

ҚАЗАҚСТАН РЕСПУБЛИКАСЫ
БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

**Г.Б. Нурпеисова, Т.Б. Нурпеисова,
И.Н. Кайдаш, Д.В. Панюкова**

РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Учебное пособие

Алматы, 2020

ЭОЖ 004(075)
КБЖ 32.973.22я7
Н 17

*Рекомендовано УМС Казахской автомобильно-дорожной академии
им. Л.Б.Гончарова*

Рецензенты:

Бекмуханбетова Ш.А. – доктор PhD, КазАДИ им.Л.Б.Гончарова
Хасенова Г. – к.т.н., доцент МУИТ

Нурпеисова Т.Б. және т.б.

Н 17 **Разработка мобильных приложений:** учебное пособие / Г.Б.Нурпеисова, Т.Б.Нурпеисова, И.Н.Кайдаш, Д.В.Панюкова. – Алматы: «Бастау», 2020. – 324 бет.

ISBN 978-601-7991-42-5

В данном учебном пособии рассматриваются современные мобильные технологии и платформы разработки мобильных приложений. Представлена их классификация.

Учебное пособие является руководством по созданию приложений для мобильных платформ Android и, частично, iOS. Пособие представляет собой комплекс теоретических и практических учебных материалов для студентов бакалавриата по направлению подготовки 6B061 Информационно-коммуникационные технологии, магистрантов, преподавателей, обеспечивающих учебный процесс по дисциплинам, связанным с мобильными и клиент-серверными приложениями.

Содержание учебного пособия соответствует требованиям ГОСО РК Соответствует Отраслевой рамке квалификаций по информационно-коммуникационным технологиям по направлений подготовки 6B061 Информационно-коммуникационные технологии.

ЭОЖ 004(075)
КБЖ 32.973.22я7

ISBN 978-601-7991-42-5

© Нурпеисова Г.Б., Нурпеисова Т.Б.,
Кайдаш И.Н., Панюкова Д.В., 2020
© «Бастау», 2020

СОДЕРЖАНИЕ

Введение	6
1 Мобильные приложения и технологии	7
1.1 Классификация мобильных приложений	7
1.2 Мобильные приложения для бизнеса.....	12
1.3 Мобильные приложения для игр.....	20
1.3.1 Этапы разработки.....	20
1.3.2 Программные средства разработки мобильных игр	22
1.3.3 Ошибки при разработке.....	23
1.3.4.Команда разработчиков игр	24
1.4 Мобильные социальные сети	26
1.5 Мобильные облачные решения.....	34
1.6 Классификация мобильных устройств	38
1.7 Коммуникационные технологии.....	44
1.7.1 Технология Wi-Fi. Стандарты.....	44
1.7.2 Стандарты GSM – поколение сетей сотовой связи	48
1.7.3 Технология Bluetooth	58
1.7.4 Проблемы безопасности мобильной связи	60
1.8 Реализация мобильных приложений на платформе Java 2 Micro Edition.....	65
1.8.1 Конфигурации и профили J2ME.....	67
1.8.2 Конфигурация Connected, Limited Device Configuration ..	68
1.8.3 Конфигурация Connected Device Configuration	69
1.8.4 Профиль Foundation Profile.....	70
1.8.5 Профиль Personal Profile	71
1.8.6 Профиль RMI	72
1.9 Мобильные операционные системы	72
1.10 Конструкторы мобильных приложений.....	77
1.11 Рынок конструкторов мобильных приложений	80
2 Процессы разработки мобильных приложений.....	95
2.1 Этапы разработки мобильных приложений.....	95
2.2 Особенности разработки интерфейсов для смартфонов. Принципы юзабилити.....	99
2.2.1 Строительные блоки визуального дизайна интерфейсов	100

2.2.2	Использование визуальных свойств для группировки элементов и создания четкой иерархии	102
2.2.3	Выравнивание	103
2.2.4	Сетка	104
2.2.5	Логические маршруты	105
2.2.6	Текст в графических интерфейсах	106
2.2.7	Цвет в графических интерфейсах	107
2.2.8	Поведение окон и определение компоновки	108
2.2.9	Проектирование для различных потребностей	109
3	Разработка мобильных приложений в среде Processing	114
3.1	Основы работы в Processing	114
3.2	Загрузка и знакомство со средой Processing	115
3.3	Функции разработчика на мобильном телефоне	123
3.4	Написание программы на мобильном телефоне	128
3.5	Использование цветов в программе	131
3.6	Голограммы. Добавление объема. Перемещение	137
3.7	Типы данных в Processing	142
3.8	Функции нажатия и протяжки на сенсорном экране	146
3.9	Прописанные условия	148
3.10	Циклы	151
3.11	Массивы данных	153
4	Как мы создали свой проект в среде Processing	156
4.1	Планирование работы над проектом	156
4.2	Вставка рисунков, как объектов	161
4.3	Установка локальных размеров объектов	165
4.4	Создание полноценных кнопок	168
4.5	Ограничение перемещений и изменение объектов	173
4.6	Создание функций	177
4.7	Добавление объектов. Создание экрана проигрыша	186
4.8	Работа с текстом	191
4.9	Создание меню	195
4.10	Последние шаги перед выгрузкой в PlayMarket	198
5	Разработка мобильных приложений в Android Studio	202
5.1	Создание нового проекта	202
5.2	Редактирование манифеста	204
5.3	Добавление и редактирование файлов приложения	206

5.4	Создание и редактирование файлов для визуализации.....	215
5.5	Добавление ресурсных файлов	221
5.6	Добавление библиотек и запуск.....	228
6	Построение базового интерфейса для IOS	231
6.1	Создание нового проекта	231
6.2	Знакомство с Xcode	235
6.3	Обзор исходного кода.....	238
6.4	Раскадровка.....	242
6.5	Создание базового пользовательского интерфейса.....	244
6.6	Предварительный просмотр интерфейса.....	252
6.7	Применение Auto Layout	256
7	Добавление логики к базовому интерфейсу в IOS	264
7.1	Базовые понятия	264
7.2	Создание выводов для элементов пользовательского интерфейса	266
7.3	Добавление действия	271
7.4	Процесс ввода данных пользователем	275
8	Создание мобильного Web-приложения	282
8.1	Web-разработка как основа мобильного приложения	282
8.2	Разработка игры как Web-приложения.....	286
8.3	Средства для разработки мобильных Web-приложений	296
8.4	Создание мобильного Web-приложения.....	302
	Глоссарий.....	308
	Вопросы для самопроверки	320
	Список использованных источников.....	323

ВВЕДЕНИЕ

Современный человек делает все для того, чтобы достигнуть максимального комфорта и удобства в своей жизни. Для этого сегодня специалистами в области информационных технологий разрабатываются мобильные приложения, которые позволяют решать огромное количество задач в разных областях жизнедеятельности человека.

Все мобильные приложения условно можно поделить на программы для рабочих целей и на развлекательные программы. Первые позволяют контролировать и оптимизировать рабочие процессы, составлять аналитическую отчетность, выполнять иные функциональные задачи. Вторые – позволяют интересно и разнообразно проводить время.

Однако, как показывает практика, большим спросом сегодня пользуется *специализированный софт*. Именно на таких программах можно создать собственный бизнес, так как современные компании не жалеют инвестиций в продукты, которые могли бы в какой-либо степени оптимизировать или упростить имеющиеся бизнес-процессы.

На протяжении последних лет показатель, характеризующий уровень спроса на мобильные устройства, постоянно растет. Такая статистика позволяет сделать вывод о том, что разработка мобильных приложений актуальна и целесообразна.

Мобильное приложение – это программный продукт, предназначенный для использования на мобильных устройствах оснащенных операционной системой. Мобильные приложения могут быть установлены на устройстве с завода-изготовителя, либо скачаны с флэш-носителей или загружены из онлайн магазинов, где за это может взиматься плата либо доступны в бесплатном доступе.

В данной книге рассмотрим краткую историю развития мобильных устройств и технологий, классификацию мобильных приложений. Изложим основные этапы их разработки с использованием языка программирования Kotlin. Читатель сможет ознакомиться не только с основами создания мобильных приложений, но и узнает особенности их адаптации к уже существующим сервисам, а также получит ценные методические указания по проектированию и написанию программных кодов для нового контента в современной среде Processing.

1 МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ И ТЕХНОЛОГИИ

1.1 Классификация мобильных приложений

Эксперты и аналитики в области ИТ сходятся во мнении, что в 2019 г. динамика роста использования мобильных устройств останется положительной. Отметим, что 90% времени пользователи проводят в различных мобильных приложениях и только 10% приходится на работу с браузером. Вместе с тем провайдеры сотовых сетей отмечают, что количество активных подключений к сети Интернет из мобильных приложений постоянно увеличивается, что свидетельствует о том, что мобильное приложение часто выступает как клиент и требует активного подключения к сети Интернет (карты-навигация, социальные сети, мессенджеры). Соответственно, это поддерживает необходимость использования стандартных протоколов и технологий сети Интернет: HTTP, HTTPS, POP3, IMAP, Web Dav и других [3].

Мобильные приложения, доступные для покупки и установки на смартфон, имеют тематический контент, четко структурированы по специальным рубрикам и ориентированы на информационные потребности определенных категорий пользователей.



Рассмотрим некоторые категории мобильных приложений.

Категории мобильных приложений по исполняемым функциям.

К наиболее востребованным категориям мобильных приложений следует отнести следующие:

- 1) развлечения (игровые приложения, мультимедиа, музыка, заказ билетов в театр, кино и тому подобное);

- 2) путешествия (заказ отеля, аренда авто, услуги гида, сервис онлайн-переводчика и тому подобное);
- 3) бизнес (финансовые приложения, планирование, торговля, приложения для города, поиск работы и тому подобное);
- 4) социальные приложения (социальные сети, глобальные «брендированные» сети, специализированные (клубные) сети и тому подобное);
- 5) еда (заказ и доставка еды, геолокация заведения питания, рецепты);
- 6) спорт (спортивные новости, покупка билетов на спортивные мероприятия, игровые симуляторы);
- 7) образование (обучающие программы, интерактивные курсы и тому подобное);
- 8) новости (дайджесты, ленты, рейтинги).

Классификация мобильных приложений по роду деятельности:

- 1) *Контентные приложения.* Обладают большой популярностью, основные задачи, которых они выполняют это: прослушивания музыки, просмотры фильмов и фотографий, чтения цифровых книг и журналов. Также к ним можно отнести информационные приложения, например, предоставляющие информацию о погоде, расписания городского транспорта, свежих новостях, рецепты или разработанные специально к каким либо намечающимся событиям, таких как спортивные чемпионаты, выставки или форумы. Ну и конечно специальные рекламные приложения.
- 2) *Бизнес приложения.* Разработаны для помощи в офисной работе, расчетах, обмене служебными данными, обеспечивающие доступ к интернет-магазинам, платежным системам и банковским счетам. На данный / момент сегмент бизнес-приложений является более интересным для инвесторов, но сложность состоит в переводе бизнес-задач на мобильные устройства.
- 3) *Мобильные игры* – это наиболее востребованный сектор мобильных приложений.
- 4) *Мобильные социальные сети.* Данный вид с каждым днем набирает все большую популярность, увеличивая многочисленную аудиторию во всех странах мира, чему способствует развитие мобильного интернета расширяющего свою доступность по всей планете.

Классификация мобильных приложений по виду монетизации:

- 1) Платное приложение, реализуемое посредством продажи в магазине.

- 2) Бесплатное приложение с платной подпиской.
- 3) Бесплатное приложение со встроенными покупками.
- 4) Бесплатное приложение с рекламой внутри приложения.

В аспекте монетизации и извлечения прибыли выбор у разработчика невелик: продажа по фиксированной цене, бесплатное приложение с отображением рекламы, продажа уникальных свойств внутри приложения.

Следует отметить, что представленный список категорий постоянно развивается и пополняется. Более того, происходит трансформация мобильных приложений в новые виды сервисов, включающие 3D-изображения, дополненную реальность, трекеры физического состояния человека, носимые гаджеты, мобильных агентов в системах безопасности, интеллектуальных агентов в гибридных сетях и когнитивных системах и многое другое.

Также необходимо отметить, что **мобильные приложения по принадлежности к разработчикам** делятся на:

- 1) *нативные* (создаются поставщиками платформ и загружаются через их магазины приложений). Это самый распространенный тип приложений. Они создаются на основе языков программирования для таких операционных систем, как Android, Windows Phone и iOS. Благодаря таким приложениям можно в полной мере использовать функционал GPS, видеокамер или датчиков ускорения. Главным преимуществом нативных приложений является возможность автономной работы без необходимости подключения к интернету. Чаще всего приложения такого типа распространяются через магазины приложений Play Market, AppStore и другие. Также нативными приложениями могут являться различные электронные библиотеки.
- 2) *кросс-платформенные* (HTML5, веб-сайт и веб-приложение, оптимизированные под мобильные устройства). Данный тип приложений использует технологию WEB для возможности работы на мобильном устройстве. Главным и несомненным преимуществом таких приложений является возможность единовременного создания на все типы платформ. Также мобильная версия сайта имеет весомое преимущество – кроссплатформенность. Однако из минусов стоит отметить, что такие приложения не позволяют использовать функции камеры или геолокации в смартфоне, а также их невозможно загрузить из магазина приложений.

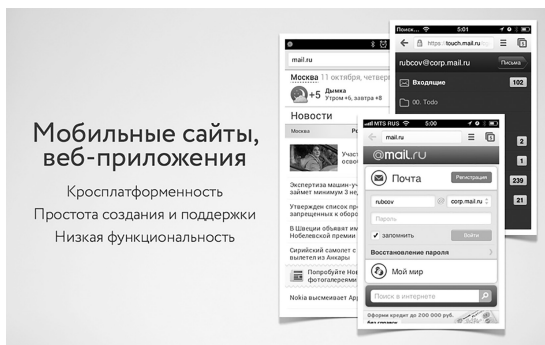


Рисунок 1.1 – Мобильные сайты, веб-приложения

- 3) *стандартные* (гибридные) приложения (содержат в себе некоторые функции нативных и веб-приложений). Программирование под Android просто не может работать без гибридных приложений. Они – это соединенные воедино нативные и веб-приложения. Главными преимуществами гибридных приложений являются кроссплатформенность на web-технологиях и возможность доступа к функциям смартфона или другого мобильного устройства. Хотя стоит отметить, что разработка приложений для IOS гибридного типа стоит значительно дороже, чем разработка приложений иного типа. Многие популярные социальные сети имеют свои гибридные приложения, которые можно свободно загрузить из онлайн-магазина.



Рисунок 1.2 – Гибридные приложения

Выбор приложения в этом случае представляет интерес для разработчиков с точки зрения быстродействия, независимости от

поставщика, безопасности работы и ряда других факторов, которые приведены в таблице 1.1.

Таблица 1.1

**Критерии сравнения
различных типов мобильных приложений**

Тип приложения (критерии)	Доступ к функционалу устройства	Скорость работы	Стоимость разработки	Распространение через магазин	Процесс одобрения
Нативное	Полный	Высокая	Высокая	Доступно	Обязательный
Гибридное	Частичный	Средняя (зависит от скорости интернета)	Доступная	Доступно	Обязательный
Веб	Отсутствует	Средняя (зависит от скорости интернета)	Доступная	Недоступно	Отсутствует

Как видно из таблицы, каждый представленный тип мобильного приложения имеет свои преимущества и недостатки, поэтому при принятии решения о выборе технологии для разработки приложения необходимо учитывать представленные в таблице функциональные, технические и коммерческие требования, а также ряд других требований, отражающих специфику конкретного приложения [4].

Тем не менее, ценность мобильных приложений часто определяется не только их уникальными свойствами, но все в большей мере их способностью быть эффективной частью большой информационной экосистемы, которая позволяет в реальном времени решать задачи поиска необходимой информации, коммуникации с людьми, сервисами и устройствами, информационно-аналитического сопровождения клиента и тому подобное.

Ниже рассмотрим наиболее востребованные секторы мобильных приложений: *мобильные приложения для бизнеса, мобильные игры, мобильные социальные сети.*

1.2 Мобильные приложения для бизнеса

Приложения для мобильных устройств сегодня проникают во многие сферы бизнеса. Они уже прочно укоренились в таких сферах как издательство, e-commerce, сфера услуг и так далее. Сегодня мобильные приложения становятся мощным маркетинговым инструментом, который позволяет решать различные задачи, стоящие перед бизнесом: привлечение клиентов; повышение продаж; создание положительного имиджа компании; оптимизация процессов коммуникации с клиентами и партнерами. Такие программы могут выступать в качестве, как основного, так и дополнительного канала коммуникации с целевой аудиторией. Разработка и создание мобильных приложений для Android и iOS, React JS полезна как стартаперам, так и уже существующим, успешно функционирующим проектам, в которые приложения смогут вдохнуть новую жизнь.

Приложения для бизнеса могут быть разные:

Интернет-магазины, главной целью которых является продажа товаров и услуг. По статистике сегодня более 70% пользователей совершают покупки именно посредством мобильных устройств. Неудивительно, что это достаточно эффективный канал продаж.

Приложения-службы, которые могут быть представлены в виде адаптивных сайтов или их различных упрощенных версий. С помощью таких приложений очень удобно информировать клиентов о своей компании, товарах и услугах, которые она предоставляет, действующих скидках и так далее.

Корпоративные мобильные приложения, которые являются дополнительным способом формирования корпоративной культуры и налаживания коммуникации внутри компании [15].

Рассмотрим некоторые категории программ, которые могут быть полезны для предпринимателей. Из каждой категории мы постарались выбрать лучшие приложения для Android и iPhone.

1) Менеджер финансов

Держать состояние счётов под строгим надзором – полезная практика для каждого. А для предпринимателя это просто жизненно необходимо. Эти программы для управления финансами помогут следить за доходами и тратами, а также оптимизировать доступный бюджет.

Money Lover

Money Lover – приложение для тех, кто хочет держать свои финансы под контролем. С его помощью можно следить за тратами,

формировать бюджет и всегда быть в курсе того, сколько имеется сбережений. Кроме того, приложение умеет фиксировать долговые обязательства и постоянные выплаты, а также напоминать о необходимости совершить очередной платёж.

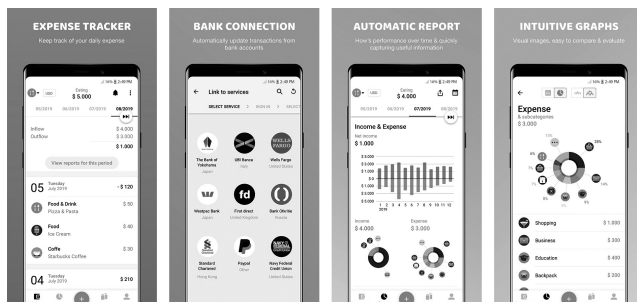


Рисунок 1.3 – Интерфейсы приложения Money Lover

Money Manager

Это очень функциональное приложение. Оно позволяет контролировать доходы, расходы, а также предоставляет исчерпывающую статистику за любой интересующий клиента период времени. Среди наиболее значимых функций можно отметить следующие: доступ к данным с компьютера, применение системы двойной записи, планирование бюджета по отдельным категориям, управление кредитными и дебетовыми картами.

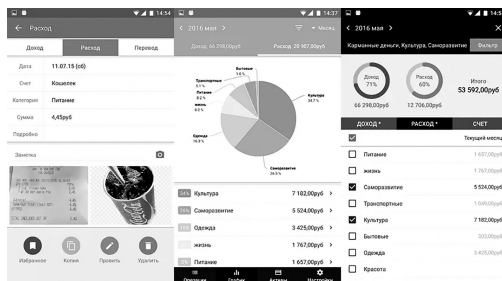


Рисунок 1.4 – Интерфейсы приложения Money Manager

Дзен-мани: учёт расходов

«Дзен-мани» умеет самостоятельно фиксировать расходы, что избавляет от необходимости вручную вносить каждый платёж и покупку. Для этого к приложению можно подключить импорт опера-

ций из «Сбербанка», «Альфа-банка», «Яндекс.Денег», Webmoney или QIWI.

Кроме того, приложение умеет получать информацию о списании денег из приходящих СМС от всех крупнейших банков России, Казахстана, Беларуси. С помощью «Дзен-мани» клиент сможет увидеть общую картину своих финансов и понять, сколько денег свободно и сколько надо зарезервировать на оплату счетов.

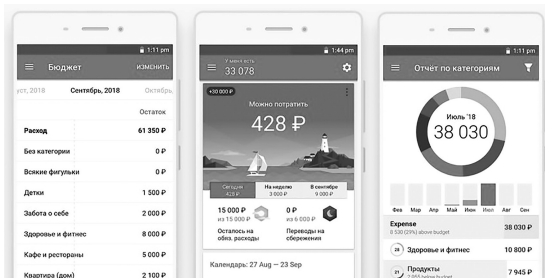


Рисунок 1.5 – Интерфейсы приложения Дзен-мани: учёт расходов

Bills Monitor

Каждому из нас приходится ежемесячно совершать большое количество обязательных выплат. Арендная плата, коммунальные услуги, кабельное телевидение, интернет, языковые курсы, тренажёрный зал и многое другое. Просрочка любого из этих платежей создаст проблемы или лишит каких-либо благ, так что лучше постараться про них не забывать.

С приложением Bills Monitor пользователи будут абсолютно уверены, что вовремя заплатили по всем счетам, а также получают представление, сколько же денег остаётся в итоге на жизнь.



Рисунок 1.6 – Интерфейсы приложения Bills Monitor

2) Мессенджер

Развитие интернета отодвигает сотовую связь на второй план. Звонить с помощью мессенджеров можно в любую точку мира, где работает Сеть. Причём абонент может общаться бесплатно и совершать не только голосовые, но и – в некоторых программах – видеозвонки. Общаться с партнёрами или устраивать международные конференции стало очень просто.

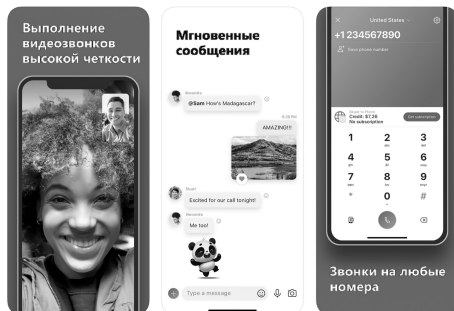


Рисунок 1.7 – Интерфейсы мессенджеров (снимки экрана iPhone)

3) Почтовый клиент

Несмотря на засилье мессенджеров и социальных сетей, главным каналом деловой переписки всё ещё остаётся электронная почта. А чтобы использовать её с максимальным удобством, нужен хороший почтовый клиент. Это может быть популярнейший Gmail, но клиентам также нравится Spark для iOS и Mac с его умными функциями сортировки входящей информации.

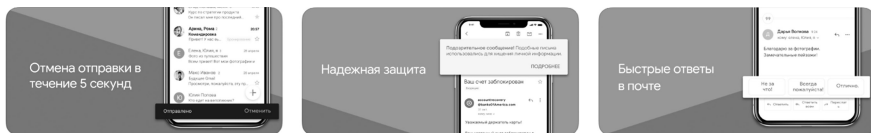


Рисунок 1.8 – Снимки экрана iPhone, iPad

4) Календарь

Время – не менее ценный для делового человека ресурс, чем деньги. Если хотите взять его под контроль и не упустить ни одного важного события, пользователям нужен наглядный и функциональный календарь.

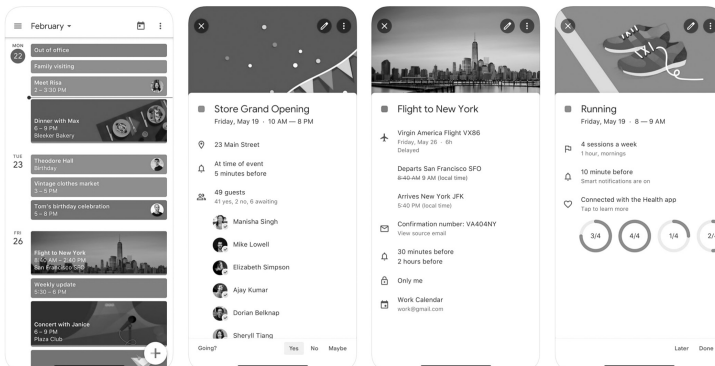


Рисунок 1.9 – Интерфейсы календаря в приложении

5) Менеджер задач

Удерживать в голове все рабочие дела бывает очень сложно. Мобильные приложения предлагают планировщики задач, которые всегда напомнят, если пользователь вдруг забудет, о каком-то важном деле.



Рисунок 1.10 – Снимки экрана iPhone, iPad, Apple Watch

6) Блокнот

Цифровой блокнот – инструмент для фиксации важных идей в формате скетчей, текстовых или голосовых заметок. Некоторые программы из этой категории – например, One Note – также подходят для работы с деловыми документами. Не обязательно носить с собой бумажные блокноты, значимые мысли можно продублировать на гаджете.

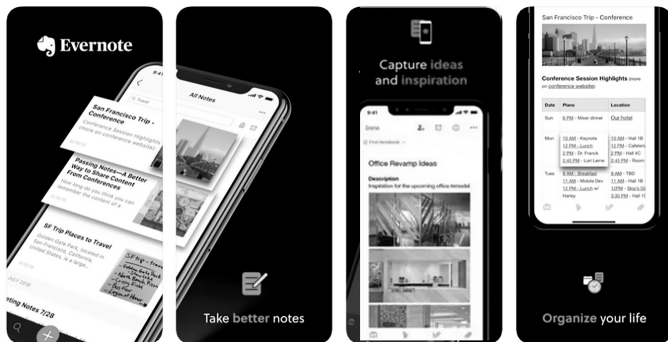


Рисунок 1.11 – Снимки экрана iPhone, iPad, iMessage, Apple Watch

7) Сканер документов

В дополнение к цифровому блокноту удобно использовать сканер документов. Такая программа быстро оцифрует почти любой бумажный документ с помощью фотокамеры смартфона. Распознанный текст затем можно сохранить в блокноте или отправить нужному человеку.



Рисунок 1.12 – Снимки экрана iPhone, iPad

8) Менеджер презентаций

Предприниматели часто имеют дело с презентациями. С помощью этого формата очень удобно доносить информацию до партнёров, инвесторов или подчинённых. Создавать презентацию на маленьком экране – не самая лучшая идея. Зато мобильный менеджер презентаций позволяет быстро повторить материал перед выступлением, когда под рукой нет компьютера.

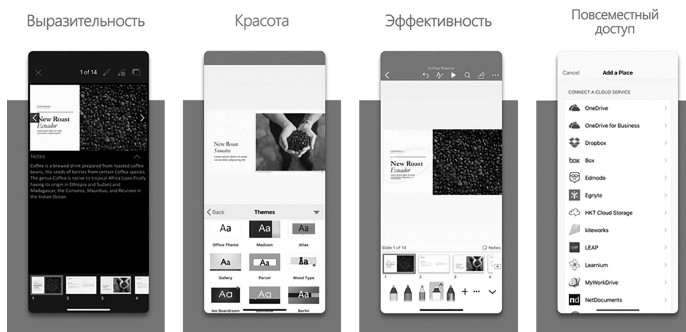


Рисунок 1.13 – Снимки экрана iPhone, iPad, Apple Watch

9) Облачный диск

Сложно придумать более удобное хранилище для снимков, видеороликов, презентаций и прочих файлов, чем облачное хранилище. Достаточно загрузить данные на сервер с помощью специальной программы, и они всегда будут доступны с любого подключённого к диску устройства.

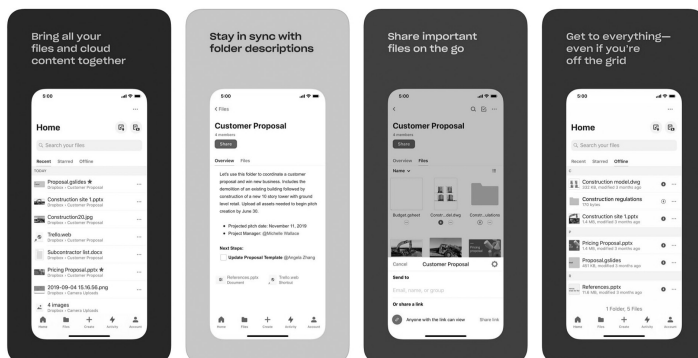


Рисунок 1.14 – Снимки экрана iPhone, iPad, iMessage

10) Агрегатор новостей

Чтобы бизнес процветал, предприниматель должен быть «начеку» и следить за событиями, будь то колебания в курсе биткоина, действия конкурентов или изменения в законодательстве. Алгоритмы социальных сетей могут не показать нужную информацию вовремя. Здесь могут помочь RSS-агрегаторы.

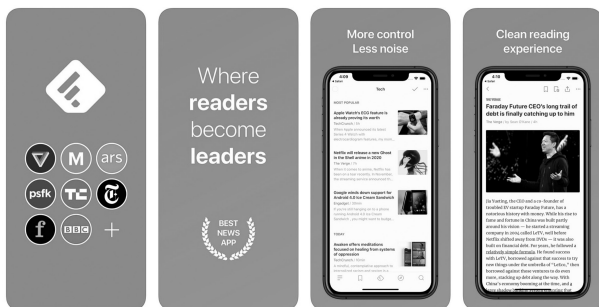


Рисунок 1.15 – Снимки экрана iPhone, iPad

11) Путеводитель

Чтобы не заблудиться в незнакомом городе и так далее, необходимо установить путеводитель на планшет или смартфон. Такие приложения отображают карты населённых пунктов вместе с данными об их инфраструктуре, маршрутах и различных интересных местах.

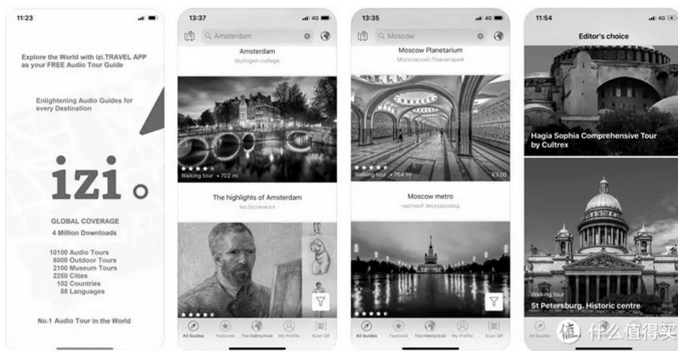


Рисунок 1.16 – Снимки экрана iPhone, iPad

12) Плеер для аудиокниг

Деловая литература – хороший источник новых знаний и идей, с помощью которых предприниматель может прорекламировать себя и свой бизнес. Но что делать, когда нет времени читать? Выход – слушать книги во время прогулок, в полёте или за рулём. В приложении «Слушай» аудиокнижки можно покупать по одной. В Bookmate можно получить неограниченный доступ к библиотеке в рамках платной подписки.

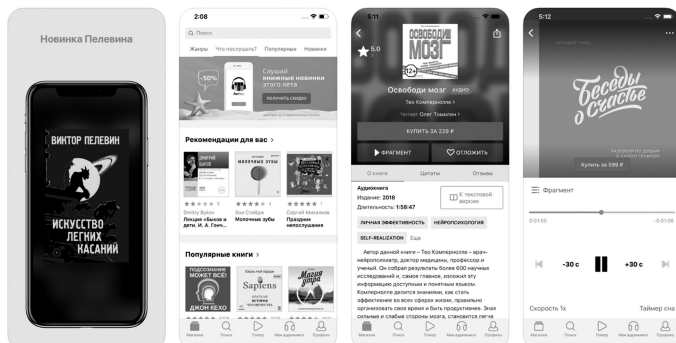


Рисунок 1.17 – Снимки экрана iPhone, iPad, Apple Watch, Apple TV

1.3 Мобильные приложения для игр

Сегодня мобильные игры считаются самым быстроразвивающимся направлением игровой индустрии. Большинство людей играет в них ради времяпрепровождения, но для некоторых это увлекательное хобби. Эта популярная категория мобильных приложений охватывает невероятно широкую аудиторию, что раскрывает потенциал для бизнеса.

Скорее всего, всем знакомы такие игры, как Angry Birds, Clash of Clans, Plants vs. Zombies. Они побудили многих людей погрузиться в индустрию мобильных приложений и начать разработку собственного продукта, несмотря на то, что это требует знания технологий, творческого мышления, умения эффективно выстраивать техпроцессы [16].

Чтобы создать качественную игру, которая сможет стать прибыльной, нужно: изучить рынок и придумать идею, разработать увлекательный игровой процесс, выбрать платформу, создать дизайн, построить план монетизации приложения. Существует несколько этапов при разработке мобильных игр.

1.3.1 Этапы разработки Формирование идеи

Это важный шаг, который будет способствовать дальнейшему успеху всего проекта. С учётом огромного количества идей, уже реализованных в мобильной индустрии, нужно предложить нечто свежее, либо предоставить что-то уже знакомое, но в новой обёртке. Главное –

она должна быть обращена к массовой аудитории для получения потенциально большей прибыли. Поэтому надо не забывать о людях, которые будут играть в созданную разработчиком новинку.

Прежде всего, надо решить, какая игра будет разработана. Выбрать категорию игр и тип аудитории, на которую необходимо ориентироваться. Определить жанр: аркада, симулятор, экшен, стратегия или RPG и прочее.

Втянуть в игровой процесс

Фактор небольшой зависимости от игрового процесса позволяет удержать большинство пользователей. Существует проверенный способ вовлечения в игру – она должна быть лёгкой и увлекательной, с постепенно увеличивающимся уровнем сложности. Как правило, игроки довольно часто теряют интерес к слишком длительным играм, а чтобы привлечь его, можно применять такие приёмы: создавать короткие уровни, где будет много разблокируемого контента, придумывать различные бонусы, секретные предметы. Всё ради того, чтобы пользователь сохранил интерес к игре.

Выбрать платформу (ОС)

При разработке игр для мобильных устройств необходимо выбрать платформу. Обычно выбор ядра сводится к Android или iOS. Удобнее было бы упростить эту задачу, перейдя к гибридной модели, но при кроссплатформенной разработке потребуются учесть дополнительные расходы. Здесь решающим фактором должен быть целевой рынок. Сосредоточится необходимо на операционную систему, которую будут использовать самые прибыльные игроки, – это поможет расставить приоритеты. Если же дополнительные расходы разработчика не пугают, то безошибочным вариантом будет поддержка сразу двух платформ.

Создать уникальный дизайн

Разработать дизайн для игры намного сложнее, чем для обычного мобильного приложения. Нужно учесть тысячу нюансов, уделить внимание деталям. Совершенство складывается из мелочей, а это обязательно заметят искушённые пользователи. В итоге именно симпатичный дизайн может стать оригинальной чертой, которая позволит выделиться на конкурентном рынке.

Спланировать стратегию монетизации

Есть много игр, которые совсем не приносят прибыль. Но разработчикам лучше заранее составить стратегию монетизации, прежде чем начинать разработку. Развитие проекта может обходиться дорого

и требовать большого количества времени, так что должен быть план по возврату инвестиций. Вот несколько распространённых способов монетизации мобильных игр:

- 1) *Покупки в приложении*. Модель Freemium – наиболее распространённый вид монетизации. Несмотря на то, что такие покупки составляют только 2% в Android-играх и чуть больше в iOS-играх, это эффективный способ, который принесёт доход.
- 2) *Реклама в приложении*. Во множестве развлекательных приложений реклама сочетается с внутри игровыми покупками. Это хорошее решение, поскольку остальные способы сами по себе не могут принести существенный доход. Только здесь важно уделять внимание содержанию рекламы, ведь если оно не связано с тематикой, то результаты будут непродуктивными.
- 3) *Премии-версии*. Модель Premium предлагает бесплатную пробную версию или демо-версию и просит игроков заплатить за дальнейшее использование. Конечно, можно сразу запросить оплату, но это значительно сократит количество покупок.
- 4) Заказать разработку игры у профессионалов.

Конечный шаг к созданию успешного продукта – выбор компании по разработке мобильных игр. Нужна команда опытных разработчиков, которые обладают необходимыми навыками, чтобы превратить идею в прибыльный бизнес. Разработка игр и так сложна, а создание уникальных и запоминающихся проектов требует профессионального подхода [16].

1.3.2 Программные средства разработки мобильных игр

Игровая индустрия предоставила разработчикам множество пригодных инструментов для создания игр. Популярными являются три игровых ПО – Unity, Unreal Engine, AppGameKit.

Unity

Это интегрированная среда разработки, которая предлагает мульти-платформенные решения для создания 2D и 3D игр. Она включает в себя огромное количество функций, предустановленных моделей, текстур и документации. Разработчики используют эту среду для создания любых игр – от простых до класса AAA.

Плюсы Unity:

- создание объектов, добавление к ним различных компонентов;
- функциональный графический редактор;
- встроенный режим Play Mode для быстрой отладки;
- поддержка множества платформ, API, технологий.

Unreal Engine

Проверенное временем программное обеспечение для разработки игр, преимущественно класса AAA. Многие игры мирового масштаба использовали его в качестве основы. Оно позволяет создавать игры, которые будут поддерживаться большинством платформ и операционных систем.

Unreal Engine предлагает разработчикам два основных инструмента: традиционный C++ и визуальные сценарии Blueprints, которые помогают быстро разобраться с игровой логикой. Программное обеспечение разработано для удовлетворения потребностей как гигантских, так и небольших игровых проектов.

AppGameKit

Программное обеспечение для мульти-платформенных игровых продуктов, подходящий для любой независимой студии разработки мобильных игр.

Плюсы AppGameKit:

- быстрый компилятор, большое количество готовых решений для удобной разработки;
- кроссплатформенность;
- встроенные средства монетизации игр.

