertView, ViewGroup parent) - Возвращает **View**, который будет выведен в ячейке под номером **position**.

Каждая ячейка идентифицируется не двумя цифрами (столбец и строка), как можно было ожидать, а одной. Нумеруются ячейки сверху вниз, слева направо. Пока описываем эти методы весьма условно, получаем что-то такое:

```
class GridAdapter extends BaseAdapter
{
    private Context mContext;
    private Integer mCols, mRows;

    public GridAdapter(Context context, int cols, int rows)
    {
        mContext = context;
        mCols = cols;
        mRows = rows;
    }

    @Override
    public int getCount() {
        return mCols * mRows;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }
}
```

```

}

@Override
public long getItemId(int position) {
    return 0;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {

    ImageView view; // выводиться у нас будет картинка

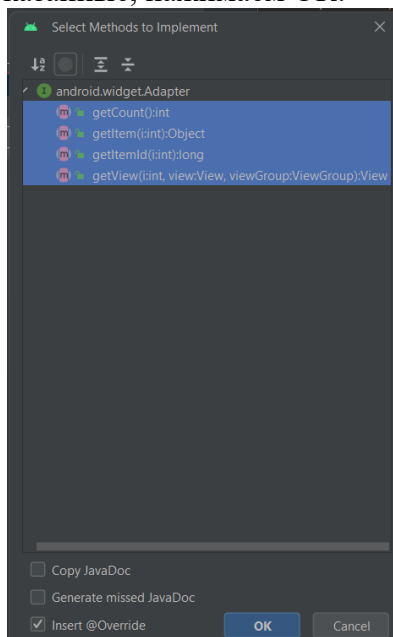
    if (convertView == null)
        view = new ImageView(mContext);
    else
        view = (ImageView)convertView;

    view.setImageResource(R.drawable.pict);

    return view;
}
}

```

Программа будет выдавать ошибку и предложит импортировать методы вышеуказанные, нажимаем ОК:



Лабораторная работа № 12. Буфер обмена

Добавьте все картинки, которые прилагаются к этой лабораторной работе, в папку **drawable** с помощью буфера обмена.

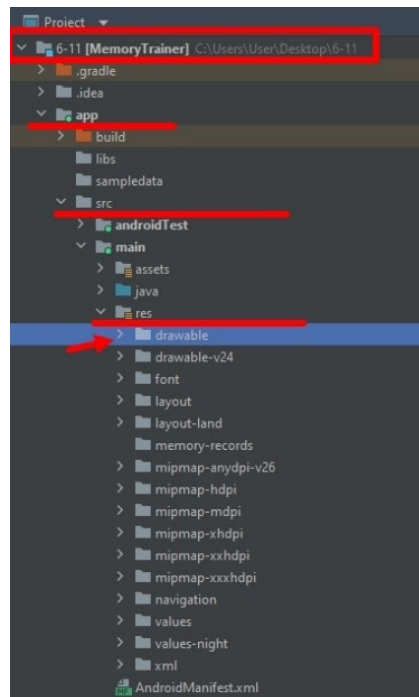


Рисунок 19 - Результат добавления картинок.

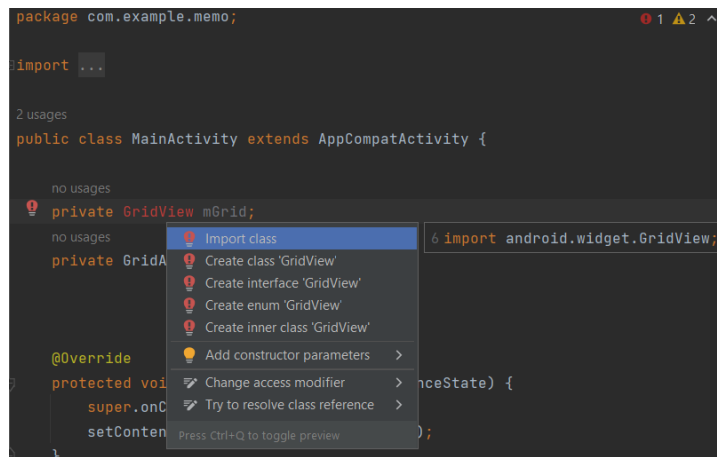
Теперь в основном классе связываем наш адаптер с таблицей:

```
public class MainActivity extends AppCompatActivity
{
    private GridView mGrid;
    private GridAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mGrid = (GridView)findViewById(R.id.field);
        mGrid.setNumColumns(6);
        mGrid.setEnabled(true);

        mAdapter = new GridAdapter(this, 6, 6);
        mGrid.setAdapter(mAdapter);
    }
}
```



Запустите проект, получаем таблицу 6*6 с одинаковой картинкой в каждой ячейке.

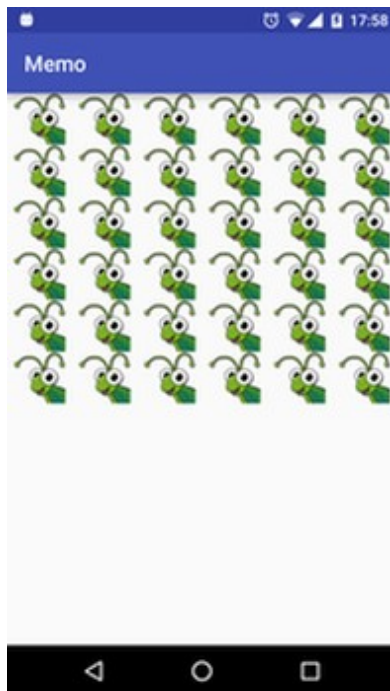


Рисунок 20 - Игровое поле.

Хорошо, но нам нужны разные картинки, причем каждая картинка должна встречаться два раза. Так как в будущем наборы картинок можно будет менять, для удобства в каждом таком наборе назовем файлы, например так: animal0.png, animal1.png, ... animal17.png или people0.png, people1.png, ... people17.png. Теперь наборы картинок отличаются префиксом, а внутри набора цифрой. Для обращения к картинке (или другого ресурса) используется его идентификатор, например, **R.drawable.picture**. Для того чтобы выбирать картинку динамически воспользуемся классом **Resources**. Этот класс предоставляет доступ ко всем ресурсам приложения. Для инициализации экземпляра класса используется метод **getResources()** (из класса **Context**). А для получения идентификатора ресурса по его названию метод

getIdentifier(String name, String defType, String defPackage), где **name** - название ресурса, **defType** - его тип (**drawable**, **string**, **anim**), **defPackage** - имя пакета приложения. Имя пакета можно также получить с помощью метода **getPackageName()** класса **Context**.

Таким образом записи

Integer identifierID = R.drawable.picture;

и

```
Resources mRes = mContext.getResources();
Integer identifierID = mRes.getIdentifier("picture", "drawable",
mContext.getPackageName());
```

по сути одинаковые и возвращают идентификатор файла **picture.png** из директории **drawable**.

Картинки в таблице должны располагаться в случайном порядке, для этого удобно использовать метод **Shuffle()** класса **ArrayList**.

ArrayList — массив для хранения данных любого типа. Добавим в массив название каждой картинки по два раза, а затем вызовем метод **Shuffle**, который перемешивает все элементы.

Допишите класс **GridAdapter**.

```
class GridAdapter extends BaseAdapter
{
    private Context mContext;
    private Integer mCols, mRows;
    private ArrayList<String> arrPict; // массив картинок
    private String PictureCollection; // Префикс набора картинок
    private Resources mRes; // Ресурсы приложения

    public GridAdapter(Context context, int cols, int rows)
    {
        mContext = context;
        mCols = cols;
        mRows = rows;
        arrPict = new ArrayList<String>();
        // Пока определяем префикс так, позже он будет браться из настроек
        PictureCollection = "animal";
        // Получаем все ресурсы приложения
        mRes = mContext.getResources();

        // Метод заполняющий массив arrPict
        makePictArray ();
    }

    private void makePictArray () {
        // очищаем массив
        arrPict.clear();
        // добавляем
        for (int i = 0; i < ((mCols * mRows) / 2); i++)
        {
            arrPict.add(PictureCollection + Integer.toString (i));
            arrPict.add(PictureCollection + Integer.toString (i));
        }
        // перемешиваем
        Collections.shuffle(arrPict);
    }

    @Override
    public int getCount() {
        return mCols*mRows;
    }
}
```

```

    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView view; // выводиться у нас будет картинка

        if (convertView == null)
            view = new ImageView(mContext);
        else
            view = (ImageView)convertView;
        // Получаем идентификатор ресурса для картинки,
        // которая находится в массиве vecPict на позиции position
        Integer drawableId = mRes.getIdentifier(arrPict.get(position), "drawable",
mContext.getPackageName());
        view.setImageResource(drawableId);
        return view;
    }
}

```

Все новые места кода прокомментированы, так что по идее все должно быть понятно. Запустите отладку. Видим вот такой вот веселенький зоопарк (рис 4.).

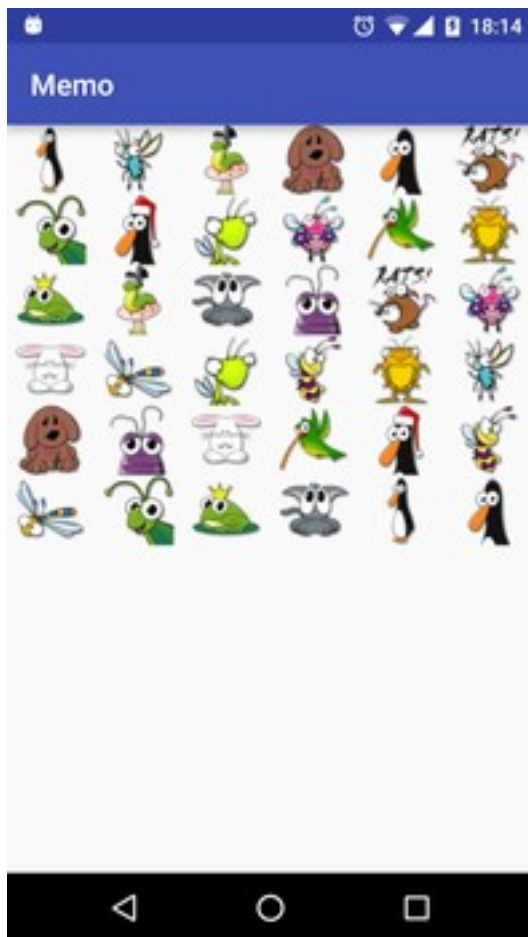


Рисунок 21 - Игровое поле.

Лабораторная 13. Ячейка состояния

Каждая ячейка может быть в одном из трех состояний: открытая, закрытая и удалённая (если две открытые картинки совпали, они убираются с поля). Для удобства добавим в **GridAdapter** константы для этих состояний:

```
private static final int CELL_CLOSE = 0;
private static final int CELL_OPEN = 1;
private static final int CELL_DELETE = -1;
или enum (он красивее смотрится)
private static enum Status {CELL_OPEN, CELL_CLOSE, CELL_DELETE};
```

В адаптере создадим массив для хранения состояния каждой ячейки, в начале игры всем ячейкам проставляется статус закрыта (**CELL_CLOSE**). И немного исправим метод **getView** чтобы картинка рисовалась в зависимости от статуса

```
class GridAdapter extends BaseAdapter
{
    private Context mContext;
    ...
    private enum Status {CELL_OPEN, CELL_CLOSE, CELL_DELETE}
    private ArrayList<Status> arrStatus; // состояние ячеек

    public GridAdapter(Context context, int cols, int rows)
    {
```

```

mContext = context;
mCols = cols;
mRows = rows;
arrPict = new ArrayList<String>();
arrStatus = new ArrayList<Status>();

// Пока определяем префикс так, позже он будет браться из настроек
PictureCollection = "animal";
// Получаем все ресурсы приложения
mRes = mContext.getResources();

// Метод заполняющий массив arrPict
makePictArray ();
// Метод устанавливающий всем ячейкам статус CELL_CLOSE
closeAllCells();
}

...

private void closeAllCells () {
    arrStatus.clear();
    for (int i = 0; i < mCols * mRows; i++)
        arrStatus.add(Status.CELL_CLOSE);
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {

    ImageView view; // выводиться у нас будет картинка

    if (convertView == null)
        view = new ImageView(mContext);
    else
        view = (ImageView)convertView;

    switch (arrStatus.get(position))
    {
        case CELL_OPEN:
            // Получаем идентификатор ресурса для картинки,
            // которая находится в векторе vecPict на позиции position
            Integer drawableId = mRes.getIdentifer(arrPict.get(position), "drawable",
mContext.getPackageName());
            view.setImageResource(drawableId);
            break;
        case CELL_CLOSE:
            view.setImageResource(R.drawable.close);
            break;
        default:
            view.setImageResource(R.drawable.none);
    }
}

```

```

        return view;
    }
}

```

Теперь надо обработать нажатие на ячейку. По нажатию будем делать следующее:

1. Проверяем есть ли уже открытые ячейки. Если открыты две ячейки, то сравниваем их и, если они одинаковые - удаляем, если разные - закрываем.
2. Открываем выбранную ячейку.
3. Проверяем остались ли открытые ячейки, если нет, то заканчиваем игру.

Можно проверять открытые картинки после нажатия, но тогда они будут закрываться (или удаляться) слишком быстро, что не удобно.

Нажатие на ячейку в таблице обрабатывает метод **setOnItemClickListener** класса **GridView**.

```

public class MainActivity extends AppCompatActivity
{
    ...

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ...

        mGrid.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View v,int position, long id){

                mAdapter.checkOpenCells ();
                mAdapter.openCell (position);

                if (mAdapter.checkGameOver())
                    Toast.makeText (getApplicationContext(), "Игра закончена",
Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

В методе **setOnItemClickListener** мы вызываем три метода из **GridAdapter**, они достаточно банальны:

```

class GridAdapter extends BaseAdapter
{
    ...

    public void checkOpenCells() {
        int first = arrStatus.indexOf(Status.CELL_OPEN);
        int second = arrStatus.lastIndexOf(Status.CELL_OPEN);
    }
}

```

```

        if (first == second)
            return;
        if (arrPict.get(first).equals (arrPict.get(second)))
        {
            arrStatus.set(first, Status.CELL_DELETE);
            arrStatus.set(second, Status.CELL_DELETE);
        }
        else
        {
            arrStatus.set(first, Status.CELL_CLOSE);
            arrStatus.set(second, Status.CELL_CLOSE);
        }
        return;
    }

    public void openCell(int position) {
        if (arrStatus.get(position) != Status.CELL_DELETE)
            arrStatus.set(position, Status.CELL_OPEN);

        notifyDataSetChanged();
        return;
    }

    public boolean checkGameOver() {
        if (arrStatus.indexOf(Status.CELL_CLOSE) < 0)
            return true;
        return false;
    }
}

```

Метод **notifyDataSetChanged()** класса **BaseAdapter** (мы его вызываем в **public void openCell(int position)**) сообщает **GridView**, что данные изменились и таблица перерисовывается. Казалось бы, что **notifyDataSetChanged()** надо вызвать и в **checkOpenCells()**, потому что там тоже меняется статус ячеек, но смысла в этом мало, потому что после **checkOpenCells()** всегда будет вызываться **openCell()** в которой таблица и перерисовуется.

Можно запускать и проверять работоспособность. Пока игра не ведет счета и после окончания игры просто показывается пустой экран без возможности запуска новой игры. Все это мы сделаем в следующих лабораторных работах.

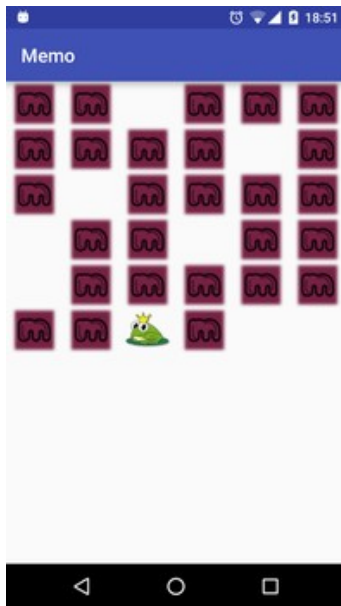


Рисунок 22- Игровое поле.

Лабораторная работа №14. Разработка игры. Учет количества ходов (или времени).

Учет количества ходов (или времени).

В прошлой лабораторной работе мы начали делать игру для тренировки памяти. Собственно, игра уже готова, осталось добавить к ней красоты. В этой лабораторной работе добавим таймер, который будет отсчитывать время с начала до окончания игры и учет ходов, то есть количество открытых картинок.

С количеством ходов все понятно, будем считать количество открытых картинок и выводить это число в **TextView**. Для таймера есть класс **Chronometer**, который делает как раз то что нам нужно — отсчитывает время. Класс унаследован от **TextView**, поэтому работать с ним можно как с обычным текстовым полем. Дополнительными являются методы:

void start () - запускает отсчет времени.

void stop() - останавливает отсчет.

void setFormat (String format) — устанавливает формат в котором будет выводиться дата. По умолчанию форматом является MM:SS (или H:MM:SS). Можно задать свой формат, при этом в строке format первое встреченное «%s», будет заменено на «HH:MM». Например: «Time: %s» будет выводить время «Time: 01:30»

Добавим в **activity_main.xml** элементы **TextView** и **Chronometer** (Элемент **Chronometer** добавляется посредством описания в разделе **Code**, в разделе **Design** добавить его не получится)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```

        android:orientation="horizontal" >

        <TextView xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/stepview"
            android:textSize="30dp"
            android:layout_weight="1"
            android:layout_marginLeft="10dp"
            android:layout_width="0dp"
            android:layout_height="wrap_content" />

        <Chronometer xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/timeview"
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:textSize="30dp" />

    </LinearLayout>

    <GridView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/field"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
</LinearLayout>

```

В классе **MainActivity** напишем их обработку:

```

public class MainActivity extends AppCompatActivity
{
    ...
    private TextView mStepScreen;
    private Chronometer mTimeScreen;

    private Integer StepCount; // кол-во ходов

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...

        mTimeScreen = (Chronometer) findViewById(R.id.timeview);
        mStepScreen = (TextView) findViewById(R.id.stepview);

        // шрифт
        Typeface type = Typeface.createFromAsset(getAssets(), "my-font.ttf");
        mTimeScreen.setTypeface(type);
        mStepScreen.setTypeface(type);

        StepCount = 0;
        mStepScreen.setText (StepCount.toString());
    }
}

```

```

mTimeScreen.start();

...

mGrid.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View v,int position, long id) {

        mAdapter.checkOpenCells ();
        mAdapter.openCell (position);

        StepCount ++;
        mStepScreen.setText (StepCount.toString());
        if (mAdapter.checkGameOver())
        {
            mTimeScreen.stop();
            String time = mTimeScreen.getText().toString();
            String TextToast = "Игра закончена nХодов: " + StepCount.toString() +
            "nВремя: " + time;
            Toast.makeText (getApplicationContext(), TextToast,
            Toast.LENGTH_SHORT).show();
        }
    }
});
}
}

```

Для добавленных **TextView** и **Chronometr** мы используем шрифт, который необходимо добавить в проект. Для этого в дереве проекта переключитесь в режим **Project** и добавьте папку **Assets** в **Memo\app\src\main**. Теперь добавьте в эту папку файл необходимого шрифта, можете использовать тот что идет в комплекте с лабораторной работой.

В коде есть такие строчки:

Typeface type = Typeface.createFromAsset(getAssets(),"my-font.ttf");

mTimeScreen.setTypeface(type);

Таким способом можно установить свой шрифт для **TextView**.

Typeface — класс стилей шрифта.

createFromAsset(AssetManager mgr, String path) — создает экземпляр класса **Typeface**, используя файл из директории **/assets**.

setTypeface(Typeface) — устанавливает шрифт в **TextView**.

Запускаем:

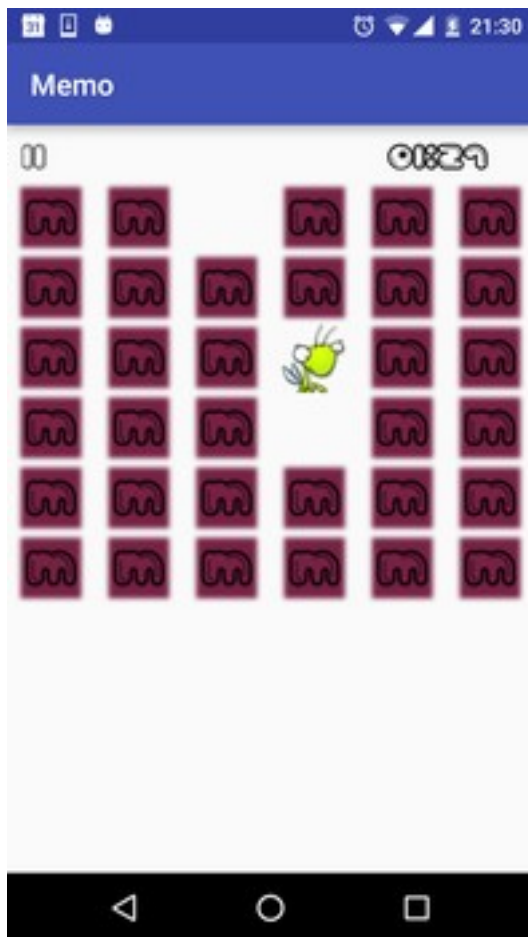


Рисунок 23 - Игровое поле.

Тут, кстати, обнаружилась небольшая ошибка в игре: нажатие на уже удаленную картинку считается ходом, что неправильно. Переделаем метод **openCell()** в классе **GridAdapter**, он должен проверять является ли карточка закрытой и только тогда ее открывать.

```
public boolean openCell(int position) {
    if (arrStatus.get(position) == Status.CELL_DELETE || arrStatus.get(position) ==
Status.CELL_OPEN)
        return false;

    if (arrStatus.get(position) != Status.CELL_DELETE)
        arrStatus.set(position, Status.CELL_OPEN);

    notifyDataSetChanged();
    return true;
}
```

А в классе **MainActivity**, будем правильно считать ходы

```
mGrid.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View v,int position, long id) {

        mAdapter.checkOpenCells ();
        if (mAdapter.openCell (position)) {
            StepCount ++;
        }
    }
});
```

```

        mStepScreen.setText (StepCount.toString());
    }

    ...
}
});

```

В xml и у **TextView** и у **Chronometer** также описаны параметры шрифта
 android:textSize="30dp"
 android:textColor="#000000"

Две строчки, конечно, можно и скопировать в два места, но если или строчек, или мест будет много, то это неудобно, да и места много занимает. Создадим стиль шрифта. Откройте файл **themes.xml** в директории **/res/values/**. Добавьте туда:

```

<resources>
    ...
    <style name="MyText">
        <item name="android:textColor">#000000</item>
        <item name="android:textSize">30dp</item>
    </style>
</resources>

```

А в **activity_main.xml** заменим эти строчки у **TextView** и **Chronometer** на:
 style="@style/MyText"

Теперь чтобы изменить стиль шрифта и в **TextView** и в **Chronometer**, надо изменить только файл **themes.xml**, например, добавим тень к тексту:

```

<style name="MyText">
    <item name="android:textColor">#000000</item>
    <item name="android:textSize">30dp</item>
    <item name="android:shadowColor">#FF0000</item>
    <item name="android:shadowDx">2</item>
    <item name="android:shadowDy">2</item>
    <item name="android:shadowRadius">3</item>
</style>

```

Лабораторная работа №15. Разработка игры. Настройки: выбор цвета фона и набора картинок.

В прошлых лабораторных работах мы уже сделали игровое поле и собственно саму игру, а также начальный экран. Осталось сделать таблицу рекордов и окно настроек. По сути эта и следующая лабораторная работа будет посвящена хранению данных приложений Android.

Данные приложений можно хранить пятью способами:

1. Shared Preferences (общие настройки). Хранилище данных простых типов (boolean, float, int, long и string) в формате ключ-значение.
2. Внутренняя память — хранение данных в файлах в памяти телефона.
3. Внешняя память - хранение данных в файлах на внешнем носителе (например, на карте памяти).
4. SQLite — хранение данных в базе данных.
5. Удаленный ресурс — хранение данных на удаленном ресурсе в сети.

Причем в 1, 2 и 4 случае данные по умолчанию будут доступны только для данного приложения, и при удалении программы из телефона они сотрутся автоматически.

Мы рассмотрим первые два способа. Настройки будем хранить в **Shared Preferences**, а таблицу рекордов в файле во внутренней памяти телефона.

Начнем с настроек. Класс **SharedPreferences** — класс для записи и чтения данных основных типов в формате ключ-значение.

Читать данные можно с помощью методов:

```
getBoolean(String key, boolean defValue),  
getFloat(String key, float defValue),  
getInt(String key, int defValue),  
getLong(String key, long defValue),  
getString(String key, String defValue).
```

где **key** — название ключа, а **defValue** — значение по умолчанию, то есть значение которое возвращает метод если в **SharedPreferences** этот ключ не определен.

Записывать значения можно с помощью подкласса **SharedPreferences.Editor**. Этот класс инициализируется методом **edit()** класса **SharedPreferences**. Сохраняем пары ключ-значение с помощью методов:

```
putBoolean(String key, boolean value),  
putFloat(String key, float value),  
putInt(String key, int value),  
putLong(String key, long value),  
putString(String key, String value).
```

где **key** — ключ, **value** — значение.

Для удаления ключа есть метод **remove(String key)**. После всех изменений необходимо вызвать метод **commit()**, который запишет все изменения.

Для того чтобы пользователь мог задавать настройки необходимо окно, его можно написать самостоятельно, а можно воспользоваться классом **PreferenceActivity**. Это класс наследник от **Activity** и он берет на себя большую часть работы по созданию окон для редактирования настроек. Вначале создадим разметку для окна, она немного отличается от разметок для простых окон. Создайте директорию **res/xml**, а в ней файл **settings.xml**.

Описание настроек начинается с тэга **PreferenceScreen**, которое описывает окно настроек.

В **PreferenceScreen** можно вложить другой **PreferenceScreen**, тогда он будет выглядеть как строчка в списке настроек, а по нажатию открывать другой экран.

Настройки можно группировать с помощью категорий **PreferenceCategory**. Категорий может быть несколько или не быть совсем. Например, на картинке две категории «Настройки» и «Еще настройки».

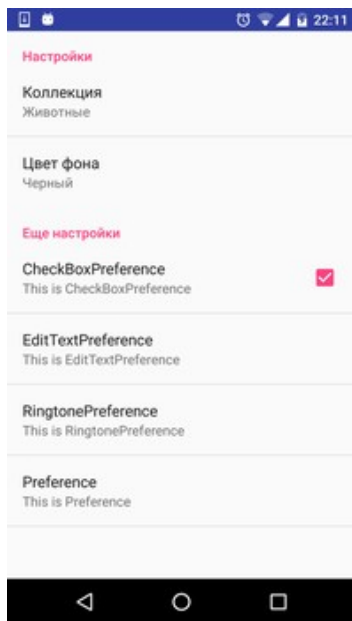


Рисунок 24 - Окно настроек.

Сами настройки могут быть следующих типов:

- **CheckBoxPreference** – чекбокс.

```
<CheckBoxPreference
    android:key="cbp"
    android:title="CheckBoxPreference"
    android:defaultValue="true"
    android:summary="This is CheckBoxPreference"
/>
```

- **ListPreference** - список из которого может быть выбран только один элемент.

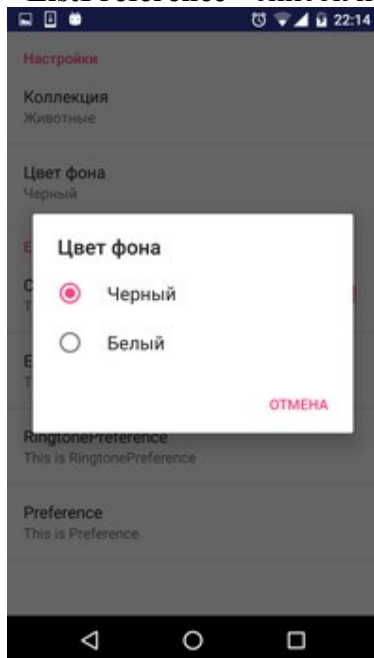


Рисунок 25 – Настройка ListPreference.

```
<ListPreference
    android:key="lp"
    android:title="ListPreference"
```

```
android:entries="@array/prefs"  
android:entryValues="@array/prefsValue"  
</>
```

- **EditTextPreference** – строка.

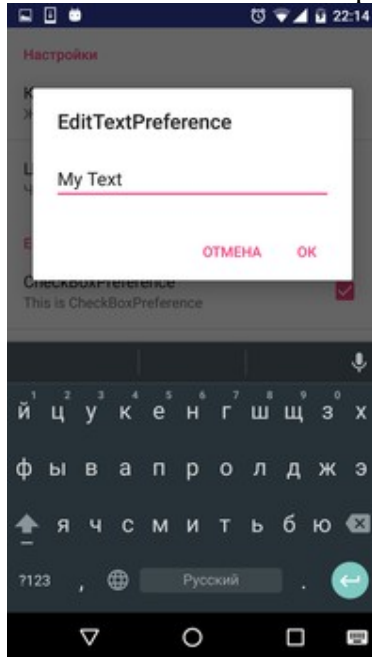


Рисунок 26 – Настройка EditTextPreference.

```
<EditTextPreference  
    android:key="etp"  
    android:title="EditTextPreference"  
    android:defaultValue="My Text"  
    android:summary="This is EditTextPreference"  
</>
```

- **RingtonePreference** - выбор рингтона.

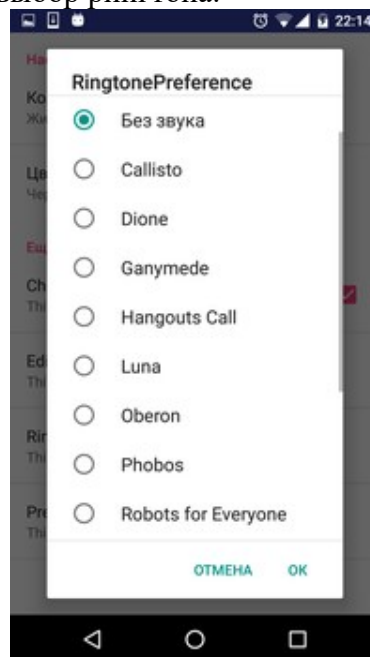


Рисунок 27 – Настройка RingtonePreference.

```
<RingtonePreference
    android:key="rp"
    android:title="RingtonePreference"
    android:summary="This is RingtonePreference"
/>
```

- **Preference** - кнопка, которой можно задать какое-то действие.

```
<Preference
    android:key="customPref"
    android:title="Preference"
    android:summary="This is Preference"
/>
```

Атрибуты:

android:key - ключ, по которому затем можно будет получить значение настройки.

android:title - название настройки (то что написано крупным шрифтом).

android:summary - описание (то что написано мелким шрифтом). Сюда для удобства можно выводить значение настройки.

android:defaultValue - значение по умолчанию.

android:entryValues - список значений. Один элемент этого списка будет является значением настройки.

android:entries - список значений, который будет видеть пользователь. Этот список может совпадать с **android:entryValues**.