1.3.3 Ошибки при разработке

Игровой мир не стоит на месте – прогресс подарил пользователям дополненную и виртуальную реальность, которая открывает новые возможности, как для разработчиков, так и для заказчиков. Теперь все стремятся выпустить супер-хит для мобильных устройств. Но многие не учитывают риски и допускают типичные ошибки в процессе работы.

Рассмотрим их:

- 1) *Перегрузка большим количеством функций.* Не стоит выпускать мобильную игру сразу со всем ассортиментом функций, пытаясь включить все задумки в первый релиз. Лучше запускать мобильную игру поэтапно. Каждая следующая версия с новыми возможностями даст представление о поведении пользователя, и разработчик сможет решить, какие очередные улучшения стоит добавлять.
- 2) *Непродуманная политика монетизации.* Нужно заранее продумать способ монетизации игры. Выбор модели зависит от целей (например, достижение доходности), а также от рыночного спроса.

- 3) *Потеря интереса пользователями.* Чтобы удержать игроков, надо постоянно прилагать усилия. Прежде чем привлекать новых пользователей, необходимо выяснить, почему уходят старые пользователи. Причинами могут быть: ошибки, отсутствие геймификации (сервисы и приложения, которые используют наработки из игровой индустрии (игровые механики) для вовлечения, удержания пользователей и многого другого), сложный интерфейс, лучшие альтернативы и прочее. Пользователю даже может просто наскучить однотипность. Поэтому придётся регулярно обновлять игру новыми уровнями, персонажами, элементами дизайна и многим другим. Здесь будет полезно изучать отзывы игроков, которые будут выступать в качестве ориентиров для устранения ошибок и внедрения новых идей.
- 4) *Слабое продвижение.* Даже идеальная игра не окупится без раскрутки. Постоянное привлечение пользователей важно для динамики доходов. Чтобы приложение стало вирусным, нужны инвестиции в маркетинг, брендинг и продвижение. Крупный бюджет позволит разработчикам получать обзоры в знаменитых блогах, размещать рекламу в социальных сетях, на игровых порталах. Только разносторонний подход принесёт желанный результаты [16].

1.3.4 Команда разработчиков игр

В процессе производства игры в дело вступают представители *множества* «игровых» профессий. При разработке небольшого проекта в рамках ограниченного бюджета один человек может совмещать в себе обязанности целой команды разработчиков. Рассмотрим основных специалистов:

Программисты

Программисты заняты работой по написанию программного кода игры. Их усилиями реализуется игровая физика, *искусственный интеллект*, с которым предстоит сражаться игроку при игре «против компьютера» и многое другое. Многие игровые программисты стали таковыми после того, как начали программировать самостоятельно, в виде хобби.

Если говорить об инструментах программиста – то практически все коммерческие игры написаны на языке C++ или C#, некоторые, особенно ответственные части игры, пишут на языке *Assembler*. Строго говоря, игру реально написать на практически любом языке програм-

мирования – например, простые игры можно создавать в Microsoft Word или Microsoft Excel, используя встроенные языки программирования, например, *Visual Basic For Applications*.

Художники

Роль художников и вообще всех, кто работает с графикой, в современном игровом строении трудно переоценить. Во все времена одним из критериев оценки игры была ее графическая составляющая, а современные средства работы с графикой позволяют создавать красивейшие игровые миры, в основе которых лежит кропотливая работа художников во всех ее проявлениях.

К тому же, аниматоры оснащены сегодня передовым аппаратным и программным обеспечением (в частности – технологией *Motion Capture*) которое, например, позволяет записывать реальные движения человека и переносить их потом на игровых персонажей.

Музыканты

Композиторы, музыканты, актеры, звукорежиссеры работают над звуковым оформлением игры. Они пишут и исполняют музыку, читают тексты персонажей. Без достойной музыки и качественного озвучивания, как и без хорошей графики, современная игра вряд ли будет успешной.

Для конструктивных и эффективных рабочих отношений композитору и разработчику мобильных игр необходимо как можно раньше выработать общее представление о саундтреке. Это важное условие успешности конечного результата. Причина для этого проста – каждый слышит и понимает музыку и звуковые эффекты немного по-разному. Кто-то может услышать «мощь» в рифмах из хэви метала, но не почувствует ее в искаженном цифровом синтезаторе (*distorted digital synths*). Следовательно, важно разработать собственное понимание того, какие инструменты и звуки передают верное настроение, и принимать окончательное решение совместно с разработчиком игры.

Сценаристы

Стратегии, головоломки, квесты для ПК, как фильмы или театральные постановки, должны иметь оригинальный сюжет, разработкой и адаптацией которого занимается сценарист компьютерных игр. Он придумывает и расписывает сценарий, выбирая актуальные темы, учитывая возраст потенциальной целевой аудитории, создавая несколько сюжетных линий.

Сценарист мобильных компьютерных игр – востребованная профессия. Сегодня разные виды игр для мобильных устройств пользуются огромной популярностью, поэтому авторов ищут крупные

отечественные и зарубежные компании, такие как Mail.Ru Group, «Яндекс» и другие.

Для работы в этой сфере необходимо хорошо развитое креативное мышление, отличная фантазия, эрудированность. Эти качества позволяют создать уникальный и интересный сценарий, который привлечет максимальное количество игроков. Чаще всего специалисты имеют высшее филологическое образование, кроме этого им рекомендуется пройти базовый обучающий курс по геймдизайну или нарративному дизайну.

1.4 Мобильные социальные сети

Мобильная связь становится все более интернет-ориентированной. Для продвинутых пользователей общение в социальных сетях с мобильного телефона не менее привычно, чем обычные звонки или SMS-сообщения.

Тем более, что безлимитные тарифные опции, становящийся массовым Wi-Fi, популяризация смартфонов и существующее множество приложений для социальных сетей буквально толкают людей в «мобильную социальность».

Все многообразие «социальных» приложений можно разделить на две большие группы. В первую из них входят официальные приложения, разработанные той или иной социальной сетью, а вторая включает в себя приложения сторонних разработчиков. Несколько особняком стоят программы производителей мобильных телефонов, причем большая часть из них – это просто ярлык для доступа к мобильному сайту социальной сети, хотя встречаются и весьма функциональные продукты [18].

Среди мобильных платформ в плане выхода официальных клиентских приложений для социальных сетей больше всего повезло iOS и Android, для которых все крупные социальные сети выпустили собственные приложения. Впрочем, это неудивительно – перечисленные платформы активно развиваются, а основная часть их пользователей сравнительно активно использует Интернет со своих мобильных устройств. Рассмотрим несколько популярных мобильных социальных сетей.

Facebook

Официальный мобильный клиент этой социальной сети существует в виде версий для iOS, BlackBerry, Android и webOS. Типичный набор

возможностей полнофункциональных клиентов Facebook включает в себя функции обновления статуса, просмотра ленты сообщений, событий, стен, информации о пользователях, обмена ссылками, добавления сообщений, загрузки фотографий и просмотра видео. В 2011 году Facebook выпустил еще одно официальное мобильное приложение под названием Facebook Messenger. Данное приложение предназначено для обмена мгновенными сообщениями и видео. Оно интегрировано с системой обмена сообщениями на основном сайте Facebook и построено на базе открытого протокола MQTT. По данным на апрель 2017 года месячная аудитория мессенджера составляла 1 млрд.

Не забыли в Facebook и владельцев обычных мобильных телефонов, чей функционал можно расширить только за счет JAVA-приложений. Была выпущена программа, суть которой следует из ее названия – Facebook for Every Phone. Список совместимости приложения включает в себя более 2500 различных моделей мобильных телефонов. Он позволяют читать ленту новостей, просматривать и загружать фото, переписываться с друзьями, редактировать свой личный профиль и даже создать новую учетную запись. Также, не выходя из приложения, можно работать с камерой устройства и получить доступ к адресной книге [17].



Рисунок 1.18 – Facebook for Every Phone

Twitter

Список платформ, поддерживаемых официальным клиентом сервиса микроблогов Twitter, содержит iOS, Android, BlackBerry и Windows Phone. Мобильный Twitter-клиент обеспечивает стандартные возможности – чтение и написание сообщений, копирование чужих постов (ReTweet), обмен прямыми сообщениями, поиск, управление списками фолловеров, обмен фотографиями, видеозаписями, ссылками, поиск, просмотр трендов и прочее. Кстати, использовать имеющуюся в этом приложении функцию добавления фотографий

непосредственно с мобильного телефона зачастую удобнее, чем вводить снимки через компьютер [17].

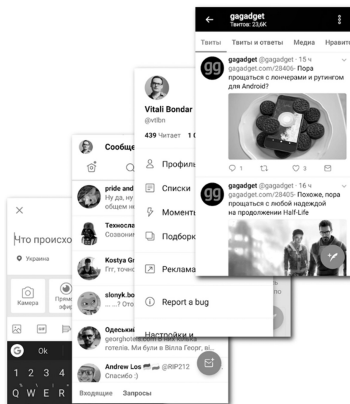


Рисунок 1.19 – Мобильный Twitter

Instagram

Пожалуй, самый простой способ рассказать о происходящем – с помощью изображения или видео. И Instagram подходит для этого. Пользователь может подписаться на обновления друзей и родственников, чтобы быть в курсе происходящего в их жизни, находить и подписываться на аккаунты незнакомцев со всех уголков мира – хоть по геометкам, хоть по тегам.



Рисунок 1.20 – Instagram

Фото перед публикацией можно обрабатывать с помощью фильтров или простого, но довольно мощного редактора. В течение дня о происходящем можно сообщать с помощью «историй» – фото- или видео сообщений, доступных подписчикам определённое время. Есть возможность отправки личных фото-, видео- и текстовых сообщений [17].

YouTube

В официальном приложении YouTube собраны самые популярные видео со всего мира – от новостей, до музыкальных клипов, от обучающих видеороликов типа «как почистить банан без рук», до серьёзного документального кино. YouTube давно используют влогеры (сокращение от «видео блогеры»). Всё, что нужно для влогерства уже встроено в это приложение. Можно снимать, загружать и редактировать видео прямо на смартфоне, публиковать его, транслировать на экран телевизора. А ещё с помощью приложения можно читать и оставлять комментарии [17].



Рисунок 1.21 – YouTube

Pinterest

Это сервис для хранения визуальных закладок, которые можно организовывать в коллекции. А также социальное медиа: следить за тем, какие коллекции и чем наполняют их люди, обновления, на которые подписаны пользователи, добавлять понравившееся себе.

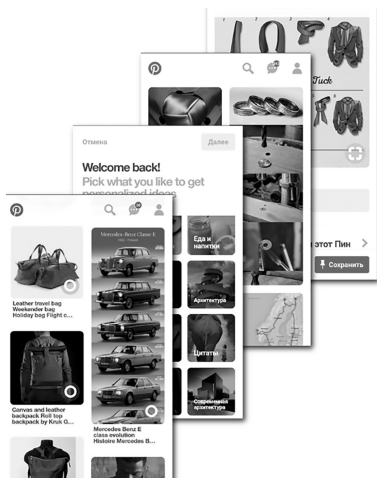


Рисунок 1.22 – Pinterest

Tumblr

Платформа для публикации блогов и одновременно социальная сеть. Пользователь может публиковать свои сообщения (текст, звук, видео, изображения), добавлять к ним теги, комментировать чужие публикации, делать «реблог» понравившегося, отправлять личные сообщения другим пользователям [17].

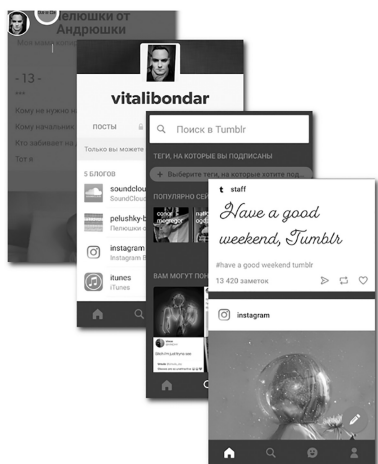


Рисунок 1.23 – Tumblr

LinkedIn

Сети профессиональных контактов, общения с коллегами, единомышленниками на профессиональные темы. Профиль LinkedIn – это резюме, которое наглядно показывает, каких карьерных и профессиональных высот достигли пользователи [17].



Рисунок 1.24 – LinkedIn

Flickr



Рисунок 1.25 – Flickr

Бесплатное хранилище в 1000 ГБ для снимков без потери их качества, удобные инструменты для загрузки и каталогизации снимков. Даже если пользователь не фотограф, но ценитель прекрасного, то он найдёт много красивых снимков и интересных людей, на которых стоит подписаться и следить за их фотожизнью и фототворчеством [17].

Snapshot

Это приложение для обмена сообщениями с прикрепленными фото и видео. К ним можно добавлять текст и рисунки, стикеры, маски и отправлять списку получателей. Для таких снимков или роликов, которые называются «snap», можно указать временной лимит для просмотра публикации («снэпа») [17].



Рисунок 1.26 – Snapchat

Line

Социальная сеть, выросшая из мессенджера, очень популярна в Китае. Главная особенность LINE – гигантское количество стикеров. Компания регулярно выпускает новые наклейки, привязывая их к различным событиям.

Пользователь может приобрести дополнительные темы оформления. Ориентированная на молодёжную (и неформальную) аудиторию, социальная сеть уже в прошлом году стоила около \$28 млрд.

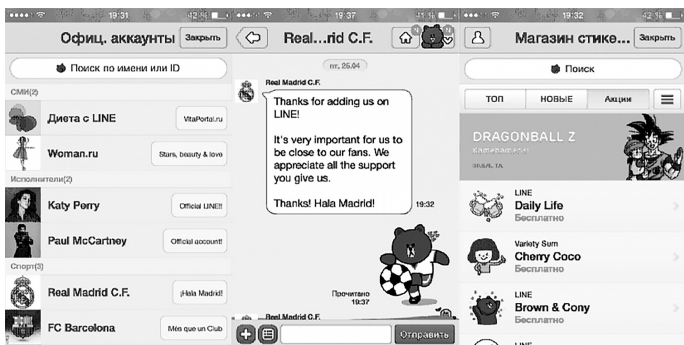


Рисунок 1.27 – Line

Path

Социальная сеть Path ориентирована исключительно на мобильные устройства. С компьютера можно увидеть отдельные посты, расшаренные из приложения, но не более того. У программы есть своя изюминка: в друзья можно добавить максимум 150 человек – именно столько постоянных социальных связей реально поддерживать одному человеку одновременно – так называемое число Данбара.



Рисунок 1.28 – Path

WeChat

Стандартный набор: анимированные стикеры, голосовые чаты, видео звонки, профили пользователей. С последним обновлением WeChat обзавёлся онлайн-магазином, через который можно приобре-

сти некоторые товары для повседневной жизни. Именно эту социальную сеть можно считать одной из самых перспективных на рынке.

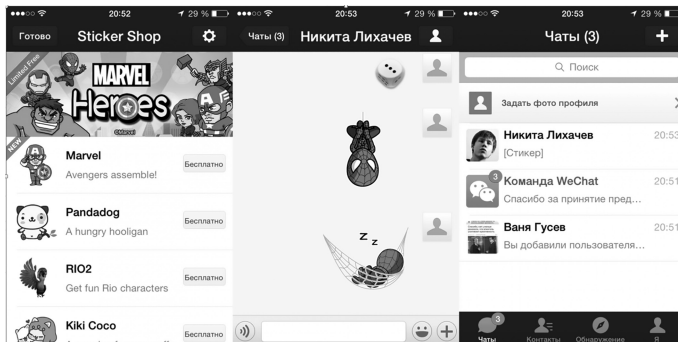


Рисунок 1.29 – WeChat

1.5 Мобильные облачные решения

Несмотря на мощность современных мобильных устройств, случается, что объёма памяти не хватает. Что могут предложить современные мобильные технологии для решения этих проблем? Достичь практически неограниченной вычислительной мощности можно, используя *технологии облачных вычислений*.

Мобильные облачные вычисления определены следующим образом: «Службы облачных вычислений, доступные в мобильной экосистеме. Это определение охватывает многие элементы, в том числе потребителя, предприятие, фемтосоты, транскодирование, сквозную безопасность, домашние шлюзы и мобильные услуги с широкополосным доступом» (*Фемтосоты* – это мелкие базовые станции сотовой сети) [19].



Рисунок 1.30 – Технология облачных вычислений

Чтобы лучше понять особенности мобильных облачных вычислений, давайте подробнее рассмотрим облачные вычисления в целом. Здесь важны два ключевых момента.

- 1) Облачные вычисления позволяют удобно, по мере необходимости получать сетевой доступ к общему пулу настраиваемых вычислительных ресурсов (таких как сети, серверы, системы хранения, приложения и службы), которые можно быстро занимать и освобождать с минимальными усилиями по управлению и без вмешательства поставщика услуг.
- 2) Существует три основных модели облачных служб: инфраструктура как услуги (IaaS), платформа как услуги (PaaS) и программное обеспечение как услуги (SaaS).

Что кроется за сложными аббревиатурами SaaS и PaaS, чем от них отличается IaaS?

iPhone или Android-смартфоны уже сделали нас активным пользователем облачных сервисов. И хотя публичные системы хранения, синхронизация настроек и фотографий, передача данных между сетями повсеместно распространилась, большинство людей до сих пор не знают или не уверены в определении того, что такое IaaS, PaaS, SaaS и другие обозначения.



Начнем с понимания, что в облачных технологиях применяется единая классификация моделей вычислений с суффиксом «...aaS» (*...as a Service*), что в переводе значит: «...как услуга». Это поможет быстрее ориентироваться в указанных аббревиатурах. Теперь рассмотрим модели облачных служб.

IaaS (Infrastructure as a Service)

«*Инфраструктура как услуга*» – пользователь избавляет себя от необходимости иметь дело с инфраструктурой. Вся ответственность за работу оборудования перекладывается на поставщика услуги. В боль-

шинстве случаев IaaS представляет собой модель продажи серверного времени с зависимостью от объёмов потребляемой вычислительной мощности и сетевого трафика.

PaaS (Platform as a Service)

«Платформа как услуга» – пользователь получает возможность развёртывания приложений без расходов на оборудование и программное обеспечение. Приложения могут быть созданы любой компанией или предлагаться в качестве сторонних веб-сервисов. Рекомендуются использовать ровно столько вычислительных ресурсов, сколько требуется, динамически изменяя нагрузки и параметры работы.

SaaS (Software as a Service)

«Программное обеспечение как услуга» – пользователь может использовать готовые программные средства и инструменты через интернет-клиент в своём рабочем смартфоне или на компьютере. Максимально удобная модель для удалённого управления приложением, которое оперативно модернизируется и обновляется, не требует контроль над инфраструктурой, операционными системами и разработками.

MaaS (Monitoring as a Service)

«Мониторинг как услуга» – позволяет отслеживать ключевые параметры инфраструктуры и программного обеспечения, которые пользователь задаёт через центральную панель управления в веб-интерфейсе онлайн. Пока ещё новое звено этой большой облачной головоломки, но уже считается неотъемлемой частью будущего IT-отрасли и обработки Big Data. Помогает снизить риски непредвиденных расходов и оптимизировать работу.

CaaS (Communication as a Service)

«Коммуникация как услуга» – решение IT-аутсорсинга для корпоративной связи, арендованной у одного провайдера. В неё входит сервис интернет-телефонии (передача голоса по IP, то есть VoIP), мессенджеры (IM-приложения), интерфейсы совместной работы или синхронизации изменений в рабочих документах, видеоконференции. Пользователь заказывает то, что требуется для связи и исключает затраты на инфраструктуру и программное обеспечение с динамическим расширением мощностей и покрытия по требованию.

XaaS («Anything» as a Service)

«Всё, что угодно как услуга» – модель предоставления информационных технологий в качестве готового сервиса с применением гибридных моделей облачных вычислений или акцентом на одной из них.

Рассмотрим компании, которые предоставляют услуги в области мобильных облачных технологий по различным моделям.

Amazon Web Services

В сфере инфраструктуры как услуги первоначально преобладала автоматизированная модель публичного облака Amazon Web Services, за которой последовали Rackspace. Amazon Web Services предлагает широкий спектр глобальных облачных продуктов, включая вычислительные сервисы, хранилища, базы данных, инструменты аналитики, сетевые технологии, мобильные сервисы и прочее. Эти сервисы помогают различным организациям развиваться быстрее, снижать расходы в сфере ИТ и обеспечивать масштабирование. Крупнейшие корпорации и стремительно растущие стартапы доверяют AWS поддержку работы широкого спектра рабочих нагрузок, включая мобильные и интернет-приложения, разработку игр, обработку и хранение данных, хранилища, архивацию и многое другое.

AnyPresence

Цель облачной мобильной платформы AnyPresence для предприятий – сократить расходы и упростить проектирование и развертывание приложений. С ее помощью можно собирать и развертывать программы HTML5, iOS и Android, не устанавливая программного обеспечения. Шаблоны мобильных приложений и возможности интеграции облегчают реализацию мобильных проектов.

Skyhigh Networks

Компания Skyhigh Networks позволяет предприятиям использовать облачные службы (SaaS, PaaS и IaaS), устраняя проблемы, возникающие из-за активного неконтролируемого использования облачных служб сотрудниками, особенно вооруженными личными мобильными устройствами. Облачная служба не создает дополнительной нагрузки на ИТ-инфраструктуру клиента. Установок и агентов для мобильных устройств не требуется. С помощью платформы ИТ-администраторы и специалисты по безопасности определяют облачные службы в сети и получают подробные отчеты о риске для каждой из них.

Microsoft

Облачная служба IaaS компании Microsoft предлагает облачные сервисы Azure Mobile Services, которые призваны облегчить разработчикам мобильных приложений создание и использование серверного бэкенда. Использование облака Windows Azure в качестве такого бэкенда позволит получить готовый функционал пуш-уведомлений, сохранения данных в облачное хранилище, аутентификации и авторизи-

зации пользователей без необходимости разворачивать собственную инфраструктуру. Кроме того, с развитием сервиса, добавится официальная инструментальная поддержка Windows Phone, iOS и Android.

Тенденции развития мобильных облачных технологий:

- 1) Компании движутся в направлении мобильного облака по требованию клиентов, которым нужны приложения для смартфонов и планшетов, позволяющие получать доступ к ключевым приложениям компаний. Сотрудникам нужен доступ с мобильных устройств. Под влиянием мобильного облака меняются модели поведения и привычки. Согласно опросу Pew Internet Project, к 2020 году большинство пользователей интернета будут работать, главным образом, в киберпространстве с приложениями на удаленных серверах, доступных через сетевые устройства.
- 2) Мобильная экосистема, меняющая модели поведения и привычки, состоит из множества устройств. Возможности по эксплуатации информации, создаваемой этими устройствами, кажутся безграничными, например, один из проектов предлагает организацию автостоянок с использованием интеллектуальных устройств, которые могут отправлять водителям сообщения о наличии свободных мест.
- 3) Еще одна тенденция указывает на «мобилизацию» денег посредством мобильного облака. Например, компания TabbedOut создала службу, которая позволяет открывать, просматривать и оплачивать чеки с помощью смартфонов. Starbucks начала предоставлять клиентам возможность оплачивать покупки с помощью электронного кошелька, который создает на экране штрих-код, сканируемый кассиром. Кошелек клиента Starbucks пополняется посредством кредитной карты или счета PayPal.

1.6 Классификация мобильных устройств

В последнее время в периодической печати все чаще стали появляться публикации многочисленных аналитиков, пытающихся обобщать деление карманных устройств на классы по тем или иным признакам. И действительно, рассматривая новинки, выставленные в магазинах, рядовой неискушенный покупатель порой просто теряется от разнообразия. По внешнему виду неспециалисту трудно отличить сотовый телефон от, скажем, карманного ПК, смартфона, видеотелефона, коммуникатора или камеро-фона. Между тем все эти термины

в той или иной степени уже прижились в русском языке, как в свое время прижились неологизмы вроде «телефон», «телевизор» и «Интернет». Рано или поздно каждому из нас придется привыкнуть к этим словам, чтобы не остаться за бортом прогресса. Тем более что все эти словесные новообразования придуманы отнюдь не для специалистов, а как раз для удобства потребителей.

На самом деле все достаточно просто, а в стремительном появлении новых слов вряд ли виноваты изобретатели всех этих новинок – просто мир вокруг нас меняется гораздо быстрее, чем порой хотелось бы.

Итак, попробуем разобраться в современной терминологии карманных устройств связи, даже если нам не удастся удовлетворить всех знатоков. Что поделаешь, даже некоторые современные плееры работают под управлением полноценных операционных систем и имеют функции, дающие большое преимущество любому карманному компьютеру прежних лет. Но сегодня мы не будем погружаться в разнообразие всех карманных электронных устройств. Ограничимся для начала устройствами, похожими на телефон.

Сегодня нет общепризнанного стандарта применения того или иного термина, с помощью которого мы без колебаний могли бы отнести, то или иное устройство. Тем не менее, некоторые устоявшиеся в специальной литературе критерии все же существуют, и сейчас мы попробуем определиться с основными терминами, обозначающими тот или иной класс устройств. Хотя бы приблизительно.

Какие устройства на самом деле являются мобильными? И почему их так называют, и в чем заключается эта всем известная мобильность?

Мобильные устройства – это такие устройства как смартфоны, интернет планшеты, электронные книги, телефоны, КПК, нетбуки. Главная их особенность это, конечно же, размер. В эпоху тотальной миниатюризации мобильные устройства вытесняют так называемые «стационарные»: на смену компьютерам приходят планшеты и смартфоны, умные часы, очки виртуальной реальности и прочая «мобильная», переносная, легкая техника.

Смартфоны

Смартфон (от английского Smartphone – умный телефон) – это мобильный телефон, который дополнен возможностями персонального компьютера. И это действительно так – сегодня смартфоны заменили многим пользователям компьютеры, поскольку предоставляют практически все те же функции. В будущем прогнозируется еще большая

популярность смартфонов как замена персональным компьютерам и ноутбукам. Это в том числе подтверждается тем фактом, что пользователи все чаще используют смартфоны для выхода в сеть интернет [7].

Основное отличие смартфона от привычного мобильного телефона – это наличие операционной системы. Однако если учесть тот факт, что некоторые телефоны снабжены операционной системой, отличий должно быть куда больше. Например, это крупный сенсорный экран.

Самым первым смартфоном, в привычном для нас виде, с сенсорным экраном, где управление осуществлялось только руками, стал первый iPhone – это произошло в 2007 году. Тогда смартфон стал самой настоящей сенсацией, ведь до этого никто ничего подобного на рынке не предлагал. Неудивительно, что в считанные месяцы iPhone стал лидером продаж.



Рисунок 1.31 – iPhone

При этом стоит отметить, что функционал самого первого iPhone был ограничен. Например, у него отсутствовала возможность передачи MMS-сообщений, что, впрочем, никак не повлияло на продажи устройства.

Что нам дает использование смартфона? Это и:

- 1) Серфинг интернета и социальных сетей.
- 2) Мультимедийный центр. На экране смартфона можно смотреть фильмы, слушать музыку, смотреть онлайн-трансляции.
- 3) Игры. В официальных онлайн-магазинах сегодня можно встретить огромное количество всевозможных игр. При этом многие игры бесплатны.
- 4) Камеры. Фотокамеры позволяют не только получать отличные фотографии, включая «селфи», но и проводить онлайн-трансляции или общаться с друзьями по видео-чату.

- 5) Беспроводные модули значительно расширяют возможности. Например, можно использовать мобильную связь для выхода в интернет, причем сейчас используются такие технологии, когда скорость мобильного интернета может быть выше, чем на домашнем компьютере.
- 6) Приложения. Приложений, как и игр, в онлайн-магазинах огромное количество на любой вкус и кошелек.

Если говорить в целом, то «умный телефон» полностью оправдывает свое название – это впрямь телефон и компьютер в одном устройстве.

Планшеты

Планшетный компьютер – это современное мультимедийное устройство, возможности которого практически безграничны. На сегодняшний момент девайсы планшетного типа используются практически во всех сферах жизнедеятельности человека, начиная от бытового использования, и заканчивая на высокотехнологичных предприятиях и, конечно же, в сфере бизнеса.

Если несколько лет назад планшеты не были настолько популярны, и мало кто о них знал, но невероятный прорыв компании Apple со своим первым iPad подтолкнул многих производителей компьютерной техники к быстрому выпуску аналогичного продукта.

Многие годы соревнования между компаниями, большими и маленькими, мы получили огромный рынок планшетных компьютеров, где можно найти совершенно любые гаджеты, как дорогие, так и недорогие, а главное, они смогли удовлетворить желания миллионов пользователей по всему миру, и этому нет предела [7].

Прототип будущего планшетного гаджета, даже сказать идея, развивалась вместе с появлением первого компьютера. Уже тогда разработчики и инженеры пытались создать что-то подобное записной книжки, только электронной, которая могла бы хранить намного больше информации, чем обычный блокнот.

По мере развития первого компьютера, развивалось и представление о самом планшете, сначала хотели сделать его похожим на небольшой персональный компьютер, то есть монитор, клавиатура, мышка, но только все в уменьшенном варианте. Из этого и появились современные нетбуки и ноутбуки, но некоторые компании, например, DEC, пошли дальше, и наконец-то выпустили, так сказать эталон будущих электронных книг. Полученный девайс прекрасно подходил для чтения документов, поддерживал перьевой ввод.

После выхода iPhone, и вслед за ним iPad, мир и человечество вошло в новую эпоху компьютерных технологий, темпы развития которой только повышались с каждым годом.

Современное планшетное устройство – это многофункциональный компьютер, который поддерживает, помимо сенсорного, перьевого и голосового ввода, прекрасно работает практически со всеми внешними периферийными устройствами, от стандартных flash-карт, до клавиатур и жестких дисков.

В последнее время некоторые сенсорные девайсы достигли такой производительности, что могут посоревноваться с бюджетными ноутбуками, а в некоторых позициях и выиграть. После появления iPad, который работал под управлением закрытой и фирменной операционной системой iOS, многие разработчики начали разрабатывать свои ОС, которые сейчас используются более чем на 80% всех выпускаемых сенсорных девайсах.



Рисунок 1.32 – Планшет

Носимые устройства

Носимые устройства – это своего рода миниатюрные компьютеры: браслеты, очки, часы и даже предметы одежды – с беспроводным локальным или удаленным подключением к другим компьютерам. Как правило, такие устройства оснащены датчиками, отслеживающими различные формы физической активности или параметры среды, в которой находится пользователь. Большинство современных носимых устройств используют смартфон с подключением к интернету. Но в будущем, по мере увеличения времени работы аккумуляторов, мощности процессоров, уменьшения размера компонентов и решения прочих технических задач, носимые устройства станут действительно

независимыми и будут использовать собственные процессоры и подключение.

Большинство потребителей считает, что носимые устройства – это что-то вроде трекеров физической активности (например, Fitbit) или аксессуаров для смартфонов (Pebble и другое). Однако многие разработчики уже исследуют возможности более активного использования носимых устройств.

Компания Vanson Bourne провела исследование, в котором приняли участие 300 британских ИТ-руководителей. Результат показал, что у 29% компаний Великобритании есть проекты, так или иначе связанные с использованием носимых устройств. Основные причины для таких проектов: благополучие сотрудников (16%), быстрый доступ к важной информации (15%) и повышение качества обслуживания клиентов (14%).

Современные носимые устройства способны отслеживать сердечный ритм, температуру тела, кровяное давление и тому подобное. Страховые компании планируют предлагать полисы по выгодной цене тем клиентам, которые согласятся носить устройства, отслеживающие состояние их здоровья и уровень активности.

Сотрудники передают свои биометрические данные работодателю, который с их помощью мотивирует работников компании придерживаться здорового образа жизни и предоставляет страховой компании информацию, позволяющую снизить стоимость корпоративных страховок.



а б

Рисунок 1.33 – Носимые устройства: а – Умные часы; б – Фитнес-браслет

В принципе мы и разобрались, в чем же заключается суть мобильных устройств. Главное – это правильно выбрать свое мобильное устройство, и тогда оно по-настоящему послужит пользователям во всех делах.

1.7 Коммуникационные технологии

1.7.1 Технология Wi-Fi. Стандарты

Наиболее быстро развивающимся сегментом телекоммуникаций сегодня является Беспроводная Локальная Сеть (WiFi). В последние годы виден все больший спрос на мобильные устройства, построенные на основе беспроводных технологий.

Стоит отметить, что WiFi-продукты передают и получают информацию с помощью радиоволн. Несколько одновременных вещаний могут происходить без обоюдного вмешательства благодаря тому, что радиоволны передаются по разным радиочастотам, известным также как каналы.

Для осуществления передачи информации WiFi-устройства должны «наложить» данные на радиоволну, также известную как несущая волна. Этот процесс называется модуляцией. Существуют различные типы модуляции, которые рассмотрим далее. Каждый тип модуляции имеет свои преимущества и недостатки с точки зрения эффективности и требований к питанию. Вместе, рабочий диапазон и тип модуляции, определяют физический уровень данных (PHY) для стандартов передачи данных. Продукты совместимы по PHY в том случае, когда они используют один диапазон и один тип модуляции [8].

Первый стандарт беспроводных сетей 802.11 был одобрен Институтом инженеров по электротехнике и радиоэлектронике (IEEE) в 1997 году и поддерживал скорость передачи данных до 2-х Мбит/с. Используемые технологические схемы модуляции стандарта: псевдослучайная перестройка рабочей частоты (FHSS – Frequency Hopping Spread Spectrum) и широкополосная модуляция с прямым расширением спектра (DSSS – Direct Sequence Spread Spectrum).

В 1999 году, IEEE одобрила еще два стандарта беспроводных сетей WiFi: 802.11a и 802.11b.

Стандарт 802.11a работает в частотном диапазоне 5 ГГц со скоростью передачи данных до 54 Мбит/с. Данный стандарт построен на основе технологии цифровой модуляции ортогонального мультиплексирования с разделением частот (OFDM – Orthogonal Frequency Division Multiplexing).

Стандарт 802.11b использует диапазон частот 2.4 ГГц и достигает скоростей передачи данных до 11 Мбит/с. В отличие от стандарта 802.11a, схема стандарта 802.11b построена по принципу DSSS.

Поскольку реализовать схему DSSS легче, чем OFDM, то и продукты, использующие стандарт 802.11b, начали появляться на рынке раньше (с 1999 года). С тех пор продукты, работающие по беспроводному протоколу радиодоступа и использующие стандарт 802.11b, широко использовались в корпорациях, офисах, дома, в загородных коттеджах, в общественных местах (хот-споты) и так далее. На всех продуктах, прошедших сертификацию альянса WECA совместимости беспроводного оборудования Ethernet имеется соответствующая отметка с официально зарегистрированным логотипом WiFi. Альянс WECA (включает в себя всех основных производителей беспроводных устройств на основе технологии WiFi. Альянс занимается тем, что сертифицирует, маркирует, а также тестирует на совместимость оборудование, применяющее технологии WiFi.

В начале 2001 года Федеральная Комиссия по Коммуникациям Соединенных Штатов (FCC – Federal Communications Commission) ратифицировала новые правила, благодаря которым разрешается дополнительная модуляция в диапазоне 2.4 ГГц. Это позволило IEEE расширить стандарт 802.11b, что привело к поддержке более высоких скоростей для передачи данных. Таким образом, появился стандарт 802.11g, который работает со скоростью передачи данных до 54 Мбит/с и разрабатывался с использованием технологии OFDM.

Частоты Wi-Fi

Обеспечить беспроводную связь с Интернет теперь доступно всем. Достаточно подключить у себя в доме, на даче или в офисе систему Wi-Fi и можно принимать сигнал, не заботясь о бесконечных проводах, телефонных подключениях, модемах и картах связи.

Роутер Wi-Fi является маршрутизатором, принимающим решение по пересылке пакетных данных для различных модульных сегментов сети. Проще говоря, если у пользователя в доме находятся один или несколько ноутбуков и все они нуждаются в подключении к сети Интернет, то эту проблему решает маршрутизатор беспроводной связи.

Система Wi-Fi самостоятельно находит ноутбуки и устанавливает соединение с Интернет. Стандартная схема беспроводного маршрутизатора предусматривает не менее одного соединения. Раздача интернета происходит на различных частотах.

В Республике Казахстан разработаны документы, которые регламентируют процесс распределения частот: «Перечень разрешенных к эксплуатации и ввозу из-за границы ВЧ средств в РК» и «Постанов-

ление Правительства Республики Казахстан от 11 сентября 2000 года № 1379 «Об утверждении Таблицы распределения полос частот между радиослужбами Республики Казахстан».

Определенные частоты являются основными, для работы в определенных диапазонах и не требуют специального разрешения. Для поиска свободного канала связи необходимо скоординировать подключение сети с администрациями других сетей. Каждая сеть должна использовать канал-частоту, отделенную от другого канала полосой 25 МГц. В таблице 1.2 приведены параметры стандартов.

Таблица 1.2

Стандарт	802.11	802.11a	802.11b	802.11g
Дата сертификации стандарта	1997	1999	1999	2003
Доступная полоса пропускания	83.5 МГц	300 МГц	83.5 МГц	83.5 МГц
Частота операций	2.4-2.4835 ГГц	5.15-5.35 ГГц	2.4-2.4835 ГГц	2.4-2.4835 ГГц
Типы модуляции	DSSS, FHSS	OFDM	DSSS	DSSS, OFDM
Скорость передачи данных по каналу	2, 1 Мбит/с	54, 48, 36, 24, 18, 12, 9, 6 Мбит/с	11, 5.5, 2, 1 Мбит/с	54, 36, 33, 24, 22, 12, 11, 9, 6, 5.5, 2, 1 Мбит/с
Совместимость	802.11	Wi-fi5	Wi-Fi	Wi-Fi со скоростью 11 Мбит/с и ниже

Благодаря использованию частоты 5 ГГц и модуляции OFDM у *стандарт 802.11a* есть два ключевых преимущества перед стандартом 802.11b. Во-первых, это значительно увеличенная скорость передачи данных по каналам связи. Во-вторых, увеличилось число не накладывающихся каналов.

Диапазон 5 ГГц (также известный как UNII) фактически состоит из трех субдиапазонов: UNII1 (5.15-5.25 ГГц), UNII2 (5.25-5.35 ГГц) и UNII3 (5.725-5.825 ГГц). При использовании одновременно двух субдиапазонов UNII1 и UNII2 можно получить до восьми непересекающихся каналов против всего лишь трех в диапазоне 2.4 ГГц. Также у

этого стандарта гораздо больше доступная полоса пропускания. Таким образом, с использованием стандарта 802.11a можно поддерживать большее число одновременных, более продуктивных, неконфликтных беспроводных соединений.

Необходимо отметить, что, так как стандарты 802.11a и 802.11b работают в различных диапазонах, то и продукты, разработанные под эти стандарты, не совместимы. Например, точка доступа WiFi, работающая в диапазоне 2.4 ГГц, стандарта 802.11b, не будет работать с беспроводной сетевой картой, рабочий диапазон которой 5 ГГц. Однако оба стандарта могут и сосуществовать. К примеру, пользователи, подключенные к точкам доступа, применяющим разные стандарты, также могут использовать любые внутренние ресурсы этой сети, но при условии, что эти точки доступа подключены к одной опорной сети.

802.11g – высокая скорость в диапазоне 2.4 ГГц. Стандарт 802.11g несет с собой более высокие скорости передачи данных, при этом поддерживая совместимость с продуктами стандарта 802.11b. Стандарт работает с применением модуляции DSSS на скоростях до 11Мбит/с, но при этом дополнительно используется модуляция OFDM на скоростях выше 11Мбит/с. Таким образом, оборудование стандартов 802.11b и 802.11g совместимо на скоростях, не превышающих 11Мбит/с. Если в диапазоне 2.4 ГГц необходима скорость выше, нежели 11Мбит/с, то нужно использовать оборудование стандарта 802.11g. Можно сказать, что стандарт 802.11g соединил в себе все лучшее от стандартов 802.11b и 802.11a.

Стандарт 802.11n. Стандарт для сетей Wi-Fi, появился в 2009 году. Получила название Wi-Fi 4. Работает в диапазонах 2,4 и 5 ГГц (устройства, поддерживающие диапазон 5 ГГц встречаются гораздо реже), позволяет достигать скоростей до 150 Мбит/с при ширине канала 40 МГц на каждую независимую антенну. Стандарт 802.11n использует совершенно новые технологии, повышающие скорость передачи данных и увеличивающие радиус покрытия. Модуляция, используемая стандартом, именуется MIMO (Multiple Input Multiple Output). Данная модуляция построена на основе применения множества антенн, соответственно, создается множество информационных потоков, что в разы увеличивает скорость передачи данных.

Стандарт Wi-Fi 802.11ac. Стандарт беспроводных локальных сетей Wi-Fi, работающий в диапазоне частот 5 ГГц. Обрато совместим с IEEE 802.11n (в диапазоне 5 ГГц) и IEEE802.11a. Получил название Wi-Fi 5. В 2019-2020 годах он будет замещён стандартом IEEE 802.11ax.

Устройства с 802.11ac обычно также реализуют стандарт 802.11n в диапазоне 2.4 ГГц.

Стандарт позволяет существенно расширить пропускную способность сети, начиная от 433Мбит/с и до 6,77 Гбит/с при 8xMU-MIMO-антеннах. Это наиболее существенное нововведение относительно IEEE 802.11n. Кроме того, ожидается снижение энергопотребления (Дж/бит), благодаря чему, в свою очередь, увеличится время автономной работы мобильных устройств.

Сфера применения беспроводной технологии

В большинстве случаев беспроводные сети (используя точки доступа и маршрутизаторы) строятся в коммерческих целях для привлечения прибыли со стороны клиентов и арендаторов. Сетевую инфраструктуру, на основе беспроводных решений, могут использовать:

- 1) офисные центры;
- 2) торговые центры;
- 3) гостиничные комплексы;
- 4) складские помещения;
- 5) места проведения коммерческих и общественных мероприятий и прочее.

1.7.2 Стандарты GSM - поколение сетей сотовой связи

Поколение сотовой связи – это набор функциональных возможностей работы сети, а именно: регистрация абонента, установление вызова, передача информации между мобильным телефоном и базовой станцией по радиоканалу, процедура установления вызова между абонентами, шифрование, роуминг в других сетях, а также набор услуг, предоставляемых абоненту [8].

Все мобильные сети делятся на разные поколения. **G** – это английская заглавная буква, обозначающая слово Generation («поколение»). Эволюция систем сотовой связи включает в себя несколько поколений 1G,2G,3G,4G. Ведутся работы в области создания сетей мобильной связи нового пятого поколения (5G). Стандарты различных поколений, в свою очередь, подразделяются на аналоговые (1G) и цифровые системы связи (остальные). Рассмотрим их подробнее.

Связь всегда имела большое значение для человечества. Когда встречаются два человека, для общения им достаточно голоса, но при увеличении расстояния между ними возникает потребность в специальных инструментах. Когда в 1876 году Александр Грэхем Белл изо-

брел телефон, был сделан значительный шаг, позволивший общаться двум людям, однако для этого им необходимо было находиться рядом со стационарно установленным телефонным аппаратом. Более ста лет проводные линии были единственной возможностью организации телефонной связи для большинства людей. Системы радиосвязи, не зависящие от проводов для организации доступа к сети, были разработаны для специальных целей (например, армия, полиция, морской флот и замкнутые сети автомобильной радиосвязи), и, в конце концов, появились системы, позволившие людям общаться по телефону, используя радиосвязь. Эти системы предназначались главным образом для людей, ездивших на машинах, и стали известны как телефонные системы подвижной связи [8].

Первое поколение мобильной связи (1G)

Официальным днем рождения сотовой связи считается 3 апреля 1973 года, когда глава подразделения мобильной связи компании Motorola Мартин Купер позвонил начальнику исследовательского отдела AT&T Bell Labs Джоэлю Энгелю, находясь на оживленной Нью-Йоркской улице. Именно эти две компании стояли у истоков мобильной телефонии. Коммерческую реализацию данная технология получила 11 лет спустя, в 1984 году, в виде мобильных сетей первого поколения (1G), которые были основаны на аналоговом способе передачи информации.

Основными стандартами аналоговой мобильной связи стали AMPS (Advanced Mobile Phone Service – усовершенствованная подвижная телефонная служба) (США, Канада, Центральная и Южная Америка, Австралия), TACS (Total Access Communications System – тотальная система доступа к связи) (Англия, Италия, Испания, Австрия, Ирландия, Япония) и NMT (Nordic Mobile Telephone – северный мобильный телефон) (страны Скандинавии и ряд других стран). Были и другие стандарты аналоговой мобильной связи – C-450 в Германии и Португалии, RTMS (Radio Telephone Mobile System – радиотелефонная мобильная система) в Италии, Radiocom 2000 во Франции. В целом мобильная связь первого поколения представляла собой лоскутное одеяло несовместимых между собой стандартов.

Во времена 1G никто не думал об услугах передачи данных – это были аналоговые системы, задуманные и разработанные исключительно для осуществления голосовых вызовов и некоторых других скромных возможностей. Модемы существовали, однако из-за того, что беспроводная связь более подвержена шумам и искажениям, чем

обычная проводная, скорость передачи данных была невероятно низкой. К тому же, стоимость минуты разговора в 80-х была такой высокой, что мобильный телефон мог считаться роскошью.

Во всех аналоговых стандартах применяется частотная (ЧМ) или фазовая (ФМ) модуляция для передачи речи и частотная манипуляция для передачи информации управления. Этот способ имеет ряд существенных недостатков: возможность прослушивания разговоров другими абонентами, отсутствие эффективных методов борьбы с замираниями сигналов под влиянием окружающего ландшафта и зданий или вследствие передвижения абонентов. Для передачи информации различных каналов используются различные участки спектра частот – применяется метод множественного доступа с частотным разделением каналов (Frequency Division Multiple Access - FDMA). С этим непосредственно связан основной недостаток аналоговых систем – относительно низкая емкость, являющаяся следствием недостаточно рационального использования выделенной полосы частот при частотном разделении каналов.

В каждой стране была разработана собственная система, несовместимая с остальными с точки зрения оборудования и функционирования. Это привело к тому, что возникла необходимость в создании общей европейской системы подвижной связи с высокой пропускной способностью и зоной покрытия всей европейской территории. Последнее означало, что одни и те же мобильные телефоны могли использоваться во всех Европейских странах, и что входящие вызовы должны были автоматически направляться в мобильный телефон независимо от местонахождения пользователя (автоматический роуминг). Кроме того, ожидалось, что единый Европейский рынок с общими стандартами приведет к удешевлению пользовательского оборудования и сетевых элементов независимо от производителя[8].

Второе поколение мобильной связи (2G)

В 1982 году СЕПТ (франц. Conférence européenne des administrations des postes et télécommunications – Европейская конференция почтовых и телекоммуникационных ведомств) сформировала рабочую группу, названную специальной группой по подвижной связи GSM (франц. Groupe Spécial Mobile) для изучения и разработки пан-Европейской наземной системы подвижной связи общего применения – второе поколение систем сотовой телефонии (2G).

Название рабочей группы GSM также стало использоваться в качестве названия системы подвижной связи. В 1989 году обязанности

CEPT были переданы в Европейский институт стандартов в телекоммуникации ETSI (англ. European Telecommunications Standards Institute). Первоначально GSM предназначалась только для стран-членов ETSI. Однако многие другие страны также имеют реализованную систему GSM, например, Восточная Европа, Средний Восток, Азия, Африка, Тихоокеанский регион и Северная Америка (с производной от GSM, названной PCS1900). Название GSM стало означать «глобальная система для подвижной связи», что соответствует ее сущности.

Первые мобильные сети второго поколения (2G) появились в 1991 году. Их основным отличием от сетей первого поколения стал цифровой способ передачи информации, благодаря чему появилась, любимая многими, услуга обмена короткими текстовыми сообщениями SMS (англ. Short Messaging Service). При строительстве сетей второго поколения Европа пошла путем создания единого стандарта – GSM, в США большинство 2G-сетей было построено на базе стандарта D-AMPS (Digital AMPS – цифровой AMPS), являющегося модификацией аналогового AMPS. Кстати, именно это обстоятельство стало причиной появления американской версии стандарта GSM – GSM1900. С развитием и распространением Интернет, для мобильных устройств сетей 2G, был разработан WAP (англ. Wireless Application Protocol -беспроводной протокол передачи данных) – протокол беспроводного доступа к ресурсам глобальной сети Интернет непосредственно с мобильных телефонов.

Основными преимуществами сетей 2G по сравнению с предшественниками было то, что телефонные разговоры были зашифрованы с помощью цифрового шифрования; система 2G представила услуги передачи данных, начиная с текстовых сообщений SMS.

Растущая потребность пользователей мобильной связи в использовании Интернет с мобильных устройств стала основным толчком для появления сетей, поколения 2,5G, которые стали переходными между 2G и 3G. Сети 2,5G используют те же стандарты мобильной связи, что и сети 2G, но к имеющимся возможностям добавилась поддержка технологий пакетной передачи данных – GPRS (англ. *General Packet Radio Service* – пакетная радиосвязь общего пользования), EDGE (англ. *Enhanced Data rates for GSM Evolution* – повышенная скорость передачи для развития GSM) в сетях GSM. Использование пакетной передачи данных позволило увеличить скорость обмена информацией при работе с сетью Интернет с мобильного устройств до 384 кбит/с, вместо 9,6 кбит/с у 2G-сетей.

Система HSCSD (англ. High Speed Circuit Switched Data – высокоскоростная передача данных) является простейшей модернизацией системы GSM, предназначенной для передачи данных. Суть этой технологии заключалась в выделении одному абоненту не одного, а нескольких (теоретически до восьми) временных интервалов. Таким образом, максимальная скорость увеличивалась до 115,2 кбит/с. HSCSD обеспечивала скорость, достаточную для выхода в Интернет, однако, при передаче данных информационные пакеты разделены неопределенными по времени промежутками, таким образом, использование этой технологии крайне расточительно. Дело в том, что сети HSCSD, как и классические сети GSM, основаны на технологии коммутации каналов, в которых за абонентом закрепляют дуплексный канал на все время сеанса связи. Из-за пауз в передаче каналный ресурс расходовался нерационально.

Дальнейшей эволюцией системы GSM стала технология GPRS. Ее внедрение способствовало более эффективному использованию каналного ресурса и созданию комфортной среды при работе с сетью Интернет. Система GPRS разработана как система пакетной передачи данных с теоретической максимальной скоростью передачи порядка 170 кбит/с. GPRS сосуществует с сетью GSM, повторно используя базовую структуру сети доступа. Система GPRS является расширением сетей GSM с предоставлением услуг передачи данных на существующей инфраструктуре, в то время как базовая сеть расширяется за счет наложения новых компонентов и интерфейсов, предназначенных для пакетной передачи.

Прогресс не стоял на месте и, для увеличения скорости передачи данных, была изобретена новая система – EDGE. Она предусматривала введение новой схемы модуляции. В результате стала достижима скорость в 384 кбит/с. EDGE была введена в сетях GSM с 2003 фирмой Singular (ныне AT&T) в США.

Технологии GPRS и EDGE в разных источниках называли по-разному. Они уже переросли второе поколение, но еще не дотягивали до третьего. Зачастую GPRS называли 2,5G, EDGE – 2,75G.

Основные цифровые стандарты систем сотовой связи второго поколения:

- D-AMPS (Digital AMPS – цифровой AMPS; диапазоны 800 МГц и 1900 МГц);
- GSM (Global System for Mobile communications – глобальная система мобильной связи, диапазоны 900, 1800 и 1900 МГц);

- CDMA (диапазоны 800 и 1900 МГц);
- JDC (Japanese Digital Cellular – японский стандарт цифровой сотовой связи) [8].

Третье поколение мобильной связи (3G)

Дальнейшим развитием сетей мобильной связи стал переход к третьему поколению (3G). 3G – это стандарт мобильной цифровой связи, который под аббревиатурой IMT-2000 (англ. International Mobile Telecommunications – международная мобильная связь 2000) объединяет пять стандартов – W-CDMA, CDMA2000, TD-CDMA/TD-SCDMA, DECT (англ. Digital Enhanced Cordless Telecommunication – технология улучшенной цифровой беспроводной связи). Из перечисленных составных частей 3G только первые три представляют собой полноценные стандарты сотовой связи третьего поколения. DECT – это стандарт беспроводной телефонии домашнего или офисного назначения, который в рамках мобильных технологий третьего поколения, может использоваться только для организации точек горячего подключения (хот-спотов) к данным сетям.

Стандарт IMT-2000 дает четкое определение сетей 3G – под мобильной сетью третьего поколения понимается интегрированная мобильная сеть, которая обеспечивает: для неподвижных абонентов скорость обмена информацией не менее 2048 кбит/с, для абонентов, движущихся со скоростью не более 3 км/ч – 384 кбит/с, для абонентов, перемещающихся со скоростью не более 120 км/ч – 144 кбит/с.

При глобальном спутниковом покрытии сети 3G должны обеспечивать скорость обмена не менее 64 кбит/с. Основой всех стандартов третьего поколения являются протоколы множественного доступа с кодовым разделением каналов. Подобная технология сетевого доступа не является чем-то принципиально новым. Первая работа, посвященная этой теме, была опубликована в СССР еще в 1935 году Д.В. Агеевым.

Технически сети с кодовым разделением каналов работают следующим образом – каждому пользователю присваивается определенный числовой код, который распространяется по всей полосе частот, выделенных для работы сети. При этом какое-либо временное разделение сигналов отсутствует, и абоненты используют всю ширину канала. При этом сигналы абонентов накладываются друг на друга, но благодаря числовому коду могут быть легко дифференцированы. Как было упомянуто выше, данная технология известна достаточно давно, однако до середины 80-х годов прошлого века она была засекреченной

и использовалась исключительно военными и спецслужбами. После снятия грифов секретности началось ее активное использование и в гражданских системах связи [8].

Поколение 3,5G

Дальнейшим развитием сетей стала технология HSPA (англ. High Speed Packet Access – высокоскоростной пакетный доступ), которую стали именовать 3,5G. Изначально она позволяла достичь скорости в 14,4 Мбит/с, однако сейчас теоретически достижима скорость 84 Мбит/с и более. Впервые HSPA была описана в пятой версии стандартов 3GPP. В ее основе лежит теория, согласно которой при сопоставимых размерах сот применение многокодовой передачи позволяет достигать пиковых скоростей.

Четвертое поколение мобильной связи (4G)

В марте 2008 года сектор радиосвязи Международного союза электросвязи (МСЭ-Р) определил ряд требований для стандарта международной подвижной беспроводной широкополосной связи 4G, получившего название спецификаций International Mobile Telecommunications Advanced (IMT-Advanced), в частности, установив требования к скорости передачи данных для обслуживания абонентов: скорость 100 Мбит/с должна предоставляться высокоподвижным абонентам (например, поездам и автомобилям), а абонентам с небольшой подвижностью (например, пешеходам и фиксированным абонентам) должна предоставляться скорость 1 Гбит/с.

Так как первые версии мобильного WiMAX (англ. Worldwide Interoperability for Microwave Access – всемирная совместимость для микроволнового доступа) и LTE (англ. Long Term Evolution – долгосрочное развитие) поддерживают скорости значительно меньше 1 Гбит/с, их нельзя назвать технологиями, соответствующими IMT-Advanced, хотя они часто упоминаются поставщиками услуг, как технологии 4G. 6 декабря 2010 года МСЭ-Р признал, что наиболее продвинутые технологии рассматривают как 4G.

Основной, базовой, технологией четвертого поколения является технология ортогонального частотного уплотнения OFDM (англ. Orthogonal Frequency-Division Multiplexing – мультиплексирование с ортогональным частотным разделением каналов). Кроме того, для максимальной скорости передачи используется технология передачи данных с помощью N антенн и их приёма M антеннами – MIMO (англ. Multiple Input/Multiple Output – множество входов/множество выходов). При данной технологии, передающие и приёмные антенны

разнесены так, чтобы достичь слабой корреляции между соседними антеннами [8].

Что такое 5G?

Выше мы рассмотрели четыре поколения мобильной связи. В настоящее время операторы при поддержке поставщиков оборудования (вендоров) активно тестируют возможности сетей пятого поколения, коммерческий расцвет которого ожидается к 2020 году. Объяснить это достаточно просто: существует, так называемое, правило десяти лет. Если заглянуть немного в прошлое, можно заметить, что каждое новое поколение мобильной связи появлялось примерно через 10 лет после появления предыдущего: первое поколение появилось в начале 80- годов, второе – в начале 90-х, третье – в начале 00-х, четвертое – в 2009 году. Напрашивается вывод, что коммерческие сети 5G начнут заполнять мир именно в 2020 году. Стандарт мобильной связи пятого поколения (5G) – это новый этап развития технологий, который призван расширить возможности доступа в Интернет через сети радиодоступа.

Стандартизацию сетей мобильной связи 2, 3, 4 и 5 поколений выполняет партнерский проект для стандартизации систем 3-го поколения (3rd Generation Partnership Project, 3GPP). В 2017 году организация 3GPP официально сообщила, что 5G станет официальным названием следующего поколения мобильной связи и представила новый официальный логотип стандарта связи.



Рисунок 1.34 – Логотип стандарта 5G

Услуги в сетях 5G

- сверхширокополосная мобильная связь (Extreme Mobile Broadband, eMBB) – реализация ультраширокополосной связи с целью передачи «тяжелого» контента;
- массовая межмашинная связь (Massive Machine-Type Communications, mMTC) – поддержка Интернета вещей (ультраузкополосная связь);

- сверхнадежная межмашинная связь с низкими задержками (Ultra-Reliable Low Latency communication, URLLC) – обеспечение особого класса услуг с очень низкими задержками.

Подробнее услуги в сетях 5G рассмотрены по ссылке.

Очевидно, что в будущем к сети будет подключено гораздо больше устройств, большинство из которых будут работать по принципу «всегда онлайн». При этом очень важным параметром будет являться их низкое энергопотребление.

Требования к сетям 5G

- 1) Пропускная способность сети до 20 Гбит/сек по линии «вниз» (к абоненту); и до 10 Гбит/с в обратном направлении.
- 2) Поддержка одновременного подключения до 1 млн. устройств/км².
- 3) Сокращение временной задержки на радио-интерфейсе до 0,5 мс (для сервисов Сверхнадежной межмашинной связи URLLC) и до 4 мс (для сервисов Сверхширокополосной мобильной связи eMBB).

Потенциальные технологии в стандарте 5G

- 1) **Массивные MIMO.** Технология MIMO означает использование нескольких антенн на приемопередатчиках. Технология, успешно применяемая в сетях четвертого поколения, найдет применение и в сетях 5G. При этом, если в настоящее время в сетях используется MIMO 2x2 и 4x4, то в будущем число антенн увеличится. Эта технология имеет сразу два весомых аргумента для применения: скорость передачи данных возрастает почти пропорционально количеству антенн; качество сигнала улучшается при приеме сигнала сразу несколькими антеннами за счет разнесенного приема (Receive Diversity).
- 2) **Переход в сантиметровый и миллиметровый диапазоны.** На данный момент сети LTE работают в частотных диапазонах ниже 3,5 ГГц. Для полноценного функционирования сетей мобильной связи стандарта 5G необходимо разворачивать сети в более свободных высокочастотных диапазонах. При повышении частоты, на которой передается информация, уменьшается дальность связи. Это закон физики, обойти его можно лишь повышая мощность передатчика, которая ограничена санитарными нормами. Однако считается, что базовые станции сетей пятого поколения будут располагаться плотнее, чем сейчас, что вызвано необходимостью создать гораздо большую емкость сети. Преи-

мощством диапазонов десятков ГГц является наличие большого количества свободного спектра.

- 3) **Мультитехнологичность.** Для обеспечения высококачественного обслуживания в сетях 5G необходима поддержка как уже существующих стандартов, таких как UMTS,GSM,LTE так и других, например, Wi-Fi. Базовые станции, работающие по технологии Wi-Fi могут использоваться для разгрузки трафика в особо загруженных местах.
- 4) **D2D (Device-to-device).** Технология device-to-device позволяет устройствам, находящимся неподалеку друг от друга, обмениваться данными напрямую, без участия сети 5G, через ядро которой будет проходить лишь сигнальный трафик. Преимуществом такой технологии является возможность переноса передачи данных в нелицензируемую часть спектра, что позволит дополнительно разгружать сеть.

В октябре 2019 г. в Казахстане начала работать первая 5G сеть Beeline. Пилотная зона пятого поколения мобильной связи развернута в городе Шымкент. Тестирование технологии продлится три месяца и будет завершено в декабре 2019 года. По результатам пилота сотовый оператор получит анализ о необходимых инвестициях в технологию, перспективах 5G в Казахстане и оценит готовность собственных сетей к повсеместному внедрению 5G.

Сеть 5g построена на оборудовании Nokia и состоит из трех базовых станций и оборудования опорной сети. Общая площадь покрытия – порядка тринадцати квадратных километров. Этого достаточно для тестов скорости, функционала, замеров распространения сигнала как снаружи, так и внутри помещений. В дальнейшем опыт Шымкента станет готовым бизнес-кейсом по строительству и внедрению 5G. То есть, опыт Казахстана станет ключевым при внедрении 5G, его используют как в сети Beeline, так и в других компаниях группы VEON, в таких странах как Россия, Украина, Кыргызстан, Армения, Пакистан и Бангладеш и других.

Польза 5G не ограничивается высокими скоростями мобильного интернета, который для конечного абонента составит не менее 1Гбит/сек. В первую очередь, коммерческое использование данного стандарта связи необходимо для цифровизации экономики, промышленности, городов. Такое направление развития технологий как «Интернет вещей» с внедрением 5G станет носить массовый характер.

Подготовку к повсеместному внедрению 5G Beeline уже начал. В компании стартовал проект по масштабной модернизации сети. Она предусматривает полный анализ покрытия, с целью строительства новых базовых станций, которые обеспечат высокие скорости интернета, без «белых пятен» в городах. Замена оборудования на ряде объектов позволит довести скорости передачи данных до уровней 4.5G и 4.9G и в дальнейшем осуществить быстрый перевод базовых станций на работу по стандартам 5G.

Сегодня связь от компании Beeline доступна в более чем 3700 городах и населенных пунктах Казахстана. Это 95% населения, 65% из которых имеют доступ к интернету стандарта 4G.

Таким образом, эволюцию стандартов мобильной связи можно представить в следующем виде (рисунок 1.35):

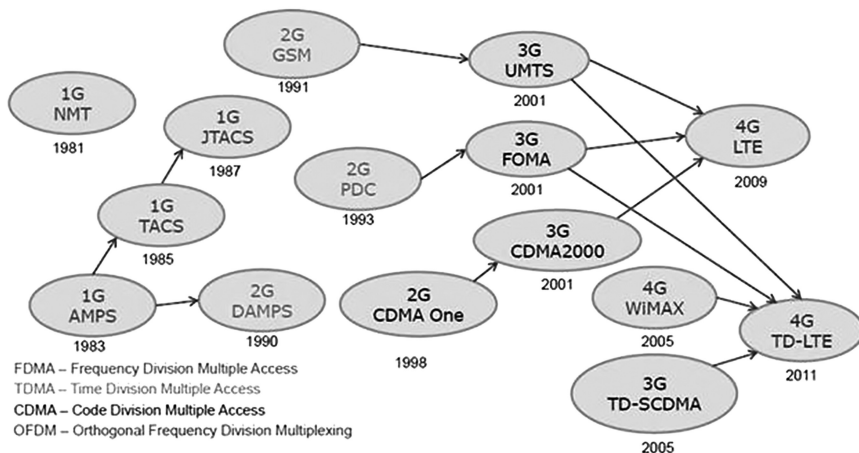


Рисунок 1.35 – Эволюция стандартов мобильной связи

1.7.3 Технология Bluetooth

Технология использует небольшие приемопередатчики малого радиуса действия, либо непосредственно встроенные в устройство, либо подключаемые через свободный порт или PC-карту. Адаптеры работают в радиусе 10 метров и, в отличие от IrDA, не обязательно в зоне прямой видимости, то есть, между соединяемыми устройствами могут быть различные препятствия, или стены.

Устройства, использующие стандарт Bluetooth, функционируют в диапазоне 2,45 ГГц ISM (Industrial, Scientific, Medical – промыш-

ленный, научный и медицинский диапазон) и способны передавать данные со скоростью до 720 кбит/с на расстояние до 10 метров и передачу 3 голосовых каналов. Такие показатели достигаются при использовании мощности передачи 1 мВт и задействованном механизме переключения частоты, предотвращающем интерференцию. Если принимающее устройство определяет, что расстояние до передающего устройства менее 10 м, оно автоматически изменяет мощность передачи до уровня, необходимого при данном расположении устройств. Устройство переключается в режим экономии энергии в том случае, когда объем передаваемых данных становится мал или передача прекращается.

Технология использует FHSS – скачкообразную перестройку частоты (1600 скачков/с) с расширением спектра. При работе передатчик переходит с одной рабочей частоты на другую по псевдослучайному алгоритму. Для полнодуплексной передачи используется дуплексный режим с временным разделением (TDD). Поддерживается изохронная и асинхронная передача данных и обеспечивается простая интеграция с TCP/IP. Временные интервалы (Time Slots) развертываются для синхронных пакетов, каждый из которых передается на своей частоте радиосигнала.

Энергопотребление устройств Bluetooth должно быть в пределах 0.1 Вт. Каждое устройство имеет уникальный 48-битовый сетевой адрес, совместимый с форматом стандарта локальных сетей IEEE 802.

Диапазон 2.45 ГГц является не лицензируемым и может свободно использоваться всеми желающими. Управляет им лишь Федеральная комиссия по коммуникациям (FCC – Federal Communication Commission), ограничивая часть диапазона, которую может использовать каждое устройство. Дело в том, что этих устройств стало очень много – начиная от беспроводных сетей, поддерживающих стандарты 802.11 и 802.11b и устройств Bluetooth и вплоть до микроволновых печей. Сейчас комиссия рассматривает просьбу увеличить используемый диапазон для Home RF (спецификация, используемая в аудио- и видеотехнике). Это увеличение может повлиять на другие устройства, работающие в этом диапазоне, количество которых увеличивается. При этом FCC заявила, что использование не лицензируемой частоты несет несомненный риск и не исключена возможность помех и конфликтов между устройствами.

Устройства стандарта Bluetooth способны соединяться друг с другом, формируя пикосети, в каждую из которых может входить до 256

устройств. При этом одно из устройств является ведущим (Master), еще семь – ведомыми (Slave), остальные находятся в дежурном режиме. Пикосети могут перекрываться, а к ресурсам ведомых устройств может быть организован доступ. Перекрывающиеся пикосети могут образовать распределенную сеть, по которой могут мигрировать данные.

Принцип действия Bluetooth

Принцип действия основан на использовании радиоволн. Радиосвязь Bluetooth осуществляется в ISM-диапазоне (англ. Industry, Science and Medicine), который используется в различных бытовых приборах и беспроводных сетях (свободный от лицензирования диапазон 2,4-2,4835 ГГц). В Bluetooth применяется метод расширения спектра со скачкообразной перестройкой частоты (англ. Frequency Hopping Spread Spectrum, FHSS). Метод FHSS прост в реализации, обеспечивает устойчивость к широкополосным помехам, а оборудование недорогое.

Согласно алгоритму FHSS, в Bluetooth несущая частота сигнала скачкообразно меняется 1600 раз в секунду (всего выделяется 79 рабочих частот шириной в 1 МГц, а в Японии, Франции и Испании полоса уже -23 частотных канала). Последовательность переключения между частотами для каждого соединения является псевдослучайной и известна только передатчику и приёмнику, которые каждые 625 мкс (один временной слот) синхронно перестраиваются с одной несущей частоты на другую. Таким образом, если рядом работают несколько пар приёмник-передатчик, то они не мешают друг другу. Этот алгоритм является также составной частью системы защиты конфиденциальности передаваемой информации: переход происходит по псевдослучайному алгоритму и определяется отдельно для каждого соединения. При передаче цифровых данных и аудиосигнала (64 кбит/с в обоих направлениях) используются различные схемы кодирования: аудиосигнал не повторяется (как правило), а цифровые данные в случае утери пакета информации будут переданы повторно [9].

Протокол Bluetooth поддерживает не только соединение «point-to-point», но и соединение «point-to-multipoint».

1.7.4 Проблемы безопасности мобильной связи

Для начала рассмотрим важные данные, которые можно найти в отчетах We Are Social и Hootsuite о глобальном состоянии цифровых технологий на 2019 год.

Сегодня в мире 5,11 миллиарда уникальных мобильных пользователей, что на 100 миллионов (2%) больше, чем в прошлом году.

В 2019 году аудитория интернета насчитывает 4,39 миллиарда человек, что на 366 миллионов (9%) больше, чем в январе 2018 года.

В социальных сетях зарегистрировано 3,48 миллиарда пользователей. По сравнению с данными на начало прошлого года этот показатель вырос на 288 миллионов (9%).

3,26 миллиарда человек заходят в социальные сети с мобильных устройств. Это на 10% больше, чем в прошлом году, когда с мобильных в соцсетях заходило на 297 миллионов человек меньше.

С даты публикации прошлогоднего отчета в сети появилось более 366 миллионов новых пользователей. Каждый день с января 2018 года в среднем один миллион человек впервые открыл для себя глобальную сеть, а это – 11 новичков в секунду [10].

Значительный прирост в 2019 году дали развивающиеся страны, где интернет был не сильно распространен. Особенно выделяется Индия, где количество онлайн-пользователей за 12 месяцев подскочило на 100 миллионов, а это более 20 процентов за год. Уровень проникновения интернета в Индии сегодня составляет примерно 41% – существенное улучшение по сравнению с прошлогодним показателем в 31%.

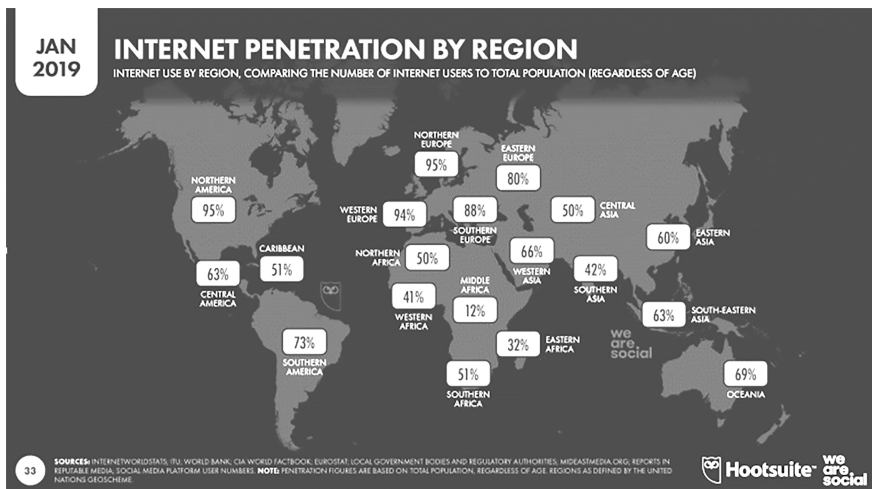


Рисунок 1.36 – Интернет-пользователи в 2019 году

Пользовательские привычки в интернете тоже быстро меняются. На мобильные телефоны сейчас приходится почти половина времени, которое люди проводят в сети.

Уже сегодня многие служащие используют смартфоны, коммуникаторы и PDA для ведения деловых переговоров и хранения важных данных, не нуждаясь при этом в ПК. В дополнение к стандартным телефонным функциям персонал может применять подобные устройства для следующих целей:

- 1) пересылки e-mail;
- 2) пересылки мгновенных сообщений (SMS, MMS, IM-сообщений с помощью ICQ или Windows Messenger);
- 3) применения приложений, в частности ERP-систем, CRM-систем, систем автоматизации продаж;
- 4) просмотра и анализа штрих-кодов;
- 5) сетевых игр;
- 6) разного рода чатов;
- 7) web-серфинга;
- 8) загрузки и совместного использования файлов в Интернете;
- 9) хранения персональной и конфиденциальной информации.

Сегодня мобильные устройства не только обеспечивают рост производительности, но и приносят новые угрозы безопасности, нарушение которой может очень дорого стоить предприятию.

Увеличивающееся число мобильных пользователей и взрывной рост Интернета уничтожили понятие периметра организации. Защита сети компании с помощью межсетевых экранов больше не отвечает современным требованиям. Пользователи часто путешествуют вне периметра, где подвергают рискам конфиденциальные и персональные данные и могут подвергнуться нападению. Мобильные устройства могут быть легко атакованы с помощью вредоносного ПО или применяться для его переноса. Другая опасность заключается в мобильности этих устройств – их легко потерять или они могут быть украдены, что поставит под угрозу данные, которые на них хранятся.

Основные риски, связанные с мобильными устройствами:

- 1) потеря конфиденциальных данных из-за хищения или утери мобильных устройств;
- 2) потеря производительности и конфиденциальных данных из-за заражения вредоносным программным обеспечением;
- 3) мошенничество; потеря производительности из-за взлома;
- 4) потеря производительности из-за взлома.

Сегодня аппаратные средства стоят во много раз меньше, чем информация, содержащаяся на устройстве. Утраченные данные могут привести к потере репутации, потере конкурентоспособности и потенциальным судебным тяжбам.

Во всем мире уже давно вопросы шифрования данных регулируются соответствующими законодательными актами. Например, в США закон U.S. Government Information Security Reform Act (GISRA) требует обязательного шифрования данных для защиты конфиденциальной информации, принадлежащей правительственным органам. В странах ЕС принята директива European Union Data Privacy Directive. Канада и Япония тоже имеют соответствующие инструкции. Защита информации в Республике Казахстан отражена в Законе Республики Казахстан «О связи» от 5 июля 2004 года N 567. Все эти законы предусматривают серьезные штрафы за утрату персональной или корпоративной информации.

Для запрета несанкционированного доступа к данным можно использовать две технологии:

- удаленное стирание данных;
- шифрование данных.

Данные могут быть стерты удаленно по специальной команде стирания или автоматически из-за нарушения политики безопасности (превышения количества неудачных входов в систему). Следует учесть, что удаленная команда стирания далеко не всегда эффективна, так как она не может быть передана, если устройство не подключено к Интернету или к беспроводной сети. В таком случае шифрование обеспечит куда более надежную защиту для ваших данных.

Кроме того, не стоит забывать о такой опасности, как несанкционированный доступ к данным во время ремонта (в том числе гарантийного) либо продажи устройств, бывших в употреблении.

Заражение вредоносным программным обеспечением. Опасность вредоносного ПО на обычных ПК стала уже привычной. Сегодня для вирусописателей все более привлекательными становятся мобильные устройства. Перспективы распространения вредоносного ПО для мобильных устройств создают серьезные риски для корпоративных пользователей.

Встроенные возможности по управлению почтовыми и текстовыми сообщениями делают данные устройства чрезвычайно привлекательными для злоумышленников. Используя Wi-Fi и Bluetooth, вредоносное ПО потенциально может распространяться внутри образованной ими одноранговой сети.

Не стоит забывать еще об одной проблеме – о спаме. У пользователей, которые отвлекаются на присылаемые SMS, значительно снижается производительность труда.