В последних версиях Android API Level появились еще интересные типы настроек, такие как **MultiSelectListPreference** (level 11), **SwitchPreference** (level 14) или **TwoStatePreference** (level 14).

Если стандартных типов настроек не хватает, можно написать класс элемента самостоятельно.

Нам из всего этого нужен только **ListPreference**. Напишите в **settings.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="@string/setTitle">

        <ListPreference android:key="PictureCollection"
            android:title="@string/Collection"
            android:entries="@array/CollectionsTitle"
            android:entryValues="@array/Collections"
            android:defaultValue="animal"
        />

        <ListPreference android:key="BackgroundColor"
            android:title="@string/Color"
            android:entries="@array/ColorTitle"
            android:entryValues="@array/Color"
            android:defaultValue="#000000"
        />
    </PreferenceCategory>
</PreferenceScreen>
```

В **/res/values/strings.xml** напишите массивы для списков:

```

<resources>
  <string name="app_name">Memo</string>
  ...

  <string-array name="Collections">
    <item>animal</item>
    ...
    <item>space</item>
  </string-array>

  <string-array name="CollectionsTitle">
    <item>Животные</item>
    ...
    <item>Космос</item>
  </string-array>

  <string-array name="Color">
    <item>#000000</item>
    ...
    <item>#FFFFFF</item>
  </string-array>

  <string-array name="ColorTitle">
    <item>Черный</item>
    ...
    <item>Белый</item>
  </string-array>

</resources>

```

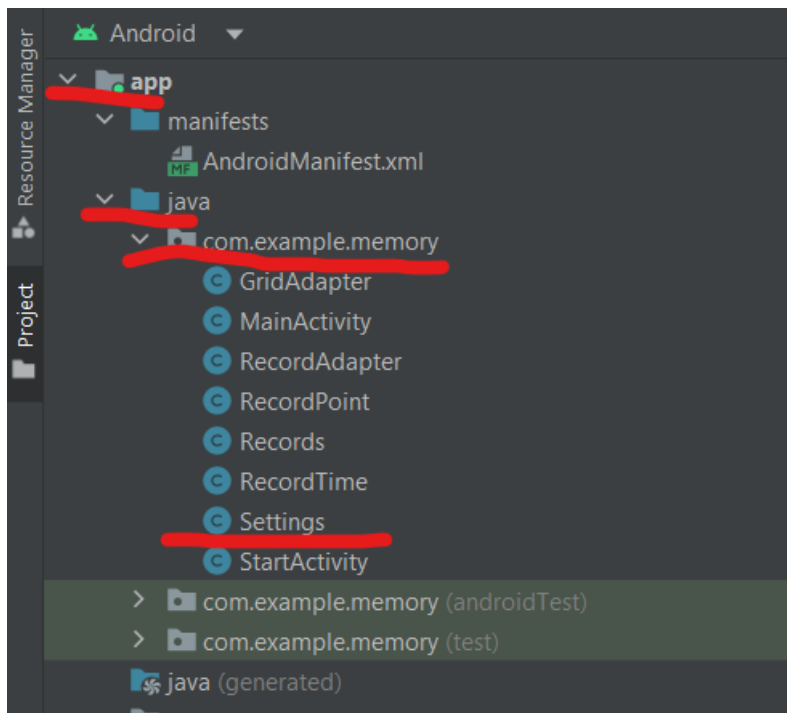
Здесь **Collections** и **Color** — массивы, чьи элементы будут сохраняться как значение настроек. **CollectionsTitle** и **ColorTitle** — списки, которые будет видеть пользователь.

В одной из прошлых лабораторных работ, мы решили что наборы картинок у нас будут называться как-то так animal0.png, animal1.png, ... animal17.png или people0.png, people1.png, ... people17.png, то есть наборы картинок отличаются префиксом, а внутри набора цифрой. Вот эти префиксы и будут элементами массива **Collections**. В массиве **CollectionsTitle** находятся человеческие названия для каждого набора.

С массивами **Color** и **ColorTitle** все понятно, это цвета в формате #RRGGBB и их русское название.

Лабораторная работа №16. Написание класса «Настройки»

Теперь напишем класс отвечающий за настройки. Создайте файл **Settings.java** (в скриншоте показаны и другие активности, но создание их будет дальше.)



И напишете в него:

```

public class Settings extends PreferenceActivity implements
Preference.OnPreferenceChangeListener
{
    ListPreference collection;
    ListPreference color;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.settings);

        collection = (ListPreference)this.findPreference("PictureCollection");
        color = (ListPreference)this.findPreference("BackgroundColor");

        // устанавливаем слушатель
        collection.setOnPreferenceChangeListener(this);
        color.setOnPreferenceChangeListener(this);

        // пишем в summary текущее значение
        collection.setSummary(collection.getEntry());
        color.setSummary(color.getEntry());
    }

    @Override
    public boolean onPreferenceChange(Preference preference, Object newValue)
    {
        preference.setSummary(((CharSequence)newValue));
        return true;
    }
}

```

```
}
}
```

Метод **onPreferenceChange(Preference preference, Object newValue)** — вызывается когда изменилась одна из настроек, где **preference** — настройка которая изменилась, **newValue** — новое значение.

При изменении настроек в **summary** записывается новое значение.

Добавьте этот класс в **AndroidManifest.xml** и вызываем это окно в **StartActivity** по нажатию на кнопку «Настройка». (Сделайте по аналогии с **MainActivity**)

Запускаем. Настройки открываются, по умолчанию в **summary** проставляется значение по умолчанию, но при выборе другого значения в списке, оно меняется на английское (из массива **Collection**), а не на русское (из массива **CollectionTitle**). Чтобы исправить это добавим следующее:

```
public class Settings extends PreferenceActivity implements
Preference.OnPreferenceChangeListener
{
    ...

    // два массива для хранения значений списков
    CharSequence[] pictValue, colValue;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        ...

        // получаем списки значений для каждой настройки
        pictValue = collection.getEntries();
        colValue = color.getEntries();
    }

    @Override
    public boolean onPreferenceChange(Preference preference, Object newValue)
    {
        // название (android:key) настройки, которая была изменена
        String Key = preference.getKey();

        // если это коллекция картинок
        if (Key.equals("PictureCollection"))
        {
            // определяем индекс нового значения
            int i = ((ListPreference)preference).findIndexOfValue(newValue.toString());
            // ищем русское название для значения с этим индексом и записываем в
summary
            preference.setSummary(pictValue[i]);
            return true;
        }

        // если это цвет
        if (Key.equals("BackgroundColor"))
```

```

        {
            int i = ((ListPreference)preference).findIndexOfValue(newValue.toString());
            preference.setSummary(colValue[i]);
            return true;
        }

        // для всех остальных настроек (хотя у нас их и нет), ставим пришедшее
        значение в summary
        preference.setSummary((CharSequence)newValue);
        return true;
    }
}

```

Лабораторная работа № 17. Добавление в класс массивы

Добавим в класс два массива **pictValue** и **colValue**, в методе **onCreate()** инициализируем их с помощью метода **getEntries()** класса **ListPreference**. Этот метод возвращает массив строк который записан в **ListPreference** в **android:entries**. И в методе **onPreferenceChange** в зависимости от того какая настройка была изменена, ищем в одном из этих массивов нужное нам значение.

Все работает. Теперь при изменении настройки у нас пишется русское значение. Писать метод для сохранения настроек не надо, все делается автоматически. Теперь если изменить настройки, выйти из приложения и зайти заново они сохранятся. Значения настроек связаны с нашим приложением и недоступны из других, при удалении программы, они сотрутся.

Теперь в классе **MainActivity** прочитаем значения наших настроек. В методе **OnCreate** напишите:

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Настройки
    SharedPreferences settings = PreferenceManager.getDefaultSharedPreferences(this);
    String PictureCollection = settings.getString("PictureCollection", "animal");
    Integer BackgroundColor = Color.parseColor(settings.getString("BackgroundColor",
"black"));

    ...

    mGrid = (GridView)findViewById(R.id.field);
    // устанавливаем цвет для фона
    View root = mGrid.getRootView();
    root.setBackgroundColor(BackgroundColor);
    mGrid.setEnabled(true);

    ...

}

```

Метод **getDefaultSharedPreferences(Context context)** класса **PreferenceManager** возвращает объект **SharedPreferences** с настройками для данного приложения. Получаем значения для **PictureCollection** и **BackgroundColor**. Устанавливаем цвет. Если написать **mGrid.setBackgroundColor(BackgroundColor)**, то цвет установится только под таблицей, что не очень красиво, поэтому вначале с помощью метода **getRootView()** получаем **View** самого верхнего элемента (у нас это **LinearLayout**) и устанавливаем цвет для него.

Для задания коллекции картинок, немного изменим **GridAdapter**, а именно добавим еще один параметр в конструктор.

```
public GridAdapter(Context context, int cols, int rows, String pictCollection)
{
    ...

    // определяем префикс
    PictureCollection = pictCollection;
    ...
}
```

И соответственно изменяем в классе **MainActivity**:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    ...

    mAdapter = new GridAdapter(this, 6, 6, PictureCollection);

    ...
}
```

Осталось только добавить соответствующие наборы картинок. Можете взять их из тех что идут в комплекте с этой лабораторной работой. Теперь можно запускать, в настройках можно менять набор картинок и цвет фона игрового поля.

Лабораторная работа №18. Разработка игры. Просмотр таблицы рекордов.

Это последняя лабораторная работа по разработке игры для тренировки памяти. У нас уже почти все готово, осталось только сохранять количество очков и затраченное время в конце каждой игры и выводить 5 лучших результатов по нажатию на кнопку «Рекорды». Окно результатов будет с двумя вкладками, в одной соответственно времени, во второй - очки. Сами результаты будем хранить в файле во внутренней памяти телефона.

Создайте новый класс **Records** (в **Records.java**) унаследованный от **TabActivity** и разметку окна **records.xml** (в **/res/layout**). Разметка окна будет такая:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
```

```

        android:layout_height="fill_parent">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">

            <TabWidget
                android:id="@android:id/tabs"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
            />

            <FrameLayout
                android:id="@android:id/tabcontent"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent" />

        </LinearLayout>
    </TabHost>

```

TabHost - контейнер для окна с вкладками. У него обязательно должны быть два дочерних элемента: **TabWidget** и **FrameLayout**.

TabWidget — список ярлыков вкладок, при нажатии на один из ярлыков в **FrameLayout** открывается соответствующая вкладка.

В классе **Records** напишите:

```

public class Records extends TabActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.records);

        TabHost tabHost = getTabHost();

        // Вкладка Время
        TabSpec timetab = tabHost.newTabSpec("Time");
        // устанавливаем заголовок и иконку
        timetab.setIndicator("по времени");
        // устанавливаем окно, которая будет показываться во вкладке
        Intent timeIntent = new Intent(this, RecordTime.class);
        timetab.setContent(timeIntent);

        // Вкладка Очки
        TabSpec pointtab = tabHost.newTabSpec("Point");
        pointtab.setIndicator("по очкам");
        Intent pointIntent = new Intent(this, RecordPoint.class);
        pointtab.setContent(pointIntent);

        // Добавляем вкладки в TabHost
    }
}

```

```

        tabHost.addTab(timetab);
        tabHost.addTab(pointtab);
    }
}

```

В качестве иконки вкладки мы написали «R.drawable.time», что обычно соответствует файлу /res/drawable/time.png. Но мы хотим сделать, чтобы рисунок активной вкладки отличался от рисунка неактивной. Например, у активной вкладки изображение будет красным (time_red.png), а у неактивной — серым (time_grey.png). Это легко сделать с помощью StateListDrawable — объект drawable определённый в xml, который позволяет использовать несколько изображений в зависимости от состояния объекта. Создадим в /res/drawable файл time.xml и напишем в него

```

<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/time_red"
        android:state_selected="true" />
    <item android:drawable="@drawable/time_grey" />
</selector>

```

Аналогично сделаем с картинками point_grey.png и point_red.png.

Добавьте еще два класса RecordTime (в **RecordTime.java**) и RecordPoint (в **RecordPoint.java**) и соответственно две разметки к ним **time_tab.xml** и **point_tab.xml**. Добавьте также все новые классы в **AndroidManifest.xml** и по кнопке «Результаты» откройте окно **Records**. Все это мы уже делали, так что проблем у вас возникнуть не должно. Запускаем:

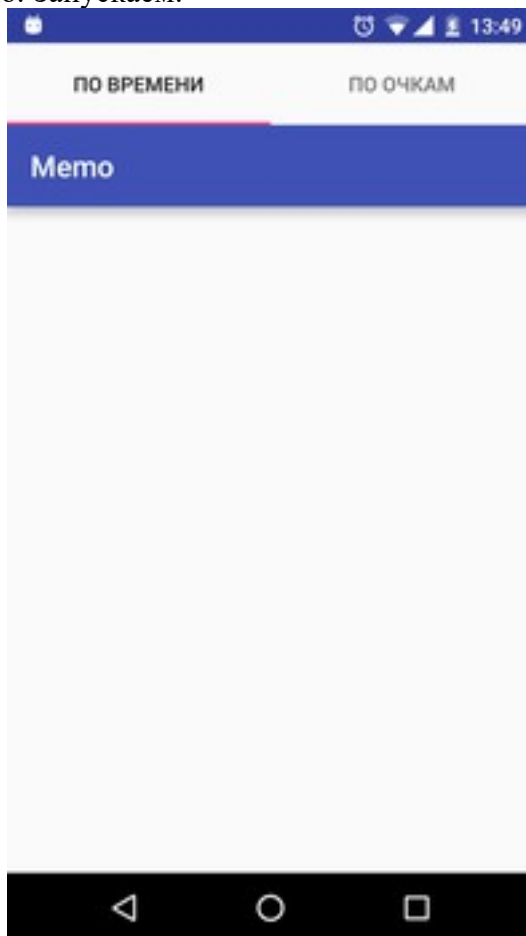


Рисунок 28 - Таблица рекордов.

Со вкладками разобрались, теперь посмотрим, как сохранять и выводить результаты. Можно сохранять результат каждой игры, а выводить только 5 лучших, но тогда получится, что мы храним кучу никому не нужных (и никогда не используемых) данных. Поэтому в файле у нас всегда будет не больше 5 лучших результатов (без дубликатов), при чем уже отсортированных, чтобы удобнее было сразу их выводить.

Для того, чтобы сохранить данные в файле нам понадобятся два класса:

FileOutputStream - выходной поток, который записывает данные в файл. Класс инициализируется методом **openFileOutput(String name, int mode)** (класса **Context**), где **name** - имя файла, который надо открыть, **mode** - режим, по умолчанию используется **MODE_PRIVATE** (создается файл доступный только из данного приложения). Также может быть **MODE_APPEND** (если файл уже существует, то данные добавляются в конец файла), **MODE_WORLD_READABLE** и **MODE_WORLD_WRITABLE** (файлы которые может читать или записывать стороннее приложение).

ObjectOutputStream - класс для сериализации любых объектов.

```
SomeObject obj;
FileOutputStream fos = openFileOutput(«FileName.txt», Context.MODE_PRIVATE);
ObjectOutputStream os = new ObjectOutputStream(fos);
os.writeObject(obj);
os.close();
```

Мы сериализуем и записываем в файл **FileName.txt**, объект типа **MyObject**. Объект может быть любого типа, например, **String**, **Integer**, **ArrayList**.

Для чтения файлов используются классы **FileInputStream** и **ObjectInputStream**. Работа с ними аналогична:

```
FileInputStream fis = openFileInput(«FileName.txt»);
ObjectInputStream is = new ObjectInputStream(fis);
SomeObject s1 = (SomeObject) is.readObject();
is.close();
```

Метод **Object readObject()** возвращает значение типа **Object** - базовый класс языка Java. Все не примитивные типы данных являются наследниками этого класса, поэтому мы можем выполнить операцию по приведению типа.

Записать в один файл можно несколько объектов (несколько раз вызвав метод **writeObject()**). Соответственно читать их надо будет в том же порядке.

Если вставить эти четыре строчки в код, то Студия их подчеркнет и напишет кучу ошибок типа «Unhandled exception type FileNotFoundException» (**IOException** или **ClassNotFoundException**). Этот код потенциально может вызывать исключения и поэтому его надо окружить операторами **try/catch**. По-хорошему надо обработать каждое исключение и написать код, что делать в случае его возникновения. Но все эти виды исключений являются наследниками класса **Exception**, поэтому все их можно отловить в одном **catch (Exception e) {}**. В нашей программе мы так и поступим, в случае любой ошибки будем писать сообщение пользователю (**Toast**).

С рекордами удобно работать в **ArrayList**, так как, во-первых, в нем их можно сортировать и искать, а во-вторых, его легко можно вывести в **ListView**. Значит в нашем файле будет храниться (в сериализованном виде) два массива - рекорды по времени и рекорды по очкам. По окончании игры, считаем массивы, добавим в них значения результатов текущей игры, отсортируем и первые 5 элементов массива запишем обратно в файл. Чтобы не засорять код класса **MainActivity** напишем всю работу с файлами в отдельном классе **RecordAdapter** (файл **RecordAdapter.java**). Это будет просто класс java не от чего не унаследованный.

```

class RecordAdapter
{
    // Название файла в котором хранятся данные
    private static String FILE_RECORDS = "memo-records";

    // Два массива для рекордов
    ArrayList<String> recTime;
    ArrayList<Integer> recPoint;

    Context mContext;

    public RecordAdapter(Context context)
    {
        recTime = new ArrayList<String> ();
        recPoint = new ArrayList<Integer> ();

        mContext = context;
        // читаем из файла два массива с рекордами
        try {
            FileInputStream fis = mContext.openFileInput(FILE_RECORDS);
            ObjectInputStream is = new ObjectInputStream(fis);
            recPoint = (ArrayList<Integer>) is.readObject();
            recTime = (ArrayList<String>) is.readObject();
            is.close();
        } catch (Exception e) {
            Toast.makeText(mContext, "Произошла ошибка чтения таблицы рекордов",
Toast.LENGTH_LONG);
        }
    }

    public void WriteRecords ()
    {
        // записываем в файл массивы с рекордами
        try {
            FileOutputStream fos = mContext.openFileOutput(FILE_RECORDS,
Context.MODE_PRIVATE);
            ObjectOutputStream os = new ObjectOutputStream(fos);
            os.writeObject(recPoint);
            os.writeObject(recTime);
            os.close();
        } catch (Exception e) {
            Toast.makeText(mContext, "Произошла ошибка записи в таблицу рекордов",
Toast.LENGTH_LONG);
        }
        return;
    }

    public void addTime (String str)
    {
        // Добавляем новое значение времени,

```

```

        // если такого еще нет в массиве
        if (!recTime.contains(str))
            recTime.add(str);

        // сортируем
        Collections.sort(recTime);

        // оставляем в массиве только 5 элементов
        for (int i = 5; i < recTime.size(); i++)
            recTime.remove(i);

        return;
    }

    public void addPoint (Integer num)
    {
        if (!recPoint.contains(num))
            recPoint.add(num);

        Collections.sort(recPoint);

        for (int i = 5; i < recPoint.size(); i++)
            recPoint.remove(i);

        return;
    }

    public ArrayList<String> getRecTime()
    {
        // возвращаем массив рекордов по времени
        return recTime;
    }

    public ArrayList<String> getRecPoint()
    {
        // переделываем массив целых чисел в массив строк
        ArrayList<String> arr = new ArrayList<String>();
        for (Integer temp : recPoint)
            arr.add(temp.toString());

        // возвращаем массив рекордов по очкам
        return arr;
    }
}

```

Теперь напишите в методе **ShowGameOver()** класса **MainActivity**, код для сохранения результатов игры:

```

private void ShowGameOver () {

    String time = mTimeScreen.getText().toString();

```

```

// Читаем файл с рекордами
RecordAdapter ra = new RecordAdapter (this);
// Добавляем новые значения
ra.addPoint(StepCount);
ra.addTime(time);
// Записываем рекорды в файл
ra.WriteRecords();

// Диалоговое окно
...
}

```

А в классах **RecordPoint** и **RecordTime** выведем список рекордов. Добавьте в **point_tab.xml** и **time_tab.xml**:

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />

</LinearLayout>

```

А в классы:

```

public class RecordPoint extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.point_tab);

        RecordAdapter ra = new RecordAdapter (this);
        ArrayList<String> arr = ra.getRecPoint();

        ArrayAdapter<String> mAdapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, arr);
        setListAdapter(mAdapter);
    }
}

```

Вместо **android.R.layout.simple_list_item_1** можно написать **R.layout.item**, а в файле **/res/layout/item.xml** описать как должна выглядеть каждая строка таблицы.

Запускаем. Теперь в конце игры результаты сохраняются и выводятся в списках по кнопке «Рекорды».

ПО ВРЕМЕНИ	ПО ОЧКАМ
01:10	86
01:28	90
01:32	106

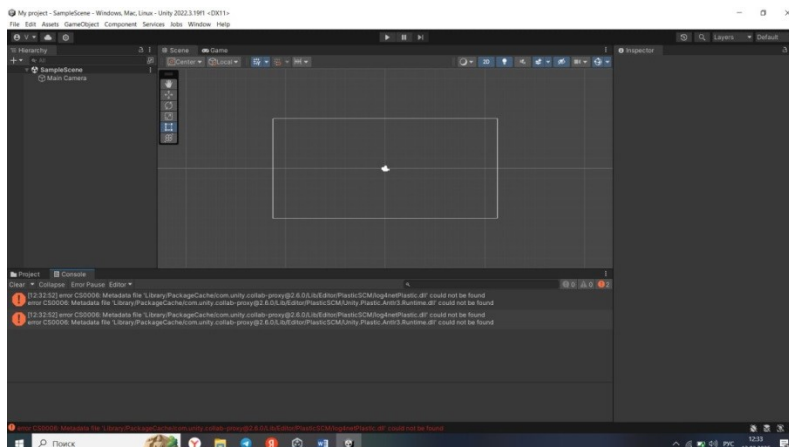
Рисунок 29 - Таблица рекордов.

Все, игра готова. Вам осталось лишь доработать по своему вкусу графическую составляющую, после чего вы сможете гордо заявить, что разработали мобильную игру!

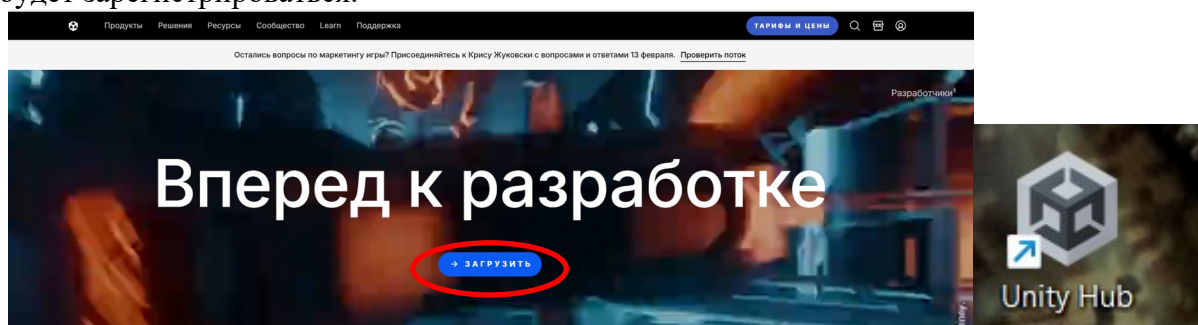
Семестр 4

Лабораторная работа №1. Space shooter. Установка Unity и регистрация в системе.

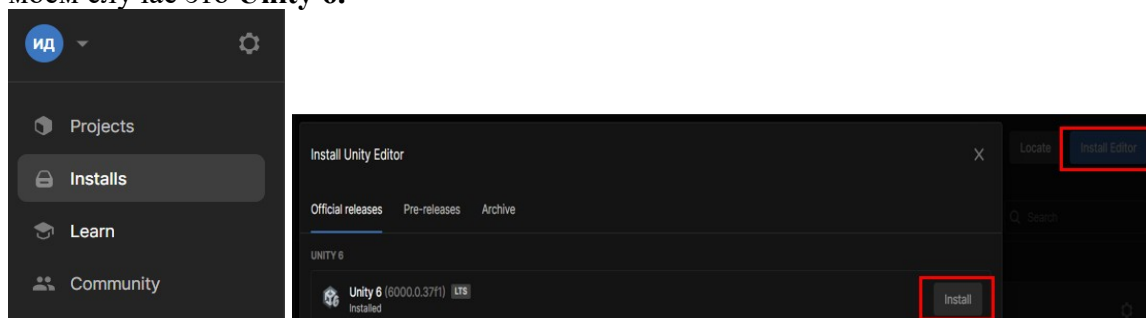
При загрузке **Unity** устаревшей версии, например, **2022.3.19f1**, при открытии проекта будут появляться ошибки. Это может быть связано с тем, что данная версия устарела.



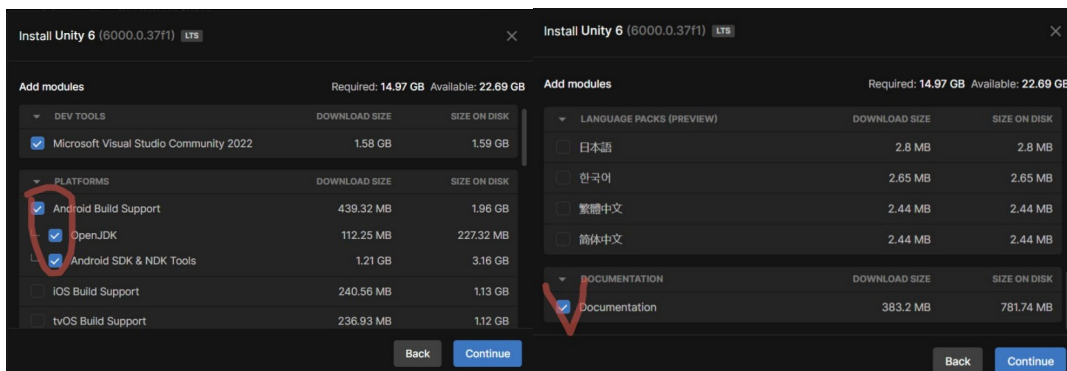
Перед тем, как установить новую версию **Unity**, переходим по ссылке <https://unity.com/ru> и устанавливаем **Unity Hub**, нажимая на «Загрузить» в самом верху, качаем установщик и устанавливаем саму программу. После установки необходимо будет зарегистрироваться.



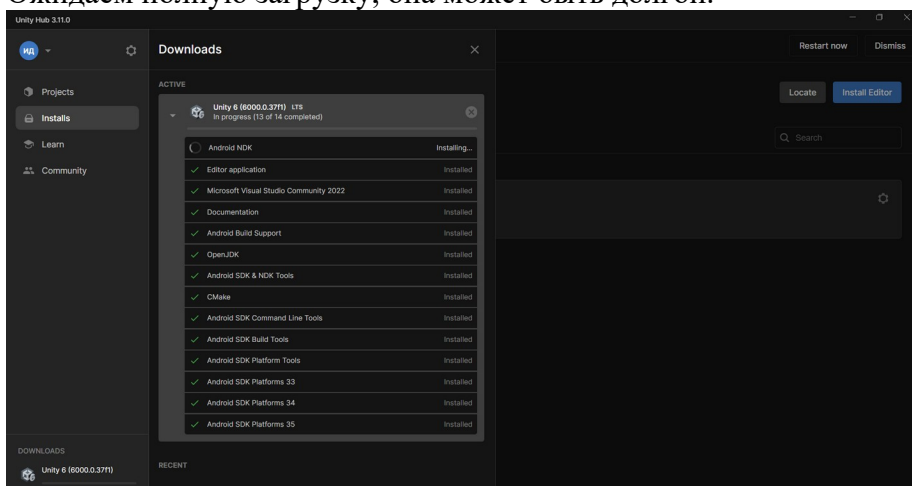
Заходим в **Unity Hub**, вы сейчас и находитесь там, и переходим в раздел установки Installs, выбираем Install Editor и версию, которую предлагает программа. В моем случае это **Unity 6**.



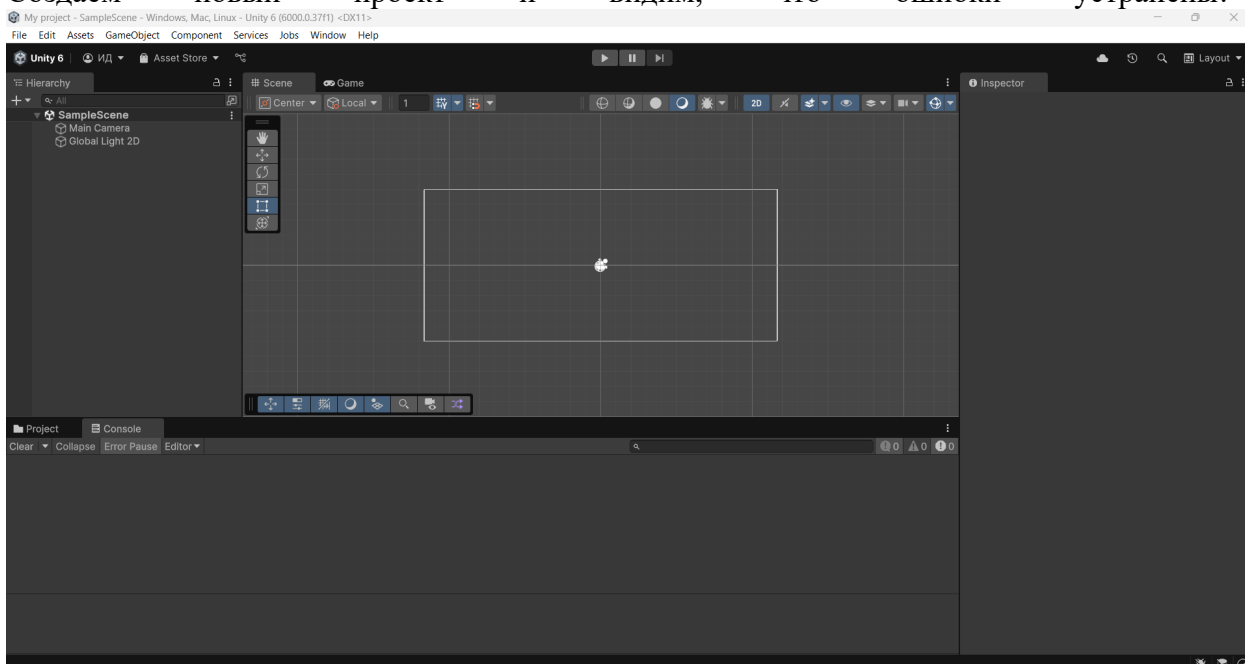
Ставим галочки, как показано на рисунке, и нажимаем «Continue».



Ожидаем полную загрузку, она может быть долгой.

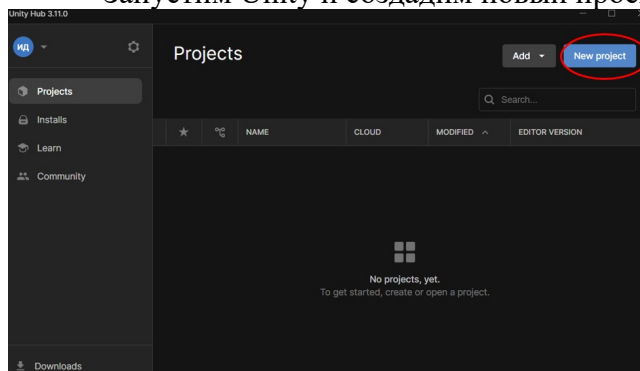


Создаем новый проект и видим, что ошибки устранены!

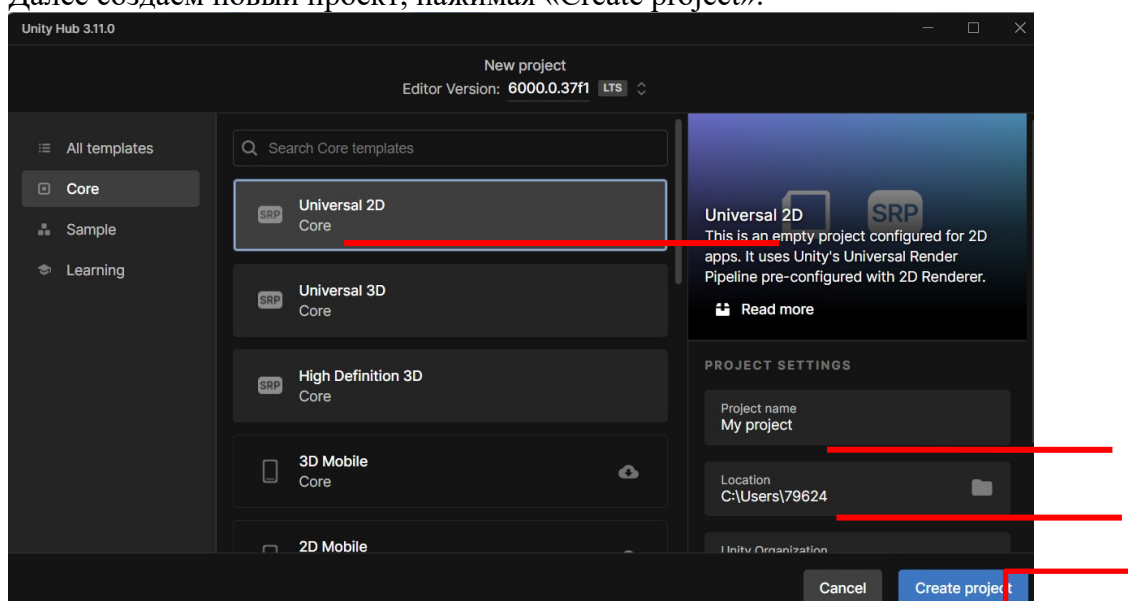


Лабораторная №2. Space Shooter. Настройка проекта. Установка Ассетов.

Запустим Unity и создадим новый проект «New project».



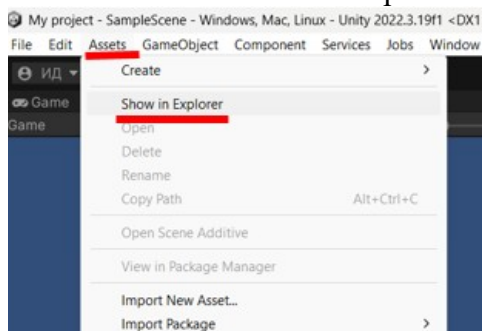
Справа мы можем выбрать способ хранения проекта в блоке Location, лучше оставить хранение на диске. Также в данном окне мы можем дать название проекту и выберем посередине 2D-шаблон. Unity аналитика нам не понадобится, отключаем ее. Далее создаем новый проект, нажимая «Create project».



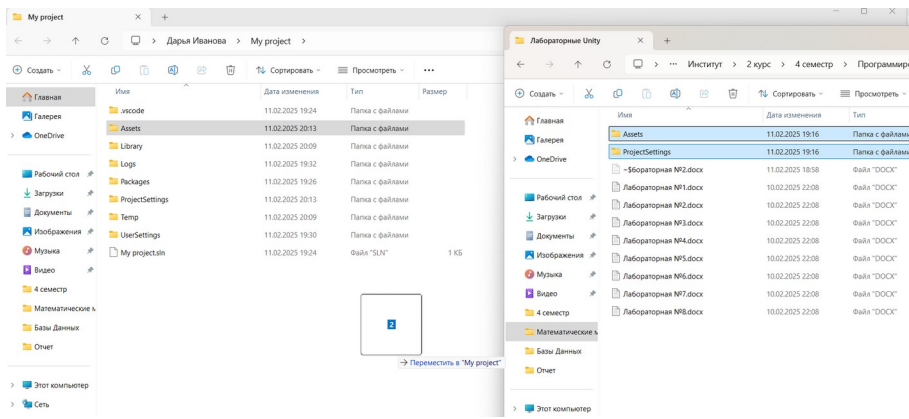
Теперь необходимо установить ассеты для будущей игры. Сначала распакуем прикрепленный архив, нажав «Извлечь все». В результате получим две папки.

Assets	11.02.2025 19:16	Папка с файлами
ProjectSettings	11.02.2025 19:16	Папка с файлами

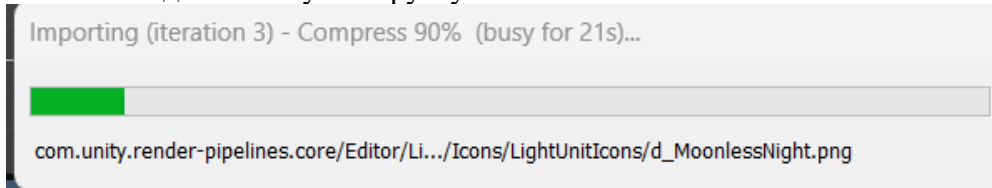
Далее нужно перенести эти папки в наш проект. Для этого в самом проекте нажмем Assets – Show in explorer.



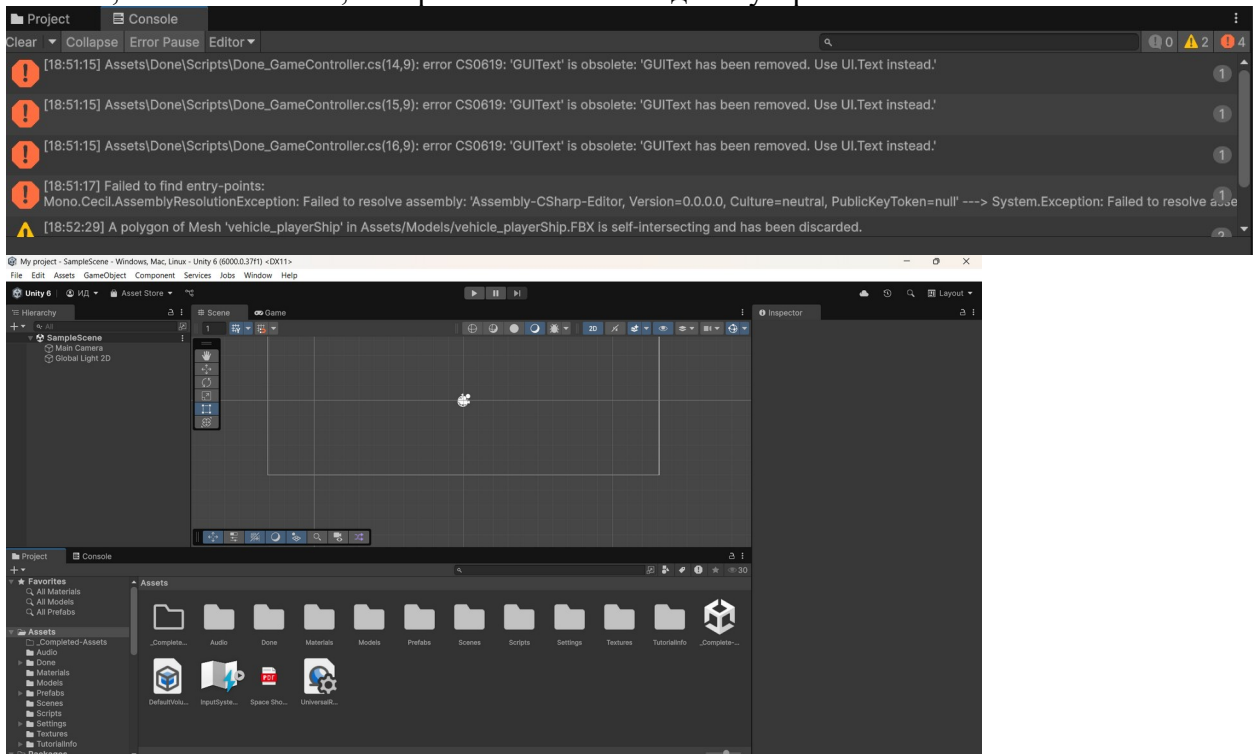
В открывшуюся директорию переместим папки.



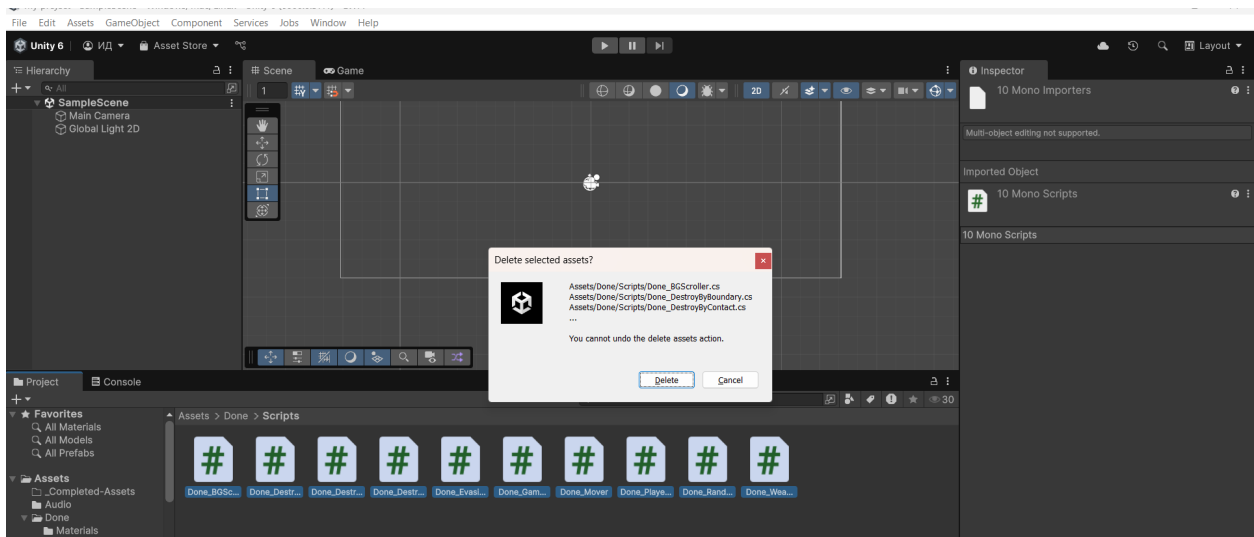
Ожидаем полную загрузку.



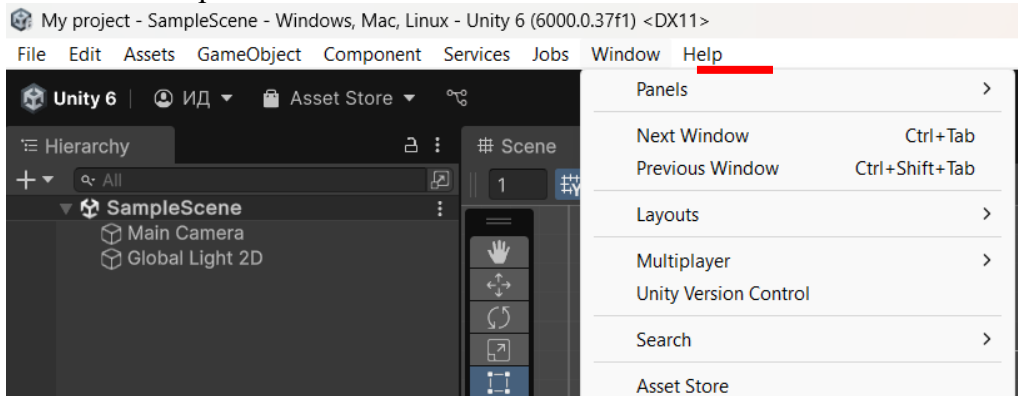
Возвращаемся в проект и замечаем, что наши файлы установились в рабочую область, но есть ошибки, которые сейчас необходимо устранить.



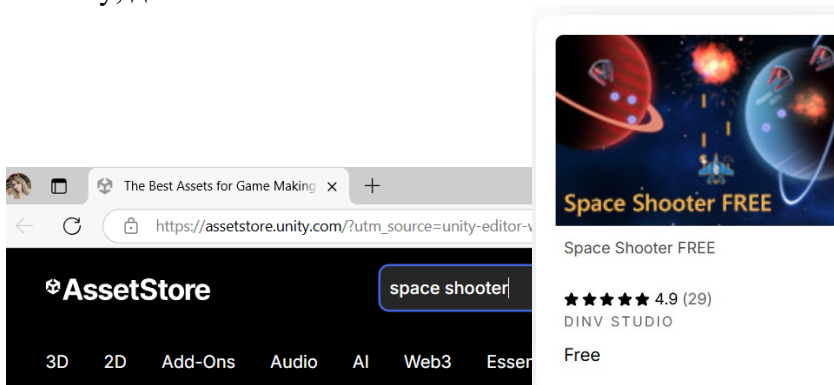
Содержимое папки **Scripts** удаляем, так как напишем собственные. Далее можем работать с проектом.



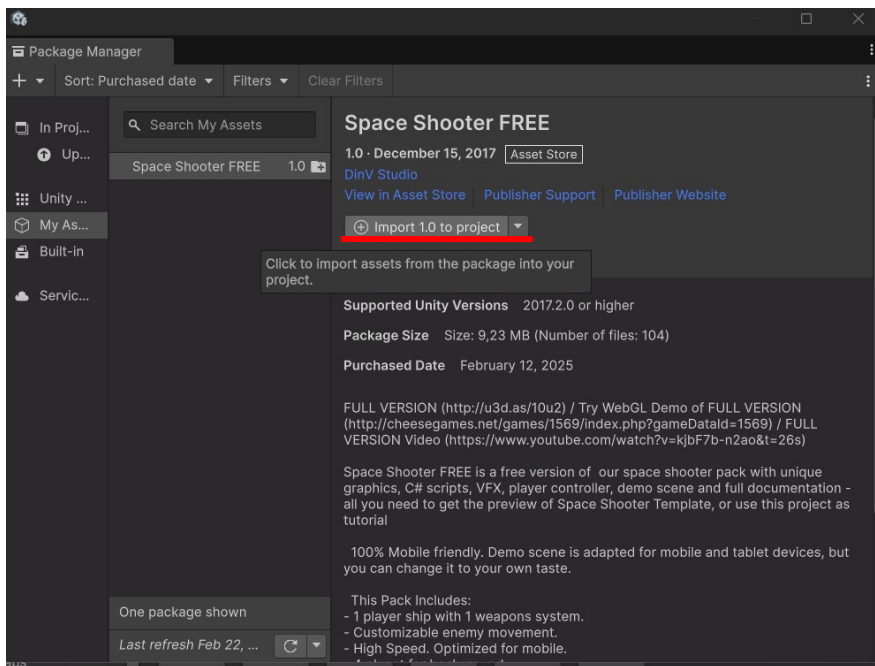
Если перейти в папку **Models**, то можно увидеть, что наши объекты: астероиды и корабли, не отображаются. Для этого нужно загрузить тот же комплект объектов. Заходим на верхней панели в **Window - Asset Store**.



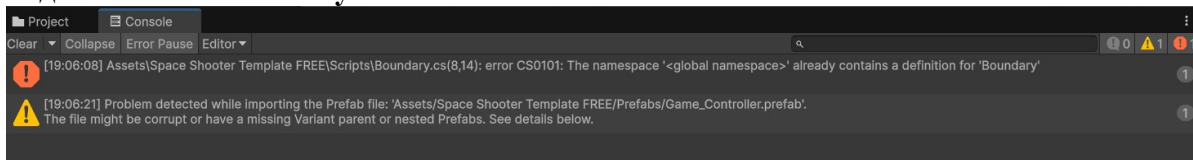
Пишем в строке поиска и выбираем самый первый бесплатный набор, нажимая по нему, добавляете в ваши ассеты.



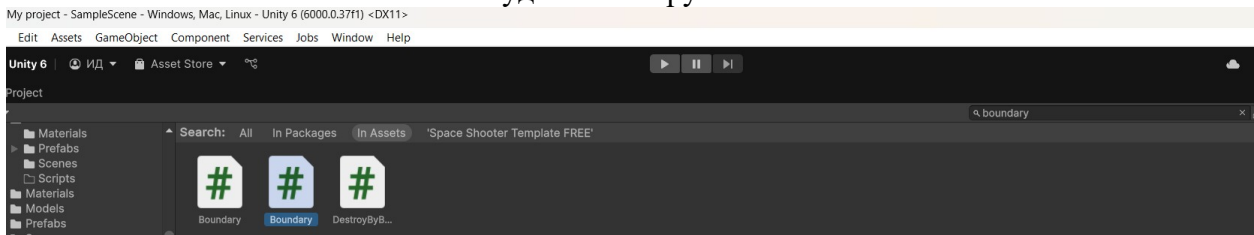
После загрузки появляется окошко, импортируете пакет в проект, везде нажимаете далее, не меняя параметров. Ждем полную загрузку.



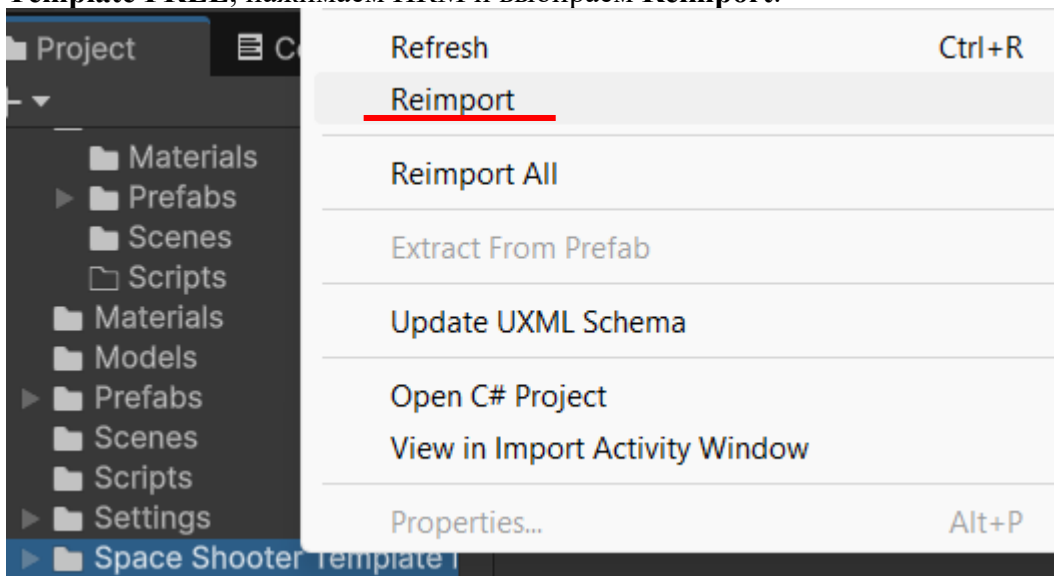
Видим, что появилась ошибка, которая означает, что у нас две одинаковые папки под названием **Boundary**.



Переходим во вкладку **Project**, два раза нажимая по ней, откроется большое окно. В поиске вбиваем название папки и удаляем вторую.



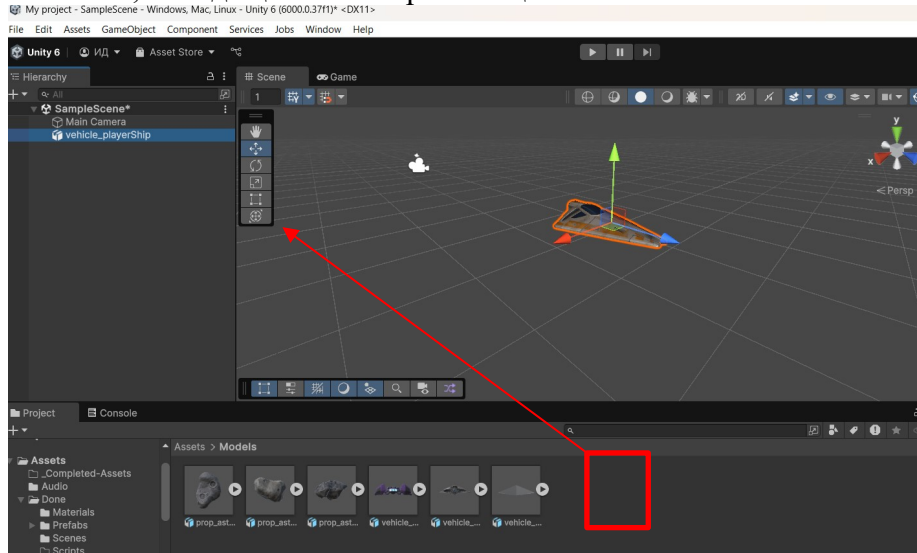
Пересобираваем скачанный пакет, выбирая в **Project** папку **Space Shooter Template FREE**, нажимаем ПКМ и выбираем **Reimport**.



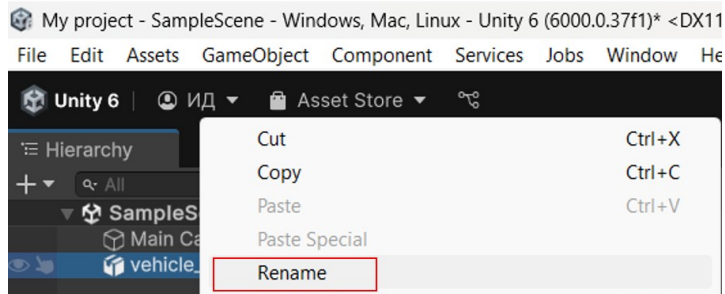
Вверху меняем вид рабочей области. Нажимаем вверху на значки  и .

Лабораторная работа №3. Game object игрока

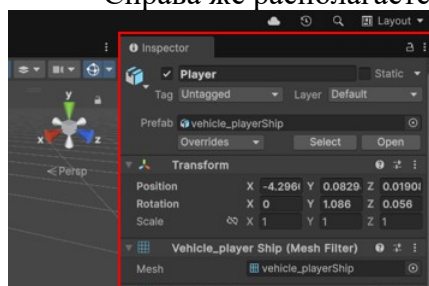
Остаемся в режиме отображение **Scene** и добавим космический корабль, которым игрок будет управлять. Для этого заходим в папку **Models**, находим там модель корабля - **vehicle_playerShip** и переносим в иерархию вверху окна(**Hierarchy**). Это совокупность объектов, находящихся на игровой сцене.



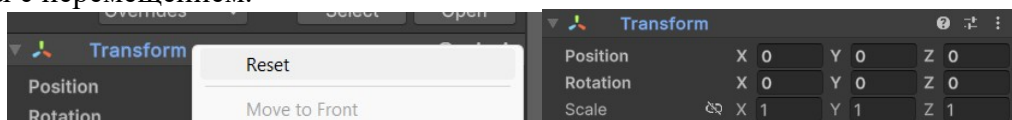
Сейчас, переименуем корабль как **Player**. Нажмем правой кнопкой мыши на игровом объекте корабля в иерархии и нажмем **Rename**.



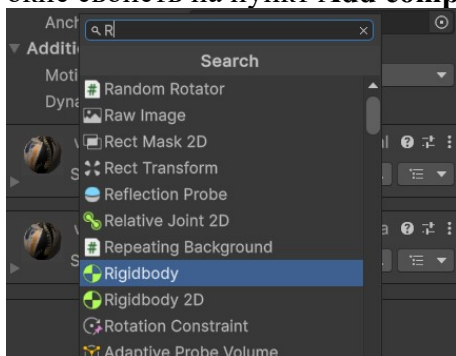
Справа же располагается окно свойств объекта Инспектор (**Inspector**).



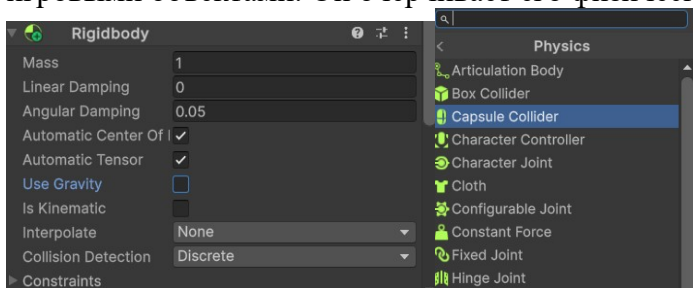
На данный момент корабль находится в координатах, обозначенных во вкладке **Transform – Position**. **Scale** - это размер игрового объекта, а **Rotation** его «поворот». Устанавливаем корабль в начало координат, нажимая ПКМ свойство **Transform** и нажмем **Reset**. Помещение в начало координат игрового объекта облегчает процесс работы с перемещением.



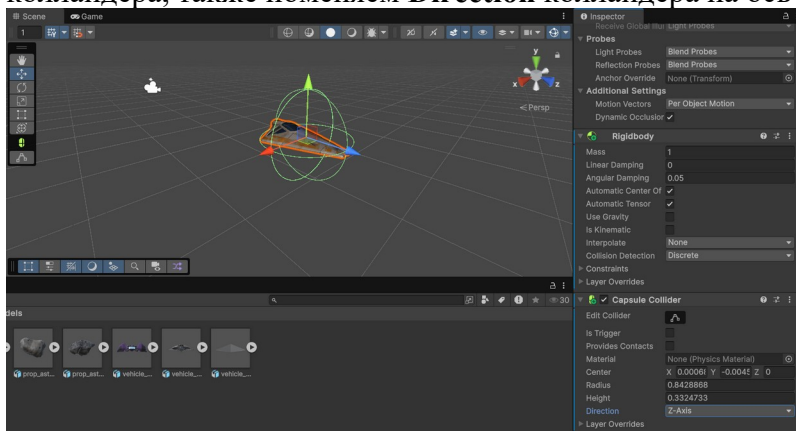
Теперь проработаем «физику» корабля, которая нужна для взаимодействия с другими игровыми объектами: вражеские корабли, снаряды и астероиды. Кликнем в окне свойств на пункт **Add component**, выберем пункт **Physics – Rigidbody**



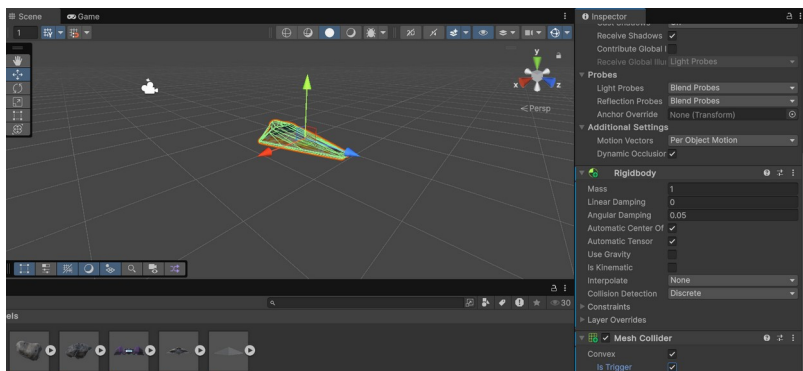
Поработаем с данным параметром: отключим гравитацию, т.к. мы находимся в космосе. Потом также, как мы добавляли **Rigidbody**, добавим к игровому объекту **Capsule Collider**, который необходим для обнаружения столкновений с другими игровыми объектами. Он очерчивает его физическую форму.



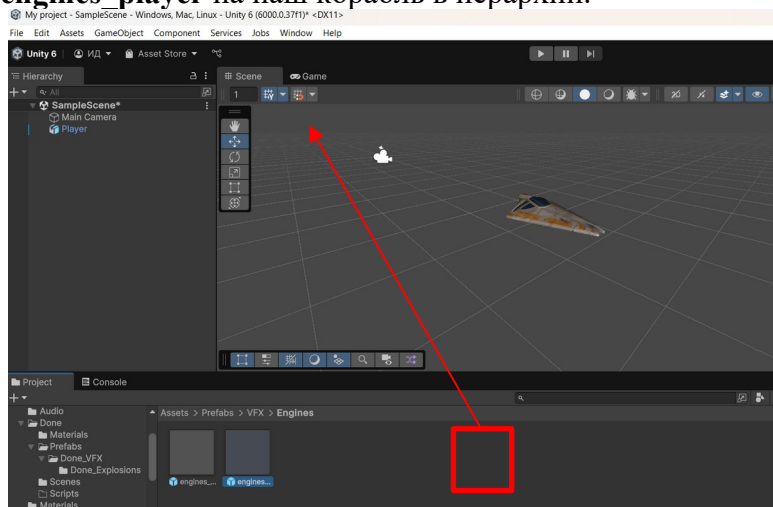
Подгоним **Capsule Collider** под размер корабля с помощью изменения свойств коллайдера, также поменяем **Direction** коллайдера на ось **Z**.



Давайте сделаем так, чтобы коллайдер покрывал наш корабль в точных пропорциях, заменив **Capsule Collider** на **Mesh Collider**. Устанавливаем флажок напротив **Convex** и **IsTrigger**, так как обработка столкновения с коллайдером будет запускать определенный скрипт, к примеру скрипт окончания игры после столкновения с астероидом.



В конце выполнения лабораторной работы необходимо сделать корабль немного эффектнее. Зайдем в папку **Prefabs – VFX – Engines** и перетащим элемент **engines_player** на наш корабль в иерархии.

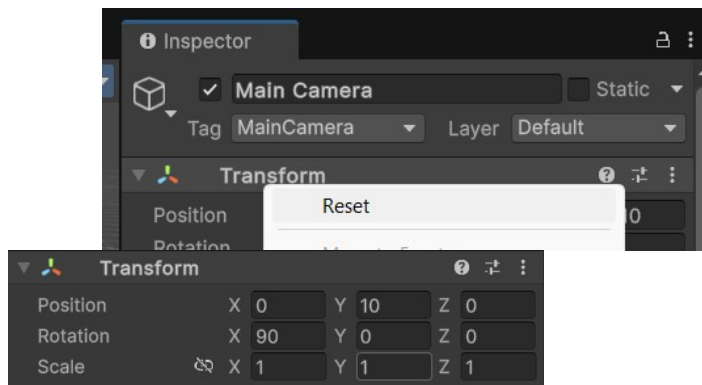


Вот такой результат получился.

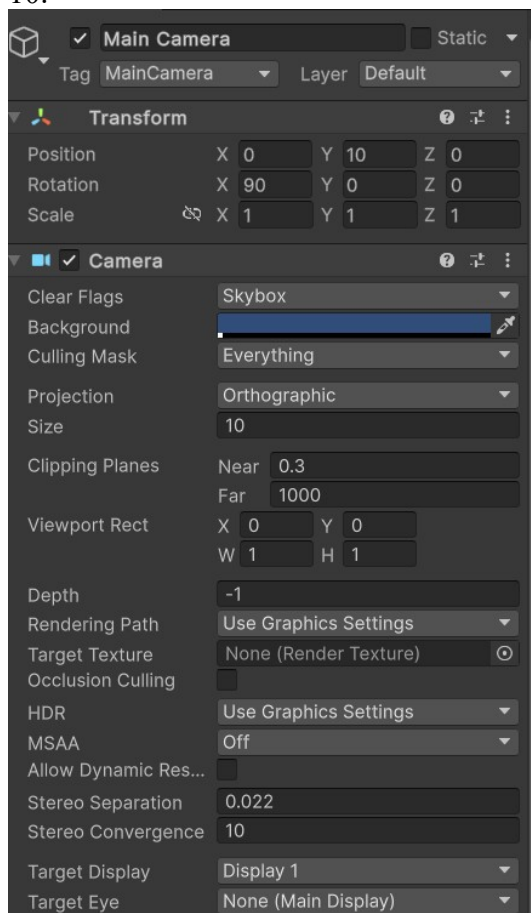


Лабораторная №4. Камера. Свет. Добавление фона.

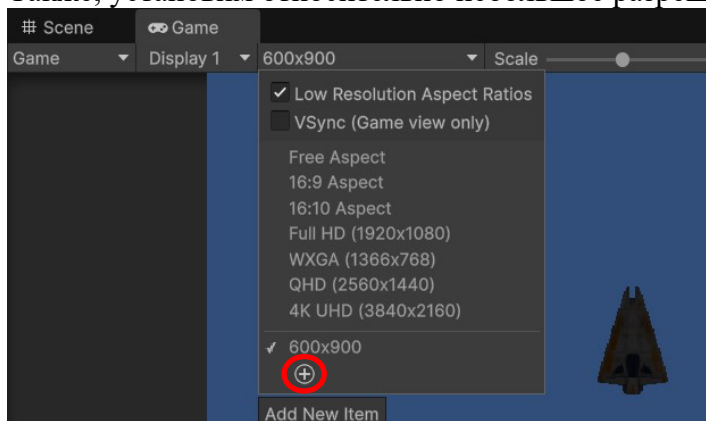
Игра будет выглядеть как top-down аркада, поэтому нужно поработать с позицией камеры, поместив ее над кораблём. Для этого, установим позицию камеры в исходное положение с помощью команды **Reset**, нажав ПКМ в окне свойств на **Transform**. Rotation по оси X установим на 90 градусов, чтобы камера была перпендикулярна кораблю. Значение позиции по оси Y установим на 10.