На рынке мобильной связи наблюдается устойчивая тенденция повышения уровня «интеллекта» средств мобильной связи. Резко увеличивается число смартфонов, что вполне естественно стимулирует рост числа проблем, ранее присущих только персональным компьютерам (ПК). Вместе с тем стоит отметить, что сегодня смартфоны гораздо хуже защищены от вредного воздействия.

О данной проблеме заговорили еще в июне 2004 года, после появления вируса Cabir для ОС Symbian. Причина вполне естественна – количество вирусов для мобильных устройств так же, как когда-то вирусов для ПК, сегодня исчисляется несколькими десятками.

Большинство известных вирусов для смартфонов относится к классу троянских программ и использует для реализации своей функциональности уязвимости ОС. Кроме того, существуют мобильные вирусы-черви.

В соответствии с прогнозами экспертов защититься от мобильных вирусов большинству владельцев портативных устройств будет довольно трудно.

Наиболее опасные вредоносные программы для мобильных телефонов создаются организованными преступными группами. В мире ПК спам- и онлайн-преступления уже приводят к многочисленным вспышкам вирусов. То же ожидает и мобильный мир. Наиболее вероятный сценарий развития событий состоит в том, что мобильные спамеры распространяют вирусы, которые заразят большое количество телефонов. Затем зараженные телефоны будут рассылать SMS-спам и мультимедийные сообщения по всем номерам, имеющимся в их телефонных книгах, а владельцам телефонов придется все это оплачивать.

По сообщениям McAfee, число подобных вирусов увеличивается на порядок быстрее по сравнению с аналогичными программами для персональных компьютеров. Кроме того, вредоносные программы для мобильных телефонов представляют, куда большую угрозу, чем их компьютерные собратья. Статистические данные свидетельствуют о том, что программное обеспечение, необходимое для защиты мобильных телефонов от неблагоприятных внешних воздействий, не пользуется популярностью среди пользователей [10].

Как известно, безопасность обеспечивается совместными усилиями людей, процессов и технологий. Для защиты мобильных устройств и корпоративных сетей от потери данных (производительности) руководство организации должно предпринять следующие действия:

- 1) создать в организации понятную политику использования мобильных устройств;
- 2) определить порядок аутентификации пользователей и устройств для доступа к данным;
- 3) зашифровать все данные;
- 4) обеспечить защищенные соединения для доступа к данным;
- 5) создать правила для межсетевых экранов и системы обнаружения вторжения для защиты от взлома;
- 6) установить защиту от вредоносного ПО на мобильных устройствах;
- 7) обеспечить централизованное управление этой защитой;
- 8) обеспечить понимание пользователями необходимости мер безопасности.

В организации должна быть создана понятная политика безопасности мобильных устройств, скоординированная с ИТ-отделом, юристами и отделом HR.

Стоит учитывать, что помимо мобильных устройств, принадлежащих организации, сотрудники будут пользоваться и собственными. Так что политика безопасности по применению мобильных устройств должна быть однозначной независимо от принадлежности устройств.

Для построения политики безопасности необходимо точное знание типа и модели устройств, используемых в организации. Проанализировав применяемые типы устройств, администратор безопасности сможет разобраться в дефектах их системы защиты, а следовательно, составит более четкое представление о том, какие именно действия необходимо предпринять для надежной защиты корпоративной сети.

1.8 Реализация мобильных приложений на платформе Java 2 Micro Edition

Изначально язык Java назывался Oak («Дуб»), разрабатывался Джеймсом Гослингом для программирования бытовых электронных устройств. Из-за того, что язык с таким названием уже существовал, вскоре Oak был переименован в Java. Назван в честь марки кофе Java, которая, в свою очередь, получила наименование одноименного острова Ява, поэтому на официальной эмблеме языка изображена чашка с горячим кофе. В соответствии с этимологией в русскоязычной литературе с конца двадцатого и до первых лет двадцать первого века

название языка нередко переводилось как Ява, а не транскрибировалось.

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной (JVM). Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина). Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Внутри Java существует несколько основных семейств технологий. Java 2 Micro Edition (J2ME) – это подмножественная платформа Java, которая была специально разработана для портативных устройств, например сотовых телефонов, карманных персональных компьютеров, ресиверов цифрового телевидения и представляет собой виртуальную машину с необходимым минимумом библиотек.

Выделяют следующие основные категории портативных устройств:

- 1) Устройства с небольшой мощностью и ограниченными функциональными возможностями, которые способны к нестационарным сетевым коммуникациям – мобильные телефоны, двусторонние пейджеры, карманные персональные компьютеры и органайзеры.
- 2) Устройства, обладающие большей мощностью и средствами пользовательского интерфейса (UI), подсоединенные к фиксированному непрерывному сетевому соединению – компьютерные приставки к телевизору, телевизоры с выходом в Интернет, навигационные автомобильные системы, смартфоны, коммуникаторы высокого класса.

Использование принципа модульного построения программ на J2ME, предоставляет возможность разработки приложений для разных типов устройств в зависимости от их функциональных возможностей, объема памяти и типа связи. С этой целью в J2ME используются конфигурации и профили [13].

1.8.1 Конфигурации и профили J2ME

Конфигурация J2ME определяет минимальную Java-платформу для категории портативных устройств. Другими словами, **конфигурация** – это спецификация, которая определяет доступные ресурсы системного уровня.

Конфигурация включает три базовых элемента:

- 1) набор свойств языка программирования Java;
- 2) набор свойств виртуальной машины Java;
- 3) набор поддерживаемых библиотек Java и программных интерфейсов приложения (API).

Выделяют две базовые конфигурации J2ME для соответствующих категорий портативных устройств:

- Connected, Limited Device Configuration (CLDC) – конфигурация для не стационарно подключаемых мобильных устройств с ограниченными возможностями.
- Connected Device Configuration (CDC) – конфигурация для постоянно подключенных устройств с объемом памяти два или более мега-байт и с большими функциональными возможностями.

Профиль J2ME определяет программный интерфейс для определенного класса устройств. Реализация профиля состоит из набора библиотек классов Java, которые обеспечивают интерфейс программного уровня, то есть определяют все виды функциональных возможностей и служб.

Профили используются для обеспечения реальной платформы для семейства устройств, которые имеют возможности, заданные конфигурацией. Таким образом, гарантируется взаимодействие, которое необязательно предполагает совместимость конечных продуктов различных производителей – между всеми устройствами одной категории для определения стандартной платформы разработки приложений на Java.

Например, профиль может поддерживать возможность сетевой коммуникации для стандарта Short Message Service (SMS). Поскольку стандарт SMS является повсеместно распространенным свойством в сотовой телефонии, поэтому имеет смысл задать эту службу в профиле, который предназначен для мобильных телефонов, вместо того чтобы встраивать ее в конфигурацию.

Профиль внедряется поверх конфигурации и включает библиотеки, которые соответствуют более специфичным характеристикам заданной категории устройств, чем библиотеки, которые содержат конфигурации. Приложения затем встраиваются поверх конфигурации и

профиля и могут использовать только библиотеки классов, предоставляемые этими двумя низкоуровневыми спецификациями. Профили могут быть встроены поверх друг друга, но приложение J2ME может содержать только одну конфигурацию. На рисунке 1.37 показаны уровни, из которых состоит платформа J2ME.

Java-приложение	
API профилей	
API конфигурации	Библиотеки
	Виртуальная машина Java
Операционная система компьютера	
Аппаратное обеспечение устройства	

Рисунок 1.37 – Платформа J2ME

1.8.2 Конфигурация Connected, Limited Device Configuration

Цель использования CLDC заключается в том, чтобы установить стандартную платформу Java для устройств. Из-за широкого выбора системного программного обеспечения на различных персональных устройствах CLDC исходит из минимальных предположений о среде, в которой она существует. Например, одна ОС может поддерживать множественные параллельные процессы, другая – может или не может поддерживать файловую систему и тому подобное.

CLDC отличается от CDC и представляет из себя ее подгруппу. Однако эти конфигурации независимы друг от друга, так что они не должны использоваться вместе при описании платформы.

На рисунке 1.38 показана связь между двумя конфигурациями и платформой J2SE.

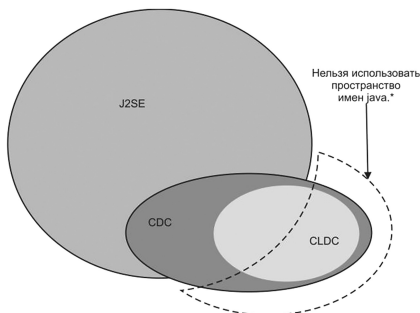


Рисунок 1.38 – CLDC является подгруппой CDC

Как и CDC, CLDC определяет требуемый уровень поддержки языка программирования Java, требуемую функциональную поддержку соответствующей требованиям виртуальной машины Java и требуемый набор библиотек классов.

Спецификация CLDC не включает поддержку следующих свойств языка Java:

- вычисления с плавающей точкой;
- финализация объекта;
- иерархия класса `Java.lang.Error` во всей его полноте.

Отсутствие поддержки плавающей точки является основным отличием на языковом уровне виртуальной машины Java.

Финализация объекта также отсутствует. Это означает, что метод `Object.finalized` был удален из библиотек CLDC.

Иерархия исключений `Java.lang.Error` также была удалена из библиотек CLDC и поэтому недоступна для приложений. Основная причина того, что обработка ошибок отсутствует, заключается в ограниченной памяти мобильных устройств. Это обычно не создает никаких неудобств при разработке приложений. Кроме того, нейтрализация ошибок на портативных устройствах, таких, как мобильные телефоны, зависит от конкретного устройства.

CLDC – это конфигурация, поверх которой встраиваются один или более профилей, поэтому CLDC предназначена для разработчиков отдельных приложений.

1.8.3 Конфигурация Connected Device Configuration

Используемая в CDC виртуальная машина Java называется компактной виртуальной машиной (Compact Virtual Machine) и имеет следующие основные свойства:

- 1) улучшенная запоминающая система;
- 2) небольшие временные интервалы сборки мусора в среднем;
- 3) полное отделение виртуальной машины от системы памяти;
- 4) модульные сборщики мусора;
- 5) сборка мусора по поколениям.

CVM была спроектирована с учетом предоставления следующих свойств:

- 1) портативность;
- 2) быстрая синхронизация;
- 3) выполнение классов Java отдельно от постоянной памяти (ROM);
- 4) поддержка естественных потоков;
- 5) зоны обслуживания малых классов;

- 6) предоставление интерфейсов и поддержка служб операционной системы реального времени (RTOS);
- 7) преобразование потоков Java непосредственно в естественные потоки;
- 8) поддержка всех свойств и библиотек виртуальной машины Java, таких как интерфейс вызова удаленных методов, интерфейс отладки виртуальной машины Java и прочее.

CDC устанавливает минимальный набор библиотек классов и API.

Она поддерживает следующие стандартные пакеты Java:

- `java.lang` – системные классы виртуальной машины Java;
- `java.util` – базовые утилиты Java;
- `java.net` – дейтаграмма Universal Datagram Protocol (UDP) и ввод/вывод (I/O);
- `java.io` – файловый ввод/вывод Java;
- `java.text` – самая минимальная поддержка интернационализации;
- `java.security` – минимальная защита на низком уровне и шифрование сериализации объекта.

1.8.4 Профиль Foundation Profile

Конфигурация вместе с профилем формирует исполняемую среду J2ME. Свойства и службы системного уровня, поддерживаемые конфигурацией, фактически спрятаны от разработчика приложений. В действительности разработчику приложения запрещен прямой доступ к ним. Если это не соблюдено, то приложение не будет считаться соответствующим требованиям J2ME.

Таблица 1.3

Пакеты профиля Foundation

Название пакета профиля Foundation	Описание
<code>java.lang</code>	Поддержка языка Java
<code>java.util</code>	Добавляет полную поддержку zip и других утилит J2SE
<code>Java.net</code>	Добавляет TCP/IP сокет и HTTP-соединения
<code>java.io</code>	Поддержка механизмов ввода/вывода
<code>java.text</code>	Обеспечивает поддержку интернационализации пакета
<code>java.security</code>	Добавляет подпись и сертификацию кодов

С точки зрения программиста, профиль необходим для «полезной» работы. Профиль определяет уровень, который содержит API. Профиль Foundation содержит в себе пакеты J2SE, перечисленные в таблице 1.3.

Профили могут быть внедрены поверх друг друга.

1.8.5 Профиль Personal Profile

Спецификация профиля Personal была разработана в Java Community, конечным результатом которой стал JSR-62. Профиль Personal обеспечивает среду с полной поддержкой AWT. Замысел его создателей заключался в том, чтобы обеспечить платформу, подходящую для Web-апплетов. Он также предоставляет способ перемещения J2ME для приложений Personal Java.

В таблице 1.4 перечислены пакеты, которые включены в профиль Personal.

Таблица 1.4

Пакеты профиля Personal

Название пакета профиля Personal	Описание
java.applet	Классы, необходимые для создания апплетов, и используемые апплетами.
java.awt	Классы AWT для создания пользовательского интерфейса программ.
java.awt.data transfer	Классы и интерфейсы для пересылки данных внутри и между приложениями.
java.awt.event	Классы и интерфейсы для обработки событий AWT.
java.awt.font	Классы и интерфейсы для работы со шрифтами.
java.awt.im	Классы и интерфейсы для описания редакторов методов ввода.
java.awt.im.spi	Интерфейсы, которые помогают в разработке редакторов методов ввода для любой среды исполнения Java.
java.awt.image	Классы для создания и изменения изображений
java.beans	Классы, которые поддерживают разработку компонентов JavaBean.
javax.microedition.xlet	Интерфейсы, используемые приложениями и диспетчерами приложений профиля J2ME Personal для коммуникации.

1.8.6 Профиль RMI

Профиль Remote Method Invocation (RMI) является профилем, созданным для платформ, которые поддерживают конфигурацию CDC. Он был задан JSR-66 и определен различными компаниями, принявшими участие в Java Community Process.

Профиль RMI требует внедрения профиля Foundation и внедряется поверх него. Продукты профиля RMI должны поддерживать следующие свойства:

- 1) полную семантику RMI вызовов;
- 2) поддержку объектов маршалинга;
- 3) RMI проводного протокола;
- 4) экспорт удаленных объектов через API UnicastRemoteObject;
- 5) распределенную сборку мусора и интерфейсы сборщика мусора, как для клиента, так и для сервера;
- 6) интерфейс активатора и протокол активации для клиента;
- 7) интерфейсы реестра RMI и экспорт реестра удаленных объектов.

1.9 Мобильные операционные системы

Рассмотрим самые распространенные операционные системы для мобильных устройств. В основе подавляющего большинства из них лежит мобильная платформа Java.

Symbian OS

The image shows the word "symbian" in a bold, lowercase, sans-serif font. The letter 'i' is stylized with a vertical bar through its center.

Была самой популярной ОС для мобильных устройств благодаря поддержке фирмы Nokia. Важную роль также сыграло то, что система имеет небольшой размер, а также то, что графический интерфейс и ядро системы отделены друг от друга. Это позволило легко портировать ее для различных мобильных устройств. Позднее была добавлена многозадачность.

Каждый разработчик создавал свой дистрибутив этой операционной системы в зависимости от ограничений аппаратной платформы, под которую она разрабатывалась. Так появились версии Series 60, Series 80, Series 90, UIQ и MOAP. Каждая версия обладала своими особенностями, что делало необходимым под каждую версию разра-

батывать свои приложения. Это было неудобно, поэтому после появления Windows Mobile, Android и iPhoneOS утратила свою популярность среди производителей мобильных устройств. Так, в этом году компании Sony Ericsson и Samsung объявили, что не будут больше поддерживать эту операционную систему. На данный момент из крупных производителей мобильных устройств только компания Nokia использует эту ОС для своих смартфонов [11].

Достоинства:

- 1) Низкие требования к памяти и процессору.
- 2) Функция освобождения неиспользуемой памяти.
- 3) Стабильность.
- 4) Малое количество вирусов для этой платформы.
- 5) Быстро выходят новые версии и исправляются нестабильности.
- 6) Большое количество программ.

Недостатки:

- 1) Для связи с ПК нужно устанавливать дополнительный софт.
- 2) Несовместимость программ для старых и новых версий.

Windows Mobile



Эта операционная система разработана мировым лидером в производстве операционных систем – компанией **Microsoft**. Эта система использует такой же программный интерфейс, что и настольная версия. Это делает написание программ более простым, а пользователям нравится удобный и понятный интерфейс, знакомый им с настольной Windows. Windows Mobile является компонентной, многозадачной, многопоточной и многоплатформенной операционной системой. Благодаря этому ОС имеет широкое распространение на мобильных устройствах.

Достоинства:

- 1) Схожесть с настольной версией.
- 2) Удобная синхронизация.
- 3) В комплекте идут офисные программы.
- 4) Многозадачность.

Недостатки:

- 1) Высокие требования к оборудованию.
- 2) Наличие большого числа вирусов.
- 3) Нестабильности в работе.

Android

Android – одна из самых молодых мобильных ОС, основанная на базе операционной системы Linux и разрабатываемая Open Handset Alliance (ОНА) при поддержке **Google**. Исходный код находится в открытом доступе, благодаря чему любой разработчик может создать свою версию этой мобильной ОС. Разработчикам приложений выдвинуто небольшое количество ограничений, благодаря чему существует множество как платных, так и бесплатных приложений, которые можно удобно загрузить с Android Market [11].

Достоинства:

- 1) Гибкость.
- 2) Открытые исходные коды.
- 3) Множество программ.
- 4) Высокое быстродействие.
- 5) Удобное взаимодействие с сервисами от Google.
- 6) Многозадачность.

Недостатки:

- 1) Множество актуальных версий – для многих устройств новая версия входит слишком поздно или не появляется вовсе, поэтому разработчикам приходится разрабатывать приложения, ориентируясь на более старые версии
- 2) Высокая предрасположенность к хакерским атакам из-за открытости кода.
- 3) Почти всегда требует доработок.

iPhone OS (IOS)



Мобильная операционная система от компании **Apple**. Данная система получила распространение только на продуктах компании Apple. Применяется в iPhone, iPod, iPad а также телевизионной приставке AppleTV.

Достоинства:

- 1) Удобство пользования.
- 2) Качественная служба поддержки.
- 3) Регулярные обновления, устраняющие многие проблемы в работе.
- 4) Возможность купить в App Store множество различных программ.

Недостатки:

- 1) Необходимость джейлбрейка (англ. jailbreak – «побег из тюрьмы», взлом) для установки неофициальных приложений.
- 2) Заблокированный характер ОС.
- 3) Отсутствие многозадачности.
- 4) Нет встроенного редактора документов.

Palm OS



Данная операционная система появилась в 1996 году. Применялась в КПК. Была очень распространена из-за широких возможностей и удобства пользователей. К настоящему моменту практически не применялась, но недавно разработчика поглотила компания HP. Благодаря этому появились надежды на воскрешение некогда популярной среди КПК операционной системы.

Достоинства:

- 1) Нетребовательна к ресурсам.
- 2) Очень удобный интерфейс пользователя.
- 3) Удобная синхронизация с ПК.
- 4) Надежность.

Недостатки:

- 1) Отсутствует полноценная многозадачность.
- 2) Не развиты мультимедийные функции.
- 3) Система не развивается (хотя возможно компания HP сможет это преодолеть).

BlackBerry OS

Операционная система работает исключительно на устройствах, выпускаемых компанией **Research In Motion Limited (RIM)**. Ориентирована на корпоративных пользователей. Свое название получила от смартфонов для которых создавалась, так как клавиатура смартфонов были похожи на ягоду ежевики. Смартфоны с этой операционной системой получили распространение в корпоративной среде, благодаря сложности перехвата сообщений.

Достоинства:

- 1) Удобное пользование электронной почтой.
- 2) Легкая синхронизация с ПК.
- 3) Широкие возможности настроек безопасности.

Недостатки:

- 1) Оптимизирована для вывода только текстовой информации, качество работы с графикой не очень хорошее.
- 2) Не очень удобный браузер.

Как видим, технические характеристики устройства отнюдь не главный параметр при выборе мобильного девайса. Ведь смысла от аппарата, который напичкан самым современным железом, но который работает на операционной системе с небольшими возможностями, минимален. Поэтому пользователь должен с большим вниманием выбирать подходящую под именно его требования мобильную операционную систему.

1.10 Конструкторы мобильных приложений

«Если вас нет в Интернете, вас нет в бизнесе» – сказал когда-то Билл Гейтс и был, безусловно, прав. В наше время стоит задуматься – есть ли вы в бизнесе, если у вас нет мобильного приложения?

С развитием интернет технологий появляются все новые и новые инструменты, автоматизирующие работу людей, а также помогающие значительно экономить время и деньги. Сегодня не обязательно быть программистом-разработчиком, чтобы делать сайты. Специальные сервисы-конструкторы позволяют за считанные часы сделать сайт для заказчика. Для этого не нужно редактировать HTML-код, достаточно всего лишь поставить нужные элементы в визуальном редакторе.

Такая же ситуация и с мобильными приложениями. Для разработки мобильных приложений существует специальный сервис – конструктор мобильных приложений. Конструктор позволяет разработать полноценное функциональное приложение всего за несколько дней.

В визуальном редакторе сервиса загружаются и настраиваются нужные элементы мобильного приложения – фоновые изображения, иконки, адрес компании, функции приложения, внешний вид, меню и прочее [14].

Конструкторы мобильных приложений появились относительно недавно – с распространением смартфонов. Разработчики быстро поняли, что программы для смартфонов очень нужны предпринимателям, чтобы продавать свои товары и услуги, но не все из них могут сделать приложение сами.

Так появились платформы-конструкторы, которые дают любому желающему возможность самостоятельно создать мобильное приложение, как коллаж из фотографий. Их множество и, как правило, все они, так или иначе, являются платными.

Платформы для создания мобильных приложений различаются между собой набором функций, ценами, а также тем, как с их помощью можно сделать приложение. По последнему признаку они делятся на две основные категории:

Генераторы. Это платформы, которые создают мобильное приложение на основе существующей веб-страницы пользователя. Пользователь дает генератору URL своего сайта, и он автоматически создает мобильное приложение с теми же разделами и контентом, что и у пользователя на сайте.

Конструкторы. Это платформы, которые позволяют пользователю собрать приложение самому из готовых элементов. В распоряжении

пользователя будут готовые шаблоны и элементы интерфейса, а также некоторые функциональные возможности, например, геопозиционирование, отправка уведомлений, работа с банковскими картами и многое другое.

Есть два типа приложений, которые умеют создавать эти платформы:

Гибридные (PWA). Это, фактически, приложения под веб, адаптированные под экран мобильного устройства. Они открываются на смартфоне при помощи браузера.

Нативные. Это приложения, которые устанавливаются в операционную систему мобильного устройства. Нативные приложения наиболее удобны для пользователя и выгодны для предпринимателя.

Создание приложений можно реализовать двумя направлениями. Во-первых, можно купить у сервиса его исходники и самостоятельно поддерживать их и распространять приложение. Кроме того, можно купить платную подписку, и тогда команда сайта сама опубликует приложение в App Store/Google Play и будет поддерживать его.

Помимо платы за поддержку придется купить аккаунт в App Store или Google Play, который стоит \$99 и \$25 соответственно. Чтобы окупить затраты, у многих платформ есть программы лояльности, которые позволяют клиентам не только сделать приложение, но и зарабатывать на нем – например, подключив рекламу.

Рассмотрим все плюсы и минусы приложений, разработанных в конструкторе.

- 1) Разрабатывая мобильное приложение в конструкторе, заказчик данного приложения **существенно экономит бюджет компании**, потому что приложения, созданные в конструкторе, стоят намного дешевле, чем приложения, написанные разработчиками. А по функциональным возможностям приложение, разработанное в конструкторе, ничем не отличается от приложения, разработанного программистом. Поэтому, создавая приложения в конструкторе, можно охватить наиболее широкую аудиторию. То есть клиентами могут стать небольшие компании малого и среднего бизнеса, которые ориентированы на конечного клиента. А это большие возможности для бизнеса на мобильных приложениях. Ведь чем больше клиентов, тем выше потенциальный доход.
- 2) Поскольку конструктор мобильных приложений так же написан программистами-разработчиками, он также состоит из кода,

который давно протестирован на тысячах других работающих приложениях, то *ошибок в разработке приложений нет*. А в приложении, которое пишется отдельно программист, может допустить ошибку, и тогда придется перерабатывать весь код заново, чтобы устранить недочеты. А это занимает много времени – несколько недель, а то и месяцев.

- 3) Функциональные возможности приложений для бизнеса предоставляют владельцу бизнеса решать маркетинговые задачи, то есть привлекать новых клиентов, удерживать уже имеющихся. Поэтому в приложениях для бизнеса необходимы функции, которые позволяют решать подобные задачи, а это – лояльность, QR-купоны, геокупоны, интернет-магазин, меню, push-уведомления и так далее. Если посмотреть приложения для бизнеса, созданные разработчиками и приложения, собранные на конструкторе мобильных приложений, то легко проследить взаимосвязь по функционалу. Вопрос в том, зачем переплачивать за приложение и ждать несколько месяцев, когда можно разработать приложение и опубликовать его за пару недель? Если функций конструктора недостаточно для заказчика, то тогда, конечно, следует обратиться к разработчику, чтобы заказать приложение с нужным функционалом. Потому, что программист может написать любую нужную функцию. В этом большой плюс работы с программистом. Но если у компании малого бизнеса доход небольшой, то нецелесообразно тратить месячный доход фирмы на разработку такого инструмента, как приложение.
- 4) **Правки «на лету»**. Приложения, разработанные в конструкторе AppGlobal, *не нужно заново публиковать в AppStore и GooglePlay после того как в них внесли изменения уже после публикации*. Допустим, у ресторана поменялось меню с летнего на зимнее. Владелец ресторана важно, чтобы в приложении отображалась актуальная информация. Конструктор мобильных приложений, позволяющий делать правки «на лету» весьма уместен. Потому, что все изменения, которые вносятся в приложение после публикации, тут же будут отображены на всех телефонах пользователей данного приложения. А для этого не придется проходить повторную модерацию в библиотеках AppStore и GooglePlay. Это несомненный плюс для владельцев приложений, а также для самих разработчиков.

- 5) *Скорость разработки приложения.* Еще один плюс работы в конструкторе возможность разработать приложение всего за несколько часов. Но на полноценную разработку потребуется несколько дней плюс неделя или две на публикацию приложения. Приложения, написанные разработчиком, создаются в течение нескольких месяцев, иногда время ожидания разработки приложения может достигать полугода. Для владельцев бизнеса это не совсем выгодно.

Подводя итоги можно сказать, что конструктор мобильных приложений – визуальный онлайн-редактор на веб-платформе, в котором пользователи могут самостоятельно создавать, тестировать и продвигать приложения для мобильных устройств. В большинстве случаев не требует для работы специальных знаний и навыков программирования.

1.11 Рынок конструкторов мобильных приложений

Рынок конструкторов мобильных приложений не такой большой, как кажется с первого взгляда. Для обзора мы учитывали несколько важных факторов, к которым отнесли:

Доступность – здесь под доступностью понимается порог вхождения и требуемые знания (программирование, дизайн и так далее).

Интерфейс – учитывается язык конструктора, простота последовательных действий, возможность протестировать приложения до публикации.

Функциональные возможности – набор модулей и возможности конечного продукта.

Индивидуальность и брендинг – возможность создания не типового шаблонного проекта, загрузка собственных иконок и экрана загрузки.

Управление приложением – возможность создавать по-настоящему динамический контент.

Ограничения – встроенная реклама, ограниченные или платные установки и так далее.

Сроки разработки – среднее время, которое потребуется для разработки приложения и его публикации.

Стоимость – конечная стоимость продукта и обоснованность ценообразования.

Служба поддержки – готовность быстро помочь на родном языке.

Предлагаем вниманию читателей некоторые конструкторы мобильных приложений.

Tadapp

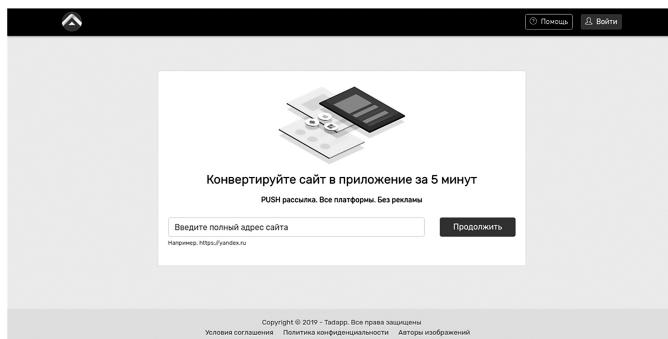


Рисунок 1.39 – Интерфейс конструктора приложений #0. Tadapp

Tadapp – сервис конвертации сайта в приложение. Если у сайта хорошая и адаптивная мобильная версия, то выбор надо сделать в пользу Tadapp. Сервис за несколько минут конвертирует сайт в приложение для девайсов Android, iOS и Windows.

Доступны бесплатные нативные PUSH рассылки. Не требуются знания программирования. Нужно только указать адрес сайта.

Интерфейс прост и понятен. В личном кабинете можно управлять сразу несколькими приложениями.

Функциональные возможности: возможность бесплатных PUSH рассылок, off-line страницы для приложения. Функционал приложения полностью повторяет функционал сайта.

Индивидуальность и брендинг: возможно загрузить собственный экран загрузки приложения, иконки, дизайн приложения соответствует мобильной версии сайта.

Сроки разработки: для получения файлов приложения понадобится несколько минут.

Стоимость: можно создать бесплатное приложение, но имеются и платные виды активаций для некоторых дополнительных функциональных возможностей. Стоимость активации платных версий от 15 USD – платеж единовременный. Абонентских плат нет.

iBuildApp

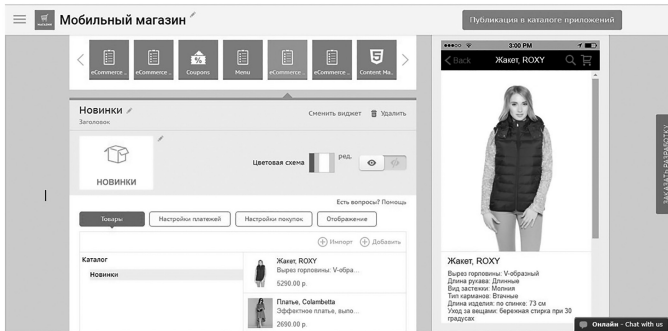


Рисунок 1.40 – Интерфейс конструктора мобильных приложений iBuildApp

iBuildApp – это конструктор мобильных приложений, номер один в мире по количеству зарегистрированных пользователей.

Клиенты iBuildApp создали более 500 000 мобильных приложений за последние 2 года.

iBuildApp разработал и запатентовал технологию «сделай сам» для быстрого создания iPhone/Android приложений без навыков программирования.

Доступность: не требуются знания программирования.

Интерфейс: понятные блоки для редактирования. Дизайн интерфейса несколько отстает от современных backend решений.

Функциональные возможности: можно создать, как и информационное приложение, так и интернет-магазин. Возможность PUSH рассылки. Формы, галереи, мультимедиа, фиды (это лента, в которую автоматически подгружаются блоки контента). Управление клиентами/пользователями доступно только на самом дорогом тарифе.

Индивидуальность и брендинг: большая коллекция шаблонов, возможно загрузить собственный экран загрузки приложения, иконки.

Управление приложением: можно добавлять новый контент и после публикации приложения без его обновления.

Ограничения: тарифные планы распространяются только на одно приложение и на 10 приложений для максимального тарифа. Встроенной рекламы в приложениях нет. На начальном тарифе есть ограничения в установках – до 500.

Сроки разработки: для небольшого приложения понадобится 2-3 дня.

Стоимость: бесплатный тестовый период на 14 дней. Абонентская плата небольшая. Начальные тарифы – только для одного приложения. Для максимального тарифа доступно 10 приложений.

Служба поддержки: на сайте активная База знаний, онлайн поддержка в чате и через тикет-систему. Скорость реакции поддержки - моментально (в чате)

Apps-Tech



Рисунок 1.41 – Интерфейс конструктора мобильных приложений Apps-Tech

Бесплатный конструктор мобильных приложений Apps-Tech позволит пользователю в короткий срок и без привлечения программистов создать мобильное приложение.

Доступность: не требуются знания программирования.

Интерфейс: большой список модулей. Есть возможность предварительного просмотра.

Функциональные возможности: можно создать как информационное приложение, так и магазин. Возможность PUSH рассылок. Формы, галереи, мультимедиа. Управление клиентами/пользователями доступно.

Индивидуальность и брендинг: большая коллекция шаблонов, можно загрузить собственные иконки.

Управление приложением: возможно добавление контента без обновления.

Ограничения: разные тарифные планы предлагают разные ограничения по количеству приложений – 1/3/10.

Стоимость: бесплатный тестовый период на 7 дней. Абонентская плата от 4.99 до 19.99 USD в месяц или от 59.88 до 239.88 USD в год.

Служба поддержки: Есть поддержка по телефону и Email. Скорость реакции поддержки – около 1 часа.

Alstrapp



Рисунок 1.42 – Интерфейс конструктора приложений Alstrapp

Alstrapp – полноценная CMS для создания и управлением приложением для Android и iOS. Приложение, созданное через Alstrapp, будет иметь чистый код и без труда пройдет модерацию в мобильных приложениях.

Доступность: не требуются знания программирования. Однако требуется время на подключение своего аккаунта Adobe Build и OneSignal. Может показаться, что это достаточно сложно, но это дает полную независимость от разработчика.

Интерфейс: интерфейс выполнен в современном стиле. Есть система локализаций, в том числе, на русском языке. Есть возможность предварительного просмотра и загрузки приложения на смартфон до публикации.

Функциональные возможности: можно использовать для создания информационного приложения, не магазина. Возможность PUSH и Alert рассылок. Формы, галереи, мультимедиа, чат с пользователями.

Управление: управление клиентами/пользователями доступно, есть система авторизации в приложении, можно хранить созданные приложения на серверах PhoneGap или скопировать приложение в локальное хранилище и использовать бесплатный тарифный план PhonGap

Индивидуальность и брендирование: два основных шаблона для Android и iOS устройств. Есть шаблон, где устройство пользователя определяется автоматически. Несколько цветовых схем. Возможность загружать собственные иконки и экраны загрузки.

Управление приложением: весь контент в приложении (кроме иконок и настроек названия) динамический. Это значит, что можно добавлять любой контент в приложение без его обновления. Можно управлять десятками приложений в одной панели.

Ограничения: одна лицензия позволяет создавать неограниченное количество приложений без технических ограничений и рекламы разработчика.

Сроки разработки: можно опубликовать приложение практически сразу, а уже потом заняться добавлением контента. Среднее время создания 1-2 дня.

Стоимость: единоразовая покупка – 64 USD. Весьма экономично, учитывая, что не придется ежемесячно вносить платежи. Alstrapp устанавливается на хостинг/сервер, поэтому могут потребоваться дополнительные расходы, но они несравнимы с платными подписками.

Служба поддержки: на сайте нет активной Базы знаний, но предоставляется доступ к документации после покупки лицензии. Есть поддержка в чате и Email. Скорость реакции поддержки – моментально (в чате).

AppsGeyser

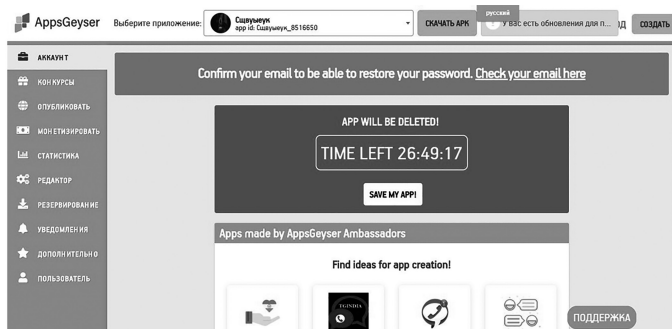


Рисунок 1.43 – Интерфейс конструктора приложений AppsGeyser

Иностранский проект конструктора приложений. Его особенность в том, что пользователь можете превратить свой сайт в приложение и оставить базовый дизайн.

Доступность: не требуются знания программирования. Есть возможность использовать русский язык.

Интерфейс: интерфейс весьма простой и все понятно с первого взгляда. Отчасти это вызвано небольшими функциональными возможностями конструктора.

Функциональные возможности: подойдет только для создания приложения, где будет отображаться сайт пользователя или подготовленный набор HTML страниц. Возможность PUSH рассылок.

Индивидуальность и брендинг: есть коллекция шаблонов (цветовых схем). Возможность загружать собственные иконки и экраны загрузки.

Управление приложением: можно добавлять контент, для этого необходимо опубликовать контент на своем сайте или заранее загрузить свои HTML страницы.

Ограничения: встроена и самая разнообразная реклама. Доход от рекламы делится с пользователем 50/50. Этот конструктор не для бизнес приложений.

Стоимость: это бесплатный конструктор. Взамен необходимо поместить рекламный баннер в своем приложении.

Служба поддержки: есть База знаний, поддержка в Email. Язык поддержки – английский. Скорость реакции – около 5 часов.

Appmachine

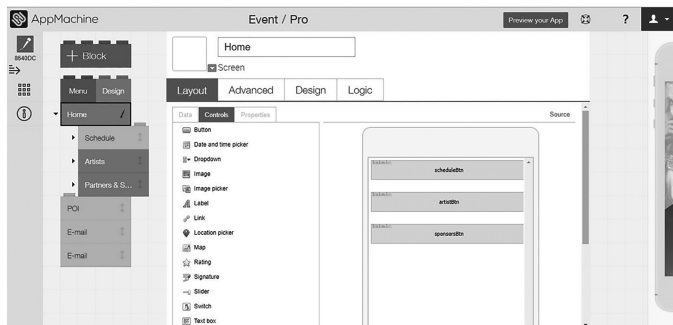


Рисунок 1.44 – Интерфейс конструктора мобильных приложений Appmachine

Достаточно мощный инструмент для создания собственного приложения. Имеет большой выбор встроенных шаблонов и модулей. Понятная логика конструктора, несмотря на отсутствие русского языка.

Доступность: не требуются знания программирования. Приложения нужно собирать из готовых блоков. К минусам можно отнести только отсутствие русской локализации.

Интерфейс: продуманный интерфейс с возможностью предварительного просмотра приложения. Есть возможность скачать приложение на свой смартфон для тестирования.

Функциональные возможности: подойдет для создания практически любого приложения. Возможность PUSH рассылки. Формы, медиаконтент и десятки других модулей. Система управления пользователями/клиентами.

Ограничения: в зависимости от тарифного плана, есть ограничения на количество приложений. На начальных тарифах – оплата только за одно приложение.

Индивидуальность и брендинг: большая коллекция шаблонов (платные и бесплатные). Возможность загружать собственные иконки и экраны загрузки.

Управление приложением: пользователь может добавлять динамический контент без предварительного обновления приложения.

Сроки разработки: среднее время разработки – 2-3 дня.

Служба поддержки: есть База знаний, поддержка в Email. Язык поддержки – английский. Скорость реакции – около 3 часов.

Shoutem

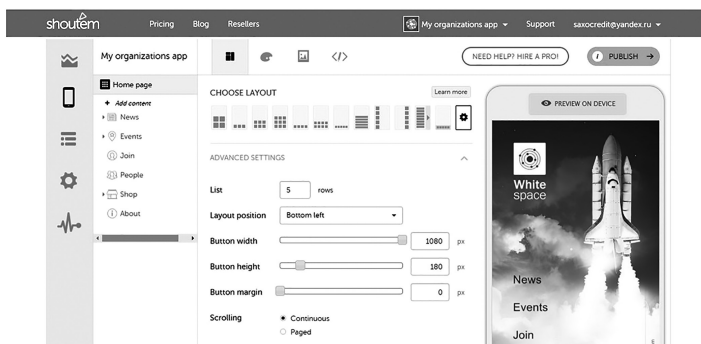


Рисунок 1.45 – Интерфейс конструктора мобильных приложений Shoutem

Один из самых понятных конструкторов. Все блоки и шаблоны демонстрируются в понятных изображениях.

Доступность: не требуются знания программирования. Приложения нужно собирать из готовых блоков. К минусам можно отнести только отсутствие русской локализации.

Интерфейс: современный интерфейс с возможностью предварительного просмотра приложения. Несмотря на английский язык – с интерфейсом можно легко разобраться.

Функциональные возможности: подойдет для создания практически любого приложения. Возможность PUSH рассылок. Формы, медиаконтент и десятки других модулей.

Ограничения: в зависимости от тарифного плана, есть ограничения по типу приложения и его функциональным возможностям. Например, на начальном тарифе, возможно создание приложения только для Android и без такой важной функции, как Push рассылка.

Индивидуальность и брендинг: шаблоны с разными типами разметки, привычные и удобные для глаз. Возможность загружать собственные иконки и экраны загрузки.

Управление приложением: обновление контента по каналам API и синхронизация контента возможна только на продвинутых тарифных планах.

Сроки разработки: среднее время разработки – 2-3 дня.

Служба поддержки: есть База знаний, поддержка в Email. Язык поддержки – английский. Скорость реакции – около 3 часов.

Flipab



Рисунок 1.46 – Интерфейс конструктора мобильных приложений Flipab

С первого взгляда – настоящий «фотошоп» для мобильных приложений. Это самый мощный и полезный инструмент, на сегодняшний день, для самостоятельного создания приложения без знания программирования. Это не онлайн конструктор. Это полноценное ПО для установки на компьютере пользователя.

Доступность: не требуются знания программирования. Однако, ввиду отсутствия русского языка и непростого интерфейса настроек, требуется потратить продолжительное время на изучение возможностей конструктора. Документация представлена на английском языке, что затрудняет использование ПО.

Интерфейс: профессиональный интерфейс. Пользователю, который знаком с другими аналогичными программами, будет комфортно работать с приложением.

Функциональные возможности: подойдет только для создания информационных приложений. Возможность PUSH рассылок отсутствует. Формы, медиаконтент и десятки других модулей включены в программу. Этот конструктор подойдет для красивых статичных приложений.

Индивидуальность и брендинг: шаблоны с разными типами разметки, но есть и одностраничные приложения. Возможность загружать собственные иконки и экраны загрузки.

Ограничения: для бесплатной версии отсутствует поддержка. В момент запуска приложения пользователь увидит логотип разработчика. С другой стороны, ограничения небольшие взамен на бесплатное использование программы для неограниченного количества приложений.

Управление приложением: пользователь может создать только статичный контент. Здесь нельзя добавлять динамический контент без обновления приложения.

Сроки разработки: среднее время разработки - 7-8 дней.

Служба поддержки: для бесплатного плана поддержка не предусмотрена.

И теперь рассмотрим топ лучших конструкторов мобильных приложений в 2019 году.

Appy Pie

Самая быстрорастущая платформа для создания приложений в мире. Appy Pie сумел оставить глубокий след в индустрии разработки. Эта платформа особенно полезна для тех, кто впервые берется за приложения и является новичком в этой области. Малые и средние предприятия сочтут платформу особенно полезной из-за большой гибкости в оплате – различные тарифные планы дают возможность начать создавать свои приложения бесплатно, а затем перейти на ту подписку, которая соответствует их бюджету.

Одна из причин, почему эта платформа так быстро завоевала популярность, заключается в большом количестве предлагаемых уникальных функций. Например, с помощью Appy Pie можно добавить в приложение встроенные покупки, рекламу, загрузить электронные книги или другой контент, подключить базы данных, интегрировать социальные сети, создать приложение для обмена мгновенными сообщениями и так далее. Это одна из самых простых в использовании платформ, основанная на drag and drop интерфейсе, позволяющая людям, не имеющим навыков программирования, без проблем создавать свои бизнес-приложения и делать это всего за несколько минут. Ценовые планы Appy Pie вполне доступны и идеально подходят для стартапов, одиночных предпринимателей и предприятий малого и среднего бизнеса [14].

Swiftic

Разработанный в 2010 году, Swiftic начинался как израильский Como, и с тех пор на нем по всему миру сделано более миллиона приложений. Компоненты или строительные блоки, предоставляемые этой платформой создания приложений, разнообразны и с их помощью можно создать карту лояльности, e-commerce магазин. Можно собирать обзоры и оценки от пользователей или реализовать приложение для мероприятия. Большинство приложений, созданных на их платформе, принадлежат ресторанам, музыкальным группам и другим организациям, которые проводят мероприятия.

Платформа предлагает семь различных шаблонов, которые можно комбинировать с шестью различными стилями навигации. Что делает приложение по-настоящему авторским. Пользователь может выбирать цвета приложения, фоновые изображения и иконки, которые можно изменить по желанию. Редактор конструктора продуман и прост, а количество функций и вариантов дизайна достаточно разнообразно [14].

GoodBarber

Платформа для создания приложений предлагает одни из самых впечатляющих шаблонов. Помимо внешнего вида, конструктор также реализует некоторые самые передовые функции – социальные сети, чаты, геофенсинг, iBeacons и многое другое. С GoodBarber можно получить собственное Android-приложение за 32 доллара в месяц, а если получить сразу Android и iOS, то ежемесячная подписка может

дойти до 96 долларов. Оба плана реализуют и красивое прогрессивное веб-приложение.

Шаблоны конструктора достаточно красивы и предлагают довольно конкурентоспособную цену для нативных приложений. Отличные дополнительные функции, такие как push-уведомления, чаты и так далее дают большую гибкость при создании приложений. Однако одним из недостатков является отсутствие собственного компонента интернет-магазина, но пользователь всегда можете интегрировать сторонние магазины, вроде Amazon, Etsy, Shopify и так далее[14].

BuildFire

BuildFire – одна из наиболее надежных платформ, с помощью которой уже более 30 000 компаний создали свои приложения. Большинство их клиентов – предприятия, инфлюенсеры или бренды. BuildFire называет себя одной из ведущих платформ на рынке ускоренной разработки приложений.

Удобные панели управления и администрирования облегчают процесс выпуска обновлений. Платформа популярна среди клиентов благодаря простоте использования, возможностям быстрой перенастройки, а также широким возможностям изменения приложения. Можно вносить изменения «на лету» и даже тестировать эти изменения в режиме реального времени. BuildFire предлагает бесплатный план и Премиум-план, который доступен за 49 долларов в месяц[14].

Mobincube

Эта платформа создания приложений зарекомендовала себя как конструктор для всех. Их девиз - «каждый должен иметь возможность создать свое собственное приложение и заработать на нем много денег через Admob». Платформа позволяет выпустить приложение с любым планом – от самого дешевого до самого дорогого.

Редактор позволяет создавать довольно сложные функции, вроде баз данных SQLite для извлечения данных с внешнего сервера. Хотя у конструктора есть огромное поле для улучшения в области UX, нет другой платформы для создания приложений, которая бы предлагала выпустить настоящее приложение бесплатно.

Платформа предлагает очень конкурентоспособные цены и большой выбор шаблонов с инновационными функциями. Однако у нее есть множество технических проблем, а шаблоны слишком ограничены в изменениях [14].

AppInstitute

В настоящее время крупнейший конструктор приложений в Великобритании, AppInstitute выиграл несколько престижных наград для стартапов. Платформа особенно хороша, когда речь заходит о функциях электронной коммерции, таких как, например, настраиваемые каталоги или программы лояльности. Самое замечательное в том, что можно получать платежи из приложения. Это очень удобно для тех, кто заботится о безопасности и конфиденциальности.

Приложения, которые создаются на платформе, можно получить за ежемесячную подписку в размере 49 долларов в месяц (Android & PWA) или 81 доллар в месяц (iOS). В подписку уже включены такие функции, как базовые push-уведомления и аналитика. Функций, на самом деле, намного больше, чем может показаться на первый взгляд. Тем не менее, в области интеграции с другими платформами и системами их явно недостаточно [14].

AppMakr

На этом конструкторе сделано почти 2 миллиона приложений. Его авторы не только утверждают, что создать приложение можно всего за 20 минут, но и позволяют сделать это довольно дешево. У них есть бесплатный план, который можно превратить в ежемесячный план с PWA за 2 доллара, или в годовой план с iOS и Android приложениями за 99 долларов. В целом, эта платформа для создания приложений на самом деле довольно дешевая.

AppMakr предлагает большое количество блоков и возможность менять дизайн приложения, но визуально редактор разочаровывает. Может в нем и есть превосходные функции [14].

Appery

Это один из облачных конструкторов, который можно использовать для создания приложений для ведущих магазинов, включая Android, iOS и Windows Phone.

Поскольку это облачная платформа, нет необходимости загружать и/или устанавливать что-либо себе на компьютер, а значит можно начать создавать приложения. Эта платформа также предлагает drag and drop интерфейс для создания пользовательских интерфейсов и позволяет подключаться к любому REST API, а затем использовать его в приложении пользователя.

Более того, данные приложения можно хранить в мгновенно добавляемой облачной базе данных. Сделанными здесь приложениями можно просто делиться в режиме реального времени – с разработчиками, бизнес-пользователями или клиентами.

Mobile Roadie

Это один из крупнейших игроков в области создания мобильных приложений, у конструктора в клиентах Disney, TED.org, Universal и другие аналогичные компании. Конструктор предлагает много вариантов дизайна для шаблонов, но есть и дополнительный плюс в том, что можно доработать их по своему вкусу.

Конструктор предлагает некоторые замечательные функции для сообществ – чат, фан-стену и тому подобное, что делает его особенно привлекательным для артистов и групп. Популярностью пользуется и музыкальный плеер, который можно интегрировать в приложение, особенно среди музыкантов. Помимо этого, платформа может похвастаться некоторыми довольно продвинутыми функциями, такими как геотаргетинг для контента.

Однако из-за широкого набора функций, которые предлагает конструктор, создание приложений на нем занимает немного больше времени. Кроме того, пользовательский интерфейс должен быть немного более интуитивным.

TheAppBuilder

TheAppBuilder предлагает два разных подхода и имеет целый набор шаблонов, которые подойдут самым разным клиентам. Приложения легко создаются с помощью онлайн-инструментов, есть обучающий курс и встроенная помощь от самого конструктора. Легко можно определить структуру приложения и заполнить его некоторыми данными.

Можно использовать выделенную AppLibrary и предложить пользователям выбор из нескольких приложений, а также есть возможность брендировать ее. Более того, есть возможность вносить изменения в структуру и содержание своего приложения даже после запуска. Любые сделанные обновления станут доступны в течение 60 секунд после их сохранения [14].

GameSalad

GameSalad – платформа, которая позволяет создавать и публиковать игры для ряда платформ, включая iOS, Android, HTML5 и OS X.

Интерфейс с перетаскиванием позволяет создавать игры без каких-либо навыков программирования.

У создателя игры есть редактор сцен и действующих лиц, в нем и будут происходить все действия. Здесь можно настроить сцену, определить свойства героев, добавить изображения, звуки и так далее. Платформа для создания игр также имеет довольно активный форум, который предлагает помощь и советы других независимых разработчиков.

BiznessApps

BiznessApps – это платформа, которая поможет создать мобильные приложения для бизнеса. Конструктор имеет множество функций – заказ еды, электронный магазин, программа лояльности, интеграция сторонних данных, push-уведомления, аналитика и многое другое.

Можно создать приложение в течение нескольких минут, используя удобную систему управления контентом, где можно настроить всё, используя свой собственный дизайн. Интересно, что пользователь может смотреть на свое приложение в реальном времени даже когда он проектирует и зарабатывает его.

С BiznessApps можно легко обновлять приложение и вносить столько изменений, сколько нужно, без необходимости выполнять длительный процесс публикации в Google Play или iTunes.

2 ПРОЦЕСС РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

2.1 Этапы разработки мобильных приложений

XXI век – эра расцвета мобильных технологий. Трудно сейчас даже вообразить человека без мобильного устройства, а уж человека-бизнеса тем более. Мобильные технологии присутствуют практически во всех сферах бизнеса. Мобильные экосистемы ежедневно меняются и развиваются на основе постоянных экспериментов. Ежедневно создаются сотни приложений для бизнеса, для образования, для развлечений и так далее. У каждого из этих приложений есть конкретный визуальный стиль и тон, в зависимости от направления и контекста приложения. Около 6140 мобильных приложений выходят в Google Play ежедневно. К 2020 году число релизов в App Store достигнет 5 миллионов. Статистика говорит об одном: пользователям необходимо разрабатывать свои мобильные приложения.



Разработка мобильного приложения «по наитию», как правило, превращает его в бесконечный долгострой с непредсказуемыми уходами в ненужные итерации, лишними действиями.

Напротив, грамотно проведенный этап проектирования позволит увидеть прозрачную картину предстоящей работы целиком и идти

не вслепую, а с четким пониманием что, зачем и для кого делается, а также, сколько это займет времени и каков объем требуемых инвестиций.

Разработка мобильного приложения с нуля и до релиза в среднем происходит за 9 основных этапов. Некоторые из них делаются за дни, а некоторые и через 3 месяца могут быть еще в процессе работы.

Некоторые этапы могут выполняться параллельно, а некоторые только последовательно.

Количество этапов в разработке приложения зависит от объема проекта и существующих вводных данных на него. Но в любом случае это – полноценная разработка самостоятельного ИТ-решения.

Ниже представлены этапы полного цикла разработки мобильного приложения от стадии зарождения идеи и до релиза [12].

- 1) Идея проекта
- 2) Техническое задание.
- 3) Прототипирование.
- 4) Дизайн мобильного приложения.
- 5) Разработка.
- 6) Тестирование.
- 7) Повторное тестирование.
- 8) Разработка иконок.
- 9) Размещение мобильного приложения в Appstore и Google.play.

Идея проекта. На самом начальном этапе необходимо тщательно продумать смысл будущего мобильного приложения и для чего оно будет использоваться.

Ниже приводятся некоторые вопросы, на которые следует обратить внимание перед публикацией приложения в одном из общедоступных магазинов приложений:

- **Конкурентные преимущества** – существуют ли аналогичные приложения? Если да, то чем данное приложение выгодно отличается от других?

Для приложений, распространяемых в корпоративной среде:

- **Интеграция с инфраструктурой** – в какую существующую инфраструктуру будет интегрироваться приложение, и какие дополнительные возможности оно привнесет?

Кроме того, приложения следует оценивать в контексте форм-фактора мобильного устройства:

- **Ценность** – какие ценностные преимущества даст пользователям приложение? Как они будут использовать его?

- **Форма и мобильность** – как это приложение будет работать с мобильными устройствами разных форм-факторов? Как повысить ценность приложения, используя различные мобильные технологии, включая службы определения местоположения, камеру и так далее?

Чтобы упростить проектирование функций приложения, рекомендуется определить субъекты и варианты использования. *Субъекты* – это роли в рамках приложения, которые часто соответствуют его пользователям. *Варианты использования* – это типовые действия или цели.

Например, в приложении для отслеживания задач можно определить два субъекта: *пользователь* и *друг*. Пользователь может *создавать задачу* и предоставлять другу *общий доступ к ней*. В этом случае можно выделить два варианта использования (создание задачи и предоставление общего доступа к ней), а также субъекты, на основании чего можно понять, какие экраны нужно создать и какие бизнес-сущности и логику требуется разработать.

После определения достаточного количества вариантов использования и субъектов разработчику будет проще приступить к проектированию приложения. Благодаря этому на стадии разработки можно сосредоточиться на том, как реализовать приложение, а не на том, что оно должно делать.

Затем необходимо определиться на какой платформе мобильное приложение будет использоваться. Как правило, сейчас приложения пишутся сразу для двух платформ IOS и Android или под две платформы сразу.

Техническое задание (ТЗ). Перед началом разработки необходимо получить ТЗ от заказчика. В случае если его нет, то заказчику дается бриф (краткая письменная форма согласительного порядка между планирующими сотрудничать сторонами) на заполнение. Данному этапу уделяется особое внимание, так как ТЗ непосредственно влияет на технические особенности результата. На данном этапе выполняются следующие виды работ:

- 1) составляется описание функционала мобильного приложения;
- 2) определяются и согласовываются сроки разработки;
- 3) рассчитываются финансовые затраты и вырабатывается модель порядка расчетов;
- 4) оформляется договор с заказчиком.

Прототипирование. Для того чтобы понять, как пользователь будет работать с мобильным приложением, создается графическая

карта взаимодействия между различными экранами программы. На этом этапе осуществляется проработка практически всего функционала мобильного приложения. На стадии проектирования UI специалисты определяют принцип работы приложения, размещение функций и кнопок на каждом из экранов. На этом этапе:

- 1) отрабатывается функционал приложения;
- 2) разрабатываются схемы экранов приложения;
- 3) продумывается связь экранов приложения и переходов по ним.

Дизайн мобильного приложения. На данном этапе создается дизайн всех экранов будущего приложения и отрисовываются различные состояния для всех сценариев пользования. После утверждения концепции дизайна отрисовываются внутренние кнопки и иконки, а также все остальные графические элементы. Как правило, отрисовка дизайна мобильного приложения предполагает юзабилити-исследования для того, чтобы убедиться, что разработанная дизайн-концепция максимально проста и удобна и поможет пользователям максимально быстро решать поставленные задачи.

Разработка. Разработчикам передается ТЗ и макеты дизайна приложения, и они приступают к реализации проекта. Программисты «трансформируют» статичную картинку в интерактивную рабочую модель. Выпускается первая версия приложения.

Тестирование. Мобильное приложение проходит тщательное тестирование, в результате чего создается таблица проверок, в которой указываются и подробно описываются все ошибки. В процессе проектирования приложения невозможно предусмотреть все погрешности реальной эксплуатации. На этом этапе формируется перечень ошибок, недочетов и недоработок приложения и определяются сроки на их устранение. Затем выпускается приложение с исправленными ошибками и, при необходимости, с измененным функционалом, что указывается в таблице после пробного тестирования.

Повторное тестирование. Приложение устанавливается на тестовые устройства, и работает точно так же, как если бы было скачано из Google Play или AppStore. Перед тем, как приложение появится в официальном каталоге программ, необходимо убедиться, что пользователи не столкнутся с ошибками разработки в процессе установки и применения программы. Поэтому на этой стадии еще раз проверяется логика продукта, работа его серверной части, приложение тестируется в самых разных условиях и на различных версиях операционных систем.

Разработка фирменных иконок. Иконка приложения – «лицо» любого современного мобильного приложения. Иконка мобильного приложения – это самостоятельный графический элемент. Создание данного элемента это многоэтапная процедура с отрисовкой изображения в нескольких размерах, а также ее тестирование на различных моделях устройств. После утверждения иконки мобильное приложение запускается.

Размещение мобильного приложения в AppStore и Google.Play. Приложение передается в AppStore и/или Google.Play для публикации. Каждое приложение перед публикацией проверяется командами Google и Apple. Публикация в Google.Play занимает не более суток, что конечно, значительно выигрывает по времени у AppStore, где публикация приложения занимает не менее 7 рабочих дней [12].

2.2 Особенности разработки интерфейсов для смартфонов. Принципы юзабилити

Силы, вложенные в разработку интерфейса, будут потрачены впустую, если разработчик не сумеет должным образом донести до пользователей принципы этого поведения. В случае мобильных продуктов, это делается визуальными средствами – путем отображения объектов на дисплее (в некоторых случаях целесообразно использовать тактильные ощущения от нажатия).

Визуальный *дизайн* интерфейсов – способен серьезно повлиять на эффективность и привлекательность продукта, но для полной реализации этого потенциала нужно не откладывать визуальный *дизайн* на потом, а сделать его одним из основных инструментов удовлетворения потребностей пользователей и бизнеса [21].

Дизайнеры создают объекты, которыми будут пользоваться другие люди. Если говорить о дизайнерах визуальных интерфейсов, то они ищут наилучшее представление, доносящее информацию о поведении программы, в проектировании которой они принимают участие. Они должны стремиться представлять поведение и информацию в понятном и полезном виде, который поддерживает маркетинговые цели организации и эмоциональные цели персонажей. Разумеется, визуальный дизайн пользовательских интерфейсов не исключает эстетических соображений, но такие соображения не должны выходить за рамки функционального каркаса.

2.2.1 Строительные блоки визуального дизайна интерфейсов

Дизайн интерфейсов сводится к вопросу о том, как оформить и расположить визуальные элементы таким образом, чтобы внятно отразить поведение и представить информацию. Каждый элемент визуальной композиции имеет ряд свойств, и сочетание этих свойств придает элементу смысл. Пользователь получает возможность разобраться в интерфейсе благодаря различным способам приложения этих свойств к каждому из элементов интерфейса. В тех случаях, когда два объекта обладают общими свойствами, пользователь предположит, что эти объекты связаны или похожи. Когда пользователи видят, что свойства отличаются, они предполагают, что объекты не связаны.

Создавая пользовательский интерфейс, необходимо проанализировать перечисленные ниже визуальные свойства каждого элемента или группы элементов. Чтобы создать полезный и привлекательный пользовательский интерфейс, следует тщательно поработать с каждым из этих свойств.

Форма

Форма – главный признак сущности объекта для человека. Мы узнаем объекты по контурам. Если увидим на картинке синий ананас, сразу его опознаем, потому что помним его форму. При этом различие форм требует большей концентрации внимания, чем анализ цвета или размера. Поэтому форма – не лучшее свойство для создания контраста, если требуется привлечь внимание пользователя.

Размер

Более крупные элементы привлекают больше внимания, особенно если они значительно превосходят размерами окружающие элементы. Люди автоматически упорядочивают объекты по размеру и склонны оценивать их по размеру; если у нас есть текст в четырех размерах, предполагается, что относительная важность текста растет вместе с размером и что полужирный текст более важен, чем текст с нормальным начертанием. Таким образом, размер – полезное свойство для обозначения информационных иерархий.

Цвет

Цветовые различия быстро привлекают внимание. В некоторых профессиональных областях цвета имеют конкретные значения, и этим можно пользоваться. Так, для бухгалтера красный цвет – отрицательные результаты, а черный – положительные.

Цвета приобретают смыслы и благодаря социальным контекстам, в которых проходит наше взросление. Например, белый цвет на Западе

ассоциируется с чистой и миром, а в Азии и арабских странах – с похоронами и смертью. При этом цвет изначально не обладает свойством упорядоченности и не выражается количественно, поэтому далеко не идеален для передачи информации такого рода. Кроме того, не следует делать цвет единственным способом передачи информации, поскольку цветовая слепота встречается довольно часто.

Чтобы создать эффективную визуальную систему, позволяющую пользователю выявлять сходства и различия объектов, используйте ограниченный набор цветов – эффект радуги перегружает восприятие пользователя и ограничивает возможности по передаче ему информации.

Яркость

Понятия темного и светлого обретают смысл преимущественно в контексте яркости фона. На темном фоне темный текст почти не виден, тогда как на светлом фоне он будет резко выделяться. Контрастность люди воспринимают легко и быстро, так что значение яркости может стать хорошим инструментом привлечения внимания к тем элементам, которые требуется подчеркнуть. Значение яркости – также упорядоченная переменная, например, более темные (с более низкой яркостью) цвета на карте легко интерпретируются: они обозначают большие глубины или большие значения других параметров.

Направление

Направление полезно, когда требуется передавать информацию об ориентации (вверх или вниз, вперед или назад). Помните, что восприятие направления может быть затруднено в случае некоторых форм и при малых размерах объектов, поэтому ее лучше использовать в качестве вторичного признака. Так, если требуется показать, что рынок акций пошел вниз, можно использовать направленную вниз стрелку красного цвета.

Текстура

Разумеется, изображенные на экране элементы не обладают настоящей текстурой, но способны создавать ее видимость. Текстура редко полезна для передачи различий или привлечения внимания, поскольку требует значительной концентрации на деталях. И, тем не менее, текстура может быть важной подсказкой. Засечки и выпуклости на элементах пользовательского интерфейса обычно указывают, что элемент можно перетаскивать, а фаски или тени у кнопки усиливают ощущение, что ее можно нажать.

Расположение

Расположение – это переменная, упорядоченная и выражаемая количественно, а значит, полезная для передачи иерархии. Расположение также может служить средством создания пространственных отношений между объектами на экране и объектами реального мира.

Расположение элементов мобильного приложения очень сильно влияет на удобство использования и зависит от того, как пользователь будет держать устройство [21].

2.2.2 Использование визуальных свойств для группировки элементов и создания четкой иерархии

Как правило, имеет смысл группировать логические наборы функциональных или информационных элементов посредством визуальных свойств, например, цвета или пространственных характеристик. Последовательно применяя эти визуальные свойства в интерфейсе можно создавать шаблонные образы, которые пользователи быстро научатся распознавать. Согласно инструкциям по проектированию дизайна мобильных приложений, обычные кнопки должны быть выпуклыми, со скругленными углами, а текстовые поля прямоугольными, обычно подчеркнутые и плоские, при этом активные элементы выделяются цветом. Благодаря систематическому применению этого образа невозможно перепутать кнопку и поле ввода, несмотря на некоторые сходства.

При создании иерархии необходимо определить, исходя из сценариев, какие функциональные и информационные элементы должны восприниматься пользователями сходу, какие являются вторичными, а какие нужны лишь в исключительных ситуациях. Такое ранжирование и служит основой для визуальной иерархии.

Чтобы создать видимые различия между уровнями иерархии, необходимо использовать цвет, насыщенность, контрастность, размер и положение. Самые важные элементы должны быть более крупными, более ярких цветов, более насыщенными и более контрастными. Их следует располагать над прочими элементами или делать выступающими. Менее важные элементы должны быть менее насыщенными, менее контрастными, более мелкими и плоскими. Нейтральные светлые цвета уводят их на второй план.

Настройку этих свойств необходимо выполнять осторожно. Не следует делать самый важный элемент огромным, красным и выпуклым. Часто бывает достаточно изменить лишь одно из свойств.

Чтобы передать связь элементов необходимо определить не только элементы со сходными функциями, но и элементы, наиболее часто используемые совместно. Совместно используемые элементы обычно следует сгруппировать в пространстве (поместить на отдельный экран), чтобы минимизировать перемещения между ними.

Пространственная группировка объясняет пользователям, каким образом одни задачи, данные и инструменты связаны с другими, и может намекать на правильную последовательность действий. Хорошая группировка посредством расположения принимает во внимание порядок задач и подзадач и движение взгляда по экрану.

Элементы, расположенные рядом, как правило, связаны друг с другом. Если необходимо создать группировку, удобно реализовывать ее посредством расстояний. Элементы, разделенные большими расстояниями, можно группировать посредством общих визуальных свойств.

Определившись с группами и визуальными особенностями этих групп, начинайте подстраивать контраст между группами – подчеркивая или, наоборот, затеняя группы сообразно их важности в текущем контексте.

Есть хороший способ убедиться, что визуальный дизайн эффективно задействует иерархию и отношения, – дизайнеры называют этот прием тестом с прищуриванием (squint test). Закройте один глаз и посмотрите на экран прищуренным вторым глазом. Обратите внимание на то, какие элементы слишком выпирают, какие стали нечеткими, а какие объединились в группы. Эта процедура часто вскрывает не замеченные ранее проблемы в композиции интерфейса.

2.2.3 Выравнивание

Выравнивание визуальных элементов – один из главных приемов, позволяющих дизайнеру представить продукт пользователям в систематизированном и упорядоченном виде. Сгруппированные элементы следует выравнивать как по горизонтали, так и по вертикали (рисунок 2.1).

В общем случае каждый элемент на экране следует выравнивать по максимально возможному числу других элементов. Отказ от выравнивания двух элементов или двух групп элементов должен быть осознанным: это допустимо только для достижения конкретного разделяющего эффекта.

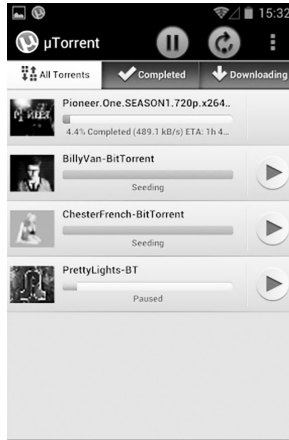


Рисунок 2.1 – Текст и функциональные элементы четко выровнены по сетке с фиксированным шагом

В числе прочего дизайнерам следует обращать внимание на:

- Выравнивание подписей. Подписи для элементов управления, расположенные друг над другом, должны быть выровнены по общей границе.
- Выравнивание внутри группы функциональных элементов. Группа связанных флажков, вариантов выбора или текстовых полей должна подчиняться выравниванию стандартной сетки.
- Выравнивание элементов, разнесенных по группам и панелям. Группы элементов управления и прочие объекты на экране везде, где это возможно, должны быть привязаны всё к той же сетке [22].

2.2.4 Сетка

Сетка – один из самых мощных инструментов визуального дизайнера. Сетка обеспечивает однородность и последовательность структуры композиции. После того как проектировщики взаимодействия определили общую инфраструктуру приложения и элементов его пользовательского интерфейса, дизайнеры интерфейса должны организовать композицию в структуру в виде сетки, которая будет должным образом подчеркивать важные элементы и структуры и оставлять жизненное пространство для менее важных элементов и элементов более низкого уровня.

Как правило, сетка делит экран на несколько крупных горизонтальных и вертикальных областей. Качественно спроектированная сетка задействует понятие шага, то есть минимального расстояния между элементами. К примеру, если шаг сетки составляет четыре пиксела, все расстояния между элементами и группами должны быть кратны четырем.

В идеальном случае сетка должна задавать и пропорции различных областей экрана. Такие отношения обычно выражаются дробями. Среди распространенных дробей – прославленное «золотое сечение» (равное примерно 1,62), которое часто встречается в природе и считается особенно приятным для человеческого глаза; величина, обратная квадратному корню из двух (примерно 1:1,41), которая является основой международного стандарта размера бумаги (например, листа А4). В программировании для мобильных устройств не следует полагаться на соотношение сторон дисплеев, так как для устройств, например на Android не существует единого стандарта размера экранов.

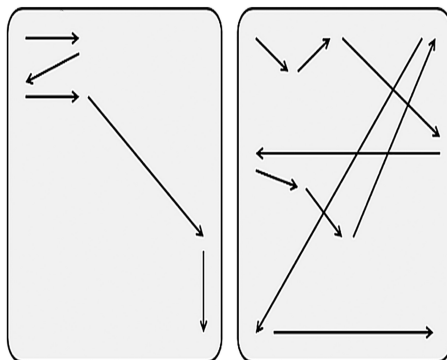
Использование сетки в визуальном дизайне интерфейсов дает ряд преимуществ:

- *Удобство применения.* Поскольку сетка делает расположение элементов единообразным, пользователи быстро приобретают навыки поиска нужных элементов в интерфейсе. Последовательность в расположении элементов и выборе расстояний между ними облегчает работу механизмов визуальной обработки в мозгу человека. Качественно спроектированная сетка упрощает восприятие экрана.
- *Эстетическая привлекательность.* Аккуратно применяя сетку и выбирая подходящие соотношения между различными областями экрана, дизайнер может создать ощущение порядка, который удобен пользователям и стимулирует их работу с продуктом.
- *Эффективность.* Создание сетки и включение ее в процесс на ранних этапах детализации проектных решений сокращает число итераций и действий по «доводке» интерфейса. Качественная и явно обозначенная сетка закладывает основу для легко модифицируемого и расширяемого дизайна, позволяя разработчикам находить хорошие композиционные решения.

2.2.5 Логические маршруты

Композиция должна не только в точности следовать сетке, но и структурировать эффективный логический маршрут через интерфейс

для пользователей, принимая во внимание тот факт, что (в случае западных языков) взгляд движется сверху вниз и слева направо (рисунок 2.2).



*Рисунок 2.2 – Слева представлен хороший логический маршрут: движение взгляда и маршрут в интерфейсе совпадают.
Справа – неудобный логический маршрут: все разбросано по экрану*

Симметрия – полезное средство организации интерфейса с точки зрения достижения визуального равновесия. Несимметричные интерфейсы обычно выглядят так, словно вот-вот завалятся на один бок. Опытные дизайнеры способны достигать асимметричного равновесия, управляя визуальным весом отдельных элементов. Тест с прищуриванием позволяет проверить сбалансированность интерфейса.

В интерфейсах чаще всего применяют два типа симметрии: вертикальная осевая симметрия (симметрия относительно вертикальной линии, проведенной через центр группы элементов) и диагональная осевая симметрия (симметрия относительно диагонали). В большинстве приложений присутствует симметрия одного из этих типов [22].

2.2.6 Текст в графических интерфейсах

Текст – неотъемлемая составляющая практических всех пользовательских интерфейсов. Следует крайне внимательно относиться к применению текста, поскольку он обладает способностью запутывать и усложнять простые вещи.

Человек распознает буквы исходя из их форм. Чем более узнаваема форма, тем проще распознать букву, поэтому СЛОВА, СОСТОЯЩИЕ

СПЛОШЬ ИЗ ЗАГЛАВНЫХ БУКВ, ЧИТАТЬ ТРУДНЕЕ, чем написанные обычным образом: в заглавных буквах отсутствуют привычные глазу начертания, и поэтому чтение требует большего внимания.

Распознавание отдельных слов – не то же самое, что чтение, при котором мы осознанно сканируем отдельные слова и интерпретируем их значения в контексте. Следует стараться минимизировать объем текста, который пользователю необходимо прочитать, чтобы сориентироваться в интерфейсе. Применение коротких простых слов облегчает навигацию при минимальной необходимости в чтении.

Что касается читаемого текста в интерфейсе, рекомендуется придерживаться некоторых принципов:

- *Используйте контрастный текст.* Убедитесь, что текст хорошо контрастирует с фоном и что не используются дополнительные цвета, способные повлиять на удобочитаемость текста.
- *Используйте подходящий шрифт и кегль (размер).* Как правило, шрифт без засечек и с резкими контурами, такой как Verdana илиTahoma, читается лучше всего. Текст мельче 10 пикселей в большинстве ситуаций трудно читать.
- *Четко формулируйте мысли.* Пользуйтесь минимальным количеством слов, необходимым для ясной передачи смысла. Избегайте сокращений. Если они все-таки необходимы, используйте общепринятые.

2.2.7 Цвет в графических интерфейсах

Цвет является составной частью визуального языка интерфейса. В большинстве приложений цвет должен использоваться сдержанно и хорошо сочетаться с прочими элементами визуального языка – символами, пиктограммами, текстом и пространственными отношениями между ними. Если не проявлять осторожность, цвет очень легко применить неправильно. Самые распространенные ошибки:

- *Слишком много цветов.* Добавление одного цвета, выделяющего важные элементы в наборе, значительно сокращает время поиска. Добавление новых цветов приводит к дополнительному ускорению работы пользователя, но при семи и более цветах скорость поиска значительно падает. Разумно предположить, что сходные результаты будут получены при любом типе навигации по интерфейсу, поскольку число семь отражает количество элементов информации, одновременно сохраняемой в кратковременной памяти человека.