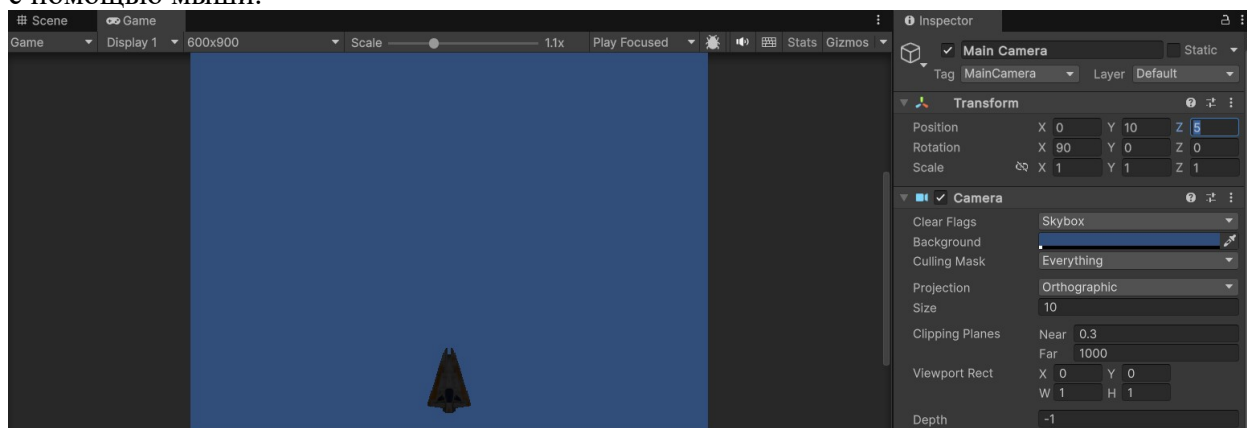
Теперь поработаем с типом камеры. Есть два типа: Perspective и Orthographic. Так как наша игра- аркада с видом сверху, не нужно использовать никакую перспективу. Выбираем тип Orthographic и устанавливаем размер отображения (Scale) примерно на 10.



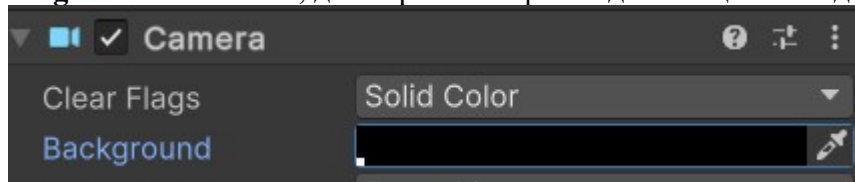
Также, установим относительно небольшое разрешение 600x900



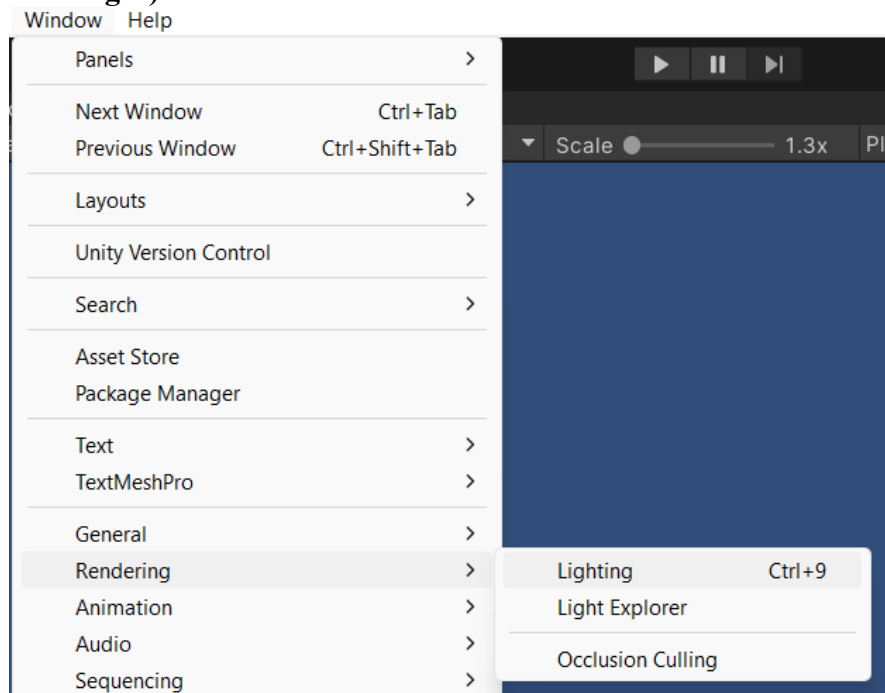
В начале игры, корабль должен находится на краю экрана, поэтому передвинем камеру по оси Z на 5 позиций, равной 5. Чтобы выбрать наиболее подходящую для вас позицию, можно открыть режим отображения **Game** и менять значение **Position** по оси Z с помощью мыши.



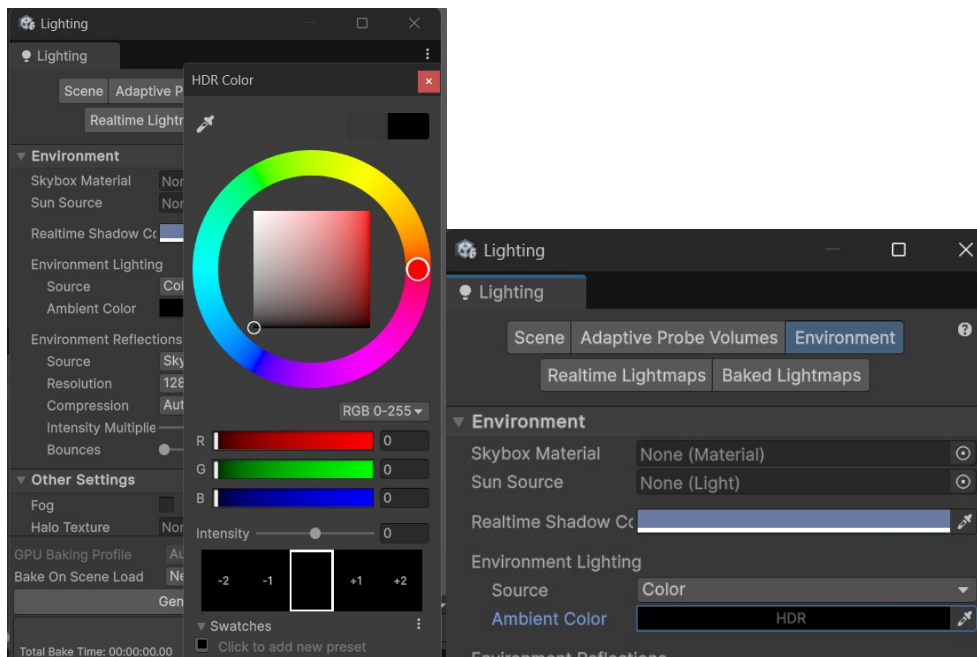
Сделаем задний фон более космическим. Для этого установим значение **Clear Flags** как **Solid Color**, для отрисовки фона единого цвета и сделаем Background черным.



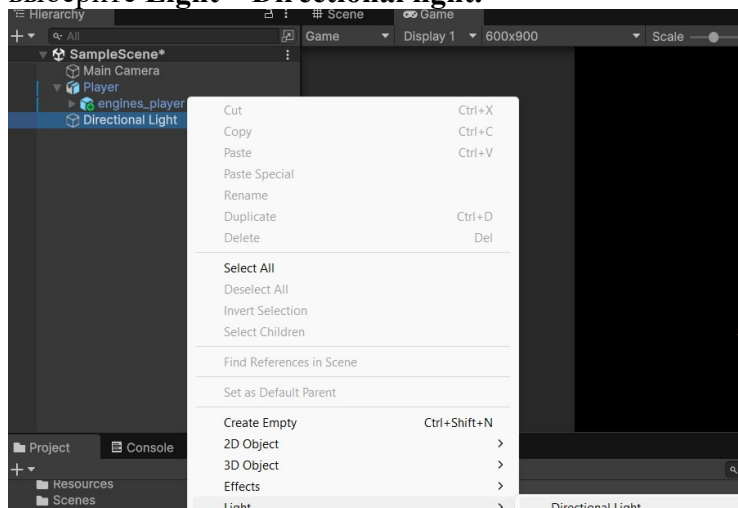
Добавим свет на нашу сцену. Перейдем в **Window – Rendering – Lightning – Lightning settings**. Обратите внимание на пункт **Environment – Ambient light**. Это освещение, распространяемое на сцену целиком, в отличие от направленного освещения (**Directional light**)



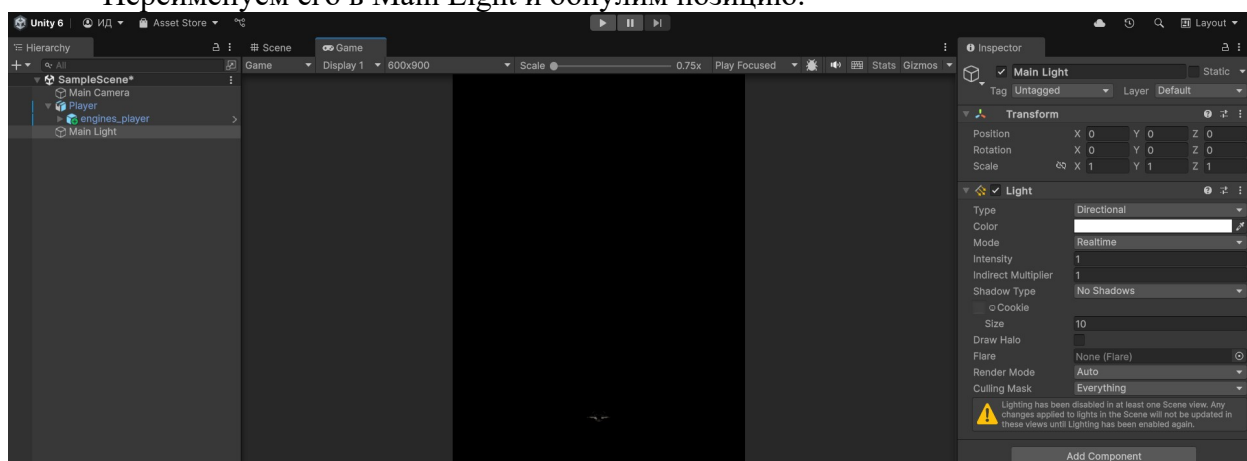
В **Ambient Color** поставим значение **Ambient light** на 0, т.е. передвинем ползунки RGB на нулевое значение.



Добавим в иерархию элемент **Directional light**. В **Hierarchy** нажмите на ПКМ и выберите **Light – Directional light**.

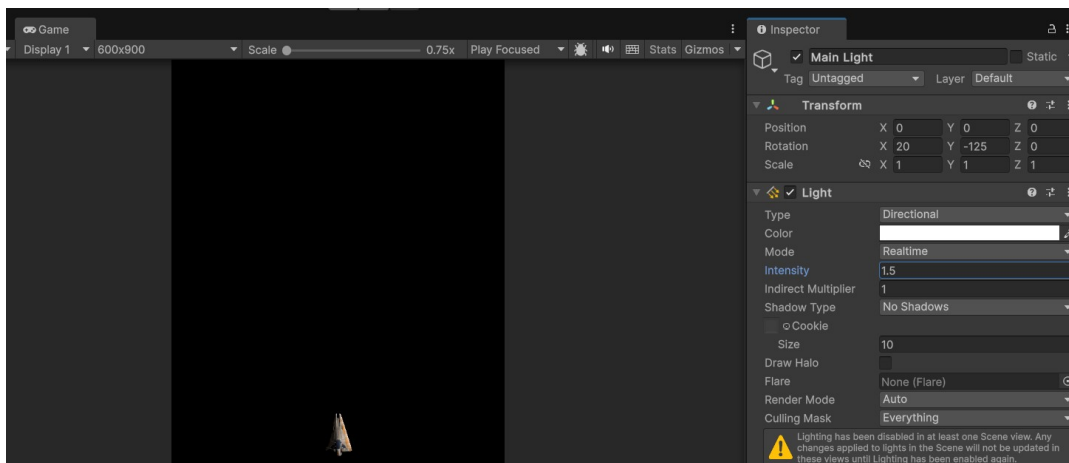


Переименуем его в **Main Light** и обнулим позицию.

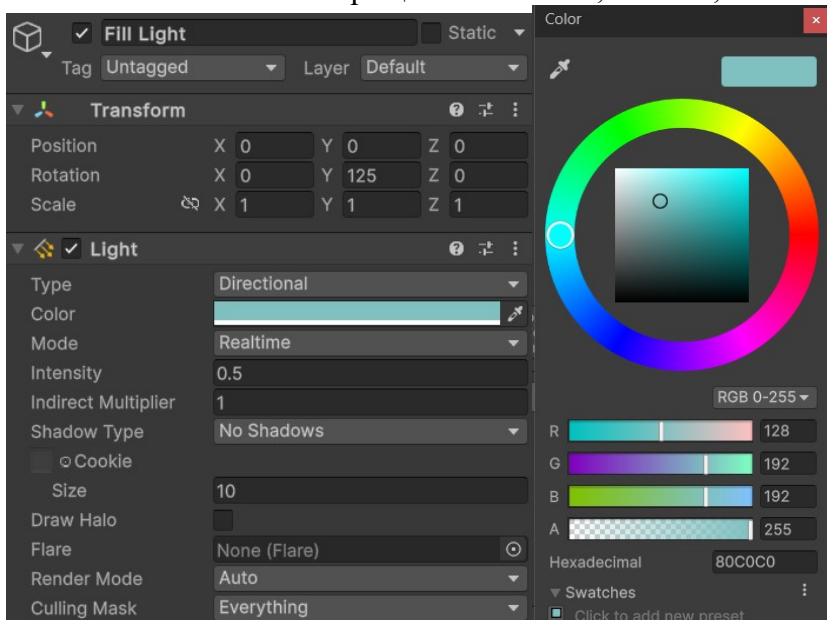


Теперь поэкспериментируем с вращением (Rotation) света. Остановимся на таких значениях: По оси X = 20, а по Y = -125.

Теперь добавим интенсивности свету. Установим на значение 1,5. Также можно изменить цвет, но мы оставим его белым по умолчанию.

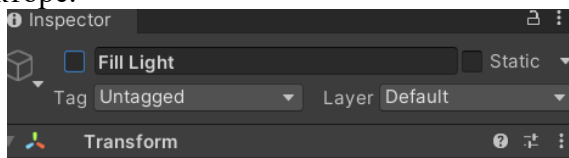


Добавим еще один **Directional light** путем копирования предыдущего. Для этого, установим курсор на нашем Main Light и нажмем сочетание клавиш Ctrl + D. Назовем новый объект Fill Light. Сбросим значения вращения и установим вращение по Y на 125. Однако, теперь кажется, что у нас два главных элемента освещения, исправим это уменьшением интенсивности. Поменяем цвет, установив значения каналов на 128, 192 и 192 и немного изменим вращение по оси X, скажем, на 5. Результат выглядит так:



Добавим последний **Directional light**, создав дубликат объекта Fill light и назовем его Rin Light.

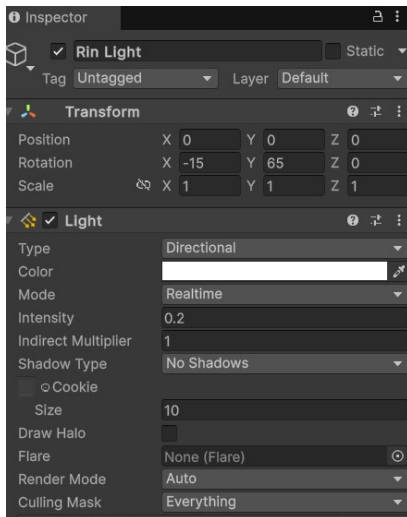
Выключим Fill Light для более удобной работы, для этого снимем флажок в инспекторе.



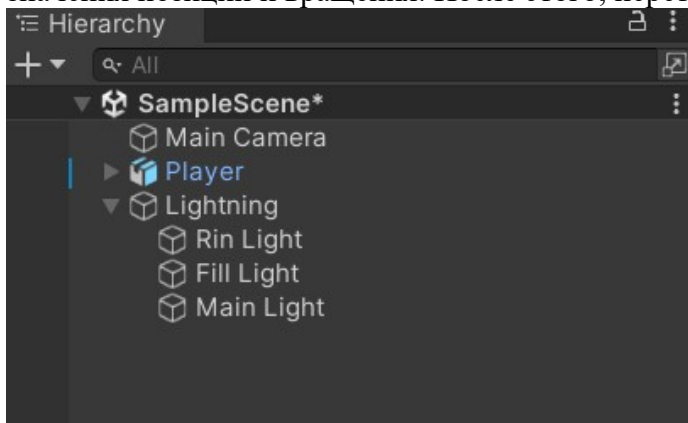
Сбросим значения **Transform** для нашего Rin Light.

Установим белый цвет по умолчанию и повернем его по оси Y на 65. Однако, теперь он выглядит как еще один Fill Light. Исправим это, установив значение вращения по X на -15 и интенсивность на 0,2.

Теперь включим Fill Light обратно и проверим результат нашей работы.



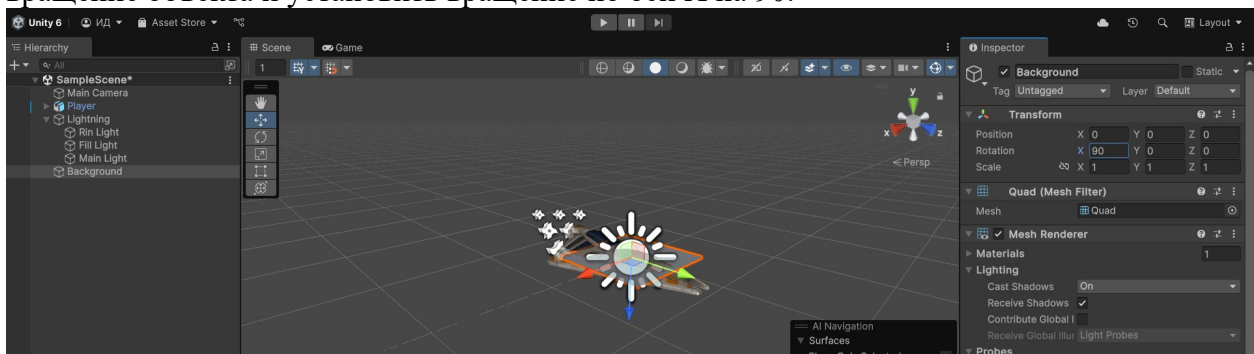
После окончания работы со светом, давайте организуем наше рабочее пространство, а именно иерархию. Для этого, создадим отдельную пустую «папку» для объектов освещения. Это делается, путем создания пустого игрового объекта в иерархии. ПКМ – create Empty. Переименуем его в Lightning. Обязательно обнуляем значения позиции и вращения. После этого, перетаскиваем в него наш «свет».



Лабораторная работа №5. Фон (Background)

Мы создаем космическую игру и космос...черный. Но, это скучновато, так что давайте сделаем красивый фон. Для начала отключим наш корабль, пока он нам не понадобится.

Создадим в иерархии элемент Quad (ПКМ- 3D Object- Quad) и назовем его Background. Чтобы наша камера смогла «видеть» фон, необходимо сбросить позицию и вращение объекта и установить вращение по оси X на 90.

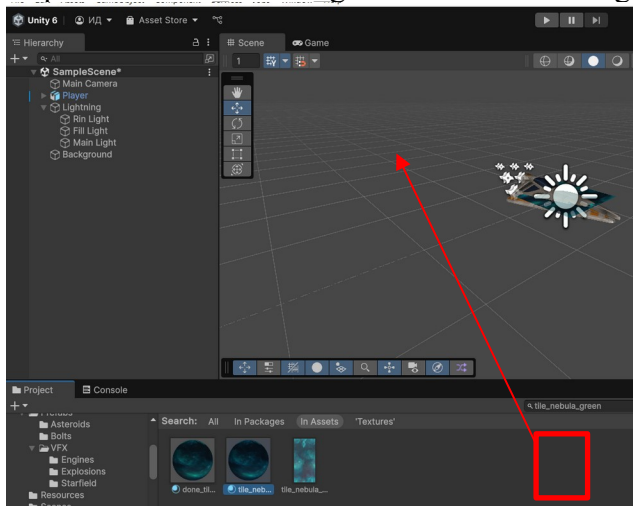


Квадрат содержит компонент Mesh Collider, который нам не понадобится, так что удалим его.

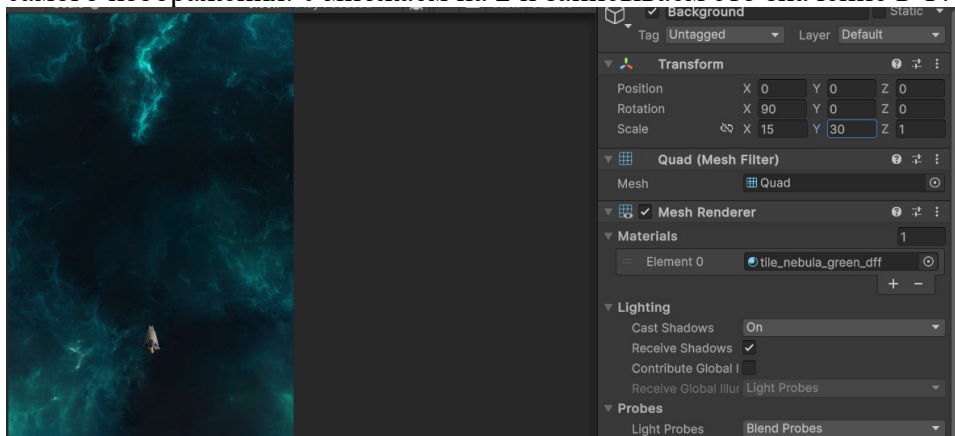
Давайте добавим текстуру для нашего квадрата, перейдем в папку Textures, и выберем файл под названием `tile_nebula_green`.

Обратим внимание на разрешение текстуры: 1024x2048. По сравнению с разрешением нашей игры, это намного больше, чем нам нужно. Обратим внимание на пропорцию 1:2, запомним ее.

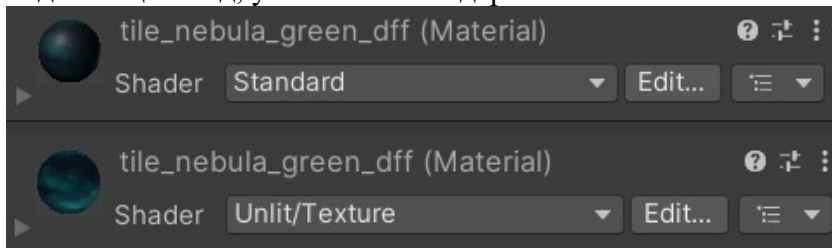
Давайте используем самый простой вариант добавления текстуры объекту. Перетащим `tile_nebula_green` на объект `Background`.



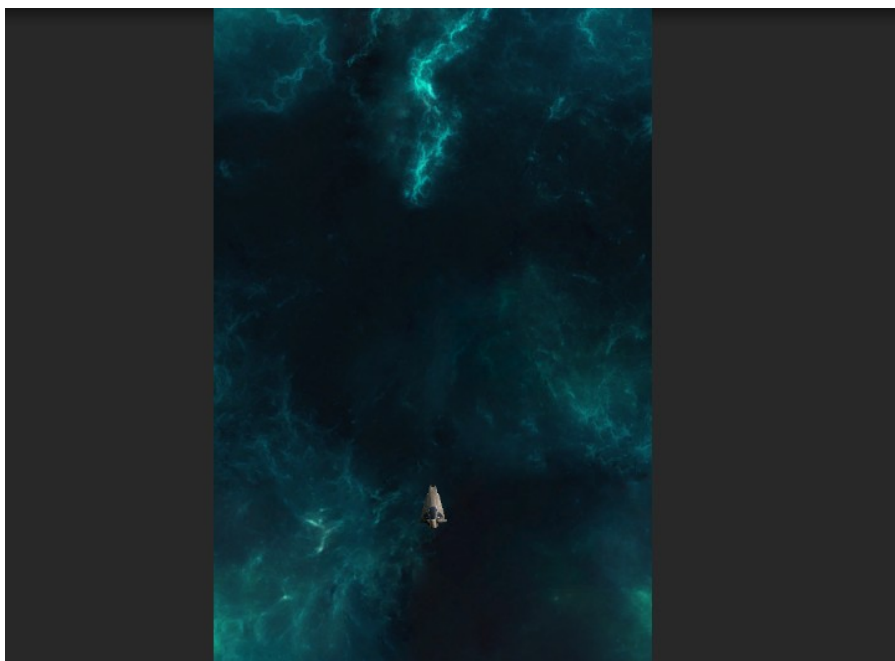
Что же делать с размером? Изменить значение `Scale`. Откроем отображение `Game` и установим значение по `X` равное 15. Установив такое значение, вспомним о пропорции самого изображения. Умножаем на 2 и записываем это значение в `Y`.



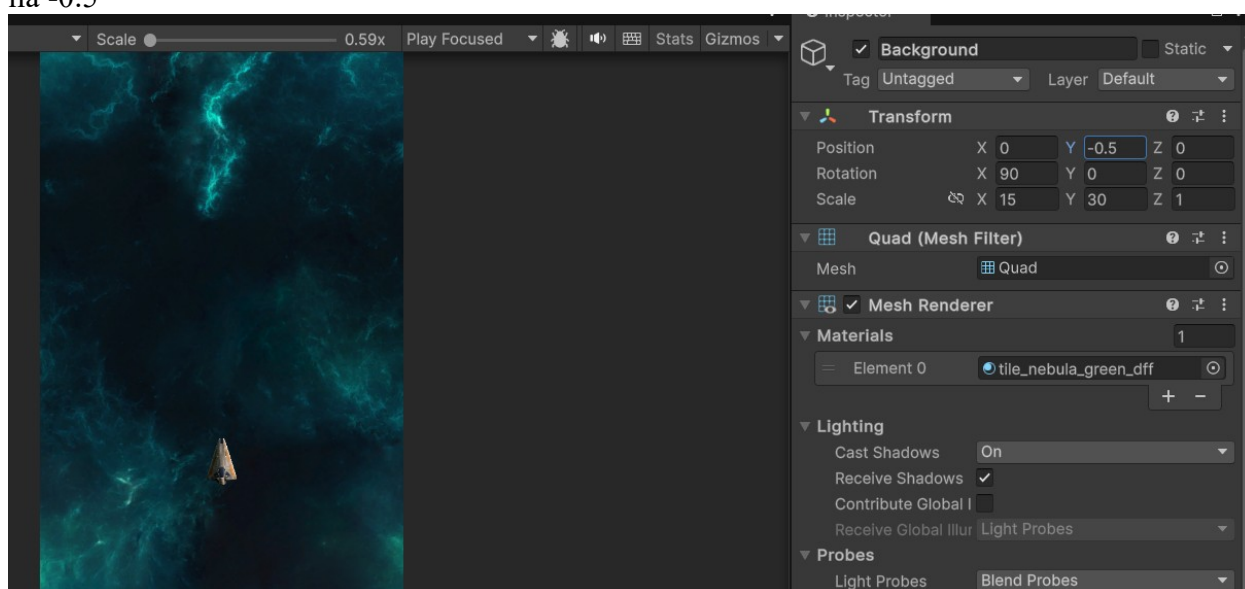
Фон все еще темноват - все дело в системе освещения. Чтобы привести его в надлежащий вид, установим шейдер **Unlit – Texture**.



Теперь наш фон независим от системы освещения и выглядит намного «живее».



Вернем корабль в активное состояние, но что-то не так. Все дело в том, что задний фон и корабль, находятся в одной плоскости. Передвинем задний фон по оси Y на -0.5



Видеоматериал:

1) <https://www.youtube.com/watch?v=s0nkSwDyeQ0&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=4>

2) <https://www.youtube.com/watch?v=ntVUkaCufJg&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=5>

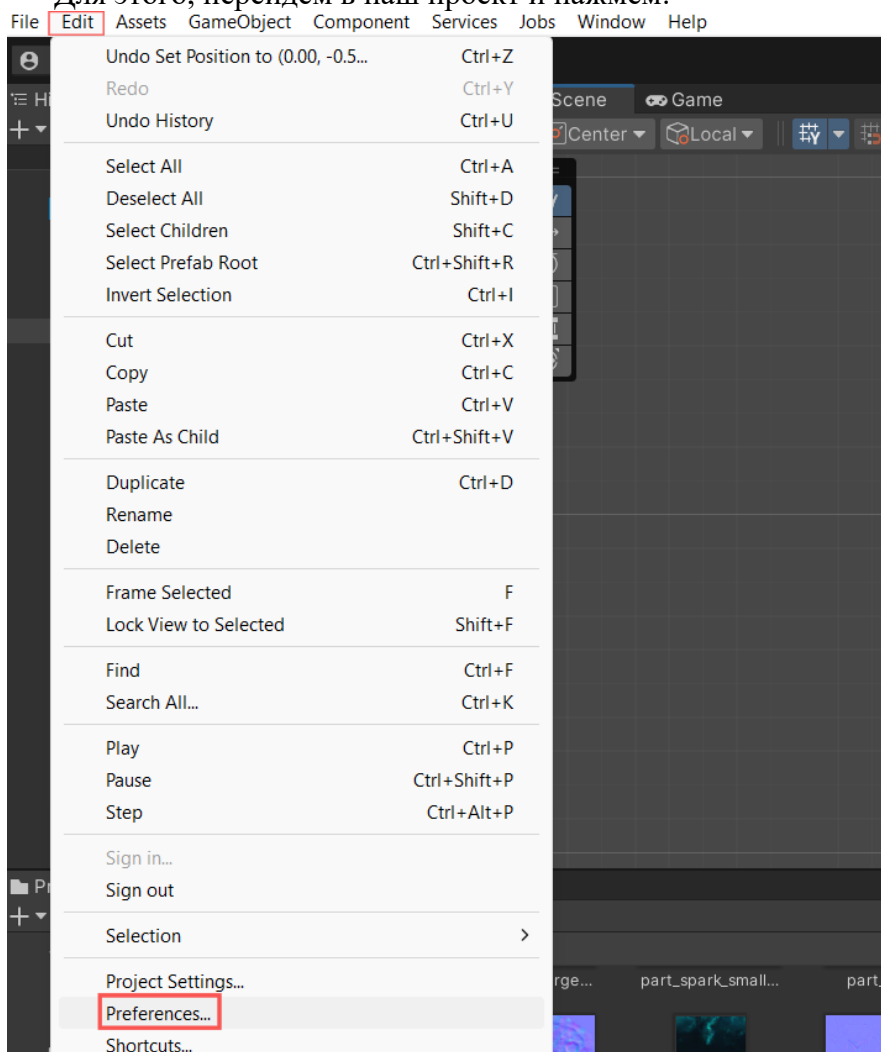
Лабораторная №6. Передвижение игрока.

Перейдем к написанию кода для осуществления передвижения игрока. Писать мы будем на языке программирования C#.

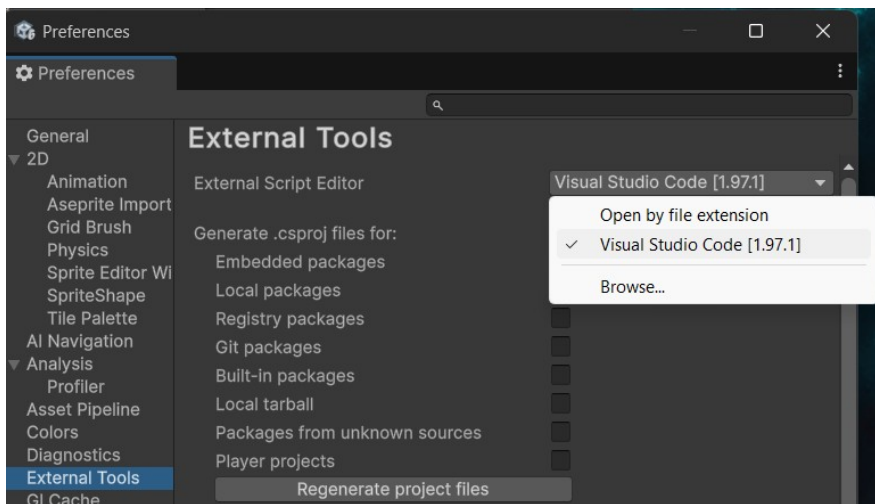
Для удобства написания кода на C# воспользуемся сторонним редактором исходного кода под названием: “**Visual Studio Code**”. Скачать данные ПО можно по ссылке: <https://code.visualstudio.com/download>

После установки необходимо связать **Visual Studio Code** и **Unity** для того, чтобы при открытии компонентов **Scripts** в **Unity**, наш файл с кодом автоматически открывался в **Visual Studio Code**.

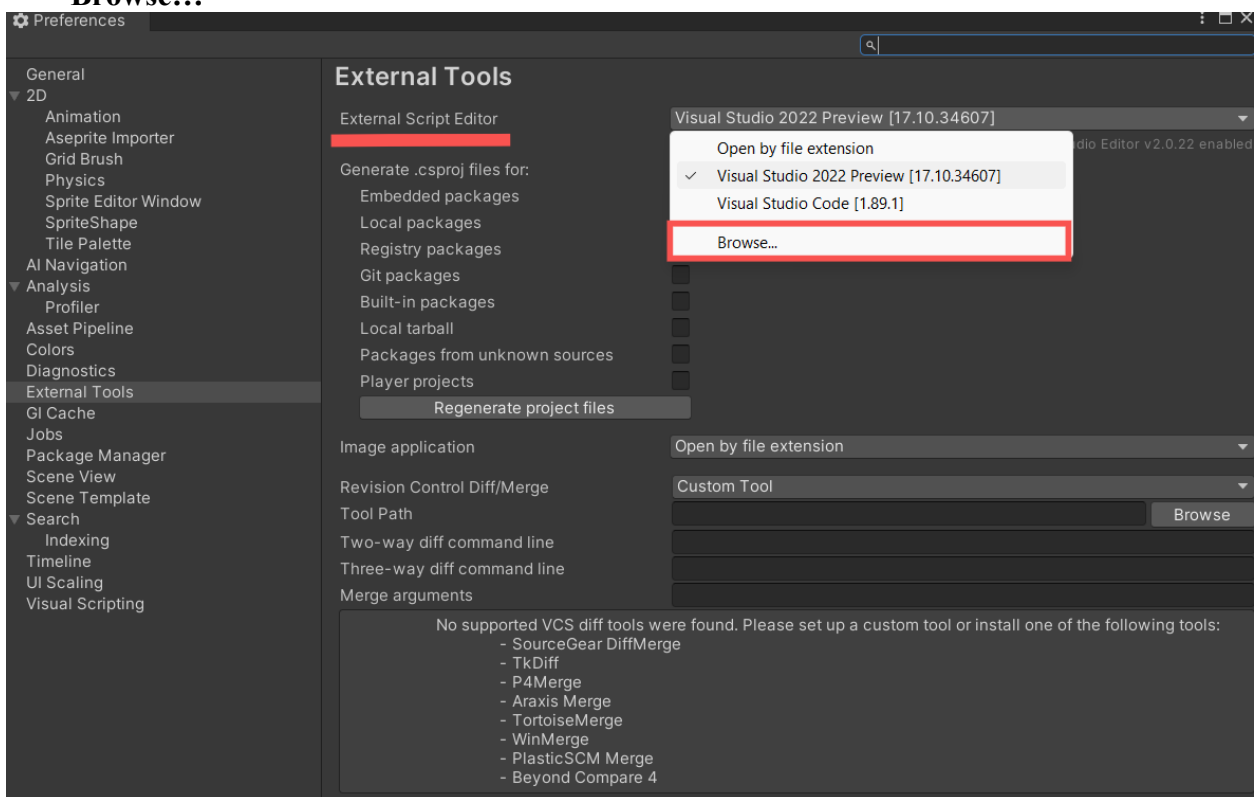
Для этого, перейдем в наш проект и нажмем:



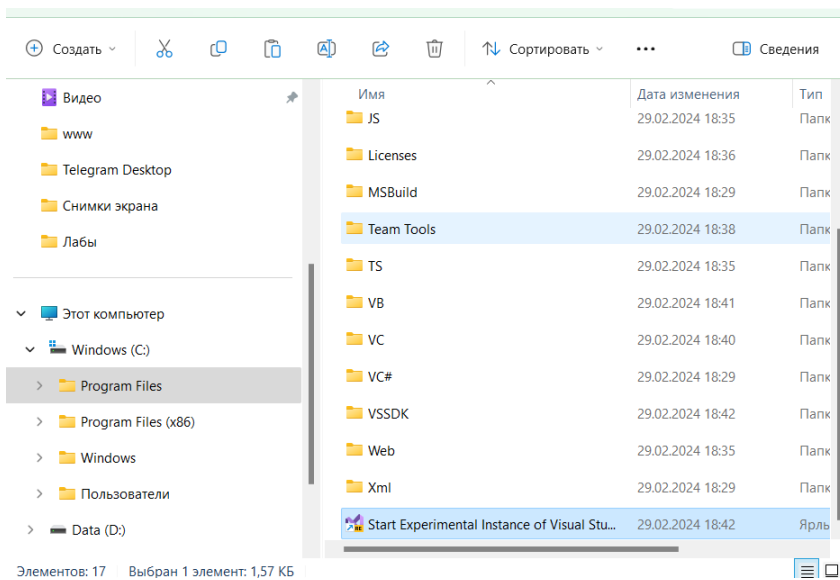
- 1)
- 2) Далее, у вас откроется окно **Unity Preferences**. Перейдем во вкладку **External Tools**. Здесь в строке **External Scripts Editor**, необходимо выбрать ранее установленный редактор кода.



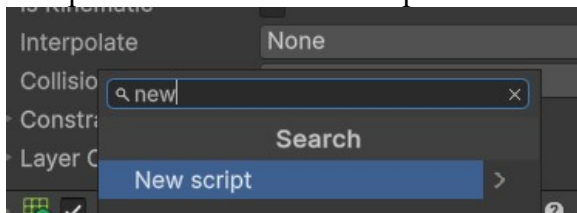
3) Если же, вы не видите ваш редактор в предложенном списке, нажмем на кнопку “Browse...”



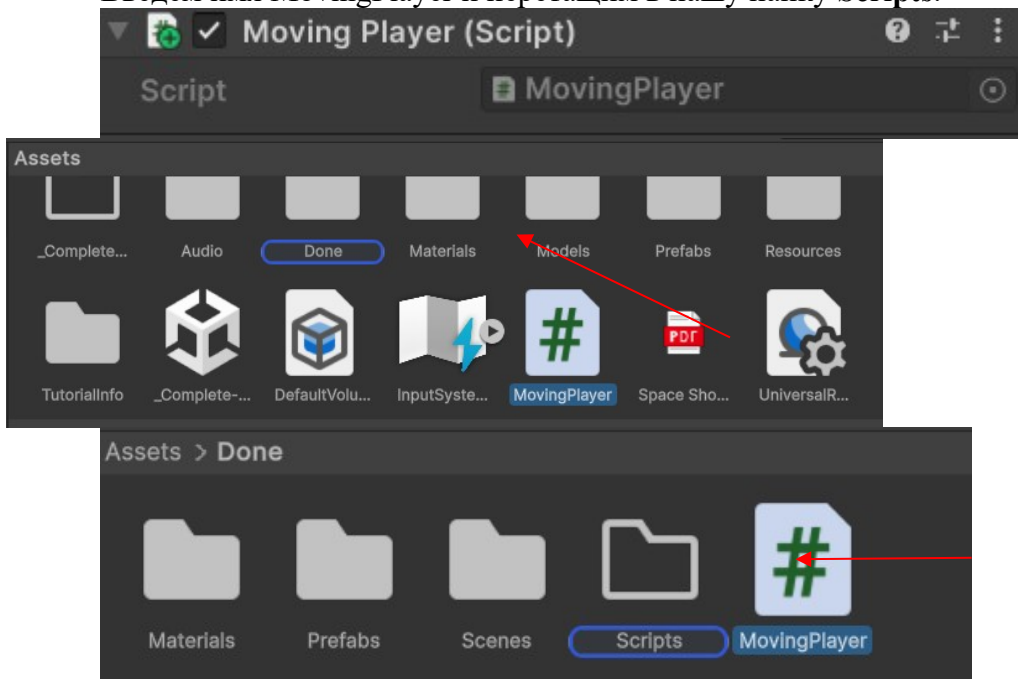
4) Найдем папку, в которой хранятся файлы **Visual Studio Code** и выберем файл открытия редактора кода – самостоятельно.



Теперь Unity сможет открывать файлы с кодом в необходимом нам приложении. Теперь, добавим к нашему кораблю **Player** C# скрипт. Для этого нажмем на кнопку Add Component в окне свойств корабля.



Введем имя MovingPlayer и перетащим в нашу папку **Scripts**.



Лабораторная работа №7. Скрипты

Откроем созданный нами скрипт. Методы Start и Update нам не понадобятся, так что удаляем их. Мы будем приводить корабль в движение, используя «физику», так что нам понадобится метод FixedUpdate.

Наш код будет выглядеть следующим образом:
[using System.Collections;](#)

```

using System.Collections.Generic;
using UnityEngine;

public class MovingPlayer : MonoBehaviour
{
    private void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
    }
}

```

Мы создали две вещественные переменные: moveHorizontal и moveVertical, которые принимают значения при нажатии клавиш, определенных в Input Manager.

Используя скрипты, мы можем управлять свойствами компонентов, привязанных к игровым объектам. Например, компонент Rigidbody.

Но перед тем как обращаться к компоненту, нам нужно ввести для него переменную, это делается через метод Start().

```

public class MovingPlayer : MonoBehaviour{
    1 reference
    private Rigidbody rb;
    0 references
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
}

```

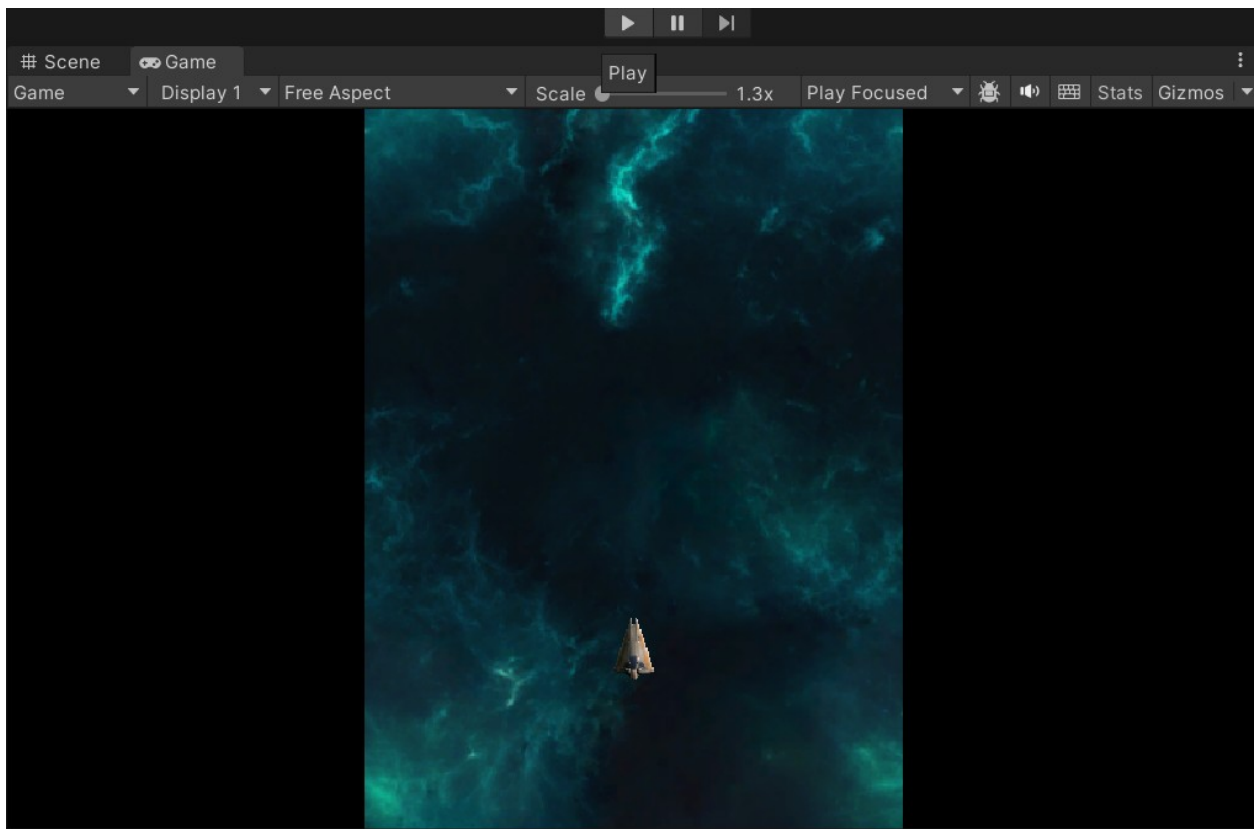
Теперь перейдем непосредственно к осуществлению движения. Создадим переменную Vector3 под названием movement и приравняем к ней новый экземпляр Vector3, т.к. корабль будет двигаться лишь в плоскости X и Z, то на место Y мы поставим 0.

```

1  using System.Collections;
2  using System.Collections.Generic; // Исправлено!
3  using UnityEngine;
4
5  0 references
6  public class MovingPlayer : MonoBehaviour
7  {
8      2 references
9      private Rigidbody rb;
10
11      0 references
12      void Start()
13      {
14          rb = GetComponent<Rigidbody>();
15      }
16
17      0 references
18      private void FixedUpdate()
19      {
20          float moveHorizontal = Input.GetAxis("Horizontal");
21          float moveVertical = Input.GetAxis("Vertical");
22          Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
23          rb.linearVelocity = movement;
24      }
25  }

```

Давайте сохраним наш код и сохраним нашу сцену (для удобства, можно воспользоваться комбинацией клавиш Ctrl + S) и перейдем к игровому режиму.



Попробуем передвигать корабль с помощью стрелок или кнопок W, A, S, D. Наш корабль движется, но движется не с той скоростью, с которой хотелось бы. Это происходит из-за того, что `Input.GetAxis` возвращает значение от 0 до 1, соответственно, корабль передвигается лишь на 1 юнит в секунду. Для изменения скорости добавим в наш скрипт новую переменную, значение которой можно будет менять прямо в инспекторе, без прямого обращения к скрипту.

Возвращаемся к нашему скрипту и создаем публичную (для редактирования из инспектора) переменную `speed`.

```
5 public class MovingPlayer : MonoBehaviour
6 {
```

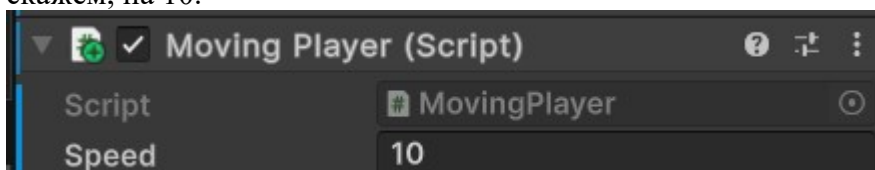
```
7     private Rigidbody rb;
```

```
8     public float speed;
```

Для того, чтобы эта переменная могла влиять на перемещение, умножаем `movement` на переменную `speed`.

```
private void FixedUpdate()
{
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.linearVelocity = movement;
    rb.velocity = movement*speed;
}
```

Сохраним скрипт и перейдем в окно инспектора. Установим значение переменной, скажем, на 10.



Запустим проект и проверим работу скрипта.

Следующий нюанс, который необходимо исправить, это вылет корабля за пределы отображения камеры. Создадим класс Boundary, в котором будем хранить значения границ по максимуму и минимуму.

```
using UnityEngine;
[System.Serializable]
0 references
public class Boundary
{
    0 references | 0 references | 0 references | 0 references
    public float xMin, xMax, zMin, zMax;
}
```

Что означает [System.Serializable]? Это выражение позволяет нам обратиться к классу непосредственно из инспектора.

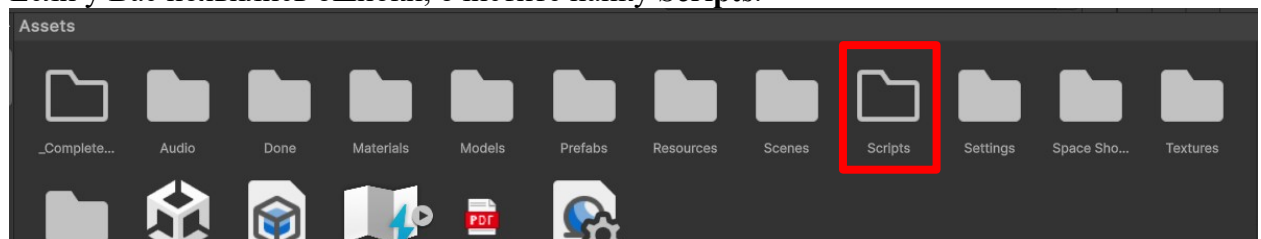
Создадим экземпляр класса

```
public class MovingPlayer : MonoBehaviour
{
    3 references
    private Rigidbody rb;
    1 reference
    public float speed;
    0 references
    public Boundary boundary;
    0 references
    void Start()
}
```

Ограничим перемещение корабля, при помощи компонента Rigidbody

```
rb.linearVelocity = movement*speed;
rb.position = new Vector3
(
    Mathf.Clamp( rb.position.x, boundary.xMin, boundary.xMax),0.0f,
    Mathf.Clamp( rb.position.z, boundary.zMin, boundary.zMax)
);
```

Если у Вас появились ошибки, очистите папку **Scripts**.



Сохраним наш код.

```
using System.Collections;
using System.Collections.Generic; // Исправлено!
using UnityEngine;
[System.Serializable]
public class Boundary
{
    public float xMin, xMax, zMin, zMax;
}

public class MovingPlayer : MonoBehaviour
{
    private Rigidbody rb;
```

```

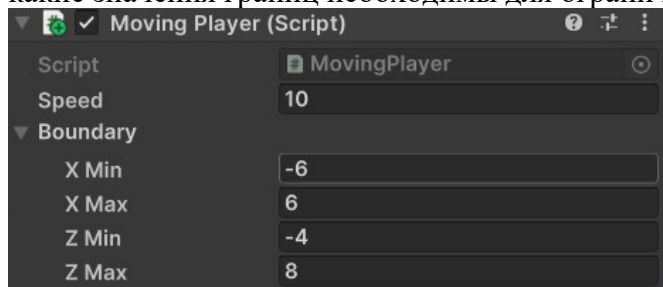
public float speed;
public Boundary boundary;

void Start()
{
    rb = GetComponent<Rigidbody>();
}

private void FixedUpdate()
{
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.linearVelocity = movement;
    rb.linearVelocity = movement*speed;
    rb.position = new Vector3
    (
        Mathf.Clamp(rb.position.x, boundary.xMin,
boundary.xMax),0.0f,Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
    );
}
}

```

Перейдем непосредственно к окну свойств. Экспериментальным путем вычислим какие значения границ необходимы для ограничения передвижения.



Запустите игровой режим и проверьте работу скрипта.

Мы закончили работу с перемещением корабля, теперь давайте добавим небольшой косметический элемент – поворот корабля при нажатии кнопки перемещения вбок.

Создадим публичную переменную, которая будет отвечать за коэффициент поворота.

```

16 public float tilt;

```

И напомним сам код в том же методе FixedUpdate():

```

    rb.rotation = Quaternion.Euler(0.0f,0.0f,rb.linearVelocity.x*-tilt);
using System.Collections;
using System.Collections.Generic; // Исправлено!
using UnityEngine;
[System.Serializable]

public class Boundary
{
    public float xMin, xMax, zMin, zMax;
}

```

```

}

public class MovingPlayer : MonoBehaviour
{
    private Rigidbody rb;
    public float speed;
    public Boundary boundary;
    public float tilt;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    private void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        rb.linearVelocity = movement;
        rb.linearVelocity = movement*speed;
        rb.rotation = Quaternion.Euler(0.0f,0.0f,rb.linearVelocity.x*-tilt);
        rb.position = new Vector3
        (
            Mathf.Clamp(rb.position.x, boundary.xMin,
boundary.xMax),0.0f,Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
        );
    }
}

```

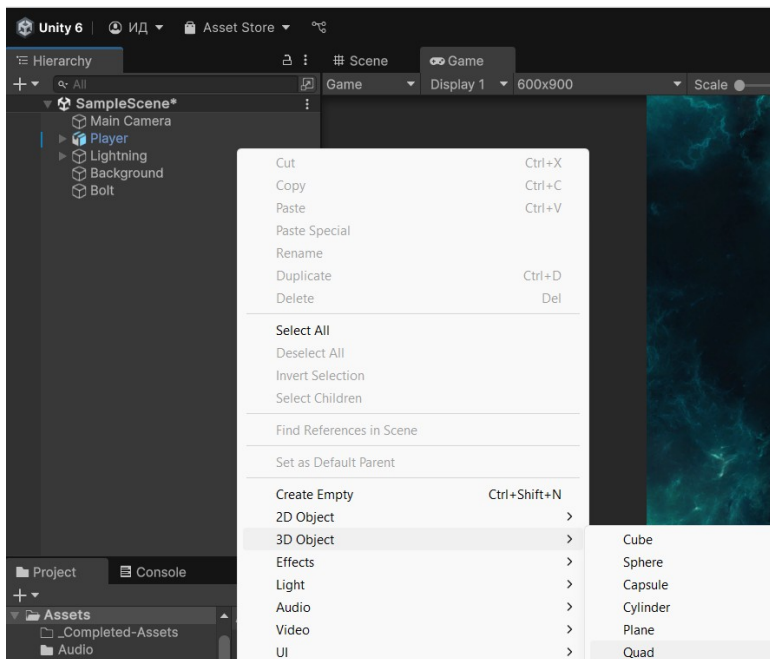
В окне свойств (инспекторе) установим значение переменной tilt, скажем, на 4. Его можно менять по вашему усмотрению.

Видеоматериалы:

1) <https://www.youtube.com/watch?v=9TqYUS0bhcc&list=PLDZieabyoJfWj5ASktozPEwNlZw30VTI&index=6>

Лабораторная №8. Выстрелы.

Создадим пустой игровой объект (ПКМ в области **Hierarchy** и выбираем **Create Empty**) и назовем его Bolt. Сбросим исходные значения позиции в окне свойства. Теперь, создадим Quad для хранения изображения визуального эффекта и назовем его VFX.



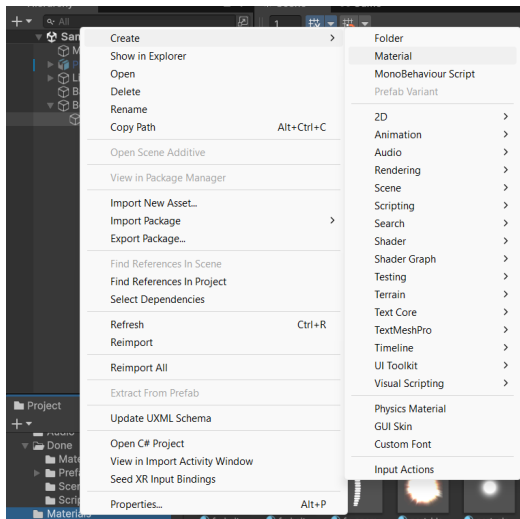
Также сбросим его позицию и перетащим на Bolt.



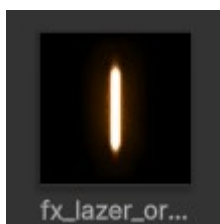
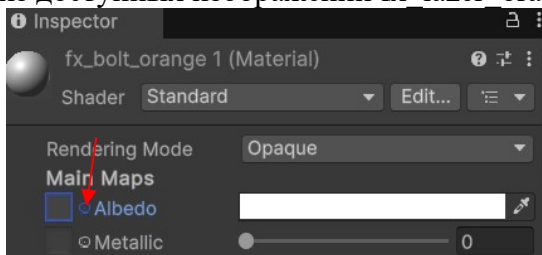
Перевернем наш VFX на 90 градусов по оси X, для корректного отображения на камере.



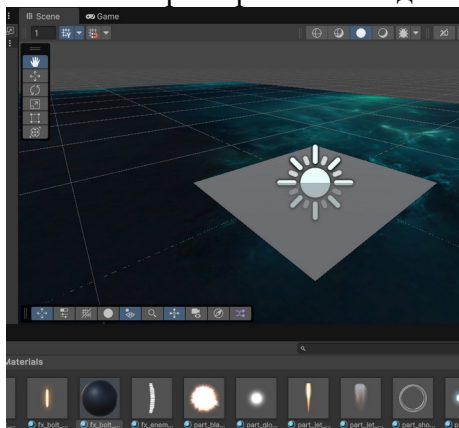
Создадим для нашего снаряда новый материал в папке. Нажимаем ПКМ на папке **Materials - Create - Material**. Пишем название для материала **fx_bolt_orange**.



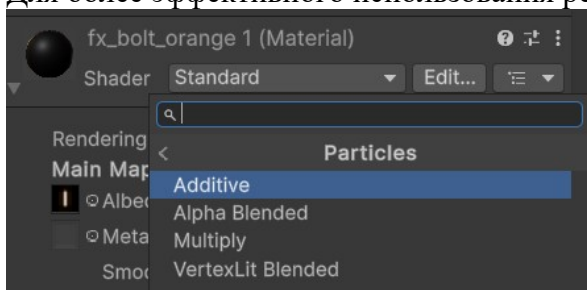
В открытой папке нажимаем рядом со свойством **Albedo** на кружок и выбираем из доступных изображений **fx_lazer_orange**.



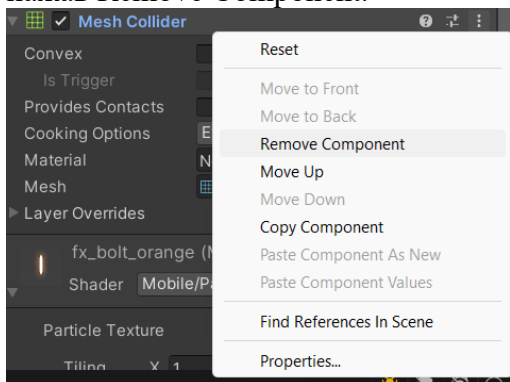
Теперь перенесем созданный материал на квадрат.



Снаряд смотрится не очень привлекательно, так что поработаем над ним. Вместо стандартного шейдера, применим на снаряд более подходящий. переходим в окно свойств VFX, спускаемся вниз и выбираем **Mobile - Particles – Additive**. Почему Mobile? Для более эффективного использования ресурсов игры.

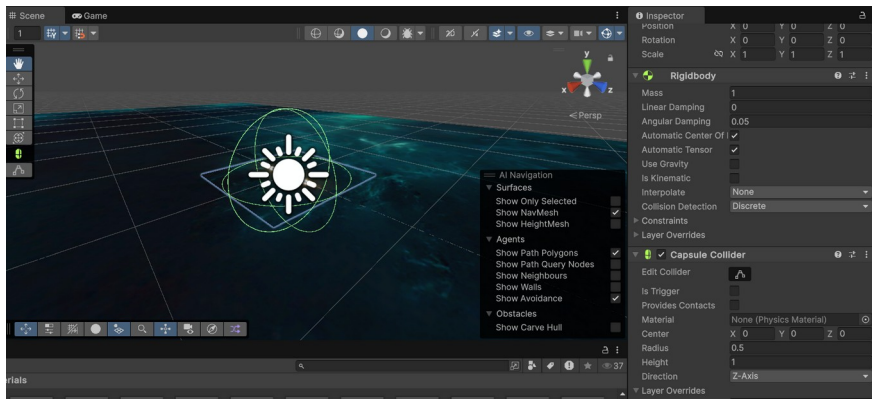


Поработаем над физикой снаряда. Перейдем к родительскому объекту Bolt в окно свойств и добавим к нему компонент Rigidbody. Снимем галочку с пункта Use Gravity, нам не нужно, чтобы снаряд падал вниз. Для обработки столкновений, нам необходим коллайдер, но перед этим следует удалить Mesh Collider с дочернего объекта VFX, нажав Remove Component.



Добавим в родительский объект Bolt компонент Capsule Collider. Может показаться, что это сфера, но это не так. Это несколько кругов, пересекающихся между собой, между которыми есть пространство. Попробуем приблизить коллайдер максимально к форме сферы.

Поменяем направление коллайдера на ось Z, как мы делали это в предыдущих лабораторных работах. Изменим ширину и размер в соответствии с размером снаряда.



Осталось поставить галочку напротив свойства Is Trigger. Добавим скрипт к объекту Bolt и назовем его Mover, нам нужно написать базовую логику движения снаряда.

Откроем скрипт. Снаряд должен двигаться сразу после добавления его на сцену. Напишем строки кода, инициализируем переменную rb как компонент RigidBody.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Mover : MonoBehaviour{
    private Rigidbody rb;
    private void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
}
```