- *Использование насыщенных дополнительных цветов.* Дополнительными являются цвета, противоположные друг другу в цветовом представлении. Если такие цвета обладают достаточно высокой насыщенностью и расположены рядом, то порождают зрительные эффекты, препятствующие легкому восприятию и мешающие сосредоточить внимание.
- *Чрезмерная насыщенность.* Сильно насыщенные цвета выглядят кричаще и привлекают слишком много внимания. Умеренное насыщение цвета допустимо для небольших областей, привлекающих внимание пользователей, но такие области всегда следует создавать с осторожностью.
- *Недостаточный контраст.* Когда цвет объекта отличается от цветов фона лишь оттенком, но не насыщенностью или яркостью, объект становится трудно воспринимать. Фигура и фон должны различаться по яркости и насыщенности, а не только по оттенку. Кроме того, необходимо избегать использования цветного текста на цветном фоне везде, где только возможно.
- *Недостаточная забота о людях с нарушениями цветового восприятия.* Примерно десять процентов мужского населения страдает цветовой слепотой той или иной степени. Это означает, что при использовании (в частности) оттенков красного и зеленого для передачи важной информации следует проявлять внимательность. Любые цвета, применяемые в интерфейсе, должны заметно различаться по насыщенности или яркости. Если интерфейс остается читаемым после преобразования в черно-белый вариант, люди с нарушениями цветового восприятия смогут работать и с цветным вариантом интерфейса [22].

2.2.8 Поведение окон и определение компоновки

Основными составляющими интерфейса приложений являются окна двух видов – главные и подчиненные. Выбор способа применения окон в программе – важный аспект общей инфраструктуры пользовательского интерфейса.

Исторически окна появились из аналогии с бумагами, которыми завален письменный стол. Когда необходимо использовать какой-то документ, человек кладет его поверх других. Эта концепция вполне логична, когда речь идет о большом экране. Однако для смартфонов следует учитывать, что *пользователь* в момент времени может увидеть *максимум* один документ, причем довольно в редких случаях целиком.

Если мы представим приложение в виде дома, то каждое окно – это отдельная комната. Сам дом будет представлен главным окном программы, а каждая комната – это окно документа, диалоговое окно или панель. Мы не пристраиваем к нашему дому комнату, если у нее нет уникальной функции, которую не способны выполнять другие комнаты. Точно так же не следует добавлять окно в программу, если у него нет предназначения, которое не могут или не должны выполнять существующие окна.

Важно подходить к вопросам предназначения с позиции будущих целей и пользовательских ментальных моделей. Думая о том, каково предназначение комнаты, подразумеваем определенную цель – не обязательно конкретную задачу или функцию.

Диалоговое окно – это еще одна комната, в которую не надо входить без веской причины.

Это один из наиболее часто нарушаемых принципов в проектировании пользовательских интерфейсов. Программисты выполняют свою работу, разбивая приложение на дискретные функции, при этом пользовательский интерфейс часто разрабатывается параллельно. И получается закономерный результат: по диалоговому окну на каждую функцию. Это сильно усложняет интерфейс.

Поэтому надо, чтобы разрабатываемое приложение как можно лучше отвечало правилу *трех кликов*. Иными словами, до любого элемента программы можно добраться не более чем за три действия.

Некоторые проектировщики считают, что каждое диалоговое окно должно вмещать единственную функцию. Результатом такого подхода становится замусоривание интерфейса окнами.

Решение многих пользовательских задач требует выполнения последовательности действий. Если для каждого действия открывается отдельное диалоговое окно, то экран вскоре визуальнo перегружается, а навигация усложняется. Каждое новое окно взваливает на плечи пользователя дополнительный интерфейсный налог, связанный с управлением окнами. При ежедневном использовании программы совокупный размер этих налогов может увеличиться до невероятных размеров.

2.2.9 Проектирование для различных потребностей

Персонажи и сценарии помогают сосредотачивать усилия проектирования на целях, поведении, потребностях и ментальных моделях реальных пользователей. Существуют также последовательные и

доступные для обобщения шаблоны пользовательских потребностей, которые следует учитывать при проектировании продуктов. Рассмотрим некоторые стратегии удовлетворения этих распространенных потребностей.

Командные векторы, рабочие наборы и персонажи

При классификации потребностей пользователей с различным уровнем подготовки весьма полезны понятия командных векторов и рабочих наборов.

Командные векторы – это самостоятельные методы, которыми пользователи могут давать команды программе. Визуальные элементы для непосредственного манипулирования, раскрывающиеся и контекстные меню, элементы управления на панели инструментов – все это примеры командных векторов.

Хороший пользовательский интерфейс предоставляет пользователю множественные командные векторы, в которых важные функции приложения представлены в разных формах, чтобы обеспечить пользователя параллельными возможностями управлять приложением. Например, многие приложения работы с картами позволяют изменять масштаб, как с помощью специальных кнопок, так и с помощью специальных жестов (рисунок 2.3).

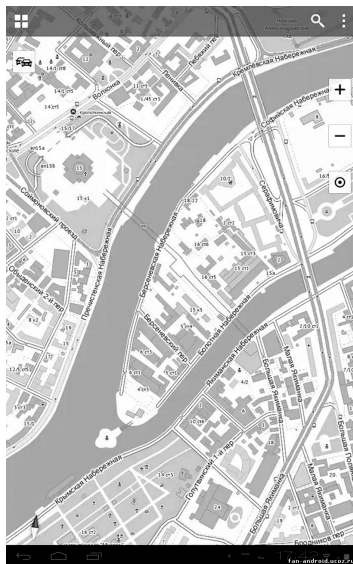


Рисунок 2.3 – Изменение масштаба изображения

На экране есть специальные кнопки; в то же время опытные пользователи сенсорных экранов ими практически не пользуются, предпочитая одновременное сведение/разведение двух пальцев (один из жестов технологии мультитач).

Среди командных векторов есть более удобные для новичков. Как правило, максимальную поддержку оказывают меню и диалоговые окна, и поэтому их называют **обучающими векторами**. Начинающие пользователи выигрывают от обучающих свойств меню, когда им необходимо сориентироваться в незнакомой программе.

Поскольку в памяти каждого пользователя невольно откладываются часто используемые команды, многие запоминают некоторое подмножество команд, которые называются **рабочим набором**. У каждого пользователя свой рабочий набор, хотя весьма вероятно, что он значительно пересекается с рабочими наборами других пользователей.

Не существует стандартного рабочего набора, покрывающего потребности всех пользователей.

Команды, составляющие рабочий набор любого пользователя – это наиболее часто используемые команды. Пользователь хочет, чтобы доступ к ним был особенно быстрым и простым. Проектировщик обязан предоставить незамедлительные командные векторы для минимального рабочего набора наиболее вероятных пользователей данного приложения. Кроме того, это облегчает освоение новичками программного продукта.

Существует исключение из правила множественных векторов. Опасные команды (такие как *Стереть все*) не должны иметь легко доступных множественных командных векторов. Их следует защищать посредством меню и диалоговых окон.

Персонализация и настройка

Проектировщикам мобильных интерфейсов часто приходится принимать решение относительно того, давать ли пользователю настраивать продукт сообразно своим предпочтениям.

Люди любят изменять окружение по своему вкусу. Даже новички любят ставить на программу свое клеймо, настраивая ее так, чтобы она выглядела и вела себя в соответствии с их предпочтениями и личным вкусом. Люди делают это по той причине, что это придает им индивидуальность.

Персонализация – украшение стабильных объектов. Персонализация делает программные продукты более симпатичными и узна-

ваемыми, более человечными и приятными для работы, становится полезным средством навигации.

С другой стороны, от перемещения стабильных объектов навигация может пострадать. Украшение стабильных объектов способствует навигации, а перемещение их – препятствует.

Настройкой называется перемещение, добавление и удаление стабильных объектов. Настройка желательна для более опытных пользователей, которые, определившись со своим рабочим набором функций, захотят настроить интерфейс таким образом, чтобы эти функции было легче находить и выполнять.

Инструменты персонализации должны быть простыми и удобными в обращении, давая пользователю возможность предварительного просмотра результатов его действий. Кроме того, действия персонализации должны легко отменяться.

Большинство конечных пользователей не будут протестовать, если им не предоставят возможность настраивать программу. Некоторые действительно опытные пользователи, возможно, почувствуют себя обделенными, но они, тем не менее, будут работать с программой и ценить ее по достоинству, если она работает так, как они того ожидают. Однако в некоторых случаях гибкость абсолютно необходима. Если проектируется продукт для быстро эволюционирующих рабочих процессов, становится чрезвычайно важным, чтобы этот продукт мог эволюционировать с такой же скоростью, как среда его применения.

Пользовательское тестирование неоднократно демонстрировало, как аудитория расходится во взглядах на эффективность той или иной идиомы, разделяясь на две примерно равные части. Половина пользователей явно предпочитает одну идиому, а вторая половина – другую. Такое четкое разделение аудитории по предпочтениям на две или несколько крупных групп указывает, что эти предпочтения *идиосинкратически модальны*.

В таких случаях не нужно устраивать дебаты, чтобы понять, какой из способов следует применять, потому что они легко превращаются в холивар (священную войну). Здесь ответ очевиден: надо применять оба. Когда пользовательская аудитория разделена предпочтениями относительно идиом, проектировщики обязаны предложить обе идиомы. Необходимо удовлетворить обе группы. Нельзя удовлетворить одну половину аудитории, оставив в негодование другую, и при этом совершенно неважно, к какой группе относят себя разработчики [22].

Локализация и глобализация

Создание приложений для различных языков и культур ставит перед проектировщиками ряд специфических проблем. И здесь снова хорошим ориентиром могут служить командные векторы. Незамедлительные векторы, такие как непосредственное манипулирование и кнопки-значки на панели инструментов, являются идиоматическими и визуальными, поэтому они без особых сложностей поддаются глобализации. Конечно, проектировщикам следует проделать подготовительную работу, чтобы убедиться, что выбранные для этих идиом цвета и символы не имеют в других культурах какого-то особого смысла, не заложенного разработчиком.

Обучающие векторы, такие как пункты меню, подписи к полям ввода, всплывающие подсказки и инструкции, зависят от языка и, таким образом, являются объектом локализации, осуществляемой посредством перевода. При создании интерфейсов, подлежащих локализации, следует принять во внимание следующие моменты:

- В некоторых языках слова и фразы длиннее, чем в других.
- В азиатских языках может вызвать сложности алфавитная сортировка.
- Формат вывода даты и соглашения о применении 12- и 24-часовых систем счисления времени в разных странах различны.
- Символы, отделяющие десятичную часть от целой в числах и триады в денежных суммах, могут быть разными.
- В ряде стран принято нумеровать недели в году. Кроме того, не во всех странах придерживаются григорианского календаря.

К переводу пунктов меню и сообщений в диалоговых окнах следует подходить целостно. Важно убедиться, что в результате перевода интерфейсы не утратили связности. Переведенные в отрыве от контекста пункты меню и метки могут вызывать путаницу, когда окажутся рядом с другими независимо переведенными элементами интерфейса. Семантику интерфейса следует сохранять на всех уровнях [22].

3 РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ В СРЕДЕ PROCESSING

3.1 Основы работы в Processing

Начнем с очень своеобразного вопроса. Кто умнее: человек или компьютер?

Много кто ответит, что компьютер. Ведь он может так быстро считать и обрабатывать информацию. Быстрее, чем многие люди. И они будут не правы. Человек умнее. И вот почему.

Предположим, Ваш друг хочет попросить заказать домой пиццу Вас или свой компьютер/смартфон. Кто лучше справиться с задачей? Конечно, Вы. Ведь Вы знаете, какую пиццу предпочитает друг. Вы можете решить в какую пиццерию лучше позвонить или оставить заказ на сайте. А что компьютер? Компьютер просто ничего не делает. Для него эта задача не *формализована*, а догадываться и придумывать сам он не может. По крайней мере, пока искусственный интеллект не достигнет человеческих высот и не станет повсеместным.

К чему это все? Любая программа для компьютера, мобильного телефона, смартфона должна быть очень четкой, без необходимости какого-то домысливания. Чем-то похоже на инструкции, только еще более подробной.

Делаем вывод: *программа* – это, проще говоря, инструкция для компьютера/смартфона, что ему нужно делать.

Следующий шаг – понять, а где же эту программу-инструкцию составить. Для этого используют *специальные программы*, которые называются *«среда разработки»* или английским сокращением IDE.

Для создания мобильных приложений существуют огромное количество различных сред разработок. Самым распространенным среди Android-разработчиков является Android Studio. Однако для Android Studio требуется довольно много вычислительных мощностей. А значит, нужен хороший мощный компьютер для разработки. К сожалению, пользователи не всегда имеют самые лучшие ПК. Поэтому полноценно работать на них не получится.

Исходя из этого, среда разработки должна быть не требовательной к вычислительным характеристикам компьютера. Конечно же, она должна быть с недорогой или даже бесплатной лицензией. А, в идеале,

еще и поддерживать несколько разных языков для проведения различных предметов и дисциплин.

Всем этим требованиям отвечает среда *разработки Processing*. Это – открытый проект (open-source) начатый в 2001 году Бенжамином Фраем и Кэйси Ризом в MIT Media Lab. Лицензии покупать не требуется. Эту среду даже нет необходимости устанавливать на компьютер. Достаточно просто скачать с официального сайта и разархивировать папку с файлами (подробнее будет описано ниже). И, что тоже не маловажно, в ней можно писать код и на компьютер, и на мобильный телефон на операционной системе Android, а еще и осваивать программирование на Python. И много чего еще.

Processing – открытый язык программирования, основанный на Java. Представляет собой лёгкий и быстрый инструмент для людей, которые хотят программировать изображения, анимацию и интерфейсы. Используется студентами, художниками, дизайнерами, исследователями и любителями, для изучения, прототипирования и производства.

К тому же Processing стал основой для разработки и других IDE, таких как Arduino, Wiring и p5.js.

Если говорить именно о мобильной разработке, то создание и загрузка готовых приложений на смартфоны на Android системе происходят очень быстро и легко.

В итоге мы – авторы, считаем именно эту среду наиболее подходящей для обучения мобильной разработке в рамках вуза или, возможно даже, спецкурса в школе.

Хотя все мы понимаем, что прогресс не стоит на месте. Тем более в сфере программирования и ИТ. И вполне вероятно, через несколько лет появится еще более удобная среда разработки.

3.2 Загрузка и знакомство со средой Processing

Первый и основной источник, который пригодится разработчику, осваивающему Processing, – официальный сайт самой среды <https://processing.org/>.

Здесь можно и скачать саму IDE, и получить краткую справку о всех системных командах и переменных с примерами, и посоветоваться с коллегами по разным вопросам на Форуме. Единственное ограничение – все это на английском языке.

Начнем с самого начала.

Заходим на официальный сайт во вкладку «Download». Здесь даны ссылки на скачивание среды для разных операционных систем. Нажимаем на нужную ссылку и скачиваем архивную папку.

Оффтоп. Если на ПК установлена ОС Windows и пользователь не знает какую версию выбрать – 32-разрядную или 64-разрядную, то необходимо найти на рабочем столе или в папке иконку «Этот компьютер» или «Мой компьютер» и нажать правой кнопкой мыши на него вызывать свойства, как показано на рисунке 3.1.

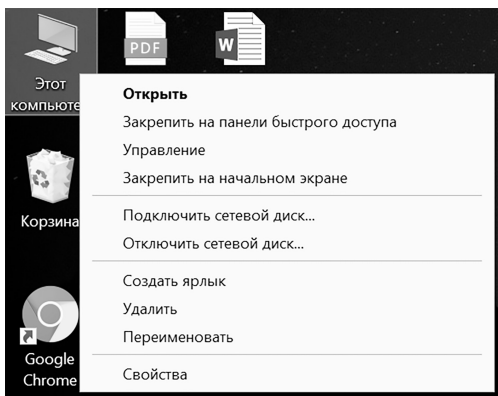


Рисунок 3.1 – Вызов свойства компьютера

В свойствах описан тип системы, где можно прочитать 32-разрядная она или 64-разрядная.

После загрузки разархивируем папку в том месте, где удобно было бы использовать среду разработки Processing (она не устанавливается и не обязательно должна быть в папке Program Files). Установка закончена.

Важно!!! Одно условие корректной работы среды Processing – адрес местонахождения папки должен быть полностью на латинице. Если в адресе есть хотя бы одна русская буква, могут возникнуть проблемы при загрузке приложения на телефон. В том числе это касается и имени пользователя.

Теперь достаточно просто дважды нажать на иконку Processing.exe (рисунок 3.2) в загруженной папке и откроется окно среды разработки (рисунок 3.3).



processing.exe

Рисунок 3.2 – Иконка Processing.exe

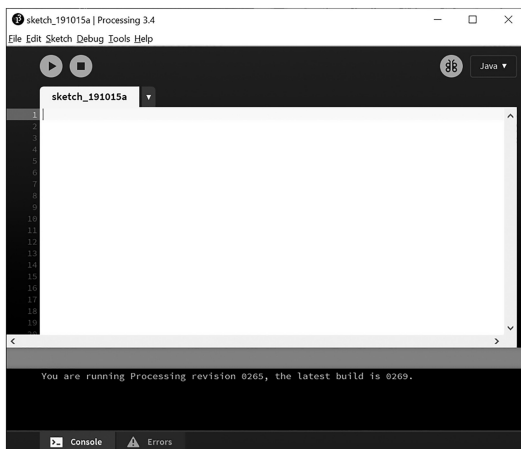


Рисунок 3.3 – Окно среды разработки Processing

Основную часть экрана занимает «чистый лист» для будущей программы или скетча. Для удобства навигации в будущем каждая строчка кода пронумерована с левой стороны. Выше находится вкладка с названием скетча, которая формируется по умолчанию при запуске и может меняться при сохранении в будущем. Еще выше слева находятся кнопки RUN и STOP. Или по-русски – запуска и остановки программы. Справа от них находятся еще две кнопки. Одна – DEBUG – включает режим поиска ошибок. Вторая – с подписью JAVA – позволяет переключаться на различные режимы программирования в среде. По умолчанию в Processing стоит режим Java, но далее мы рассмотрим подробно, как можно добавить новый режим на примере режима Android.

Еще выше находятся классические вкладки File (работа с файлами), Edit (инструменты для редактирования), Help (вызов различных справочных материалов). Из относительно специфических режимов добавлены Sketch (работа со скетчами), Debug (работа с режимом поиска ошибок), Tools (дополнительные инструменты).

Чуть ниже скетча находится серая полоса, на которой IDE будет иногда выводить какие-то оперативные подсказки или «возмущаться» каким-то непонятным для нее частям кода. Еще ниже находится черный экран с двумя вкладками: *Console*, в которой будет выводиться описание выполняемых процессов при запуске программы или поиске в ней ошибок. И *Errors*, в которой будут описаны найденные ошибки и несоответствия в коде.

Вызов функции установка

Прежде чем перейти непосредственно к коду немного разберем теорию и каждый элемент кода в отдельности.

Любой человек, поработавший с компьютерами на уровне хотя бы продвинутого пользователя, знает, что работа с программой начинается с Setup или установить. Так и выполнение программы в Processing начинается с вызова функции установки или «точки входа» или с ключевых слов ***void setup()*** в окне скетча.

Оффтоп. Ключевое слово ***void*** появилось еще в языке программирования C, после чего перешло «по наследству» в ряд других языков, в том числе и в Java. Переводится с английского оно как «пустота», «пробел», «вакуум». По сути, означает функцию, после выполнения, которой не остается каких-то значений для использования в других функциях. Если же функция возвращает какое-то значение, то вместо слова ***void*** пишется тип переменной, которую она возвращает.

Обратите внимание на два вида скобок, написанных сразу после слова ***setup***. Как и в большинстве других сред и языков программирования за каждым знаком или скобкой закреплена только одна задача или функция. Так, круглые скобки используются для перечисления свойств или параметров, и они всегда сопровождают как функцию (в случае ***setup***), так и команду. Фигурные скобки несут другую функцию. В них перечисляются команды или действия. Они встречаются как у ***void***, так и у других элементов программы.

Важно!!! Всегда отслеживайте, чтобы каждая открытая скобка была закрыта парной скобкой. Желательно в нужном месте. Иначе программа не будет выполнять код.

Важно!!! Функция установки, точка входа или `void setup()` выполняется только один раз при запуске программы, а значит все прописанные внутри действия тоже.

Далее. Одно из первых действий при запуске программы будет задание размера ее окна. Если этого не сделать, то размеры окна по умолчанию будут 100 на 100 пикселей. Это действие надо прописать в фигурных скобках функции.

Задать размер экрана можно с помощью команды `size()`, в которой задаются не менее двух параметров в виде натуральных чисел или переменных. При мобильной разработке, в отличие от компьютерных программ, принято, чтобы приложение занимало весь экран смартфона или планшета. Но размеры и расширение экрана на разных устройствах различаются, поэтому прописывать конкретные значения будет неправильно. Они могут подходить под одно устройство, но не соответствовать другому. В таком случае помогают две системные переменные `displayWidth` и `displayHeight`. При запуске программы они считывают размеры экрана устройства в виде натуральных чисел и могут быть сразу использованы.

Важно!!! Обращайте внимание на прописные и строчные буквы. Если в них ошибиться, среда просто «не признает» функцию, команду или переменную и подчеркнет, подчеркнув это слово красным цветом. В случае «зарезервированных» или системных функций, команд или переменных среда выделяет их цветом. Жирным и синим – для системных функций, просто синим – для команд, и красным – для системных переменных. Есть и исключения. Например, для ключевых слов, относящихся к дополнительному режиму разработки. Их система узнает и не подчеркивает, но и не выделяет цветом. Например, команда `orientation()` с параметрами `LANDSCAPE` или `PORTRAIT`, которая блокирует автоматический поворот экрана на смартфоне или планшете относится к командам режима Android и не подсвечивается цветом.

Теперь программа выглядит следующим образом:

```
void setup(){
  size(displayWidth, displayHeight);
}
```

Обратите внимание на три особенности. Во-первых, перечисляя параметры необходимо использовать запятую. В этой среде разработки она как раз и зарезервирована за перечислением параметров в круглых скобках. Во-вторых, после команды стоит точка с запятой. В этой среде и в большинстве других языков программирования точка с запятой означает окончание действия или команды. Точку ставить нельзя, так как точка уже зарезервирована для написания дробных чисел и не может быть использована в других целях.

***Важно!!!** Не забывайте ни про одну запятую или точку с запятой. Практически во всех средах разработки и языках программирования забытый знак препинания не даст выполнить код.*

В-третьих, надо обратить внимание на отступы и форматирование кода. Своеобразная «красная строка» означает равенство строки налево, а не направо. Кроме этого, закрывающая фигурная скобка также находится с правой стороны. А вот весь внутренний код в **void** написан со сдвигом вправо, где каждое законченное действие находится на отдельной строке.

Правильное форматирование не требование самой среды. Если есть все необходимые знаки препинания и скобки, она выполняет код. Форматирование помогает именно человеку лучше понимать и воспринимать код программы. Это пригодится при работе над совместным проектом или проведении code review.

Оффтоп. Термин code review используется у программистов и означает проверку части кода одним или несколькими коллегами перед его полноценным запуском или «релизом». В ходе такого code review могут находиться различные ошибки или «баги», оптимизироваться код или даже просто корректироваться форматирование.

Итак, у нас уже есть функция начала программы, в которой задан размер экрана, подстраивающийся под размеры смартфона, планшета или компьютера. Теперь надо начать прорисовку самого интерфейса приложения. Такая прорисовка выполняется в еще одной системной функции **void draw(){}.** Согласно принятому форматированию между функциями принято оставлять отступ.

```
void setup(){  
    size(displayWidth, displayHeight);  
}
```

```
void draw(){  
}
```

Важно!!! Функция **void draw(){}** выполняется по циклу сразу после завершения последней своей команды, а это значит, что все прописанные внутри действия тоже. Но существует возможность этот цикл остановить. Например, с помощью команды **noLoop()**, а потом снова запустить с помощью команды **loop()**.

Теперь используем ряд команд для прорисовки четырех основных плоских объектов на интерфейсе: *отрезок, прямоугольника, эллипса и треугольника*. Начнем с линии.

Для того, чтобы нарисовать отрезок, надо задать две точки. В среде Processing любая точка задается через координаты. Если создаются плоские фигуры, то для одной точки задаются две координаты: одна – по X или по ширине, другая по – по Y или по высоте.

Важно!!! В среде Processing, в отличие от классического подхода в геометрии, точка начала координат находится в левом верхнем углу экрана. Это очень важно учитывать при работе с размерами и местоположением.

Для прорисовки отрезка используется команда **line()**, в которой задается четыре параметра: первые два – местонахождение точки начала по ширине и высоте, третья и четвертая – местонахождение конечной точки по ширине и высоте. Если рисуется линия в объеме, то для каждой точки добавляется еще третий параметр по глубине.

```
line(500, 500, 1000, 1000);
```

Прорисовка прямоугольника. В команде **rect()** необходимо задавать не менее четырех параметров.

```
rect(500, 500, 200, 200);
```

Первые два – это местонахождение прямоугольника по ширине и высоте. А именно: левого верхнего угла прямоугольника. Следующие два – это ширина и высота прямоугольника. Этой же командой можно

нарисовать и квадрат. Для этого достаточно задать один размер и на ширину, и на высоту прямоугольника. Можно задать и больше параметров. Например, добавив пятый параметр можно сгладить углы. А добавив еще три, настроить сглаживание углов для каждого угла в отдельности.

Ellipse во многом похож на прямоугольник. Он рисуется с помощью команды ***ellipse()***, в которой также задаются четыре параметра.

```
ellipse(1000, 1000, 400, 200);
```

Отличие лишь в том, что первые два параметра местонахождения определяют не левый верхний угол, которого у эллипса нет, а центр эллипса. Третий и четвертый параметр задает диаметр эллипса по ширине и высоте. И, конечно, если диаметры задать одинаковые. Получается круг.

В команде для *рисования треугольника* ***triangle()*** параметров больше – целых шесть.

```
triangle(1000, 500, 750, 1000, 1250, 1000);
```

Все потому, что рисование треугольника требует соединения трех точек. Каждая точка задается двумя параметрами на плоскости или тремя в объеме.

Для полной наглядности предлагаем немного скорректировать параметры местоположения и размеров, а также порядок прорисовки объектов. В итоге получим примерно следующий код:

```
void setup(){  
  size(displayWidth, displayHeight);  
}  
  
void draw(){  
  rect(displayWidth-500, displayHeight-500, 200, 200);  
  ellipse(500, displayHeight-500, 400, 200);  
  line(500,displayHeight-500,displayWidth-500, displayHeight-500);  
  triangle(displayWidth/2, displayHeight/2-200, displayWidth/2-100,  
displayHeight/2, displayWidth/2+100, displayHeight/2);  
}
```


Обратите внимание, что в параметрах можно не только задавать числа, но использовать системные переменные и делать математические вычисления. При необходимости, внутри команды может быть прописана другая команда. Кроме этого в среде Processing, почти, как и в любом другом случае программирования, принципиально важен порядок написания команд. Именно поэтому линию надо перенести после прямоугольника и круга, иначе другие объекты ее просто перекроют, и ее не будет видно.

При запуске полученного кода с помощью кнопки RUN, среда открывает вот такой экран – рисунок 3.4.

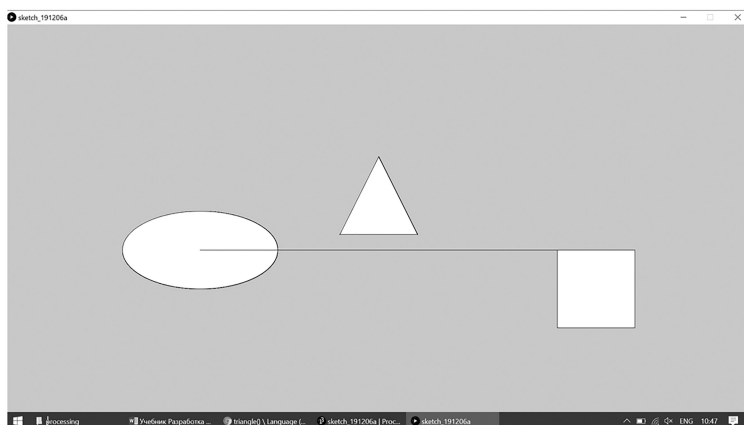


Рисунок 3.4 – Прорисовка базовых фигур в среде Processing

На этом рисунке по нарисованной линии хорошо видно, что местонахождение прямоугольника и эллипса задается для разных точек.

Приведенные выше примеры рассматривались в среде разработки и классическом программировании. Теперь рассмотрим разработку для мобильного приложения.

3.3 Функции разработчика на мобильном телефоне

Начнем с того, что операционных систем для мобильных телефонов, смартфонов и планшетов довольно много. Каждая имеет свои особенности, требования к приложениям и даже языки программирования.

Как мы отмечали, выше одной из самых распространенных операционных систем на данный момент является Android. Это относится и к iOS, но у нее много особенностей в работе и разработке приложений, которые заслуживают отдельного рассмотрения. Кроме того, iOS используется только для устройств фирмы Apple. А Android работает на смартфонах и планшетах большого числа компаний. Поэтому в дальнейшем речь пойдет именно о ней.

Оффтоп. Операционная система Android разработана на основе ядра Linux. Изначально разрабатывалась одноименной компанией, которая была куплена Google. Большинство приложений под эту операционную систему писались на языке Java, но сейчас идет постепенный переход к более новому языку Kotlin.

Любая загрузка программы на смартфон или планшет начинается с трех шагов: включения режима Android в среде разработки, разрешение загрузки приложения в режиме разработчика с компьютера на самом смартфоне и непосредственно сама загрузка на телефон.

Начнем с включения режима Android в Processing. В правом верхнем углу IDE надо найти клавишу с надписью Java. Надпись на этой кнопке как раз и показывает включенный в данный момент режим. Если нажать на нее и в появившейся вкладке выбрать «Add Mode...» или добавить режим, то появится окно «Contribution Manager» или «Менеджер Дополнений» (рисунок 3.5).

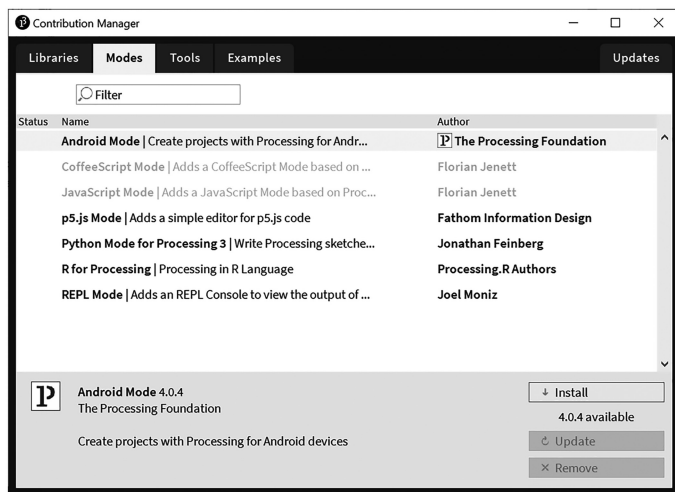


Рисунок 3.5 – Contribution Manager или Менеджер Дополнений

Через него можно подгрузить как дополнительные режимы программирования, так и различные библиотеки, инструменты и примеры. Processing – открытый проект, и значит такие дополнения могут быть как от базового разработчика – The Processing Foundation, так и конкретного участника сообщества Processing. При желании разработчики сами потом смогут создавать и добавлять инструменты, библиотеки или даже целые режимы для этой среды.

В Менеджере Дополнений в первую очередь нас интересует режим Android Mode. Выбираем его, и с помощью кнопки Install справа снизу загружаем этот режим в IDE. После чего можно выключить окно Менеджера до другого раза.

Далее, в основном окне справа сверху необходимо поменять режим Java на режим Android. Всплывет окно о необходимости загрузки SDK – рисунок 3.6.



Рисунок 3.6 – Среда не может найти SDK

Вероятнее всего, SDK на компьютере нет. Поэтому правильнее будет загрузить SDK автоматически (левая кнопка в окне). После загрузки прочитайте лицензию на Android SDK и согласитесь с условиями – рисунок 3.7.

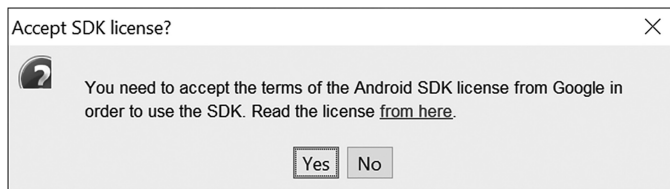


Рисунок 3.7 – Согласие с лицензионными условиями на Android SDK от Google

Оффтоп. SDK – Software Development Kit – средства разработки для определённого пакета программ, аппаратной платформы, операционных систем и так далее. SDK обычно разрабатывается самим разработчиком этой платформы. Android SDK – средство разработки приложений для любых устройств, работающих на операционной системе Android. А это не только смартфоны и планшеты, но и другие устройства, начиная с умных очков и часов, до телевизоров и даже автомобилей.

Теперь, когда среда разработки готова, можно переходить непосредственно к смартфону.

Важно!!! Обратите внимание, что включение подменю «Для разработчиков» для разных моделей и компаний производителей смартфонов и планшетов может отличаться. Поэтому, если инструкция ниже оказалась не подходящей, найдите в Интернете инструкцию непосредственно по Вашей модели и компании производителя.

На смартфоне надо зайти в меню «Настройки» и выбрать раздел «О телефоне (О планшете)» (рисунок 3.8).

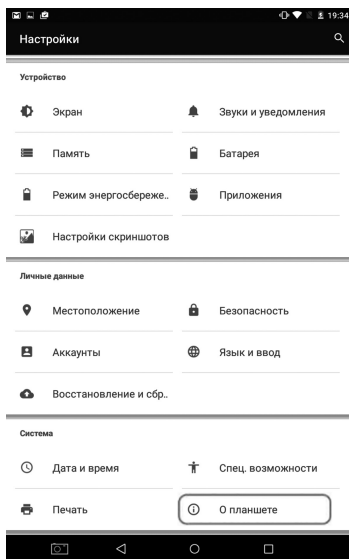


Рисунок 3.8 – Меню «Настройки» на планшете

В этом разделе надо найти строчку «Номер сборки» и нажать на нее семь раз (рисунок 3.9).

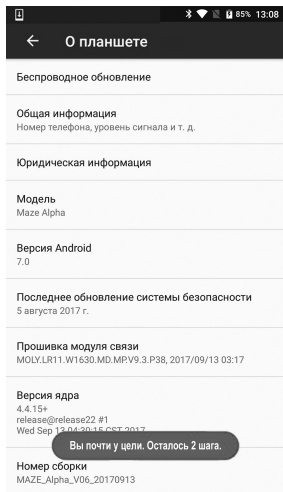


Рисунок 3.9 – Подменю «О планшете»

После седьмого нажатия система сообщит, что Вы стали разработчиком (рисунок 3.10).

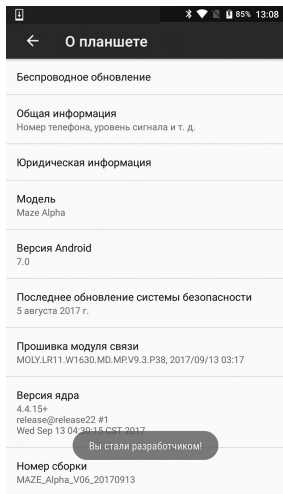


Рисунок 3.10 – Включение на планшете подменю «Для разработчиков»

Теперь можно в любой момент найти подмену «Для разработчиков» в настройках смартфона. Например, в Android 6.0 пункт размещен в категории «Система».

3.4 Написание программы на мобильном телефоне

Через вкладку «Файл» необходимо открыть новый скетч и начать написание функции начала и заданию размеров приложения:

```
void setup(){  
  size(displayWidth, displayHeight);  
}
```

Далее, в функции рисования интерфейса написать команду для эллипса и нарисовать круг:

```
void draw(){  
  ellipse(500, 500, 20, 20);  
}
```

А теперь применим две системные переменные *mouseX*, *mouseY* для местоположения этого круга:

```
void draw(){  
  ellipse(mouseX, mouseY, 20, 20);  
}
```

Переменные *mouseX*, *mouseY* отслеживают, где находится компьютерная мышь по ширине и высоте и возвращает эти значения в код, там, где они используются.

Далее необходимо запустить программу в режиме Java (рисунок 3.11).

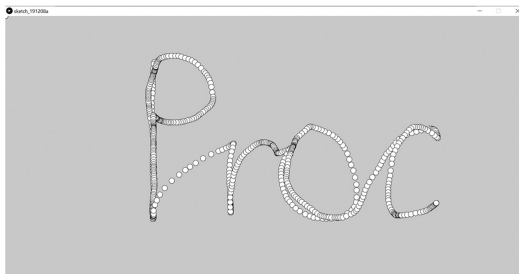


Рисунок 3.11 – Маленькая программа для рисования

При использовании этих переменных в мобильных приложениях они отслеживают не мышь, а палец, нажимающий на сенсорный экран. Поэтому рисунок можно сделать более прерывистым.

Теперь займемся загрузкой этой программы на смартфон или планшет. Для этого сначала сохраняем скетч, желательно поменяв его название. Потому что именно с этим названием приложение будет отражаться на телефоне. Переключаем на режим Android.

Подключаем смартфон по USB-кабелю к компьютеру. Теперь в меню «Режим разработчика» надо включить «Отладка по USB» и «Активный режим» (не отключать экран при зарядке). Все дело в том, что если при загрузке приложения телефон заблокируется, то связь с компьютером прервется и компиляция остановится из-за ошибки связи.

Оффтоп. Когда не загружается приложение на устройство желательно выключить «Отладка по USB», чтобы обезопасить свой телефон от загрузки вредительских приложений при подключении в другие разы или к другим компьютерам.