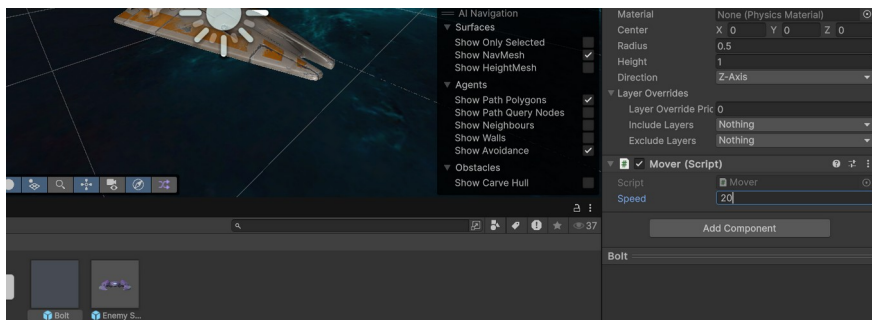
Снаряд должен двигаться по оси Z, в направлении от корабля игрока. Также, нам понадобится скорость снаряда и возможность менять ее из окна свойств – добавим публичную переменную вещественного типа speed.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Mover : MonoBehaviour{
6     1 reference
7     public float speed;
8     2 references
9     private Rigidbody rb;
10    0 references
11    private void start()
12    {
13        rb = GetComponent<Rigidbody>();
14        rb.velocity = transform.forward*speed;
15    }
16 }
```

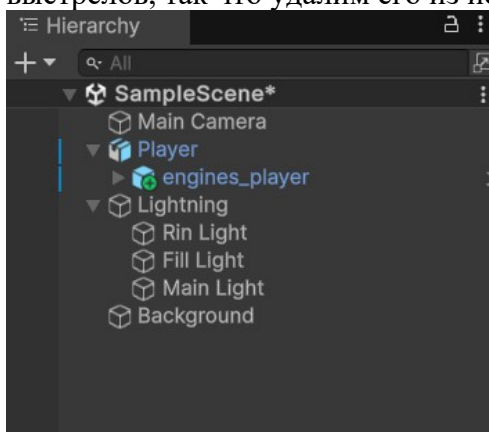
Velocity приводит объект в движение, а строка transform.forward берет значение позиции по положительному направлению оси.

Наш корабль будет использовать множество снарядов, так что нам необходимо сделать из Bolt префаб (prefab). Перенесем Bolt из иерархии в папку под названием Prefabs.

Значение скорости установим, приняв во внимание скорость нашего корабля. С какой скоростью он движется? 10? Снаряд должен двигаться намного быстрее, пусть будет 20.



Объект снаряда не должен находиться на сцене в начале игры, лишь во время выстрелов, так что удалим его из иерархии.



Как проверить работу снаряда без написания скрипта на стрельбу? Включим игровой режим и перетащим Bolt в иерархию. Если снаряд движется медленно, кликните на Bolt в панели Prefabs и измените его скорость вне игрового режима.

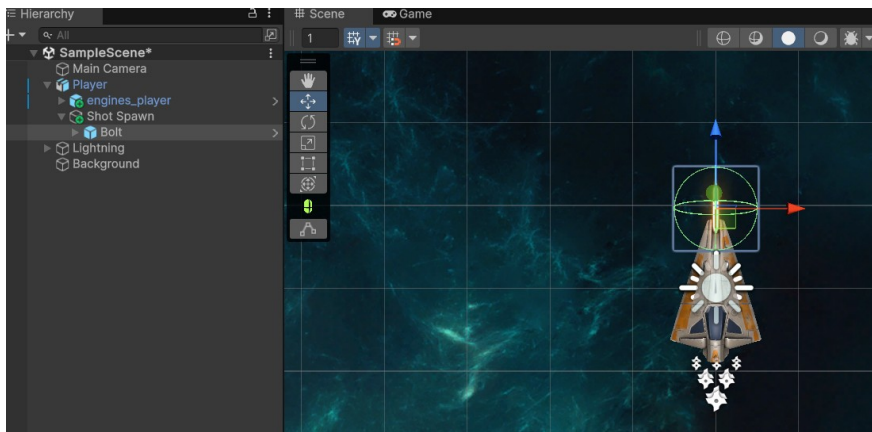
Лабораторная работа №9. Стрельба

Перейдем к редактированию скрипта **корабля**. Какие переменные нам понадобятся? Во-первых, сам игровой объект выстрела, во-вторых, точка спауна выстрела и, в-третьих, частота стрельбы.

```
17 | public GameObject shot;
    | 2 references
18 | public Transform shotSpawn;
    | 1 reference
19 | public float fireRate;
```

Как игра узнает откуда должен полететь выстрел? Необходимо добавить точку спауна. Создадим в иерархии пустой игровой объект и назовем его Shot Spawn. Переместим его на объект Player, тем самым присвоив его как дочерний объект.

Сдвинем точку спауна на нос корабля, и перетащим Bolt, для того чтобы проверить подходит ли эта позиция.



Подходит. Не забываем удалить Bolt.

Вернемся к скрипту.

Напишем метод Update() и напишем представление объекта выстрела таким образом, используя наши переменные:

```
private void Update()
{
    Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
}
```

(Если вам не понятен тот или иной отрезок кода, всегда можно обратиться к документации, также можно нажать клавишу Ctrl и кликнуть по непонятной конструкции)

Если мы запустим игру сейчас, то можем увидеть выстрелы, запускающиеся каждый фрейм. Выстрелы должны происходить по нажатию, так что нам необходимо использовать условный оператор, который будет проверять нажата ли та или иная клавиша.

Введем еще одну приватную вещественную переменную nextFire, с помощью нее мы будем проверять частоту выстрелов, чтобы это значение не превышало какого-то числа.

```
11 public class MovingPlayer : MonoBehaviour
12 {
13     8 references
14     private Rigidbody rb;
15     1 reference
16     public float speed;
17     4 references
18     public Boundary boundary;
19     1 reference
20     public float tilt;
21     1 reference
22     public GameObject shot;
23     2 references
24     public Transform shotSpawn;
25     1 reference
26     public float fireRate;
27     2 references
28     private float nextFire;
29     void Start()
30     {
31         rb = GetComponent<Rigidbody>();
32     }
33     0 references
34     private void Update()
35     {
36         if(Input.GetButton("Fire1") && Time.time > nextFire)
37         {
38             nextFire = Time.time + fireRate;
39             Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
40         }
41     }
42 }
```

```
using System.Collections;
using System.Collections.Generic; // Исправлено!
using UnityEngine;
[System.Serializable]
```

```
public class Boundary
{
    public float xMin, xMax, zMin, zMax;
}
```

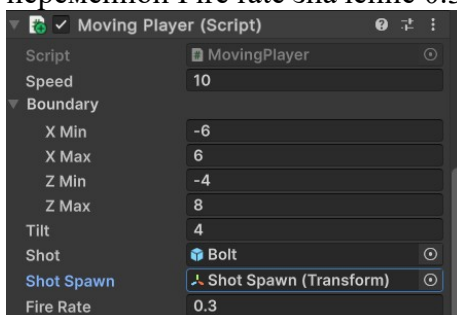
```

public class MovingPlayer : MonoBehaviour
{
    private Rigidbody rb;
    public float speed;
    public Boundary boundary;
    public float tilt;
    public GameObject shot;
    public Transform shotSpawn;
    public float fireRate;
    private float nextFire;
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
    private void Update()
    {
        if(Input.GetButton("Fire1") && Time.time > nextFire)
        {
            nextFire = Time.time + fireRate;
            Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
        }
    }

    private void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        rb.linearVelocity = movement;
        rb.linearVelocity = movement*speed;
        rb.rotation = Quaternion.Euler(0.0f,0.0f,rb.linearVelocity.x*-tilt);
        rb.position = new Vector3
        (
            Mathf.Clamp(rb.position.x, boundary.xMin,
boundary.xMax),0.0f,Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
        );
    }
}

```

Сохраним скрипт и перейдем в инспектор корабля. В инспекторе, в компоненте MovingPlayer, перетащим на переменную Shot объект Bolt, т.е. предадим выстрелу вид созданному нами снаряду. Тоже самое сделаем со спавном выстрела. Присвоим переменной Fire rate значение 0.3 и проверим работу скрипта.



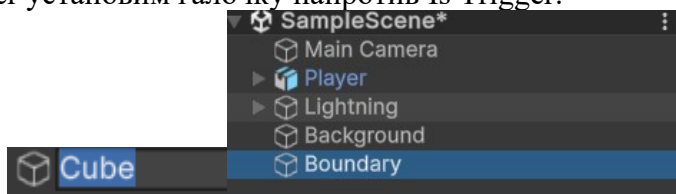
Видеоматериалы:

- 1) https://www.youtube.com/watch?v=h_CzOZaA0Zk&list=PLDZieabyoJfWj5ASktozPEwNlZw30VTI&index=7
- 2) <https://www.youtube.com/watch?v=jCbAq3qBY58&list=PLDZieabyoJfWj5ASktozPEwNlZw30VTI&index=9>

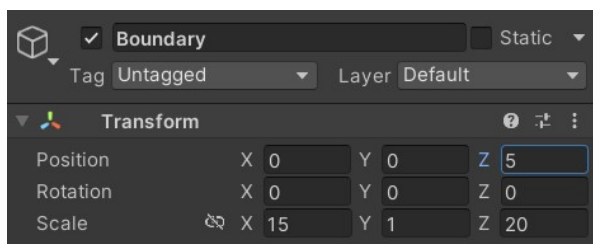
Лабораторная работа №10. Границы.

Мы создали выстрелы, но есть проблема: игровые объекты выстрелов каждый раз собираются в иерархии и занимают место в памяти. Исправим это, создав границы.

Создадим новый куб в нашей иерархии и переименуем в Boundary, также сбросив позицию. Отключим компонент Mesh Renderer, он нам не понадобится, в элементе Box Collider установим галочку напротив Is Trigger.



Установим позицию нашего куба по оси Z на отметку 5, чтобы он находился по середине. Теперь изменим его размер, растянув в соответствии с размером камеры x=15, z=20.



Наши границы должны выполнять определенную функцию. Добавим на куб новый скрипт DestroyByBoundary. Листинг скрипта будет выглядеть так:

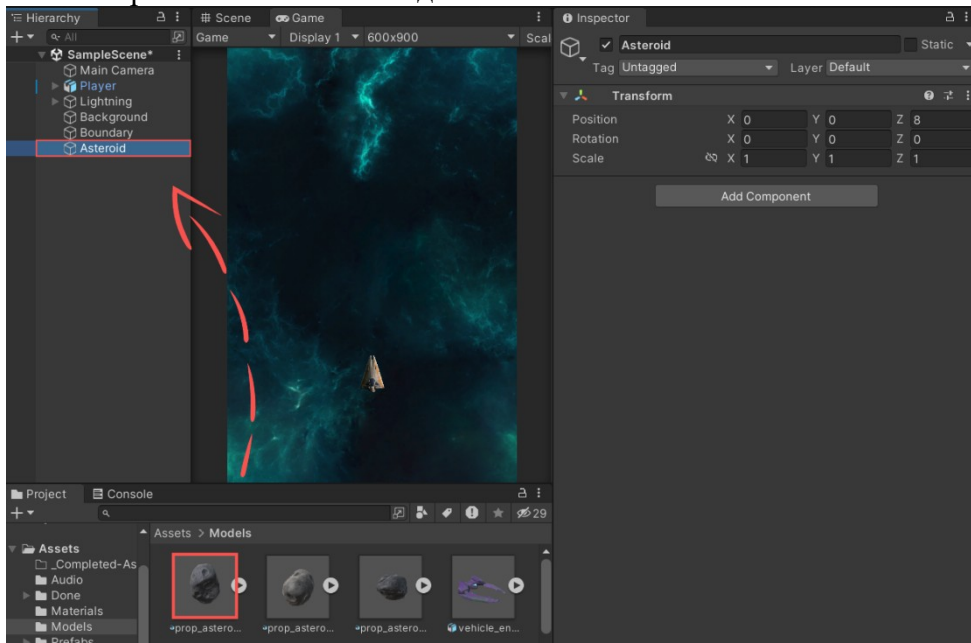
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroyByBoundary : MonoBehaviour
{
    void OnTriggerExit(Collider other) {
        Destroy(other.gameObject);
    }
}
```

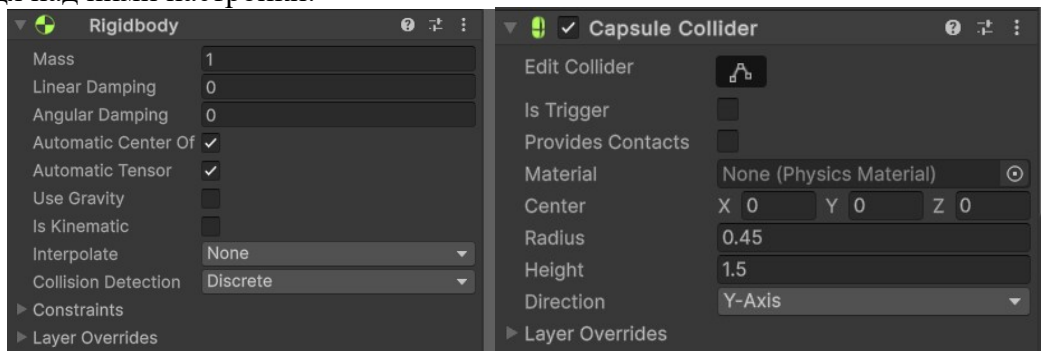
Сохраним код и перейдем к Unity. Удалим компоненты Mesh Renderer и Mesh Filter. Запустим игровой режим и проверим работу.

Лабораторная работа №11. Астероиды

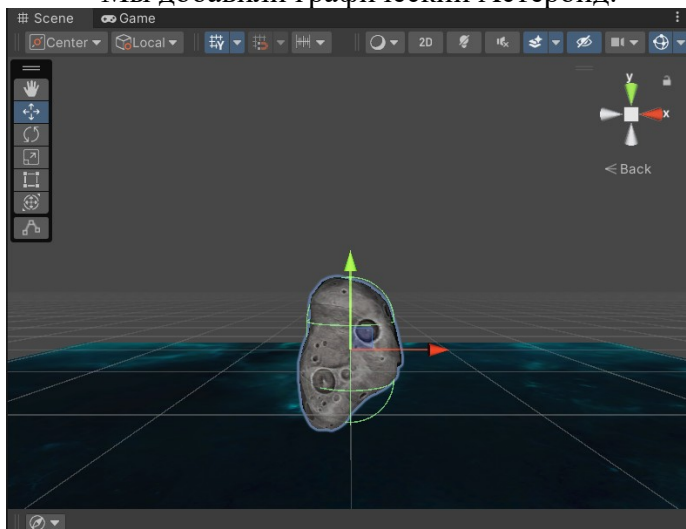
Добавим в нашу игру астероиды. Создадим новый пустой игровой объект Asteroid, сбросим позицию. Установим позицию по оси Z на 8. Это делается для более удобного тестирования. Далее, из папки Models, возьмем первую модель астероида. Также сбросим Transform модели.



Добавим на главный объект Asteroid компоненты Rigidbody и CapsuleCollider, проведя над ними настройки:



Мы добавили графический Астероид.

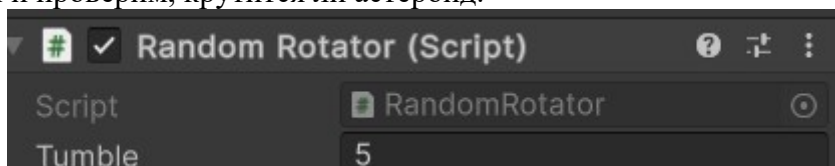


Добавим к нему скрипт RandomRotator, и пропишем код, который будет вращать его. Листинг скрипта выглядит следующим образом:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RandomRotator : MonoBehaviour {
    private Rigidbody rb;
    public float tumble;
    void Start()
    {
        rb = GetComponent<Rigidbody>();
        rb.angularVelocity = Random.insideUnitSphere * tumble;
    }
}
```

Пропишем в инспекторе значение переменной tumble равное 5. Включим игровой режим и проверим, крутится ли астероид.



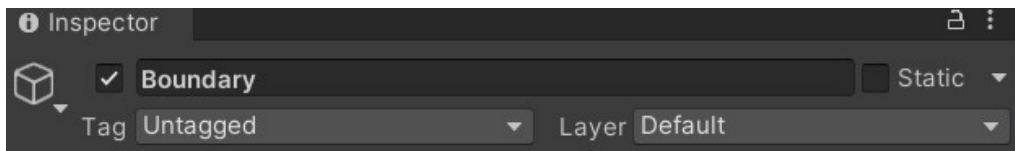
Добавим еще один скрипт, который будет отвечать за уничтожение астероида после попадания в него снаряда. Назовем его DestroyByContact. Листинг кода:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroyByContact : MonoBehaviour {
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

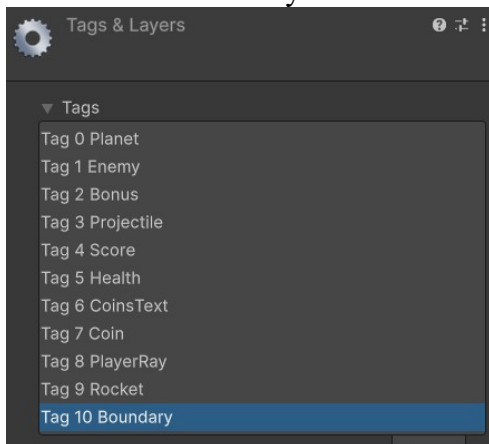
Какую же работы выполняет tag? Когда пуля попадает в астероид, он уничтожается - это правильно. Но почему-то пуля уничтожает ещё и границу игрового мира (Boundary), что не нужно. Чтобы это исправить, надо сделать так, чтобы когда пуля сталкивается с границей, она просто проходила сквозь неё, не наносила ей никакого вреда и не прекращала полёт. Нужно добавить в код проверку, чтобы при столкновении с объектом, помеченным тегом "Boundary", выстрел игнорировался и ничего не происходило.

Выбираем объект Boundary в иерархии и в меню инспектора находим поле tag

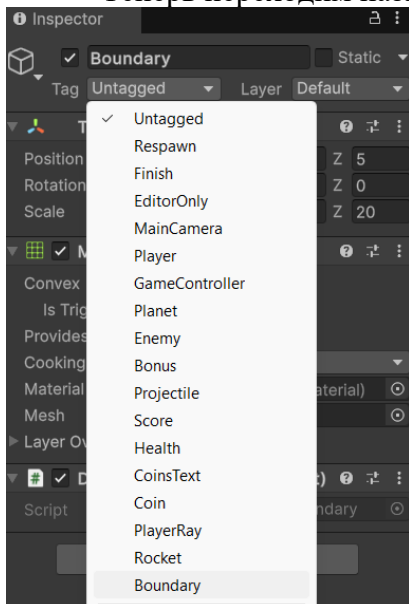


Щелкаем на списке и нажимаем поле Add tag.

В окне инспектора появилось меню тегов. Добавим новый тег, щелкнув на +, назовем его Boundary.



Теперь переходим назад к объекту Boundary и присвоим ему наш новый тег.



Лабораторная работа № 12. Взрывы

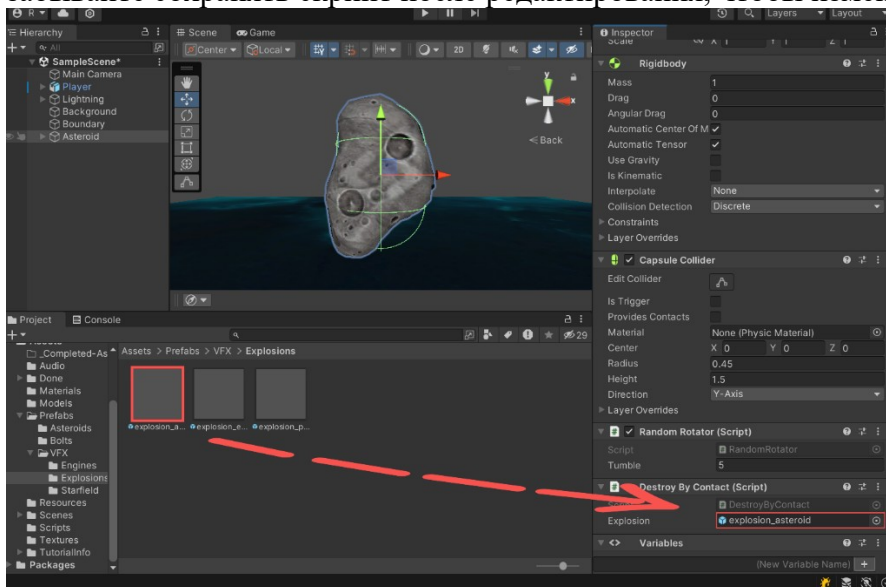
Для начала, нам нужно добавить переменную, для эффекта взрыва. Перейдем к скрипту `DestroyByContact`. Также мы добавили строчку `Instantiate(explosion, transform.position, transform.rotation);` (мы использовали ее в скрипте `MovingPlayer`)

```

DestroyByContact.cs
Assets > Done > Scripts > DestroyByContact.cs > DestroyByContact > OnTriggerEnter
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
0 references
5 public class DestroyByContact : MonoBehaviour {
6     1 reference
7     public GameObject explosion;
8     0 references
9     void OnTriggerEnter(Collider other)
10    {
11        if (other.tag == "Boundary")
12        {
13            return;
14        }
15        Instantiate(explosion, transform.position, transform.rotation);
16        Destroy(other.gameObject);
17        Destroy(gameObject);
18    }
19 }

```

Вернемся обратно в Unity. В папке **Prefabs – VFX – Explosions** выберем файл **explosion_asteroid** и перетащим в поле нашей переменной в инспекторе астероида (Не забывайте сохранять скрипт после редактирования, чтобы изменения вступили в силу).



Включим игровой режим и проверим работу скрипта.



Выглядит красиво, но с другой стороны, подводя корабль к астероиду мы увидим тот же эффект. Усовершенствуем наш скрипт. Добавим переменную для хранения эффекта взрыва корабля и клонируем строку `Instantiate`, но с изменениями.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

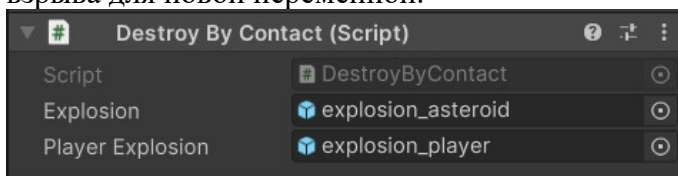
```
public class DestroyByContact : MonoBehaviour {
    public GameObject explosion;
    public GameObject playerExplosion;
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Instantiate(explosion, transform.position, transform.rotation);
        Instantiate(playerExplosion, other.transform.position, other.transform.rotation);
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

Только нам не хватает условия проверки коллизии с объектом, обладающим тегом «Player».

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class DestroyByContact : MonoBehaviour {
    public GameObject explosion;
    public GameObject playerExplosion;
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Instantiate(explosion, transform.position, transform.rotation);
        if (other.tag == "Player")
        {
            Instantiate(playerExplosion, other.transform.position, other.transform.rotation);
        }
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

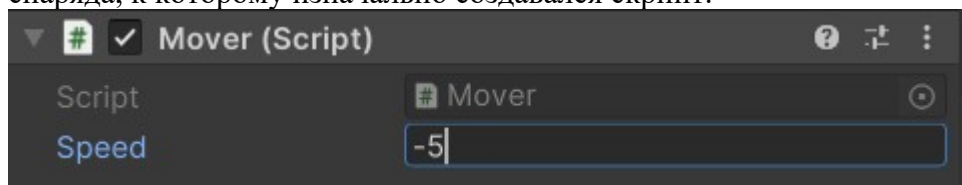
Добавим к Player тег Player, как мы делали в предыдущей главе и добавим эффект взрыва для новой переменной.



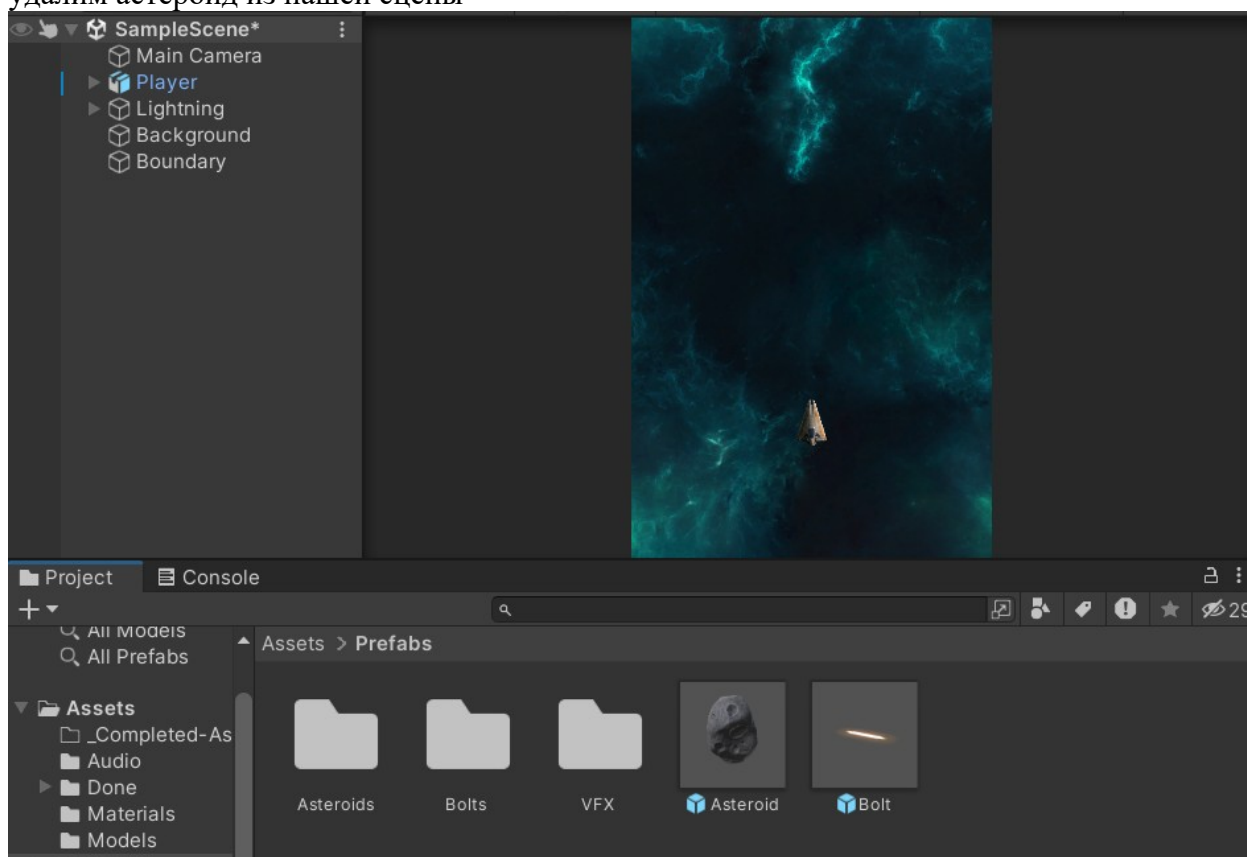
Проверим все в игровом режиме.



Осталось добавить скрипт движения нашего Астероида. Для этого, из папки Scripts перетащим скрипт Mover в инспектор нашего астероида. Установим значение скорости -5. Почему минус? Он означает движение противоположное от движения снаряда, к которому изначально создавался скрипт.



Проверим работу скрипта и сохраним Asteroid в папку Prefabs. После этого, удалим астероид из нашей сцены



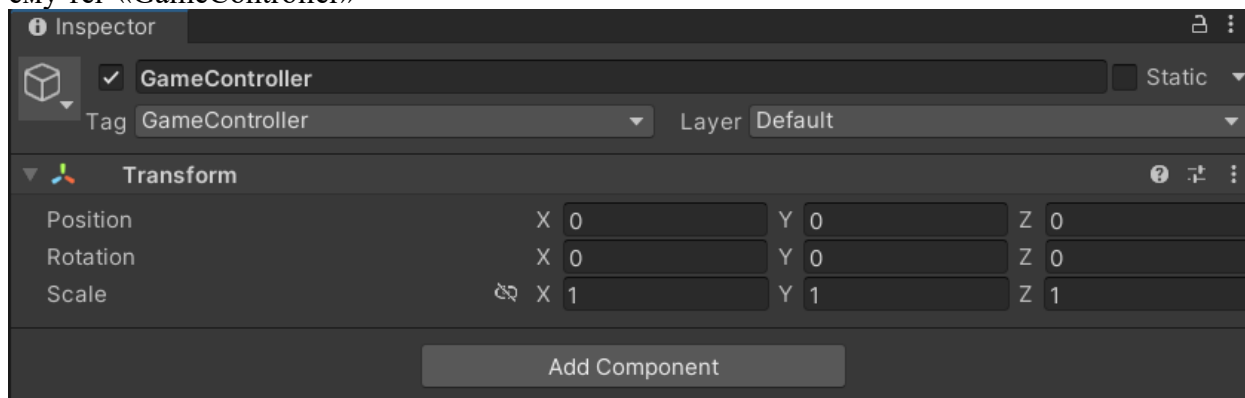
Видеоматериалы:

- 1) https://www.youtube.com/watch?v=nv_sNFZEls&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=10
- 2) https://www.youtube.com/watch?v=r48NT_2ckO0&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=11

Лабораторная работа №13. Game controller. Spawing waves.

У нас есть корабль, астероид и снаряды. Нам нужно объединить все это в одну игру. Поэтому, нам необходимо создать пустой объект и назвать его Game controller, чтобы создавать волны врагов и астероидов, показывать таблицу результатов и т.п.

Создадим игровой объект GameController и сбросим положение. Затем присвоим ему тег «GameController»



Перейдем к написанию основной логики. Добавим к игровому объекту GameController новый скрипт GameController. Он будет выполнять несколько задач, но основной будет создание волн астероидов.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

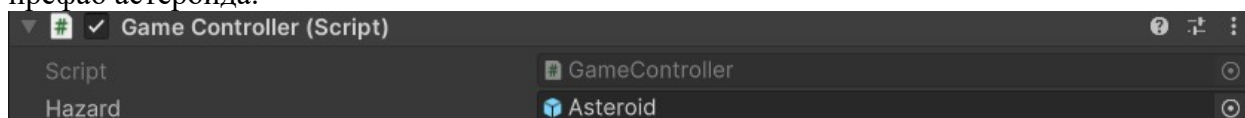
```
public class GameController : MonoBehaviour {

    public GameObject hazard;

    private void Start()
    {
        SpawnWaves();
    }

    void SpawnWaves()
    {
        Vector3 spawnPosition = new Vector3();
        Quaternion spawnRotation = new Quaternion();
        Instantiate(hazard, spawnPosition, spawnRotation);
    }
}
```

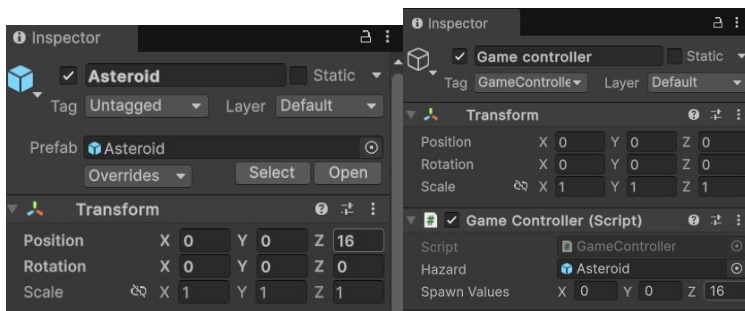
Мы определили переменную для хранения астероидов и написали функцию, которая будет вызываться при старте. Присвоим в инспекторе нашей новой переменной префаб астероида.



Вернемся к нашему скрипту и пропишем переменную, которая будет содержать в себе значения координат появления астероидов `public Vector3 spawnValues;`

```
1 reference
public GameObject hazard;
0 references
public Vector3 spawnValues;
```

Вернемся в инспектор и пропишем значения. По оси Z установим значение равное 16, это координата, находящаяся за пределами главной камеры.



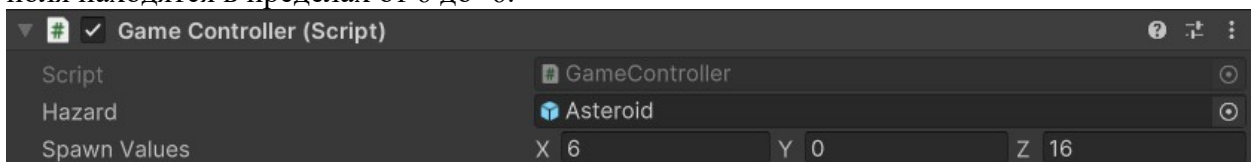
Что касается остальных осей, то нам нужно, чтобы астероиды появлялись в случайном порядке. Пропишем это в скрипте.

```

15 void SpawnWaves()
16 {
17     Vector3 spawnPosition = new Vector3(Random.Range(-spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
18     Quaternion spawnRotation = new Quaternion();
19     Instantiate(hazard, spawnPosition, spawnRotation);
20 }
21

```

В инспекторе, значение x установим на 6, т.к. правая и левая границы игрового поля находятся в пределах от 6 до -6.



Осталось разобраться со случайным поворотом астероида (нижняя строка).

```

Vector3 spawnPosition = new Vector3(Random.Range(-spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
Quaternion spawnRotation = Quaternion.identity;

```

Сохраним скрипт и протестируем.

Лабораторная работа №14. Волны астероидов

Перейдем к скрипту GameController. Для осуществления нашей задумки, необходимо будет написать цикл появления астероидов.

Создадим переменную для хранения целого числа астероидов и поместим код внутри функции, отвечающей за генерацию астероидов, в цикл *for* с некоторыми параметрами.

```

5 public class GameController : MonoBehaviour {
6
7     1 reference
8     public GameObject hazard;
9     4 references
10    public Vector3 spawnValues;
11    1 reference
12    public int hazardCount;
13
14    0 references
15    private void Start()
16    {
17        SpawnWaves();
18    }
19
20    1 reference
21    void SpawnWaves()
22    {
23        for(int i = 0; i < hazardCount; i++)
24        {
25            Vector3 spawnPosition = new Vector3(Random.Range(-spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
26            Quaternion spawnRotation = Quaternion.identity;
27            Instantiate(hazard, spawnPosition, spawnRotation);
28        }
29    }
30 }

```

Сохраним скрипт и протестируем его работу в Unity. Установим значение астероидов, скажем, на 10.



Hazard Count 10

Работает, но астероиды уничтожаются от соприкосновения друг с другом. Чтобы исправить это, нужно прописать появление каждого следующего астероида по порядку, а не одновременно.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;
    public int hazardCount;
    public float spawnWait;
    public float startWait;

    private void Start()
    {
        StartCoroutine (SpawnWaves());
    }

    IEnumerator SpawnWaves()
    {
        yield return new WaitForSeconds(startWait);
        for(int i = 0; i<hazardCount; i++ )
        {
            Vector3 spawnPosition = new Vector3(Random.Range(-spawnValues.x,
spawnValues.x), spawnValues.y, spawnValues.z);
            Quaternion spawnRotation = Quaternion.identity;
            Instantiate(hazard, spawnPosition, spawnRotation);
            yield return new WaitForSeconds(spawnWait);
        }
    }
}
```

Сохраним скрипт и перейдем в инспектор. Значение spawnWait установим на 0,5, т.е. мы будем создавать 2 астероида каждую секунду. startWait установим на 1, т.е. игроку дается 1 секунда перед началом волны астероидов.

Теперь нам нужно сделать перерыв между волнами. Ограничить их друг от друга.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;
    public int hazardCount;
    public float spawnWait;
    public float startWait;
```

```

public float waveWait;

private void Start()
{
    StartCoroutine (SpawnWaves());
}

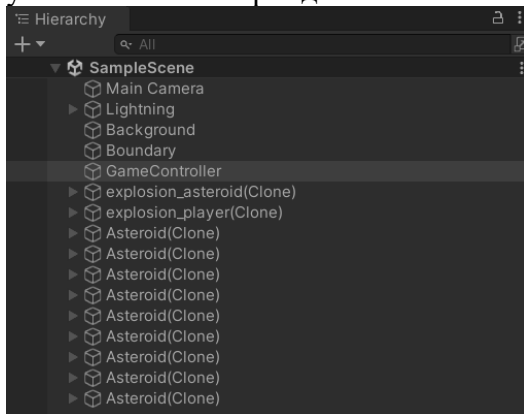
IEnumerator SpawnWaves()
{
    yield return new WaitForSeconds(startWait);
    while(true)
    {
        for(int i = 0; i<hazardCount; i++ )
        {
            Vector3 spawnPosition = new Vector3(Random.Range(-spawnValues.x,
spawnValues.x), spawnValues.y, spawnValues.z);
            Quaternion spawnRotation = Quaternion.identity;
            Instantiate(hazard, spawnPosition, spawnRotation);
            yield return new WaitForSeconds(spawnWait);
        }
        yield return new WaitForSeconds(waveWait);
    }
}
}

```

Для переменной waveWait пропишем значение 4.

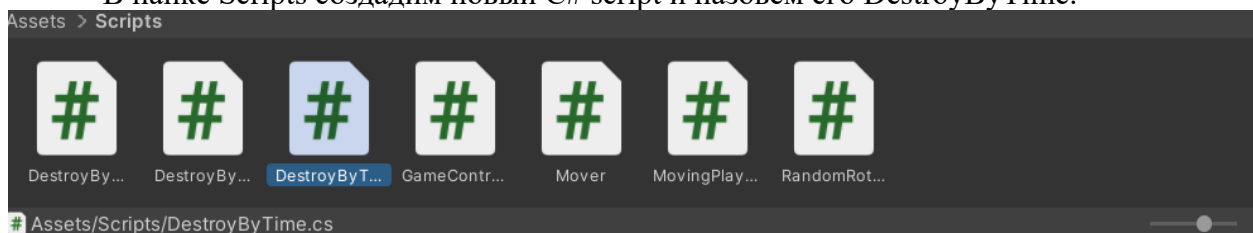
Протестируем работу скрипта: после уничтоженной волны астероидов, должна быть пауза перед следующей.

Теперь нам нужно решить проблему накопления эффектов взрыва после уничтожения астероидов.



Они не соприкасаются со снарядами и с границей игрового поля, так что нужно прописать для них отдельный отрезок кода.

В папке Scripts создадим новый C# script и назовем его DestroyByTime.



Он будет очень простым:

```
using System.Collections;
```

```

using System.Collections.Generic;
using UnityEngine;

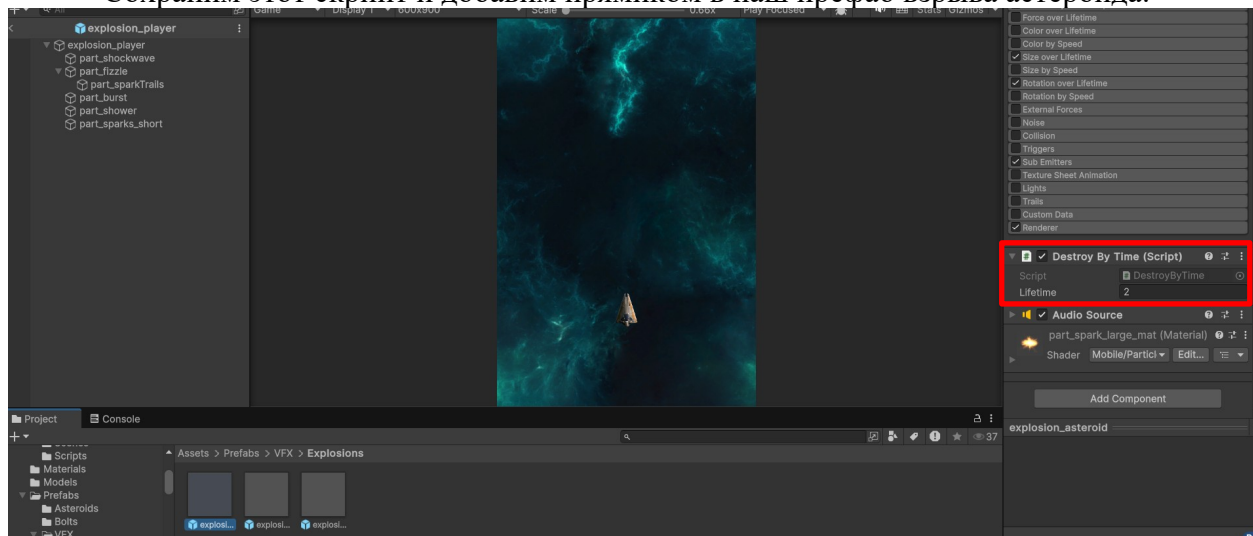
public class DestroyByTime : MonoBehaviour {

    public float lifetime;

    void Start()
    {
        Destroy(gameObject, lifetime);
    }
}

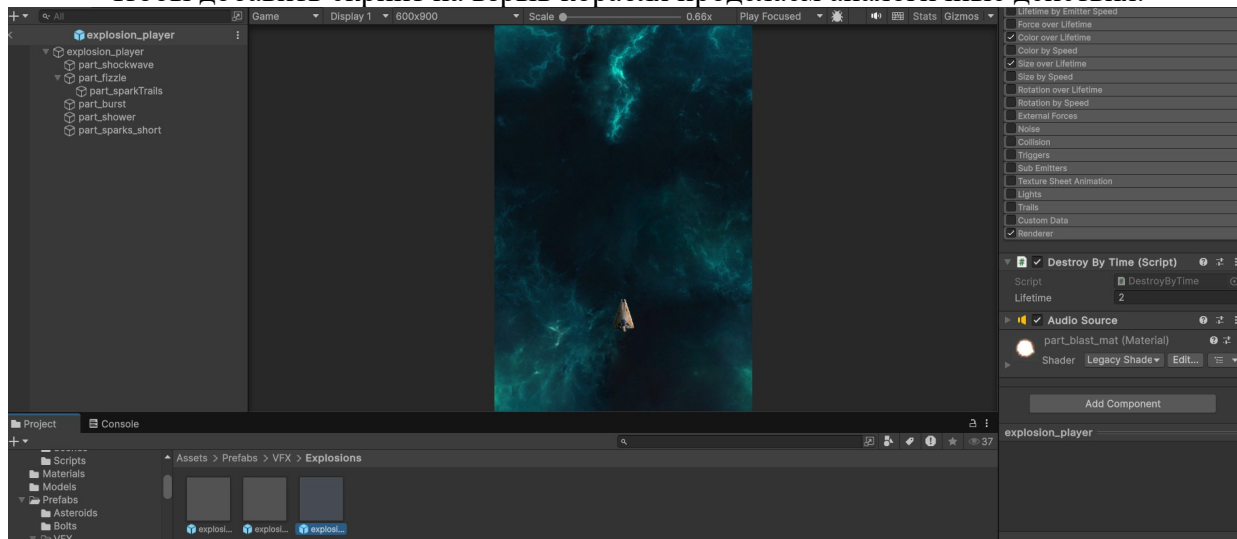
```

Сохраним этот скрипт и добавим пряником в наш префаб взрыва астероида.



Значение переменной lifetime установим на 2.

Чтобы добавить скрипт на взрыв корабля сделаем аналогичные действия.



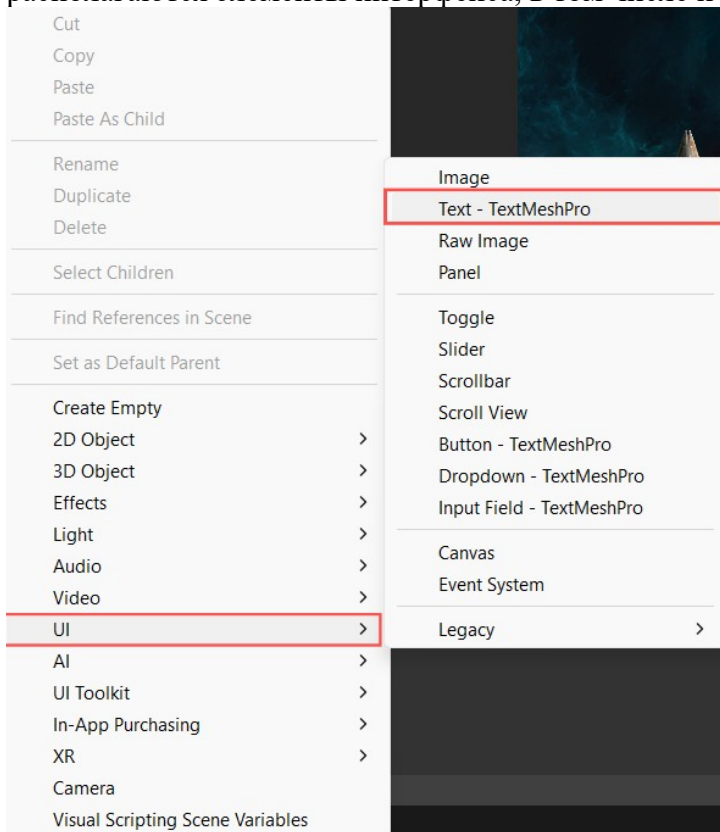
Включим игровой режим и протестируем проделанную работу.

Видеоматериалы:

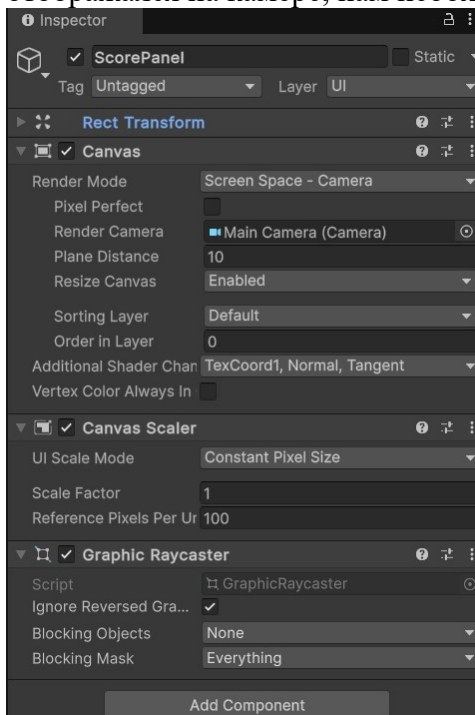
- 1) <https://www.youtube.com/watch?v=HvhAoesy6UM&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=12>
- 2) https://www.youtube.com/watch?v=wyGHp_uXC7Y&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=13

Лабораторная работа №15. Подсчет очков и отображение результатов. Конец игры. Building the game.

Каждый уничтоженный астероид будет приносить игроку по 10 очков. Прежде чем мы приступим к написанию кода, создадим табло для отображения очков. Создадим Text в иерархии. Юнити автоматически создает элемент Canvas, на котором располагаются элементы интерфейса, в том числе и текст.

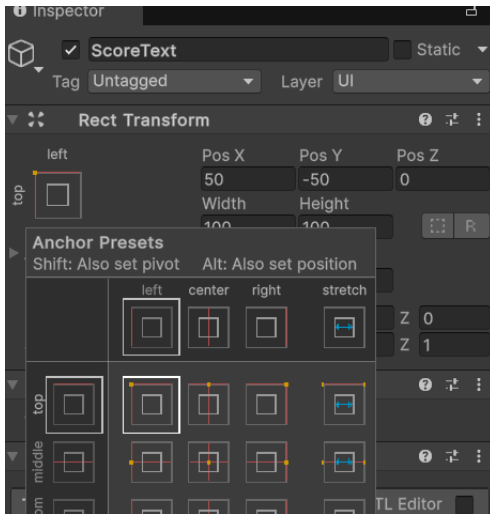


Переименуем Text (TMP) в ScoreText, а Canvas в ScorePanel. Чтобы наш Canvas отображался на камере, нам необходимо сделать следующие настройки:

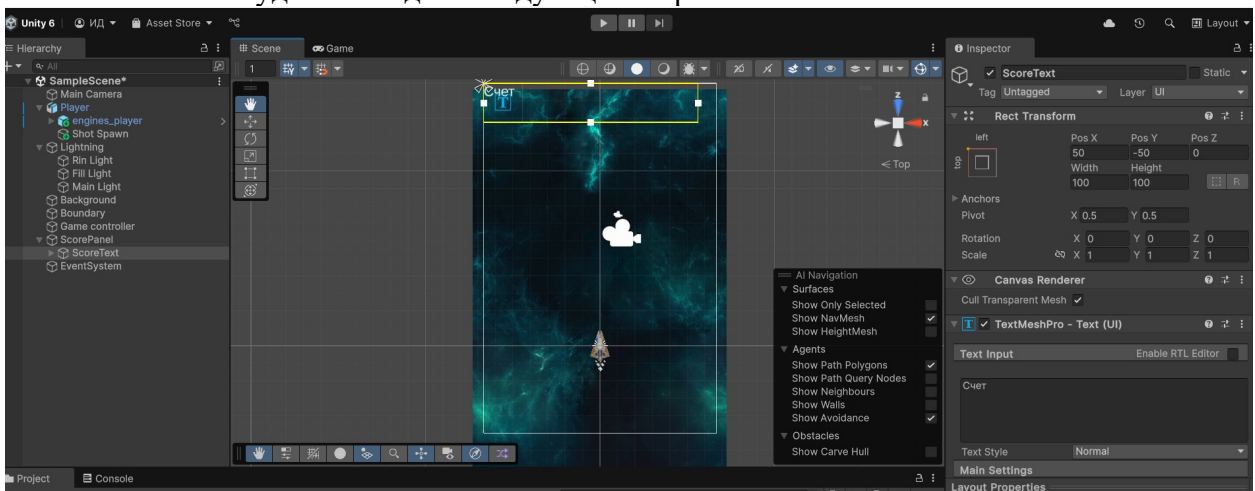


А также сбросить позицию для элемента Text.

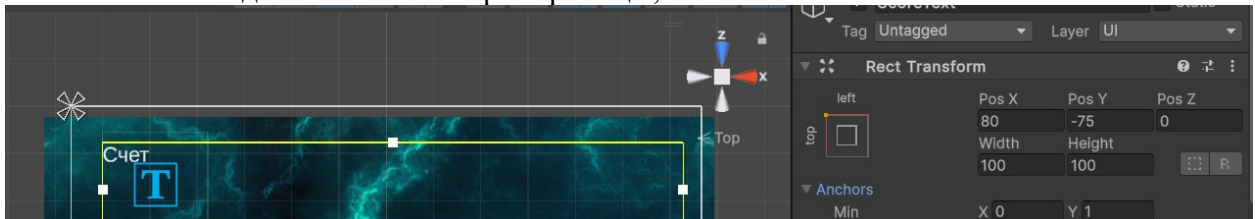
Расположим текст в левом верхнем углу экрана, для этого нажмем на квадрат в компоненте Rect Transform и с зажатой клавишей alt нажмем на левый верхний квадрат



ScoreText будет выглядеть следующим образом:



Немного сдвинем текст от края границы, чтобы он не вливался в нее.



Перейдем в скрипт GameController и пропишем переменные для хранения информации о результате, хранении поля вывода результата и определим функцию обновления и добавления результата.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using TMPro;
using System.Threading;
using Unity.VisualScripting;
using UnityEngine.SocialPlatforms.Impl;
```

```

public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;
    public int hazardCount;
    public float spawnWait;
    public float startWait;
    public float waveWait;
    public TextMeshProUGUI scoreText;
    public static int score;

    private void Start()
    {
        GameObject scoreTextObject = GameObject.FindWithTag("ScoreText");
        if (scoreTextObject != null)
        {
            scoreText = scoreTextObject.GetComponent<TextMeshProUGUI>();
        }
        if (scoreText == null)
        {
            Debug.Log("Cannot Find 'scoreText' Script");
        }
        score = 0;
        UpdateScore();
        StartCoroutine (SpawnWaves());
    }

    IEnumerator SpawnWaves()
    {
        yield return new WaitForSeconds(startWait);
        while(true)
        {
            for(int i = 0; i<hazardCount; i++ )
            {
                Vector3 spawnPosition = new Vector3(UnityEngine.Random.Range(-
spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
                Quaternion spawnRotation = new Quaternion();
                Instantiate(hazard, spawnPosition, spawnRotation);
                yield return new WaitForSeconds(spawnWait);
            }
            yield return new WaitForSeconds(waveWait);
        }
    }

    public void AddScore(int newScoreValue)
    {
        score += newScoreValue;
        UpdateScore();
    }
}

```

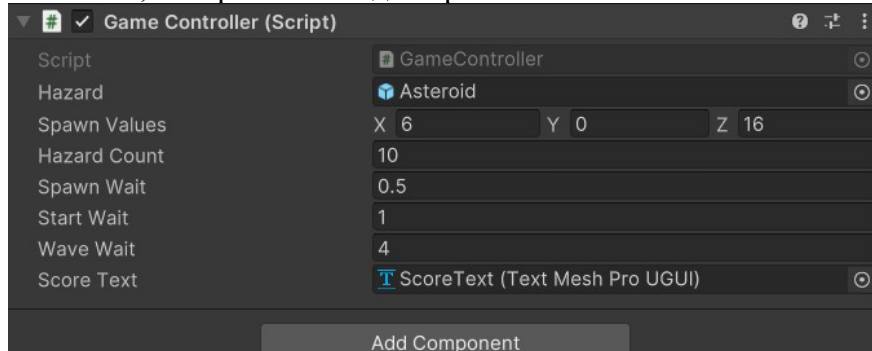
```

void UpdateScore()
{
    scoreText.text = "Score: " + score;
}
}

```

Также, нам необходимо добавить тег ScoreText для элемента ScoreText.

Перейдем в Юнити и в GameController в поле scoreText вставим наш объект ScoreText, который мы создали ранее.



Теперь перейдем к скрипту DestroyByContact и пропишем код, для начисления 10 очков за каждый разрушенный астероид. Добавим функцию Start(), и допишем функцию OnTriggerEnter().

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroyByContact : MonoBehaviour {
    public GameObject explosion;
    public GameObject playerExplosion;
    public int scoreValue;
    public GameController gameController;
    private GameObject newProjectile;
    void gameController_Main()
    {
        GameController gameController = new GameController();
        GameObject gameControllerObject = GameObject.FindWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Невозможно найти скрипт 'GameController' ");
        }
        gameController.AddScore(scoreValue);
    }
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {

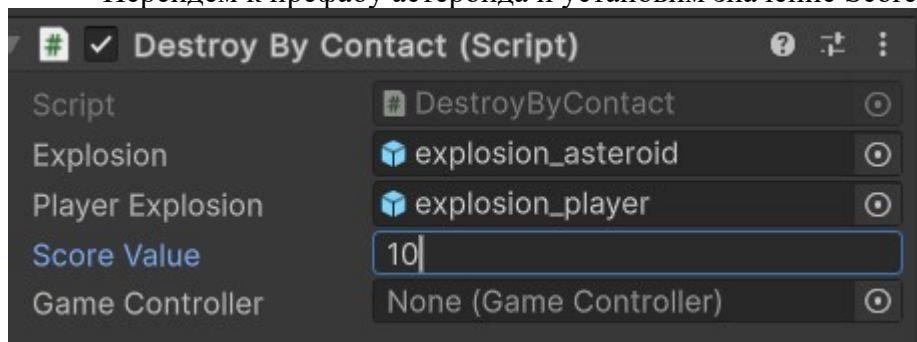
```

```

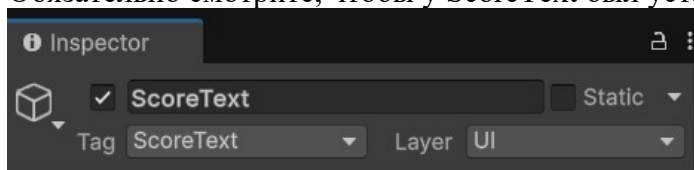
        return;
    }
    newProjectile = Instantiate(explosion, transform.position, transform.rotation) as
GameObject;
    if (other.tag == "Player")
    {
        newProjectile = Instantiate(playerExplosion, other.transform.position,
other.transform.rotation) as GameObject;
    }
    gameController_Main();
    Destroy(other.gameObject);
    Destroy(gameObject);
}
}
}

```

Перейдем к префабу астероида и установим значение ScoreValue на 10.



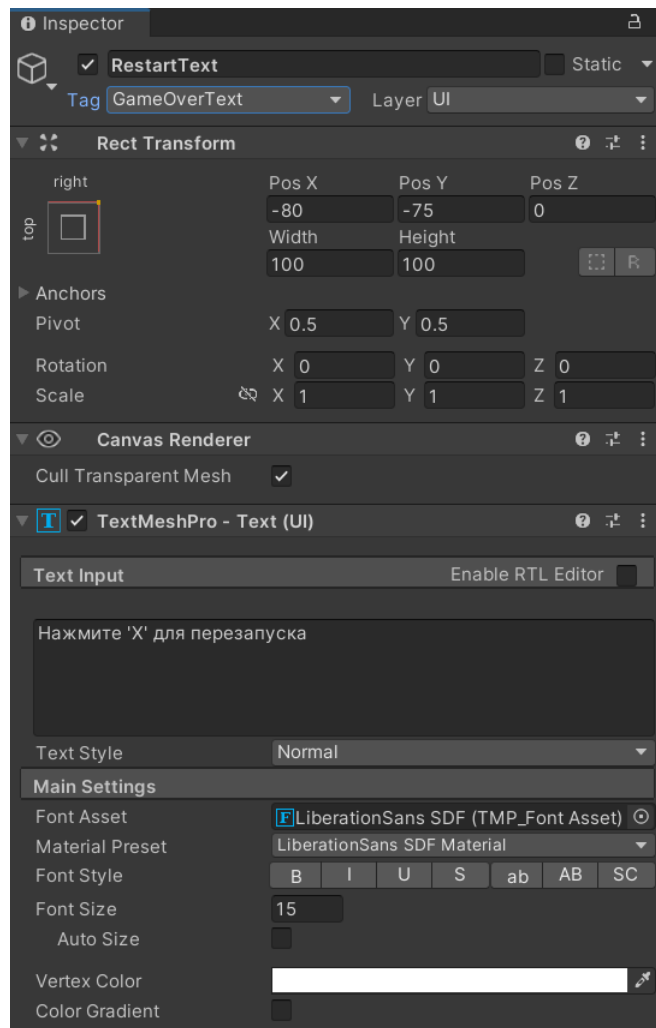
Обязательно смотрите, чтобы у ScoreText был установлен тег.



Протестируем работу в игровом режиме.