После разрешения на отладку по USB в выпадающем сообщении смартфона или планшета надо переключиться с режима подключения «Зарядка» на «Передача файлов мультимедиа» или MTP.

Когда смартфон и компьютер «увидят» друг друга на экране смартфона появится всплывающее окно с вопросом: разрешить ли отладку USB с этого компьютера? Можно в этом окне поставить галочку на «Всегда разрешать отладку с этого компьютера» или разрешить только на этот раз. Теперь телефон готов к загрузке на него приложения из Processing.

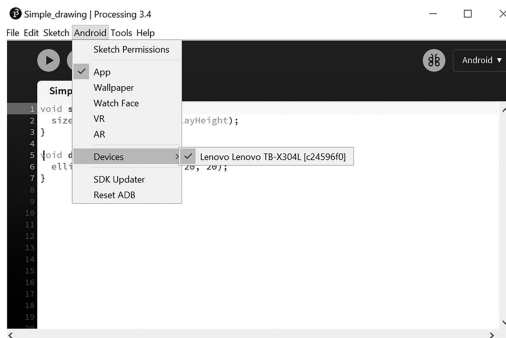



Рисунок 3.12 – Среда разработки полностью подключилась к планшету и готова к компиляции и загрузке

Последний шаг – убедиться, что среда разработки «видит» устройство. Для этого в меню «Android», подменю «Devices» необходимо убедиться, что смартфон появился с галочкой подключения (рисунок 3.12). Иногда это занимает какое-то время.

Важно!!! Убедитесь, что, при первом запуске приложения на смартфоне, компьютер подключен к Интернету, потому что для компиляции среда разработки подгружает необходимые программные инструменты из Интернет-ресурсов Android SDK, Java и Processing.

Далее необходимо нажать кнопку  – «Run on device» и дождаться когда приложение откроется на телефоне. Этот процесс довольно долгий.

Важно!!! Именно из-за долгой загрузки проверка работы приложения при доработке сразу на устройстве не выгодна с точки зрения существенной потери времени. Но желательно все равно проверять работу приложения на смартфоне регулярно, потому что ее работа может критично отличаться не только на компьютере и смартфоне, но и на устройствах разных моделей.

В конце компиляции и загрузки внизу среды разработки в консоли выведется сообщение об окончании процесса и время, потраченное на него – рисунок 3.13.

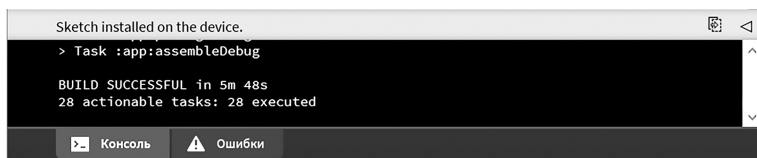


Рисунок 3.13 – Компиляция и загрузка приложения на устройство закончено

После этого с небольшой задержкой приложение запустится на смартфоне, и можно проверить правильно ли оно выполняется.

Теперь среди приложений смартфона появилось новое – ВАШЕ ПЕРВОЕ ПРИЛОЖЕНИЕ. Можно в любой момент запустить это приложение на устройстве даже без подключения к компьютеру.

3.5 Использование цветов в программе

Пока все части программы были черными, белыми или серыми. Причем по умолчанию. В программе это отдельно никак не задавалось. Теперь научимся задавать цвет.

Сначала необходимо определиться, как задается цвет в цифровой среде. Существует несколько вариантов кодировок цвета. В Processing используются три: уровень серого, RGB (red, green, blue – красный, зелёный, синий) или HSB (Hue, Saturation, Brightness – тон, насыщенность, яркость).

Уровень серого задается целым числом в диапазоне от 0 до 255, где 0 – черный, а 255 – белый. Число между ними дает более темный серый цвет при приближении к нулю, и светло серый цвет – при приближении к 255.

Кодировка RGB, в чем-то схожа с обозначением уровня серого. Здесь также используются целые числа в диапазоне от 0 до 255. Отличие в том, что нужна уже не одна, а три числа. Первое число дает уровень красного, второе – уровень зеленого, третье – уровень синего. Здесь 0 – означает, что цвета нет совсем, а 255 – максимальный уровень. Все цвета являются различной комбинацией именно этих трех цветов. Если добавить еще и четвертое число в диапазоне от 0 до 255, то можно регулировать еще и прозрачность цвета.

Кодировка HSB также составляется из трех значений: цветовой тон (от 0 до 359°), насыщенность (от 0 до 99%) и яркость (от 0 до 99%). После задания этих трех параметров формируется код цвета в виде хештега (#) и шести шестнадцатеричных цифр.

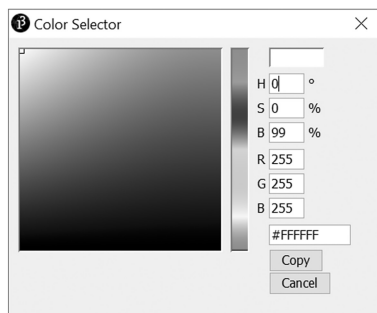


Рисунок 3.14 – Выбор цвета

Для выбора цвета и поиска ее кодировки можно воспользоваться, как и различными онлайн палитрами, так и встроенным инструментом Color Selector или Выбор цвета из вкладки Tools или Инструменты в среде Processing – рисунок 3.14.

Для этого достаточно в столбике-палитре в середине выбрать цвет, а потом выделить нужный оттенок в поле слева. После выделения в ячейках справа появятся все возможные кодировки.

Теперь давайте попробуем поменять цвета. Первым в списке, конечно же, будет фон. В среде Processing фон задается командой **background()**. Как и любые другие команды, она должна писаться внутри функции или **void**. Для того, чтобы решить в каком именно **void** команда фона должна находиться важно представлять, когда именно она должна происходить.

Для примера возьмем написанную ранее программу с фигурой круг следующим за мышкой и добавим фон в **void setup()**:

```
void setup(){
  size(displayWidth, displayHeight);
  background(0);
}

void draw(){
  ellipse(mouseX, mouseY, 20, 20);
}
```

В результате программа будет также выглядеть, как простейшая «рисовалка», но уже с черным фоном – рисунок 3.15.

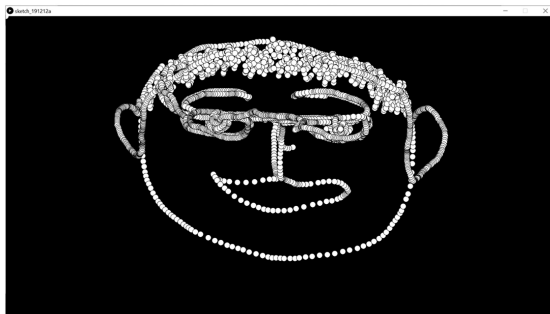


Рисунок 3.15 – Черный фон в **void setup()**

Теперь переместить команду *background()* в *void draw()* перед кругом:

```
void setup(){  
  size(displayWidth, displayHeight);  
}
```

```
void draw(){  
  background(0);  
  ellipse(mouseX, mouseY, 20, 20);  
}
```

Получим только один круг, следующий за мышкой – рисунок 3.16.

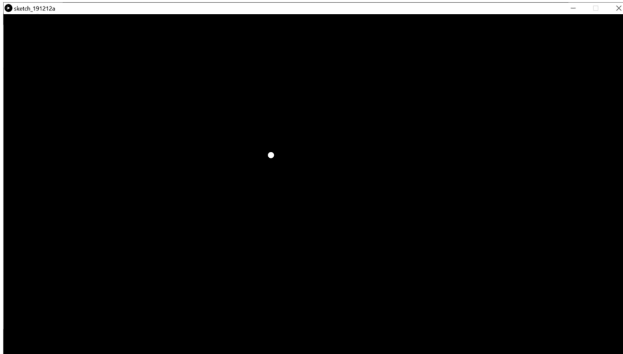


Рисунок 3.16 – Черный фон в *void draw()* перед объектом

Если же поставить команду фона после прорисовки круга, то на экран выведется просто черный фон.

Почему же так происходит?

В первом случае фон находится в функции *void setup()*, которая выполняется только один раз в самом начале. Поэтому и фон прорисовывается один раз в начале, после чего прорисовываются все объекты из *void draw()*. В нашем случае это круг.

Во втором случае фон рисуется в *void draw()*, а значит, он рисуется постоянно по циклу. А после него рисуется круг. В итоге мы получаем последовательную прорисовку по циклу: фон-круг, фон-круг и так далее.

В последнем случае, когда фон прописан после круга, получается черный экран, так как в каждом кадре сначала рисуется круг, который потом закрашивается фоном.

Важно!!! Практически в любом программировании принципиально важна последовательность функций и/или команд.

Дальше будем закрашивать объект командой `fill()`. Например, сделаем круг в программе ярко красным:

```
void setup(){  
  size(displayWidth, displayHeight);  
  background(0);  
}
```

```
void draw(){  
  fill(255, 0, 0);  
  ellipse(mouseX, mouseY, 20, 20);  
}
```

В программе теперь можно рисовать не белым, а красным кругом – рисунок 3.17.

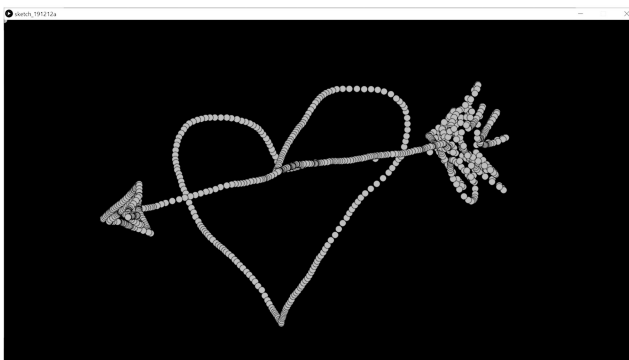


Рисунок 3.17 – Красная кисть на черном фоне

Обратите внимание, что команду закраски надо писать до объекта. Иначе первый круг будет белым. А закрасенным, только следующие.

Если для приложения необходим полностью прозрачный объект, то можно выключить закраску, написав команду *noFill()*.

Контур, который задается командой *stroke()*, тоже надо прописывать до объекта, по все той же причине. Например, давайте для контрастности зададим белый контур в предыдущей программе:

```
void setup(){  
  size(displayWidth, displayHeight);  
  background(0);  
}
```

```
void draw(){  
  fill(255, 0, 0);  
  stroke(255);  
  ellipse(mouseX, mouseY, 20, 20);  
}
```

И получим приложение как на рисунке 3.18.



Рисунок 3.18 – Красная кисть с белым контуром

Так же, как и с закраской, если контур не нужен, можно воспользоваться командой *noStroke()*.

Когда рисуется не один, а несколько объектов разного цвета, то необходимо прописывать команду закраски для каждого цвета. Но не обязательно для каждого объекта. Например:

```

void setup(){
  size(displayWidth, displayHeight);
  background(0);
}

void draw(){
  fill(255, 0, 0);
  stroke(255);
  ellipse(mouseX, mouseY, 20, 20);
  fill(0, 255, 0);
  rect(100, displayHeight-100, 50, 50);
  rect(displayWidth-100, displayHeight-100, 50, 50);
}

```

Получаем еще два зеленых квадрата в нижних углах программы (рисунок 3.19). Профессиональные программисты порекомендовали бы выделить код написанные в *void draw()* в две отдельные функции для простоты чтения кода.

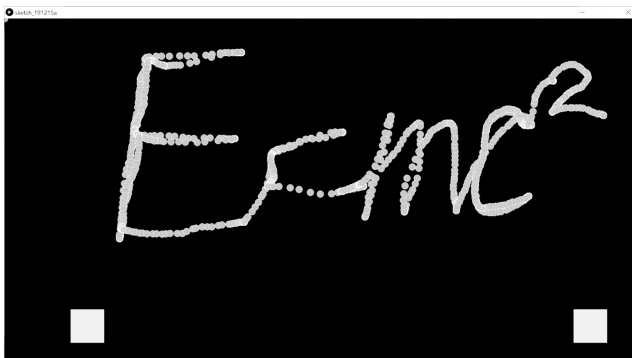


Рисунок 3.19 – Несколько объектов разного цвета

Теперь мы знаем три базовых элемента для работы с цветом и сможем создавать красочные и приятные глазу приложения. Конечно же, правильный подбор цветов – это большая отдельная задача, которая не описывается в этом учебном пособии. Это скорее вопрос дизайна и художественного восприятия. Именно поэтому даже в небольших

командах разработчиков приложений часто работает хотя бы один дизайнер.

3.6 Голограммы. Добавление объема. Перемещение

Ранее мы описывали, как прорисовывать плоские объекты, но среда Processing позволяет рисовать и манипулировать еще и с объемными объектами. Для этого в первую очередь надо включить координату *z*, которая до этого не задавалась.

Это делается в команде *size()*, которая задает размеры экрана. Но в отличие от первых двух параметров – ширины и высоты экрана, глубина для современных экранов пока параметр не физический, а виртуальный (хотя, вполне возможно через несколько лет в продаже появятся устройства с голограммными проекциями). Поэтому и третий параметр задается не физическим размером, а специальным параметром *P3D*:

```
void setup(){  
  size(displayWidth, displayHeight, P3D);  
  background(0);  
}
```

Только после этого можно приступить к рисованию объемных фигур. Стандартных фигур всего две: параллелепипед – *box()* и сфера – *sphere()*. В каждой из этих команд требуется как минимум один параметр, а именно размер стороны для куба или радиус для сферы:

```
void draw(){  
  box(300);  
  sphere(200);  
}
```

В итоге получаем обе фигуры, наложенные друг на друга, в верхнем левом углу (рисунок 3.20).

Дело в том, что для объемных фигур не задается местоположение, они всегда рисуются в точке начала координатной оси. Для сферы можно задавать только радиус, а для параллелепипеда один параметр или три параметра – размер по ширине, высоте и глубине.



Рисунок 3.20 – Базовые объемные фигуры

Если же надо перенести такой объемный объект, придется перенести всю систему координат командой *translate()*, где тремя координатами (шириной, высотой, глубиной) задается, куда она переносится.

Координаты можно задавать положительными и отрицательными числами, переменными и даже математическими вычислениями:

```
void draw(){  
  translate(displayWidth/2, displayHeight/2, -100);  
  box(300);  
  translate(displayWidth/2, displayHeight/2, 0);  
  sphere(200);  
}
```

Получаем объекты, нарисованные уже в других точках (рисунок 3.21).

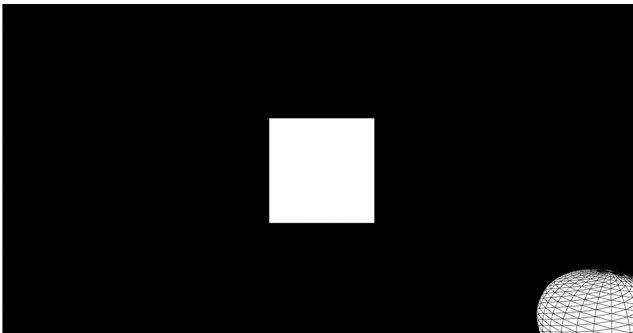


Рисунок 3.21 – Перенос начала координат

Важно!!! Обратите внимание, что несколько команд **translate()** дополняют работу друг друга. Так, в примере, первая команда перенесла куб на центр экрана, а вторая – перенесла сферу еще на половину экрана. И получилось, что сфера оказалась в правом нижнем углу.

Для того, чтобы задавать перенос координат для каждого объекта или группы объектов отдельно, можно использовать команды **pushMatrix()** и **popMatrix()**. Они позволяют, соответственно создавать отдельную систему координат и возвращаться к исходной:

```
void draw(){
  pushMatrix();
  translate(displayWidth/2, displayHeight/2, 0);
  box(300);
  popMatrix();

  pushMatrix();
  translate(displayWidth/3, displayHeight/3, 0);
  sphere(200);
  popMatrix();
}
```

Теперь можно задавать перенос объектов, отдельно для каждого не думая о том, как переносы влияют друг на друга. Здесь также рекомендуется создать две отдельные функции для рисования куба и сферы. Как это сделать – описано ранее.

Но на рисунке 70 Вы заметили еще один нюанс. Кодом задавался куб, а на экране виден просто квадрат. Все дело в виде на этот куб. Если на куб смотреть ровно спереди, то ни одну из его граней видно не будет. Для того, чтобы показать грани, куб надо немного развернуть. Разворот фигур в объеме выполняется с помощью команды **rotate()** или более частных **rotateX()**, **rotateY()**, **rotateZ()**. Здесь **rotate()** работает аналогично **rotateZ()**, то есть вращает вокруг оси **Z**.

Важно!!! Все команды вращения сдвигают объекты относительно точки начала координат. Поэтому категорически важно задавать команды переноса координат и вращения в нужном порядке.

В этих командах вращения задается только один параметр – угол поворота. Если хотите вращать по часовой стрелке, то задаете угол с положительным значением. Если – нет, то с отрицательным.

Важно!!! В командах вращения угол задается в радианах, а не в градусах. Из курса школьной математики Вы знаете, что $45^\circ = \pi/4$ радианов, $60^\circ = \pi/3$ радианов, $90^\circ = \pi/2$ радианов, $180^\circ = \pi$ радианов, $360^\circ = 2\pi$ радианов.

Константа π в Processing задается системным словом `PI`.

Теперь, используя описанные команды, развернем куб одной из граней, а сферу повернем вокруг оси **Z**:

```
void draw(){
  pushMatrix();
  translate(displayWidth/2, displayHeight/2, 0);
  rotateX(PI/4);
  box(300);
  popMatrix();

  pushMatrix();
  rotate(PI/4);
  translate(displayWidth/3, displayHeight/3, 0);
  sphere(200);
  popMatrix();
}
```

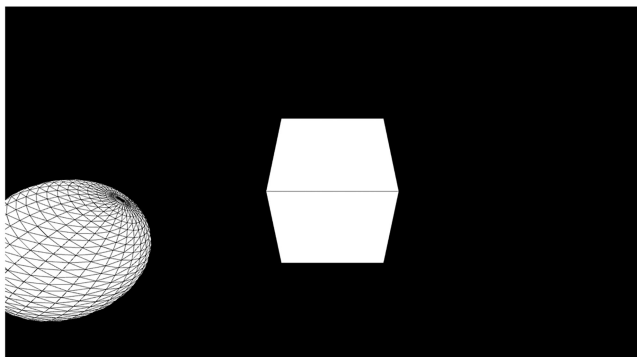


Рисунок 3.22 – Разные варианты разворота и переноса координат

Как видно из кода и на рисунке 3.22, куб развернулся вокруг своей оси, потому что команда вращения была прописана после команды переноса координат. А сфера развернута вокруг изначальной оси **Z**.

Часто разработчикам неудобно использовать радианы для обозначения угла. В таком случае может выручить команда **radians()**, которая преобразует углы в градусах в радианы за разработчика:

```
void draw(){  
  pushMatrix();  
  translate(displayWidth/2, displayHeight/2, 0);  
  rotateX(radians(45));  
  box(300);  
  popMatrix();  
  
  pushMatrix();  
  rotate(radians(45));  
  translate(displayWidth/3, displayHeight/3, 0);  
  sphere(200);  
  popMatrix();  
}
```

Команда **radians()** дает числовое значение. Поэтому ее можно использовать как самостоятельно, так и вставлять вместо параметра в другую команду.

Для еще более красивого вращения можно воспользоваться системной переменной **frameCount**, которая отсчитывает количество кадров прорисованных функцией **void draw()** с запуска приложения:

```
void setup(){  
  size(displayWidth, displayHeight, P3D);  
}  
  
void draw(){  
  background(0);  
  
  pushMatrix();  
  translate(displayWidth/2, displayHeight/2, 0);  
  rotateX(radians(frameCount));
```

```

    box(300);
    popMatrix();

    pushMatrix();
    rotate(radians(frameCount));
    translate(displayWidth/3, displayHeight/3, 0);
    sphere(200);
    popMatrix();
}

```

Запустите этот код. Объекты в приложении находятся в постоянном движении. Его стационарная картинка не дает понять, что делает код, как работающее приложение.

3.7 Типы данных в Processing

При изучении команд, которые рассматривали ранее, при наборе кода появлялись небольшие красные подсказки системы, немного выше консоли (рисунок 3.23). Чаще всего высвечивалась просьба добавить в параметры *int* и *float*. Эти слова означают числа. Они появляются тогда, когда требуются числовые параметры для команды. Теперь рассмотрим типы данных более подробно.

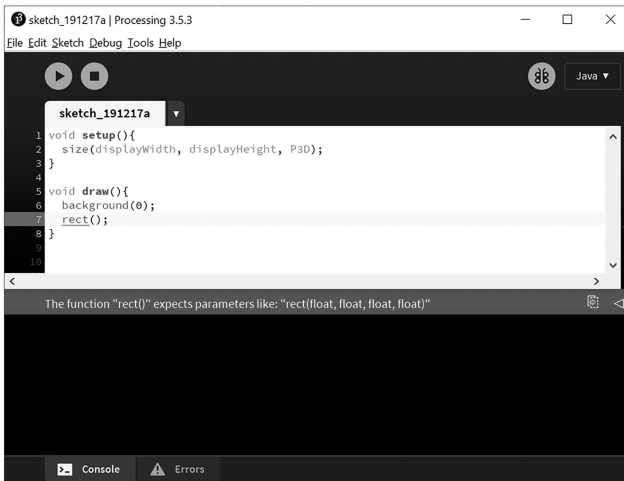


Рисунок 3.23– Подсказки среды разработки

Любая программа работает в первую очередь с информацией. Она может ее принимать, перерабатывать, выдавать и иногда даже генерировать. Информация – это главное и единственное, с чем любая программа работает.

Мы передаем информацию чаще всего в виде слов. В программировании для работы со словами и строками текста используется такой тип данных, как **String**. Любые значения в него записываются через кавычки. Например, «Hello, World!». Если нужна только одна буква или символ, то – тип данных **char**. Значения в него записываются через апостроф. Например, '!’.

Числа в программировании используются чаще всего. И их типов довольно много. Базовые это – **byte**, **int** и **float**.

Тип **byte**, как можно понять из названия, может хранить в себе 8 бит информации с целым числом от -128 до 127. В типе **int** (сокращении от *integer* – целые) могут храниться целые числа от -2 147 483 647 до 2 147 483 648 или 32 бита информации. Слово **float** переводится с английского языка как поплавок, подразумеваемая запятую в числе, которая может смещаться в числе как поплавок, образуя дробное число. И, соответственно, используется для дробных чисел от $-3.40282347 \cdot 10^{38}$ до $3.40282347 \cdot 10^{38}$.

А теперь представьте кивающего головой человека. Он тоже передает информацию. Но не в виде слов или чисел, а в виде логического значения: **true-false** (правда-ложь), включить-выключить, да-нет или есть-нет. Такой тип данных называется **boolean**.

Это базовые типы данных, которые используются в среде Processing. Конечно, есть и другие, но некоторых из них рассмотрим ниже. А сейчас надо обсудить, как же эти типы данных использовать.

Типы данных всегда задаются при создании новых переменных. В примерах выше, мы пробовали пользоваться только системными переменными. Теперь будем создавать свои.

Переменная в программировании – это, по сути, зарезервированная часть памяти устройства с определенным названием. Для сравнения, в реальном мире переменной можно назвать полку, сосуд или сумку, куда складываются определенные вещи или материалы. Так же, как и в физическом мире, такие переменные – «емкости» имеют определенные свойства: что может храниться и в каком количестве.

Любые переменные делятся на два вида: *локальные* и *глобальные*. Локальные переменные можно сравнить с полкой, прибитой на стене комнаты – ее можно использовать только в этой комнате. То есть

локальная переменная может быть использована только в той функции, где она создана. Глобальные переменные можно сравнить с сумкой или рюкзаком. Значение из нее можно использовать во всем коде повсеместно.

Теперь перейдем непосредственно к созданию новой глобальной переменной в коде. Есть несколько вариантов как это можно сделать:

```
int color1=0;  
int color2=0, color3;  
  
void setup(){  
  size(displayWidth, displayHeight);  
  color3=0;  
}  
  
void draw(){  
  background(color1, color2, color3);  
}
```

В любом случае сначала указывается тип переменной. В примере это `int`. Потом прописывается ее название (`color1`, `color2` и `color3`). Это обязательная часть для создания или, правильнее сказать, декларирования переменной. При этом после декларирования типа данных можно перечислить сразу несколько названий переменных через запятую.

***Важно!!!** С точки зрения среды разработки название переменной может быть почти любым, на усмотрение разработчика. Среда лишь требует, чтобы она была написана латинскими символами и цифрами и не совпадала с системными и уже используемыми названиями. Программисты рекомендуют называть переменные и функции таким образом, чтобы было уже по названию понятно, за что они отвечают.*

После декларирования переменной рекомендуется задать или записать в нее начальное значение. Это можно сделать сразу после декларирования или внутри функции. Если начальное значение не задается, а в коде оно потребуется, то система выдаст значение по умолчанию. Для числовых типов – это 0, для символа – “”, а для логической пере-

менной – *false*. Но для строковой переменной значения по умолчанию нет, потому Processing выдаст ошибку *NullPointerException*.

Переменные могут меняться, поэтому разработчики могут их изменять в коде. Для этого используется ряд специализированных знаков. Один из них «=*=*». Знак равенства стирает предыдущее значение, которое хранилось в переменной, и записывает новое. Но есть и другие, которые не стирают полностью, а модифицируют значения переменной: +=, -=, *=, /=, ++ и --.

Здесь знаки += и ++ имеют схожее действие. ++ добавляет в переменную значение 1, и это полностью совпадает с +=1. Хотя += более универсален, так как может добавлять в переменную не только один, но вообще любое заданное после него значение:

```
color1++;  
color2+=1;  
color3+=8;
```

Аналогично со знаками -= и --. Но здесь, соответственно, заданное значение или единица отнимаются от переменной.

Используя *= или /= можно умножить или разделить значение в переменной на заданное число.

Теперь используем эти знаки для получения постепенно меняющегося фона:

```
int color1=255;  
int color2=0, color3;  
  
void setup()  
size(displayWidth, displayHeight);  
color3=0;  
}  
  
void draw()  
color1--;  
color2++;  
color3++;  
background(color1, color2, color3);  
}
```

С применением локальных переменных мы встретимся. Но принцип декларирования для них тот же: пишется тип данных, название и задается начальное значение. Единственное отличие – они задаются внутри какой-либо функции и используются только там:

```
void setup(){  
  size(displayWidth, displayHeight);  
}
```

```
void draw(){  
  int color1=0;  
  if (mousePressed){  
    color1=255;  
  }  
  background(color1);  
}
```

В данном коде нам не знакомы *if (mousePressed){ }*. Теперь рассмотрим их.

3.8 Функции нажатия и протяжки на сенсорном экране

Во всех предыдущих приложениях воздействие пользователя ограничивалось только ведением мышки по экрану, которое считывалось системными переменными *mouseX* и *mouseY*. Но нужны еще кнопки, а значит считывание нажатия на сенсорный экран.

Для этого в среде Processing существует целый ряд системных функций: *mousePressed()*, *mouseDragged()*, *mouseMoved()*, *mouseClicked()* и *mouseReleased()*.

Функция *mousePressed()* срабатывает при нажатии на кнопку мыши на компьютере или на сенсорном экране смартфона, или планшета, и, кроме того, может также быть и системной переменной типа Boolean. Давайте нарисуем прямоугольник, сдвигающийся при нажатии, применив эту функцию:

```
int rectX=200, rectY=200;  
  
void setup(){  
  size(displayWidth, displayHeight);  
}
```



```
void draw(){
    background(0);
    rect(rectX, rectY, 50, 50);
}
```

```
void mousePressed(){
    rectX+=10;
    rectY+=10;
}
```

В этом приложении цвет экрана будет меняться не сам по себе, а только по нажатию на экран. Ведь действия по изменению переменных цвета прописаны в **void mousePressed()**.

***Важно!!!** Команды, прописанные в системных функциях нажатия и протяжки, срабатывают только во время такого нажатия и протяжки. То есть, если написать в фигурных скобках **void mousePressed()** команду **rect()**, прямоугольник появится только в то мгновение, когда мышь или экран нажат, после чего будет закрашен объектами из **void draw()**.*

Теперь попробуем применить **mouseDragged()**. Эта функция считывает протяжку на сенсорном экране, то есть передвижение по сенсорному экрану нажатым пальцем. Но сначала узнаем еще некоторые системные переменные: **pmouseX** и **pmouseY**. Эти переменные хранят в себе не просто значение координат для мыши, а эти значения в предыдущем кадре (прорисовке функции **void draw()**). И сравнив их с нынешними **mouseX** и **mouseY** можно определить в какую сторону была протяжка. В итоге получим вот такой код:

```
int rectX=200, rectY=200;

void setup(){
    size(displayWidth, displayHeight);
}

void draw(){
    background(0);
```

```
rect(rectX, rectY, 50, 50);  
}
```

```
void mouseDragged() {  
    rectX += pmouseX - mouseX;  
    rectY += pmouseY - mouseY;  
}
```

Функции *mouseMoved()*, *mouseClicked()* и *mouseReleased()* используются на многого реже. Но стоит иметь их в своем арсенале. *mouseMoved()* выполняется, когда не нажатая мышь или палец на сенсорном экране сдвигается. *mouseClicked()* выполняется сразу после нажатия и отпускания (клика) мыши или пальца. *mouseReleased()* выполняется после отпускания кнопки мыши или сенсорного экрана.

При помощи этих функций можно прописать существенную часть функционала для мобильного приложения, если достаточно чтобы весь экран был как одна кнопка. Но если нужно несколько кнопок, то понадобится еще один инструмент, а именно условия.

3.9 Прописанные условия

Когда задается условие, звучит это примерно так: «Если выполнил задание, то можешь отдохнуть». Ключевыми слова даже в словесной форме являются «Если ..., то...». После слова «Если» пишется непосредственно условие, а после «то» – что же при выполнении условия делать.

В среде Processing ключевым словом и обязательными элементами для условия являются *if () {}*. Так же, как в словесной форме в круглых скобках, сразу после *if* пишется условие, а в фигурных – действие или действия, которые надо будет выполнять:

```
void draw() {  
    if (mousePressed) {  
        color1 = 255;  
    }  
    background(color1);  
}
```

Кроме того, словесное условие можно дополнить: «Если выполнил задание, то можешь отдохнуть, иначе – работать». Так и программное условие можно дополнить системным словом *else*{}:

```
void draw(){  
  if (mousePressed){  
    color1=255;  
  } else {  
    color1=150;  
  }  
  background(color1);  
}
```

Для условия в круглых скобках можно использовать логическую переменную или какое-либо сравнение двух параметров или значений. Такое сравнение можно выполнить с помощью ряда знаков: >, <, >=, <=, != и ==. Со знаками > и <. Знаки >= и <= имеют схожее значение, но добавляется еще возможность равенства двух величин. Знаки == и != имеют противоположенное друг другу значение: две величины равны и две величины не равны.

Важно!!! Для выяснения равны ли две величины нельзя использовать знак =, так как он зарезервирован за присвоением или записью значения. Кроме того, если хотите сравнить две строки на совпадение значений необходимо использовать команду *equals()*:

```
if (str1.equals(str2)) {  
  background(255);  
}
```

Теперь разделим экран на две виртуальные половины: при нажатии на левую – экран будет светлеть, а на правую – темнеть. Для этого сначала пропишем код с переменной в параметре фона, которая меняется при нажатии на экран:

```
int color=150;  
  
void setup(){  
  size(displayWidth, displayHeight);  
}
```

```
void draw(){  
    background(color);  
}
```

```
void mousePressed(){  
    color+=10;  
}
```

Теперь добавим команду и синтаксис условия, и противоположенное действие:

```
void mousePressed(){  
    if(){  
        color+=10;  
    }  
    else {  
        color-=10;  
    }  
}
```

Для рабочего кода не хватает непосредственно условия, а именно: попадание нажатием на левую сторону. Граница между сторонами проходит ровно посередине экрана. Ее мы можем вычислить, разделив ширину экрана *displayWidth* на 2, а считать, куда попали при нажатии можно с помощью *mouseX*. Осталось только сравнить эти две величины. Чтобы положение мыши оказалось слева ее координата должна быть меньше середины ширины экрана. Вот и условие:

```
void mousePressed(){  
    if(mouseX<displayWidth/2){  
        color+=10;  
    }  
    else {  
        color-=10;  
    }  
}
```

Почти полностью тот же код можно использовать только в *void draw()*, заменив функцию *void mousePressed()* на логическую пере-

менную *mousePressed*. Но тогда придется соединять два условия. Для этого хорошо подходят специальные логические операторы И, ИЛИ, НЕ – `&&`, `|`, `!`.

Если необходимо одновременно совпадение двух условий используется оператор И `-&&`. Например, «Отличник и спортсмен получает ректорскую стипендию». Здесь, если человек отличник, но не спортсмен, он стипендию не получает. Или, если он спортсмен, но не отличник, тоже.

Если необходимо выполнение хотя бы одного условия, то используется оператор ИЛИ – `|`. Например, «Уступайте места пассажирам с детьми, инвалидам или престарелым». Здесь при выполнении хоть одного условия стоит уступить место.

НЕ – `!` – дает противоположенное значение. Так, «неправда» это – «ложь» или *true!==false*.

В итоге получим такой код для управления цветом фона:

```
void draw(){  
  background(color);  
  if(mousePressed && mouseX<displayWidth/2){  
    color+=10;  
  } else if (mousePressed && mouseX>displayWidth/2){  
    color-=10;  
  }  
}
```

3.10 Циклы

Иногда в программировании разработчик встречается с однотипными задачами. Всегда можно воспользоваться комбинацией копировать-вставить, но можно задачу или код оптимизировать, сделать короче, четче и понятнее.

Один из инструментов как сократить код для выполнения однотипных задач – *применение циклов*.

Циклы – это своеобразные программные конвейеры. Все повторяющиеся действия прописываются один раз и задаются, в какой последовательности и до каких пор эти действия должны повторятся.

В среде Processing используется два вида циклов: *for* и *while*. Их отличие легко понять, представив конвейер, который делает, например, конфеты. С помощью цикла *for* разработчик задает, сколько

таких «конфет» уже готово, сколько «конфет» делается за один заход и до скольких «конфет» нужно еще сделать. Цикл *while* намного проще по форме. В нем задается только условие остановки. Для примера с конфетами таким условием остановки может быть «когда коробка с готовыми конфетами будет полной».

Давайте нарисуем одинаковые квадраты циклом *for* и одинаковые круги циклом *while*:

```
void setup(){  
  size(displayHeight, displayHeight);  
}
```

```
void draw(){  
  for(int i=10;i<=displayHeight/2-70; i+=50){  
    rect(i,i,70,70);  
  }  
  int j=displayHeight;  
  while(j>=displayHeight/2+70){  
    ellipse(j-70,j-70,70,70);  
    j-=50;  
  }  
}
```

Результат представлен на рисунке 3.24.

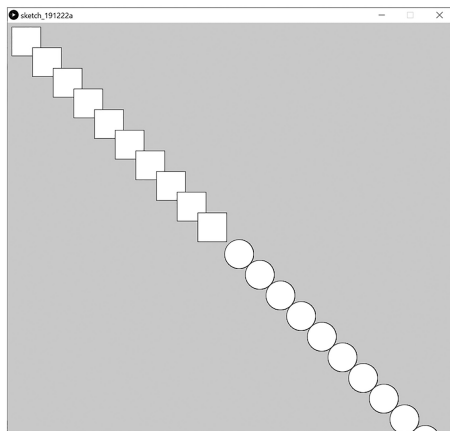


Рисунок 3.24 – Рисование однотипных объектов

Оба цикла задаются системными словами и двумя парами скобок – круглыми для параметров цикла и фигурными для повторяющихся действий. В каждом из них, так же, почти всегда используется счетчик, позволяющий считать количество раз или итераций работы цикла. Обычно это отдельная переменная типа *int*, которую принято называть *i*, *j*, *k* и т.д. И теперь разберем каждый цикл подробнее.

В круглых скобках цикла *for* задаются три основных элемента. Первый элемент – с чего начинается отсчет. В примере это *i=10*;. Далее задается условие остановки: *i<=displayHeight/2-70*;. И в конце шаг, с которым счетчик изменяется: *i+=50*. В фигурных скобках уже задается, что должно выполняться в цикле. Причем, чаще всего, но не всегда, счетчик используется как параметр в этих действиях, чтобы задать смещение или изменение объектов, или их свойств.

Важно!!! Условие остановки любого из циклов, задается ли оно с помощью счетчика или нет, должно быть достижимым. Иначе программа не будет выполняться, зависнет и затормозит работу устройства.

Для цикла *while* в круглых скобках задается лишь условие завершения цикла – *j>=displayHeight/2+70*. Поэтому, при использовании счетчика, задавать его начальное значение и шаг изменения приходится отдельно. Соответственно – до цикла и внутри цикла.

Также можно заметить, что шаг можно задавать как в положительную сторону – *i+=50*, так и в отрицательную – *j-=50*;. Но и условие остановки тогда должно иметь другой знак: *i<=displayHeight/2-70* и *j>=displayHeight/2+70* соответственно. Иначе можно получить бесконечный цикл, который затормозит и приложение, и устройство.

3.11 Массивы данных

Ранее переменные сравнивались с полками или сумками. Мы уже знаем, что в одной переменной может храниться только одно значение. Но в программировании есть специальная структура, которая позволяет хранить сразу ряд однотипных данных. Называется она – *массив*.