Лабораторная работа №16. Рестарт уровня

Добавим в игру возможность рестарта уровня. Для начала создадим Текст рестарта в нашем канвасе и определим для него такие характеристики:



Затем перейдем к скрипту GameController и пропишем функцию Update

```

65 |
66 | 0 references
67 | public void Update()
68 | {
69 |     if (Input.GetKeyDown(KeyCode.Escape))
70 |         Application.Quit();
71 |     if (Input.GetKeyDown(KeyCode.X))
72 |         SceneManager.LoadScene("SampleScene");
73 | }

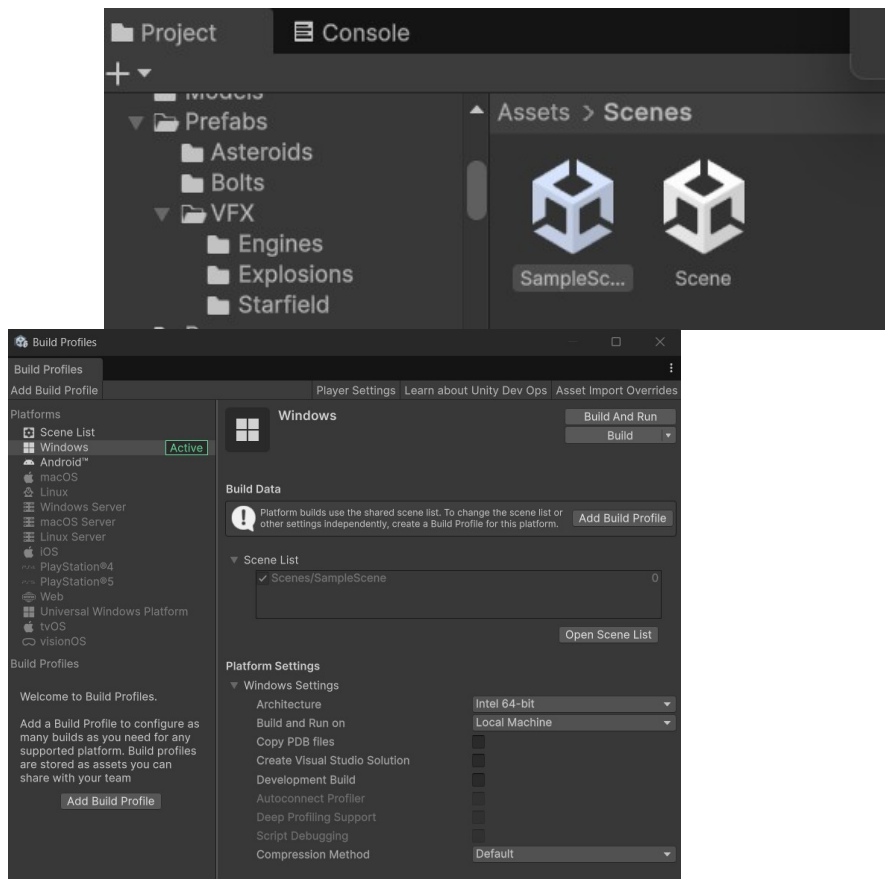
```

Данный код, считывает нажатие клавиши X и перезапускает игровую сцену, т.е. делает рестарт, а также при нажатой клавише Escape закрывает игру.

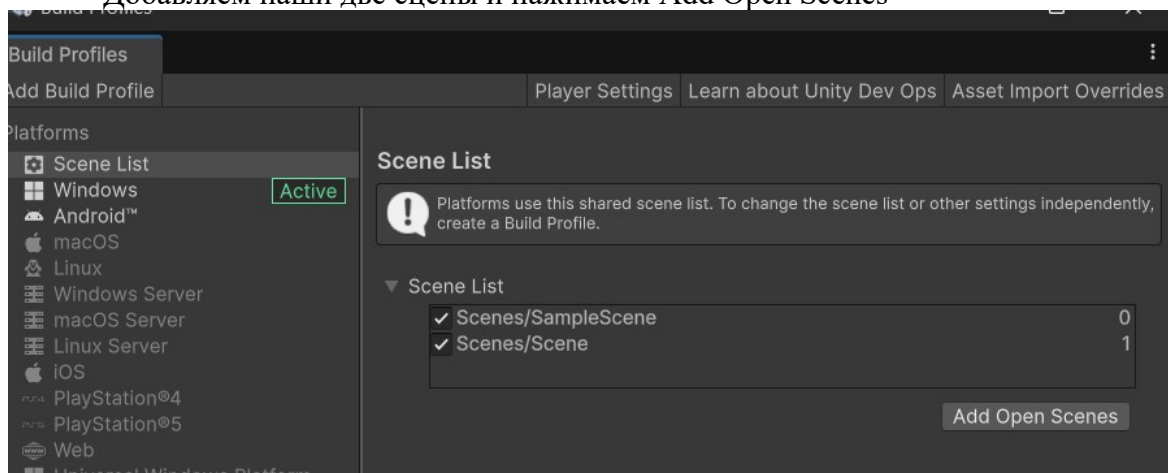
Сборка проекта

Создадим исполняемый файл. Для того, чтобы собрать наш проект в игру, нам необходимо провести следующие манипуляции:

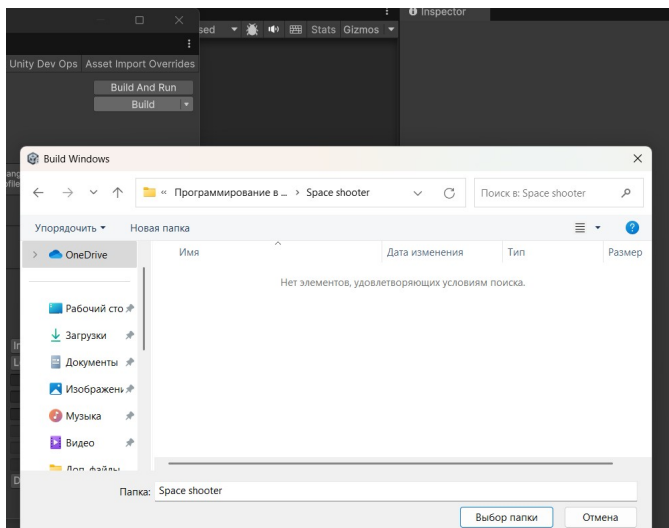
- 1) Перейти в меню Build Profiles (File – Build Profiles).
- 2) Из раздела Project папки Scenes перетащить наши основные сцены в поле Scene List. Нажмите кнопку Open Scene List.



Добавляем наши две сцены и нажимаем Add Open Scenes



3) Нажать справа кнопку Build и выбрать папку, в которой будет храниться игра



4) По завершению сборки откроем игру и проверим ее работу

Folder	D3D12	21.03.2025 0:11	Папка с файлами	
Folder	MonoBleedingEdge	21.03.2025 0:11	Папка с файлами	
Folder	Space Shooter_BurstDebugInformation_...	21.03.2025 0:12	Папка с файлами	
Folder	Space Shooter_Data	21.03.2025 0:12	Папка с файлами	
Application	Space Shooter.exe	21.03.2025 0:11	Приложение	657 КБ
Application	UnityCrashHandler64.exe	21.03.2025 0:11	Приложение	1 496 КБ
File	UnityPlayer.dll	21.03.2025 0:11	Расширение при...	32 822 КБ

Видеоматериалы:

- 1) https://www.youtube.com/watch?v=6AV_DwAs_xc&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=14
- 2) <https://www.youtube.com/watch?v=519wwXqM4no&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=15>
- 3) <https://www.youtube.com/watch?v=RJk8VgfgD6k&list=PLDZieabyoJJfWj5ASktozPEwNlZw30VTI&index=16>

Дополнительные материалы:

- 1) <https://unity3d.com/ru/learn/tutorials/projects/space-shooter-tutorial/audio?playlist=17147>
- 2) <https://unity3d.com/ru/learn/tutorials/topics/user-interface-ui/ui-text?playlist=17111>

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для обучающихся по организации и проведению самостоятельной работы
по дисциплине «Программирование в бизнес-системах»
направление подготовки 38.03.05 «Бизнес-информатика»
направленность (профиль): «Информационная бизнес-аналитика и
цифровые технологии»

ВВЕДЕНИЕ

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

Целью данной дисциплины является формирование набора (ОПК-3) компетенций будущего бакалавра по направлению подготовки 38.03.05 «Бизнес-информатика».

Задачами дисциплины «Программирование в БС» являются: изучение стратегических перспектив, увязывании финансовых и стратегических целей компании, разработке и формировании приоритета стратегических мероприятий, управлении реализацией стратегии компании, а также сокращении издержек и увеличении производительности.

Общая характеристика самостоятельной работы студента при изучении дисциплины «Программирование в БС»

Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СР являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ.

Методические рекомендации по изучению теоретического материала Работа с книгой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Различают два вида чтения; первичное и вторичное. *Первичное* - это внимательное, неторопливое чтение, при котором можно остановиться на трудных местах. После него не должно остаться ни одного непонятого слова. Содержание не всегда может быть понятно после первичного чтения.

Задача *вторичного* чтения полное усвоение смысла целого (по счету это чтение может быть и не вторым, а третьим или четвертым).

Правила самостоятельной работы с литературой

Как уже отмечалось, самостоятельная работа с учебниками и книгами (а также самостоятельное теоретическое исследование проблем, обозначенных преподавателем на лекциях) - это важнейшее условие формирования у себя научного способа познания. Основные советы здесь можно свести к следующим:

- Составить перечень книг, с которыми Вам следует познакомиться;
- Сам такой перечень должен быть систематизированным.
- Обязательно выписывать все выходные данные по каждой книге (при написании курсовых и дипломных работ это позволит очень сэкономить время).
- Разобраться для себя, какие книги (или какие главы книг) следует прочитать более внимательно, а какие - просто просмотреть.
- При составлении перечней литературы следует посоветоваться с преподавателями и научными руководителями (или даже с более подготовленными и эрудированными сокурсниками), которые помогут Вам лучше сориентироваться, на что стоит обратить большее внимание, а на что вообще не стоит тратить время...

•Естественно, все прочитанные книги, учебники и статьи следует конспектировать, но это не означает, что надо конспектировать «все подряд»: можно выписывать кратко основные идеи автора и иногда приводить наиболее яркие и показательные цитаты (с указанием страниц).

Чтение научного текста является частью познавательной деятельности. Ее цель - извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют *четыре основные установки в чтении научного текста*:

1. информационно-поисковый (задача - найти, выделить искомую информацию);
2. усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как самисведения, излагаемые автором, так и всю логику его рассуждений);
3. аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему);
4. творческая (создает у читателя готовность в том или ином виде - как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. - использовать суждения автора, ход его мыслей, результат

наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

1. Аннотирование - предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;
2. Планирование - краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;
3. Тезирование- лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;
4. Цитирование - дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;
5. Конспектирование - краткое и последовательное изложение содержания прочитанного.

Конспект - сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические указания по составлению конспекта

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта.
2. Выделите главное, составьте план.
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора.
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

Список рекомендуемой литературы

Основная литература:

1. Борисенко, В. В. Основы программирования : учебное пособие / Борисенко В. В. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2020. - 323 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-9556-00039-0, экземпляров неограничено
2. Блюмин А. М. Проектирование систем интеллектуального обслуживания: учебник - Москва: Дашков и К°, 2018
3. Карпович Е. Е. Языки программирования интеллектуальных систем: учебник - Москва: МИСиС, 2018

Дополнительная литература:

1. Гарибов, А. И. Основы разработки приложений для мобильных устройств на платформе Windows Phone / А.И. Гарибов. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2020. - 460 с., экземпляров неограничено
2. Самойлова, Т. А. Разработка гибридных приложений для мобильных устройств под Windows Phone / Т.А. Самойлова. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2019. - 461 с., экземпляров неограничено
3. Цифровая экономика. Социально-экономические и управленческие концепции Электронный ресурс: Коллективная монография / Л. И. Антонова [и др.] ; ред. А. А. Степанова. - Москва: Научный консультант, Виктория плюс, 2018. - 186 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-6040573-2-2

Интернет-ресурсы:

1. www.biblioclub.ru - Электронно-библиотечная система «Университетская библиотека онлайн»
2. e.lanbook.com - Электронно-библиотечная система «Лань»

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению и защите курсовой работы
по дисциплине «Программирование в бизнес-системах»
направление подготовки 38.03.05 Бизнес-информатика
направленность (профиль) «Информационная бизнес-аналитика и
цифровые технологии»

Ставрополь 2026 г

Оглавление

1. Введение
2. Цель, задачи и реализуемые компетенции
3. Формулировка задания
4. Структуры работы
5. Общие требования к написанию и оформлению работы
6. Последовательность выполнения задания
7. Критерии оценивания работы
8. Порядок защиты работы
9. Список рекомендуемой литературы

1. Введение

Методические указания по выполнению и защите курсовой работы по направлению подготовки 38.03.05 Бизнес-информатика, по дисциплине «Программирование в БС» разработаны в целях установления общего порядка выполнения и защиты курсовой работы (далее - «КР») студентами магистратуры.

Методические указания по выполнению и защите курсовой работы по направлению подготовки 38.03.05 Бизнес-информатика, по дисциплине «Программирование в БС» разработана на основании:

- - Федеральный закон от 29.12.2012 г. № 273-ФЗ «Об образовании в Российской Федерации»;

- Федеральный государственный образовательный стандарт высшего образования – бакалавриат по направлению подготовки 38.03.05 Бизнес-информатика, утвержденный приказом Министерства науки и высшего образования Российской Федерации от 29.07.2020 № 838(далее ФГОС ВО);

- Положения о выполнении и защите курсовых работ (проектов) в Федеральном государственном автономном образовательном учреждении высшего образования «Северо-Кавказский федеральный университет» (новая редакция) принятого ученым советом СКФУ протокол 1 от 29.08.2019г.

2. Цель, задачи и реализуемые компетенции

Курсовая работа является самостоятельной исследовательской работой студента и представляет собой логически завершенное и оформленное научное исследование.

Цель - формирование у студента навыков научно-исследовательской работы, повышение уровня его профессиональной (теоретической и практической) подготовки, углубление знаний по учебной дисциплине, развитие интереса и навыков самостоятельной работы с научной и справочной литературой.

Системой курсовых работ (проектов) студент подготавливается к выполнению выпускной квалификационной работы.

В процессе выполнения курсовой работы студентом должны решаться следующие задачи:

- Приобретение новых теоретических знаний в контексте современного развития методики финансового учета коммерческих предприятий;

- Умение систематизировать, обобщать и логично излагать концепции, альтернативные точки зрения по исследуемой проблеме.

- Развитие учебно-исследовательских и методических навыков, необходимых для системного научного анализа изучаемого явления.

- Совершенствование профессиональной подготовки (профессиональные навыки по использованию современного учетного инструментария, которые могут использоваться при решении широкого спектра экономических и управленческих задач).

Курсовую работу (проект) целесообразно планировать ходе изучения курса «Программирование в БС».

Перечень оцениваемых компетенций

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине, характеризующие этапы формирования компетенций, индикаторов
ОПК-3 Способен управлять процессами	ИД -1 ОПК-3 Реализовывает и обеспечивает поддержку	Владеет навыками создания и

создания и использования продуктов и услуг в сфере информационно-коммуникационных технологий, в том числе разрабатывать алгоритмы и программы для их практической реализации	процессов, относящихся к различным фазам жизненного цикла информационных систем ИД -3 ОПК-3 Разрабатывает алгоритмы и приложения на бизнес-ориентированных языках программирования	использования продуктов и услуг в сфере ИКТ, в том числе разрабатывать алгоритмы и программы для их практической реализации
--	---	---

3. Формулировка задания

Для выполнения курсовой работы по дисциплине необходимо собрать в течении недели примерный список исследуемой литературы и обсудить его с научным руководителем, затем студент составляет план курсовой работы.

Примерные темы курсовых работ

1. Информационные потоки в цифровой экономике
2. Применение технологии блокчейн для бизнес-процессов
3. Монетизация компьютерных игр
4. Факторы, вызывающие изменения в структуре и политике компании. Внешние и внутренние факторы
5. Финансовый контроллинг в эпоху цифровой экономики
6. Характеристика искусственной нейронной сети. Сферы применения в программных продуктах
7. Интеграция бизнес-систем в единое информационное пространство
8. Классификация языков программирования для Интернет
9. Информационная стратегия как ключевой фактор успеха. Примеры успешного внедрения информационных технологий
10. Цифровизация в масштабе города Ставрополь
11. Статистический обзор мирового рынка цифровой экономики и электронной торговли»
12. Концепция Enterprise 2.0
13. Игра как новая форма ведения бизнеса
14. Человекоцентричная разработка и дизайн-мышление. Стадии процесса дизайн-мышления
15. Геймификация как мотивационная форма работы с сотрудниками организации»
16. Определение феномена электронных предприятий. Основы деятельности электронных предприятий
17. Технологии компьютерных игр
18. Техничко-экономическое обоснование разработки игрового приложения
19. Состояние и перспективы развития игровой индустрии. Разработка игр как бизнес явление
20. Индустрия 4.0. Перспективы развития программирования
21. Открытое программное обеспечение (open-source). Проблемы и перспективы развития
22. Игровое пиратство. Плюсы и минусы явления. Экономических расчет потерь
23. Составляющие и особенности Web-программирования
24. Интеграция интернет-вещей и бизнеса
25. Взаимодействие между технологиями и бизнесом – как сложная и комплексная проблема

26. Виртуальная и дополненная реальность. Сферы применения
27. Понятие геймификации. Бизнес-симуляторы
28. Индустрия 4.0. Перспективы развития программирования
29. Необходимость инвестирования средств в информатизацию бизнеса. Причины и предпосылки
30. Понятие игрофикации. Сферы и примеры применения игрофикации
31. Применение систем искусственного интеллекта для решения бизнес задач
32. Информационные потоки в цифровой экономике (услуги, производство, образование)
33. Методы продвижения цифровизации в научно-исследовательских организациях

4. Структура работы

Структура текста должна отражать логику исследования. Это предполагает разделение текста на отдельные логически соподчиненные части, каждая из которых снабжается кратким и ясным заголовком, отражающим ее содержание. Структурно курсовая работа должна состоять из следующих частей:

- титульный лист;
- содержание;
- введение;
- теоретическая часть (первая глава);
- практическая часть (вторая главы);
- заключение;
- список использованных источников;
- приложения.

Титульный лист оформляется в соответствии с положениями, изложенными в нормативных документах.

Содержание представляет собой расширенное оглавление, помогающее получить полное представление о содержании работы и ее структуре с указанием страниц начала каждой части работы.

Во Введении (ориентировочно, 2-3 страницы) обосновывается актуальность выбранной темы исследования; отражаются объект, предмет, задачи, цели, методы, новизна, теоретическая и практическая значимость исследования. Здесь же должно быть осуществлено представление предприятия, на примере которого подготовлена курсовая работа.

Основная часть курсовой работы состоит из двух глав, которые можно, в свою очередь, разделить на параграфы (рекомендуемый объем первой главы 15-18 страниц, объем второй главы – 18-20 страницы). Названия глав и параграфов не должны дублировать название темы курсовой работы. Главы и параграфы необходимо соотносить друг с другом по объему представленного материала. Оптимально равное соотношение объемов разделов и параграфов. Объем параграфов не должен превышать объема любой из глав работы. Заголовки глав и параграфов должны быть лаконичными и соответствовать их содержанию.

В теоретической части (первой главе) курсовой работы обобщаются сведения из разных литературных источников по данной теме, излагается аргументированный авторский подход к рассмотренным концепциям, точкам зрения. Определяется сущность, понятия, виды и предмет курсовой работы.

В практической части (второй главе) курсовой работы рассматривается описание системы экспериментального исследования, обоснование методов исследования, разработку инфологической модели, анализ результатов экспериментального исследования, схемы, графические и математические способы интерпретации полученных данных, выводы.

В курсовой работе практическая часть не обязательно должна носить обширный характер, но вместе с тем должна быть такой, чтобы студент мог освоить практические, статистические, математические, диагностические и т.п. методы конкретной науки.

В конце каждой главы рекомендуется сделать выводы.

Заключение содержит краткое изложение выводов по теме работы. Заключение не должно носить характер сжатого пересказа всей работы, в нем должны быть изложены итоговые результаты (2-3 стр.). Эта часть исполняет роль концовки, обусловленной

логикой проведенного исследования, которая носит форму синтеза накопленной в основной части работы.

Заключительная часть предполагает также наличие обобщенной итоговой оценки проделанной работы. При этом важно указать, в чем заключается ее главный смысл, какие важные побочные научные результаты получены, какие встают новые научные задачи в связи с проведением исследования. В некоторых случаях возникает необходимость указать пути продолжения исследования темы, формы и методы ее дальнейшего изучения, а также конкретные задачи, которые будущим исследователям придется решать в первую очередь.

Список использованной литературы содержит наименование работ, источников, которые были непосредственно использованы автором при работе над курсовой работой. В работе необходимо использовать нормативно-правовую литературу, особенно с последними изменениями и дополнениями, учебно-методическую литературу и периодические издания.

Количество использованных источников и литературы в курсовой работе, как правило, должно быть не менее 15–20.

Приложения. Вспомогательные или дополнительные материалы, которые загромождают текст основной части работы, помещают в приложения. Приложения являются обязательным компонентом курсовой работы.

Приложения могут быть различными: таблицы, схемы, графики, диаграммы, иллюстрации, копии постановлений, и т. п.

Если приложение выполнено на листах иного формата, чем текстовая часть работы, то оно должно быть сложено по формату А4.

Приложения не засчитываются в заданный объем курсовой работы. Количество приложений не ограничено в курсовой работе, но студенту необходимо исходить из принципа целесообразности.

5. Общие требования к написанию и оформлению работы

Курсовую работу (проект) рекомендуется представлять в объеме 1-2 печатных листа. Текст работы должен быть напечатан через 1,5 интервала на одной стороне стандартного листа белой бумаги (А-4). Текст и другие отпечатанные элементы работы должны быть черными, контуры букв и знаков четкими, без ореола и затенения. Шрифт TimesNewRoman, кегель 14. Названия глав и параграфов выделяются полужирным шрифтом. Лист с текстом должен иметь поля: слева - 30 мм, справа - 10 мм, сверху - 20 мм, снизу - 20 мм. Нумерация страниц текста делается в правом нижнем углу листа. Проставлять номер страницы необходимо со страницы, где печатается «Введение», на которой ставится цифра «3». После этого нумеруются все страницы, включая приложения.

Между названием главы и названием параграфа этой главы ставится пробел равный двум интервалам, а название параграфа не должно отделяться от текста этого параграфа пробелом. Названия параграфов отделяются от текста предыдущего параграфа пробелом, равным двум интервалам. Каждая глава, а также введение, выводы, приложения и список использованной литературы начинаются с новой страницы. Слово «Глава» не пишется. Главы имеют порядковые номера в пределах всей работы, обозначаемые арабскими цифрами (например: 1, 2, 3), после которых ставится точка. Слово «параграф» или значок параграфа в названии не ставятся. Параграфы имеют порядковые номера в пределах глав, обозначаемые арабскими цифрами (например: 1.1. и 1.2.). Заголовки глав и параграфов в тексте работы должны располагаться по центру, точку в конце названия главы и параграфа не ставят. Не допускается переносить часть слова в заголовке.

Нумерация таблиц и рисунков может быть сквозной или соотноситься с номером главы и параграфа. Например, если таблица или рисунок включены в текст первого параграфа второй главы, нумерация следующая: Таблица 2.1.1., рис. 2.1.1. Последняя

цифра означает порядковый номер таблицы (или рисунка) в данном параграфе. Таблица помещается в качестве следующей страницы после первого упоминания о ней в тексте.

Список использованных источников должен быть оформлен в соответствии с ГОСТом «Библиографическая запись. Библиографическое описание». (ГОСТ 7.1— 2003)

Задание на подготовку курсовой работы заполняется на стандартном бланке (Приложение 2). В нем указываются тема курсовой работы, целевая установка и основные вопросы, подлежащие разработке. Задание подписывает научный руководитель курсовой работы, заведующий кафедрой и студент.

Титульный лист оформляется на стандартном бланке и содержит название темы, фамилию, имя и отчество студента, факультет; фамилию, имя и отчество, звание (должность) руководителя курсовой работы (Приложение 1). Титульный лист курсовой работы подписывается студентом, руководителем курсовой работы, консультантом, заведующим кафедрой.

6. Последовательность выполнения задания

Рекомендуется следующая последовательность выполнения курсовой работы:

1. Изучаются действующие правовые и нормативные акты, положения, инструкции по выбранной теме.
2. Изучается специальная литература: учебники, монографии, сборники научных статей, брошюры, журнальные и газетные статьи.
3. При чтении литературы необходимо делать соответствующие выписки. Делать это следует с указанием источника, чтобы потом сгруппировать их по отдельным разделам курсовой работы.
4. Составление расчетов, относящихся к теме исследования и использованных в работе в качестве иллюстративного практического материала.
5. Распределение материала в соответствии с планом курсовой работы.
6. Написание курсовой работы и сдача ее преподавателю для предварительной проверки.

Законченная и надлежащим образом оформленная курсовая работа сдается в строго установленный срок на кафедру и должна быть зарегистрирована лаборантом кафедры. Работа проверяется научным руководителем. Все замечания по ней сообщаются автору. При необходимости работа возвращается на доработку. После доработки назначается срок защиты курсовой работы.

7. Критерии оценивания работы

Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил *Оценка «отлично»* выставляется студенту, если теоретическое содержание курса освоено полностью, без пробелов, необходимые практические компетенции сформированы, исчерпывающе, последовательно, четко и логически стройно излагает материал курса, умеет увязывать теорию с практикой, свободно справляется с задачами, вопросами и другими видами применения знаний, причем не затрудняется с ответом при видоизменении заданий, использует в ответе дополнительный материал, все предусмотренные программой обучения учебные задания выполнены, качество их выполнения оценено числом баллов, близким к максимальному.

Оценка «хорошо» выставляется студенту, если теоретическое содержание курса освоено полностью, необходимые практические компетенции в основном сформированы, все предусмотренные программой обучения учебные задания выполнены, качество их выполнения достаточно высокое. Студент твердо знает материал, грамотно и по существу

излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении практических вопросов и задач, владеет необходимыми навыками и приемами их выполнения.

Оценка «удовлетворительно» выставляется студенту, если теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, необходимые практические компетенции в основном сформированы, большинство предусмотренных программой обучения учебных задач выполнено, но в них имеются ошибки, при ответе на поставленный вопрос студент допускает неточности, недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении.

Оценка «неудовлетворительно» выставляется студенту, если он не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы, необходимые практические компетенции не сформированы, большинство предусмотренных программой обучения учебных заданий не выполнено, качество их выполнения оценено числом баллов, близким к минимальному.

8. Порядок защиты работы

Защита курсовой работы является обязательной формой проверки выполнения работы. Защита производится специальной комиссией, утверждаемой директором института, состоящей обычно из 3 преподавателей кафедры, при непосредственном участии руководителя, в присутствии студентов. Результаты наиболее интересных курсовых работ могут быть доложены на научных конференциях. Публичная защита стимулирует научный интерес, творчество, ответственность студентов.

Защита состоит в коротком докладе студента по выполненной работе и в ответах на вопросы присутствующих на защите. Научный руководитель зачитывает отзыв на курсовую работу студента (*Приложение 3*).

Оценка курсовой работы заносится в зачетную книжку студента и зачетно-экзаменационную ведомость, составляемую в 2-х экземплярах, один из которых хранится на кафедре в течение всего срока обучения студента, другой представляется в дирекцию института.

Защита курсовых работ, предусмотренных учебным планом, проводится не позднее, чем за две недели до начала зачетно-экзаменационной сессии.

Студент, не представивший в установленный срок курсовую работу (проект) или не защитивший ее по неуважительной причине, считается имеющим академическую задолженность.

Выполненные работы (проекты) после их защиты должны храниться на кафедре в течение 2 лет, не считая года написания; затем работы, не представляющие для кафедры интерес, уничтожаются по акту.

9. Список рекомендуемой литературы основная литература:

1. Борисенко, В. В. Основы программирования : учебное пособие / Борисенко В. В. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2020. - 323 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-9556-00039-0, экземпляров неограничено

дополнительная литература:

1. Гарибов, А. И. Основы разработки приложений для мобильных устройств на платформе Windows Phone / А.И. Гарибов. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2020. - 460 с., экземпляров неограничено

- 2.Самойлова, Т. А. Разработка гибридных приложений для мобильных устройств под Windows Phone / Т.А. Самойлова. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2019. - 461 с., экземпляров неограничено
- 3.Соколова, В. В. Разработка мобильных приложений / В.В. Соколова. - Томск : Издательство Томского политехнического университета, 2019. - 176 с. - ISBN 978-5-4387-0369-3, экземпляров неограничено

Интернет-ресурсы:

1. <http://www.garant.ru> - Информационно-правовой портал.
2. www.biblioclub.ru – Электронно-библиотечная система «Университетская библиотека онлайн».
3. www.iprbookshop.ru - Электронно-библиотечная система «IPRbooks»
4. <http://www.consultant.ru> - Официальный сайт компании «Консультант Плюс»
5. Профессиональная база данных «Всероссийская система данных о компаниях и бизнесе «За честный бизнес» // Режим доступа: <https://zachestnyibiznes.ru>
6. Профессиональная база данных Росстата // Режим доступа:http://www.gks.ru/wps/wcm/connect/rosstat_main/rosstat/ru/statistics/databases/

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЭКОНОМИКИ И УПРАВЛЕНИЯ
КАФЕДРА «ЦИФРОВЫХ БИЗНЕС-ТЕХНОЛОГИЙ И СИСТЕМ УЧЕТА»

КУРСОВАЯ РАБОТА

по дисциплине

«Программирование в БС»

на тему:

Выполнил:

студент 2 курса группы БИН-б-о-__ - __
38.03.05 Бизнес-информатика
очной формы обучения

(подпись)

Руководитель работы:

Передереева Е.В.
(старший преподаватель кафедры ЦБТиСУ)

Работа допущена к защите _____

(подпись руководителя)

(дата)

Работа выполнена и
защищена с оценкой _____ Дата защиты _____

Члены комиссии: _____

(должность)
Фамилия)

(подпись)

(И.О.

Ставрополь, 202_г.

Институт Экономики и управления
Кафедра Цифровых бизнес-технологий и систем учета
Направление 38.03.05 Бизнес-информатика

ЗАДАНИЕ
на курсовую работу

студента _____

по дисциплине «Программирование в БС»

1. Тема работы _____
2. Цель: _____
3. Задачи: Выполнение курсовой позволит освоить компетенции ОПК-3
5. Исходные данные:
 - а) по литературным источникам основная и дополнительная литература в области конфигурировании информационных систем.
 - б) по вариантам, разработанным преподавателем
 - в) иное: _____
6. Список рекомендуемой литературы
7. Контрольные сроки представления отдельных разделов курсовой работы:

25 % - _____ 202__ г.
50 % - _____ 202__ г.
75 % - _____ 202__ г.
100% _____ 202__ г.
8. Срок защиты студентом курсовой работы _____ 202__ г.

Дата выдачи задания _____ 202__ г.

Руководитель курсовой работы

Старший преподаватель _____

Задание принял к исполнению студент _____

Отзыв

на курсовую работу студента _____ курса _____
ФИО

Актуальность: Курсовая работа посвящена

В первой главе

Вторая глава

Выводы, сделанные в Заключение, соответствуют целям, поставленным во
Введении.

Проанализирован достаточный объём основной, а также дополнительной
литературы.

За время работы студент проявил себя как

Таким образом, работа выполнена на хорошем уровне, соответствует требованиям,
предъявляемым к курсовым работам, и заслуживает положительной (____) оценки.

Научный руководитель

Старший преподаватель кафедры

Цифровых бизнес-технологий и систем учета

/ ФИО

«.....».....202__ г.