Основной элемент, по которому можно отличить массив данных в среде Processing, это квадратные скобки – []. Давайте сначала научимся массивы данных декларировать. Так как они схожи во многом

с переменными, декларировать их можно вне функций (если они глобальные) и внутри функции (если они локальные). Декларирование массива состоит из заявления типа данных, хранимых в массиве, квадратных скобок, названия массива, равенства и либо записи элементов массива (в фигурных скобках), либо заявления количества элементов в массиве (в квадратных скобках):

```
float [] rectsizeX = { 90, 150, 30 };  
float [] rectsizeY = new float [3];
```

После этого с элементами массива можно работать практически как с обычной переменной, записывая туда значения, используя как параметры в коде, изменяя значения. Единственное отличие при обращении к элементу массива – необходимо прописывать квадратные скобки, а внутри номер элемента:

```
rect( rectsizeX[0], rectsizeY[0], 40,40);
```

*Важно!!! Нумерация элементов в массиве всегда начинается с нуля. И, если, например, в массиве 3 элемента, значит первый, то элемент будет **rectsizeX[0]** и последний – **rectsizeX[2]**.*

Теперь воспользуемся инструментами, которые изучили ранее и сделаем небольшое «новогоднее» приложение, в котором будет идти снег:

```
float [] elX = new float [100];  
float [] elY = new float [100];  
float [] color = new float [100];  
  
void setup(){  
  size(displayWidth, displayHeight);  
  
  for (int i=0;i<100;i++){  
    elX[i]=random(0,displayWidth);  
    elY[i]=random(0,displayHeight);  
    color[i]=random(0,255);  
  }  
}
```



```

void draw(){
  background(0);
  for (int i=0;i<100;i++){
    elY[i]+=5;
    fill(255,255,255,color[i]);
    ellipse(elX[i],elY[i],30,30);
    if (elY[i]>displayHeight){
      elY[i]=0;
      elX[i]=random(0,displayWidth);
      color[i]=random(0,255);
    }
  }
}
}
}

```

Итоговая программа будет динамично изменяться, и выглядеть примерно, как на рисунке 3.25.



Рисунок 3.25 – Новогоднее приложение

В этом коде единственная незнакомая команда это **random()**. Она позволяет генерировать случайные значения в заданном диапазоне. Так, если задать в ее круглых скобках одно число, то будет генерироваться значение от 0 до этого числа. Если задать два значения, то команда будет выдавать случайное значение в диапазоне между ними.

4 КАК МЫ СОЗДАЛИ СВОЙ ПРОЕКТ В СРЕДЕ PROCESSING

4.1 Планирование работы над проектом

Как мы отмечали в предыдущей части нашего пособия, приступая к работе над проектом необходимо сформулировать этапы проектирования. Для нашего проекта сформулируем три пункта (рисунок 4.1):

- 1) цель;
- 2) конечный приемлемый результат;
- 3) сроки или дедлайн.



Рисунок 4.1 – Постановка задачи

Цель дает направление, в котором двигаться, работать. Результат дает понимание до каких пор работать. А дедлайн ограничивает во времени, без чего работа будет тянуться или откладываться бесконечно. Это касается как рабочих или учебных проектов, так и собственных pet – project.

Оффтоп. Дедлайн – это крайний срок, к которому должен быть готов какой-то определенный конечный результат работы. При любой работе на кого-то обязательно ставятся дедлайны. Но и при работе «на себя» надо ставить собственные дедлайны для того, чтобы более продуктивно работать.

Пет-проект (от англ. pet – домашнее животное) – это проект, создаваемый разработчиком, так сказать, для себя. Профессиональные программисты создают такие проекты для саморазвития, изучения новых технологий и, возможно, будущего собственного стартапа.

Стартап – новая компания с минимумом ресурсов, в основе которой лежит какая-либо инновационная идея. Один из самых распространенных в современном мире вариантов начала собственного бизнеса.

После такой базовой формулировки можно приступать к более детальному планированию. Оно заключается в разбиении этой боль-

шой задачи на маленькие подзадачи и их приоритизацию. Те есть на небольшие шаги в виде таких же трех пунктов в порядке важности (рисунок 4.2).



Рисунок 4.2 – Деление на подзадачи

Такое деление на подзадачи позволяет получать минимально рабочий проект, который уже можно использовать как альфа-версию для первичного тестирования. А уже потом дорабатывать, добавляя новый функционал.

Оффтоп. В разработке программного обеспечения принято использовать деление на альфа-, бета- и гамма- версии. Альфа-версия используется для внутреннего тестирования. К проверке бета-версии часто присоединяются небольшая группа пользователей. А уже гамма-версия идет в релиз.

Релиз – это полноценный и, иногда, окончательный выпуск программы или приложения для пользования широким кругом людей и компаний.

Кроме деления на подзадачи рекомендуется прописать для каждой из них алгоритм выполнения программы. Такой алгоритм может быть в базовой словесной форме или в виде детальной блок-схемы. Чем подробнее будет прописан или прорисован алгоритм, тем легче будет потом реализовать его в коде. А еще блок-схема будет нужна для подачи заявки на авторское свидетельство, если разработчик захочет защитить свою программу и идею как интеллектуальную собственность.

Оффтоп. Алгоритм – инструкция в виде последовательности коротких и однозначных действий для получения определенного результата.

Блок-схема – графический алгоритм в виде стандартизированных блоков, соединенных стрелками-линиями.

Как пример давайте рассмотрим создание мобильного приложения игры в среде Processing.

В первую очередь сформулируем, цель проекта. Цель – получить практические навыки разработки полноценного мобильного приложения-игры. Сроки – до конца этой главы.

Результат: Игра *Spaceship*.

Описание: *Снизу экрана симметрично слева и справа находятся кнопки. С левой стороны экрана двигается «космический корабль». При нажатии на правую кнопку «корабль» сдвигается вверх, при нажатии на левую – вниз. Справа налево летит «метеор». Он исчезает за левым краем экрана. При прохождении метеора в правом верхнем углу на один увеличивается количество очков. С определенной периодичностью на разной высоте появляется новый «метеор». Скорость движения «метеоров» и их количество постепенно увеличиваются. При попадании «метеора» в «корабль» на одно уменьшается количество жизней в левом верхнем углу. Начальное количество жизней 3. При уменьшении жизней до 0 выводится экран проигрыша.*

Ключевые три пункта для всего проекта теперь сформулированы.

Следующий шаг разбить всю разработку на подзадачи, которые включают следующее:

- 1) нарисовать ключевые элементы интерфейса игры (кнопки, корабль, метеор);
- 2) прописать попадание на кнопки;
- 3) прописать движение объектов (корабль и метеор);
- 4) ограничить движение объектов (корабля и метеора);
- 5) добавить условие проигрыша (попадание метеора на корабль) и экран проигрыша;
- 6) добавить массив «метеоров»;
- 7) добавить изменяющиеся очки и жизни.

Как видим, уже после третьего этапа можно дать игру на предварительную оценку коллегам, а после пятого – даже лояльным игрокам.

Но кроме этого, можно усовершенствовать игру:

- 8) добавив меню в начале игры.

Цели, которые достигнуты при этом:

- 1) вставка рисунков, как объектов и использование локальных размеров;
- 2) создание кнопок;
- 3) ограничение изменения объектов;
- 4) поиск новых команд;
- 5) работа с текстом;
- 6) создание меню и подменю.

Сроки тоже ограничены одним-двумя подпунктами этого учебного пособия.

Если говорить об алгоритмах, то уже список подзадач, сформулированных выше, уже является алгоритмом. Но не для программы, а для самого разработчика. Непосредственно же алгоритмы в словесной форме или в виде блок-схем будут приведены далее для конкретной подзадачи, для наглядного сравнения с итоговым кодом. Здесь же разберем только базовые принципы построения.



Так алгоритмы должны обладать следующими свойствами:

- 1) *Конечность*: Алгоритм обязательно должен заканчиваться, через какое-то количество шагов. Иначе выполняющее его устройство выйдет в бесконечный цикл и «зависнет».
- 2) *Определенность*: Каждый шаг алгоритма должен быть полностью определен и не оставлять возможности додумывать или догадываться.
- 3) *Ввод*: информация для выполнения действий должна быть введена до шага с этими действиями.
- 4) *Вывод*: Алгоритм всегда должен выдавать, какое-то значение или результат. Такой вывод информации может быть в середине и/или в завершающей части алгоритма.
- 5) *Эффективность*: Алгоритм должен состоять из базовых операций, которые можно в точности выполнить за ограниченное время просто ручкой на бумаге.

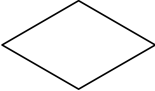

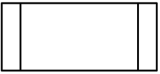


Блок-схема имеет те же свойства, но состоит из шагов не в виде текста, а в виде стандартизированных схематичных блоков – Таблица 5.1.

Таблица 5.1

Стандартные элементы блок-схем

Элемент	Название	Назначение
1	2	3
	Действие	Определяет любой тип операций, выполняемых в процессоре или памяти
	Данные (ввод/вывод)	Используется для ввода и вывода информации

Продолжение таблицы 5.1

1	2	3
	Вопрос (условие или решение)	Используется для любого закрытого вопроса с бинарным ответом (Да-Нет, Правда-Ложь)
	Соединитель	Позволяет составить блок-схему без пересечения и протяжки линий через всю схему
	Предопределенный процесс (функция)	Используется для обозначения под-алгоритма или функции
	Ограничитель	Показывает место начала и окончания алгоритма, процесса или функции
	Стрелки	Указывают направление последовательности действий

Базовые правила построения блок-схем:

- 1) Все блоки должны быть соединены стрелками.
- 2) Из предыдущего блок стрелка направлена в середину верхней части следующего блока. А в последующий блок рисуется из середины нижней части блока. Исключение - блок вопроса.
- 3) Блок вопроса имеет две выходных стрелки. Они рисуются с каждого боку или одна - снизу из середины, вторая - сбоку.
- 4) Обычно последовательность действий в блок-схеме показаны стрелками сверху-вниз. Но бывают случаи движения снизу-вверх, пока они не превышают 3-х шагов.
- 5) Соединитель используется для прерываний в блок-схеме (с одной страницы на другую или снизу блок-схемы вверх, на более чем 3 шага).
- 6) Каждая функция имеет свою собственную отдельную блок-схему.
- 7) Все блок-схемы начинаются блоком ограничителя или функции.
- 8) Все блок-схемы заканчиваются блоком ограничителя.

Еще раз хочется напомнить, что умение составлять словесные алгоритмы и блок-схемы могут пригодится не только в программиро-

вании, но в работе с людьми, планировании собственной деятельности и даже в повседневной семейной жизни.

4.2 Вставка рисунков, как объектов

Первым этапом в разработке игры была сформулирована задача нарисовать кнопки, корабль и метеор.

Конечно можно нарисовать все эти элементы с помощью стандартных фигур *rect()*, *ellipse()*, *triangle()* и *line()*. Но не каждый из разработчиков умеет хорошо рисовать. И чтобы облегчить разработку, можно прорисовать объект не в коде, а заранее в специальных графических редакторах, просто вставив потом, этот рисунок в нужном месте.

Любой рисунок вставляется в три этапа: создание переменной, загрузка файла в эту переменную, расположение рисунка в приложении.

Чтобы четко разобраться в работе с рисунками зададим фон загруженной картинкой. Для этого создаем папку *Spaceship*. В ней создаем внутреннюю папку *data*. Потом находим или рисуем картинку в формате .png и сохраняем ее в этой папке, желательно с названием латиницей, которое дает понять, что это за картинка. А уже потом, прописываем код в Processing и сохраняем его как скетч *Spaceship* в одноименной папке, которую создали чуть раньше. Код следующий:

```
PImage sky;
```

```
void setup(){  
  size(displayWidth, displayHeight);  
  sky=loadImage(«sky.png»);  
}
```

```
void draw(){  
  image(sky,0,0,displayWidth, displayHeight);  
}
```

В коде *PImage* – это тип переменной, в которой хранятся рисунки, *sky* – название переменной. Это первый шаг.

Второй шаг – загрузка изображения в переменную *sky* с помощью команды *loadImage()* из файла *sky.png*. Заметим, что загрузку обычно прописывают в *void setup()*, чтобы выполнить ее один раз в самом начале.

Следующий шаг – размещение изображения *sky* в нужном месте – ***0, 0, displayWidth, displayHeight*** – с помощью команды ***image()***. В примере место прописано четырьмя числами: первые два – местонахождение правого верхнего угла, последние два – размеры изображения по ширине и высоте (при этом изображение может растягиваться или сжиматься). Если задается только два числа – например, ***image(sky,0,0);***, то изображение выведется в своем реальном разрешении.

***Важно!!!** При загрузке на смартфон кода с большим количеством мультимедиа (картинки, музыка, видео) вероятнее всего потребуются существенные вычислительные мощности как компьютера, так и смартфона. Особенно, если в приложении мультимедийные файлы не только выводятся, но и обрабатываются. Например, растягиваются или сжимаются изображения. Поэтому, не удивляйтесь, если приложение с картинками, видео и музыкой долго компилируются и долго запускаются для использования.*

Оффтоп. Компиляция – это своеобразный перевод с «человеческого» кода в «машинный». Компьютеры, смартфоны и другая вычислительная техника работают на единицах и нулях. А код, который прописывается в среде разработки выглядит намного ближе к человеческой речи. Так вот, компиляцию или «перевод» за программиста выполняет обычно среда разработки.

Но бывают специалисты, которые умеют писать на «низкоуровневом» языке (очень близком к машинному коду). Самые известные языки низкого уровня – языки группы *Assembler*.

А такие распространенные языки как *C#, Java, JS, Python* и другие называются «высокоуровневыми» языками, так как они существенно ближе к человеческой речи. (К слову, стоит добавить, что языки *JS* и *Python* не компилируемые, а интерпретируемые).

Теперь перейдем к непосредственной задаче: нарисовать кнопки, корабль и метеор. Оставим ранее нарисованный фон. И для начала напишем алгоритм и нарисуем блок-схему кода.

Алгоритм будет следующим:

- 1) Начало.
- 2) Загрузка файлов *sky.png, ship.png* и *meteor.png* в переменные *sky, ship* и *meteor*.

- 3) Вывод изображения *sky* во весь экран, *ship* – с правого края посередине, *meteor* – с левого края посередине.
- 4) Прорисовка двух кнопок в виде кругов в правом и левом нижних углах экрана.
- 5) Конец.

Блок-схема для такого алгоритма будет выглядеть как на рисунке 4.3.

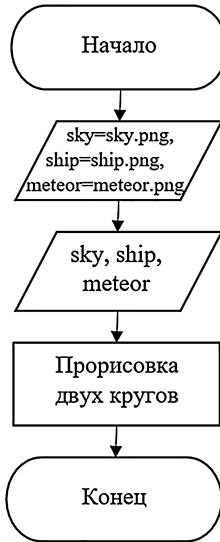


Рисунок 4.3 – Блок-схема прорисовки базового интерфейса

Теперь реализуем код на основе этой блок-схемы:

PImage sky, ship, meteor;

```
void setup(){  
  size(displayWidth, displayHeight);  
  sky=loadImage(«sky.png»);  
  ship=loadImage(«ship.png»);  
  meteor=loadImage(«meteor.png»);  
}
```

```
void draw(){  
  image(sky,0,0,displayWidth, displayHeight);
```

```

image(ship,10,displayHeight/2-50, 200,100);
image(meteor,displayWidth-120, displayHeight/2-35,120,70);
ellipse(150, displayHeight-150, 100, 100);
ellipse(displayWidth-150, displayHeight-150, 100, 100);
}

```

И получаем интерфейс как на рисунке 4.4.



Рисунок 4.4 – Первоначальный интерфейс игры

Может возникнуть вопрос, почему при расположении изображения корабля *ship* были использованы параметры *(10, displayHeight/2-50, 200, 100)*? Последние два числа – *200, 100* – выбираются разработчиком «на глаз», чтобы корабль визуально красиво и удобно размещался на экране. Число 10 для местоположения по ширине подобрано так, чтобы корабль размещался почти у самого левого края экрана.

С местоположением по высоте немного сложнее. Сначала находится середина экрана *displayHeight/2*. Но если разместить корабль там (а размещается изображение относительно своего правого верхнего угла), то корабль будет не ровно посередине, а чуть ниже. Для попадания ровно посередине надо, чтобы середина корабля совпала с серединой экрана, а значит нужно сместить изображение вверх на половину ее высоты: $100/2=50$ и получаем *displayHeight/2-50*.

Проводим расчеты для изображения метеора *meteor*. Размеры подбираем визуально: *120, 70*. По высоте – середину метеора размещаем в середине экрана *displayHeight/2-35*. По ширине – размещаем с правого края сместив влево на ширину самого метеора *displayWidth-120*, чтобы он оказался не за экраном, а ровно на левом краю.

Схожие вычисления проделываем и для кнопок. По размерам и высоте они должны быть одинаковы. Поэтому визуально подбираем размеры **100, 100** и располагаем их внизу экрана *displayHeight*, сместив вверх на их высоту **100** и визуальный отступ от края еще **50**. Отличаются кнопки лишь по расположению по ширине. Одна кнопка располагается в левом углу **0** со смещением **150** (**100** – размер, **50** – визуальный отступ) вправо: **150**. Вторая – в правом углу *displayWidth* со смещением **150** влево: *displayWidth-150*.

Проектируемый интерфейс может немного отличаться в зависимости от того, какие изображения разработчик сохранил в папке `data`.

***Важно!!!** Рекомендуется в конце каждого шага загружать приложение на телефон, чтобы проверить насколько корректно оно работает именно как мобильная игра.*

4.3 Установка локальных размеров объектов

Различные устройства имеют разное разрешение экрана. Это утверждение верно, в том числе и для смартфонов с планшетами. Поэтому объект, чей размер задан определенным количеством пикселей, может существенно отличаться в своей прорисовке на экране разных устройств.

В среде Processing все размеры задаются пикселями. Это значит, что, подобрав размеры объектов в приложении для одного устройства, можно получить неприглядный интерфейс в другом устройстве. Объекты могут стать слишком маленькими или слишком большими. Рисунки могут накладываться друг на друга. А кнопки могут оказаться, вообще, за пределами экрана и их невозможно будет использовать.

С этой проблемой можно бороться, используя так называемые «локальные единицы измерения» размеров. То есть, измеряя расстояния на экране не через пиксели, а через созданные единицы измерения, зависящие от разрешения экрана.

Разрешение или размер экрана в Processing может проясняться и фиксироваться в памяти при запуске программы с помощью переменных *displayWidth*, *displayHeight*. От них и стоит отталкиваться, создавая локальные переменные. В реальном мире, это можно сравнить с тем, как можно задавать размеры прямоугольника не с помощью

линейки и измерения в сантиметрах, а расчертив лист бумаги на определенное количество равных частей, отдельно по ширине и по высоте.

Сначала создадим две переменные для локальных размеров по ширине – w , и по высоте – h :

```
float w,h;  
PImage sky, ship, meteor;
```

Затем, после определения в команде *size()* фактических размеров экрана *displayWidth*, *displayHeight*, зададим значения этих локальных единиц, разделив экран на 500 частей по ширине и высоте:

```
void setup(){  
  size(displayWidth, displayHeight);  
  w=displayWidth/500;  
  h=displayHeight/500;  
  sky=loadImage(«sky.png»);  
  ship=loadImage(«ship.png»);  
  meteor=loadImage(«meteor.png»);  
}
```

Теперь все размеры надо переписать, используя эти локальные единицы измерения:

```
void draw(){  
  image(sky,0,0,displayWidth, displayHeight);  
  image(ship,10*w,displayHeight/2-50*h, 200*w,100*h);  
  image(meteor,displayWidth-120*w, displayHeight/2,120*w,70*h);  
  ellipse(150*w, displayHeight-150*h, 100*w, 100*h);  
  ellipse(displayWidth-150*w, displayHeight-150*h, 100*w, 100*h);  
}
```

Важно!!! Обратите внимание, что в местах, где используются системные переменные *displayWidth*, *displayHeight*, локальные единицы измерения не используются. Там уже идет подстройка к фактическим размерам экрана.

В итоге получим интерфейс как на рисунке 4.5.

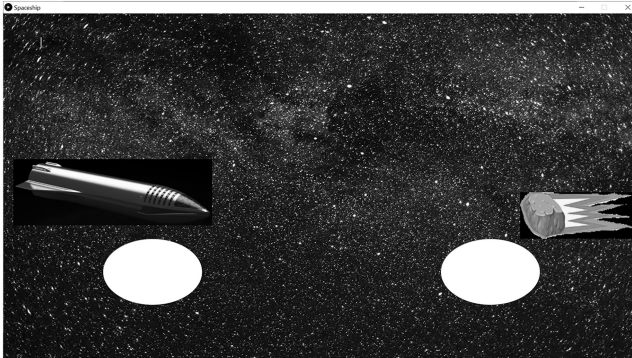


Рисунок 4.5 – Интерфейс игры на локальных единицах измерения

Как видно, подобранные ранее размеры оказались не подходящими и их надо скорректировать:

```
float w,h;
PImage sky, ship, meteor;

void setup(){
  size(displayWidth, displayHeight);
  w=displayWidth/500;
  h=displayHeight/500;
  sky=loadImage(«sky.png»);
  ship=loadImage(«ship.png»);
  meteor=loadImage(«meteor.png»);
}

void draw(){
  image(sky,0,0,displayWidth, displayHeight);
  image(ship,10*w,displayHeight/2-25*h, 100*w, 50*h);
  image(meteor,displayWidth-50*w, displayHeight/2-25*h,50*w,25*h);
  ellipse(60*w, displayHeight-75*h, 40*w, 50*h);
  ellipse(displayWidth-60*w, displayHeight-75*h, 40*w, 50*h);
}
```

И теперь, со скорректированными локальными единицами измерения получаем интерфейс, который представлен на рисунке 4.6.



*Рисунок 4.6 – Скорректированный интерфейс игры
на локальных единицах измерения*

В дальнейшем, все размеры и местоположения надо задавать именно через них.

4.4 Создание полноценных кнопок

Следующим шагом в плане разработки, является программирование полноценных кнопок. Эту задачу можно прописать в виде следующего алгоритма:

- 1) Начало.
- 2) Если нажали на экран, то шаг 3. Иначе шаг 9.
- 3) Если попали по левой кнопке, шаг 4. Иначе шаг 6.
- 4) Перенос корабля на определенное расстояние вверх.
- 5) Переход к шагу 9.
- 6) Если попали по правой кнопке, шаг 7. Иначе шаг 9.
- 7) Перенос корабля на определенное расстояние вниз.
- 8) Переход к шагу 9.
- 9) Конец.

В виде блок-схемы этот алгоритм будет выглядеть как на рисунке 4.7.

Для того, чтобы проверить правильно ли прописано попадание на кнопку, сначала необходимо скорректировать код так, чтобы корабль можно было сдвигать вдоль левого края экрана.

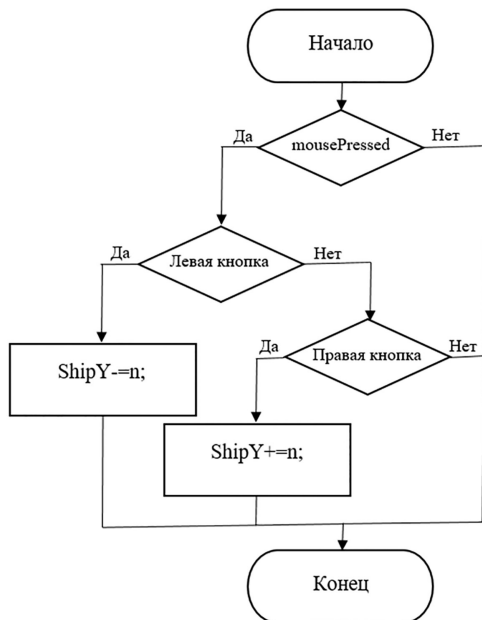


Рисунок 4.7 – Блок-схема попадания на кнопки

А значит требуется создать переменную для перемещения корабля по вертикали, задать начальное местоположение и применить ее в коде:

```

float w,h;
float ShipY;
PImage sky, ship, meteor;

void setup(){
  size(displayWidth, displayHeight);
  w=displayWidth/500;
  h=displayHeight/500;
  ShipY=displayHeight/2-25*h;
  sky=loadImage(«sky.png»);
  ship=loadImage(«ship.png»);
  meteor=loadImage(«meteor.png»);
}

```

```

void draw(){
    image(sky,0,0,displayWidth, displayHeight);
    image(ship,10*w, ShipY, 100*w, 50*h);
    image(meteor,displayWidth-50*w, displayHeight/2-25*h,50*w,25*h);
    ellipse(60*w, displayHeight-75*h, 40*w, 50*h);
    ellipse(displayWidth-60*w, displayHeight-75*h, 40*w, 50*h);
}

```

Теперь реализуем блок-схему с рисунка 4.7. Сначала пропишем «нажатие» на экран:

```

void mousePressed(){
}

```

Далее разделим экран на две половины ($displayWidth/2$): попадая пальцем/мышью ($mouseX$) на левую половину – корабль сдвигается вверх, на правую половину – вниз. Реализуем это через два условия:

```

void mousePressed(){
    if(mouseX<displayWidth/2){
        ShipY-=10;
    }
    if(mouseX>displayWidth/2){
        ShipY+=10;
    }
}

```

Можно возразить, что не обязательно использовать два условия *if*, можно обойтись формой *if ... else*. Но немного позже в коде понадобятся именно два условия для двух отдельных кнопок.

Теперь для правой кнопки сместим границу с середины экрана к правому краю кнопки $displayWidth - 60*w$ (центр кнопки) + $20*w$ (половина ширины кнопки) = $displayWidth - 40*w$:

```

void mousePressed(){
    if(mouseX< displayWidth - 40*w){
        ShipY+=10;
    }
}

```


Здесь же добавим границу попадания на левый край кнопки $displayWidth - 60*w$ (центр кнопки) – $20*w$ (половина ширины кнопки) = $displayWidth - 80*w$:

```
void mousePressed(){
    if(mouseX < displayWidth - 40*w && mouseX > displayWidth - 80*w){
        ShipY+=10;
    }
}
```

Осталось ограничить попадание на кнопку по высоте *mouseY* снизу ($displayHeight-75*h+25*h$) и сверху ($displayHeight-75*h-25*h$):

```
void mousePressed(){
    if (mouseX < displayWidth - 40*w && mouseX > displayWidth -
    80*w && mouseY < displayHeight - 50*h && mouseY > displayHeight
    - 100*h){
        ShipY+=10;
    }
}
```

То же самое проделываем и для левой кнопки:

```
void mousePressed(){
    if (mouseX < 80*w && mouseX > 40*w && mouseY < displayHeight
    - 50*h && mouseY > displayHeight - 100*h){
        ShipY-=10;
    }
}
```

Кнопки готовы, однако попадание по кнопкам прописано как попадание по прямоугольникам, а не по эллипсам. Если разработчик «дружит» с математическим описанием геометрических объектов на координатной плоскости, то может разобраться и как сделать попадание по эллипсам, но об этом далее. Если же нет, рассмотрим это далее.

В общем виде эллипс математически описывается следующим образом:

$$x^2/a^2 + y^2/b^2 = 1,$$

переменные откуда показаны на рисунке 4.8.

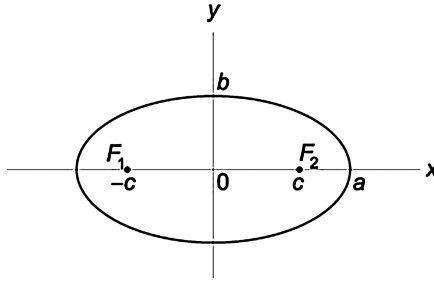


Рисунок 4.8 – Эллипс в общем виде

По аналогии с кругом, если эллипс находится не в начале координат, то уравнение эллипса преобразуется так:

$$(x - c_x)^2/a^2 + (y - c_y)^2/b^2 = 1,$$

где c_x, c_y – координаты центра эллипса по X - и Y -оси соответственно.

Для математического описания не только контура эллипса, а еще и всей площади внутри необходимо скорректировать уравнение в неравенство:

$$(x - c_x)^2/a^2 + (y - c_y)^2/b^2 \leq 1.$$

Теперь достаточно добавить полученное математическое описание в условие попадания на кнопку. Для левой кнопки получаем:

$$\begin{aligned} & (sq(mouseX-60*w)/sq(20*w)+ \\ & +sq(mouseY-(displayHeight-75*h))/sq(25*h))\leq 1). \end{aligned}$$

Для правой кнопки:

$$\begin{aligned} & (sq(mouseX-(displayWidth-60*w))/sq(20*w)+ \\ & +sq(mouseY-(displayHeight-75*h))/sq(25*h))\leq 1). \end{aligned}$$

Из новых команд здесь добавляется $sq()$, которая возводит в квадрат все, что написано в ее скобках. Так же важно не забыть и не пропустить все необходимые скобки, иначе условие попадания не работает, так, как надо.

Но, если для разработчика математика такого уровня еще сложна, можно ограничивать даже круглые кнопки прямоугольными границами, как было сделано раньше.

4.5 Ограничение перемещений и изменение объектов

Третья приоритетная задача из процесса разработки уже наполовину выполнена: уже прописано движение корабля для проверки работы кнопок. Теперь надо прописать движение метеора по горизонтали. Для этого необходимо ввести новую переменную:

```
float MeteorX;
```

Задать начальное значение в ***void setup()***:

```
MeteorX=displayWidth;
```

Используем ее в прорисовывании метеора:

```
image(meteor,MeteorX, displayHeight/2-25*h,50*w,25*h);
```

Метеор должен сдвигаться в каждом кадре, а значит перемещение должно быть прописано в ***void draw()***:

```
MeteorX-=5;
```

Цифра **10** – это скорость смещения метеора. Если ее уменьшить, метеор замедлится. Если увеличить, ускорится.

Давайте сразу введем дополнительную переменную для этой скорости:

```
float MeterSpeed=5;
```

и используем ее:

```
MeteorX-=MeterSpeed;
```

Теперь оба объекта: и корабль, и метеор у нас сдвигаются почти как надо. Переходим к четвертой задаче: Ограничение движения объектов (корабля и метеора).

Что это за ограничения и зачем они нужны?

Обратите внимание, что при долгом нажатии только на левую или только на правую кнопку корабль уходит за край экрана. Для того, чтобы это предотвратить, и нужно ввести ограничения. Делается это с помощью команды *if*.

Алгоритм получится следующий:

- 1) Начало.
- 2) Считывание местоположение корабля.
- 3) Если корабль выше верхнего края экрана, то шаг 4, иначе шаг 5.
- 4) Сдвигаем корабль обратно на верхний край.
- 5) Если корабль ниже нижнего края экрана, то шаг 6, иначе шаг 7.
- 6) Сдвигаем корабль обратно на нижний край.
- 7) Конец.

Блок-схема получается, как на рисунке 4.9.

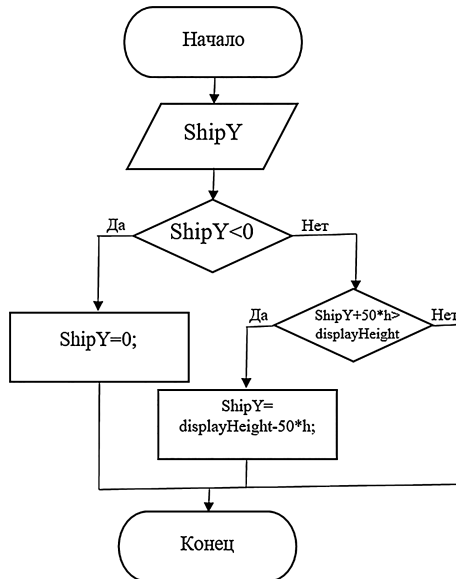


Рисунок 4.9 – Блок-схема ограничения движения корабля

Далее, необходимо прописать ограничения корабля сверху в коде программы. Для этого определяем, когда корабль выходит за край. Это

происходит, когда верхняя часть корабля *ShipY* становится выше, чем верхняя часть экрана *0* или *ShipY<0*. Тогда надо будет вернуть корабль к краю экрана или *ShipY=0*. В итоге, перед прорисовкой корабля на экране пишем следующий код:

```
if (ShipY<0){  
ShipY=0;  
}  
image(ship,10*w, ShipY, 100*w, 50*h);
```

Теперь ограничим корабль снизу. Корабль выходит за край, когда нижняя часть корабля *ShipY+50*h* (потому, что верхний край *ShipY*, а высота корабля *50*h*) становится ниже, чем нижняя часть экрана *displayHeight* или *ShipY+50*h>displayHeight*. Тогда необходимо вернуть нижнюю часть корабль к нижнему краю экрана или *ShipY=displayHeight-50*h*. В итоге получается такой код:

```
if (ShipY<0){  
ShipY=0;  
}  
if (ShipY+50*h>displayHeight){  
ShipY= displayHeight-50*h;  
}  
image (ship,10*w, ShipY, 100*w, 50*h);
```

Теперь перейдем к движению метеора. Задача: когда метеор уйдет за левый край экрана, надо вернуть его за правый край ускорив его движение.

Алгоритм:

- 1) Начало.
- 2) Считываем положение метеора по горизонтали.
- 3) Если метеор за левым краем экрана, то шаг 4. Иначе шаг 6.
- 4) Переносим метеор ровно за правый край.
- 5) Увеличиваем скорость полета метеора.
- 6) Конец.

Блок-схема следующая:

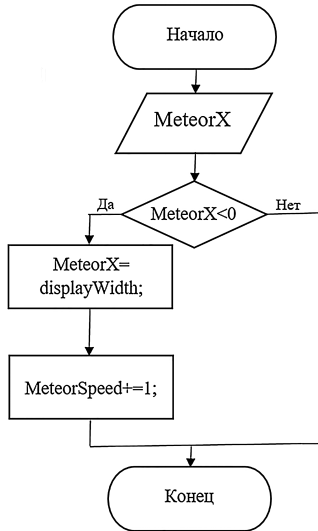


Рисунок 4.10 – Блок-схема ограничения движения метеора

Для написания кода определим, когда метеор выходит за край. Это происходит, когда левая часть метеора *MeteorX* становится левее, чем левый край экрана 0 или *MeteorX < 0*. Тогда надо вернуть левый край метеора к правому краю экрана или *MeteorX = displayWidth*; и ускорить движение метеора, увеличив его скорость *MeteorSpeed += 1*; . Перед прорисовкой метеора на экране пишется следующий код:

```

if(MeteorX < 0){
    MeteorX = displayWidth;
    MeteorSpeed += 1;
}
image(meteor, MeteorX, displayHeight/2-25*h, 50*w, 25*h);
  
```

Для того, чтобы метеор появлялся в случайном месте по вертикали экрана, необходима еще одна переменная:

```
float MeteorX, MeteorY;
```

Сразу задаем случайное местоположение метеора при запуске программы в *void setup()*:

```
MeteorY=random(0, displayHeight-25*h);
```

Здесь важно учесть, что нижняя граница для случайного расположения метеора не *displayHeight* (иначе метеор может оказываться ниже экрана), а выше границы на высоту метеора $25*h$. Поэтому и получилось *displayHeight-25*h*.

Добавим такую же строчку в код перемещения метеора:

```
if(MeteorX<0){  
  MeteorX=displayWidth;  
  MeteorY=random(0,displayHeight-25*h);  
  MeteorSpeed+=1;  
}
```

И теперь метеор будет каждый раз появляться справа на разной высоте, как видно на рисунке 4.11.



Рисунок 4.11 – Игра с полноценным движением корабля и метеора

4.6 Создание функций

На этом этапе разработки мы видим, что код получился довольно большой. Целых 46 строк с отступами:

```
float w,h;  
float ShipY;
```

```

float MeteorX, MeteorY;
float MeteorSpeed=5;
PImage sky, ship, meteor;

```

```

void setup(){
  size(displayWidth, displayHeight);
  w=displayWidth/500;
  h=displayHeight/500;
  ShipY=displayHeight/2-25*h;
  MeteorX=displayWidth;
  MeteorY=random(0,displayHeight-25*h);
  sky=loadImage(«sky.png»);
  ship=loadImage(«ship.png»);
  meteor=loadImage(«meteor.png»);
}

```

```

void draw(){
  image(sky,0,0,displayWidth, displayHeight);
  if(ShipY<0){
    ShipY=0;
  }
  if(ShipY+50*h>displayHeight){
    ShipY= displayHeight-50*h;
  }
  image(ship,10*w, ShipY, 100*w, 50*h);
  if(MeteorX<0){
    MeteorX=displayWidth;
    MeteorY=random(0,displayHeight-25*h);
    MeteorSpeed+=1;
  }
  image(meteor,MeteorX, MeteorY,50*w,25*h);
  ellipse(60*w, displayHeight-75*h, 40*w, 50*h);
  ellipse(displayWidth-60*w, displayHeight-75*h, 40*w, 50*h);
  MeteorX-=MeteorSpeed;
}

```

```

void mousePressed(){
  if ((sq(mouseX-60*w)/sq(20*w))+
  +sq(mouseY-(displayHeight-75*h))/sq(25*h))<=1){

```



```

ShipY-=10;
}
if ((sq(mouseX-(displayWidth-60*w))/sq(20*w)+
+sq(mouseY-(displayHeight-75*h))/sq(25*h)) <=1){
ShipY+=10;
}
}
}

```

Важно!!! Обратите внимание на пробелы после декларации переменных и каждого `void`. Они принципиальны не для среды, а для самих разработчиков. Такое форматирование позволяет легче читать и понимать код. Такая своеобразная «красная строка».

То же касается и отступов внутри каждой пары фигурных скобок `{}`. Такие отступы помогают понять, какая часть кода находится, в какой паре скобок. А кроме того, позволяют увидеть, где эти скобки открываются, а где – закрываются. Так легче отследить скобку, которая была не закрыта.

В полученном коде все работает так, как и планировалось. Но с каждым новым функционалом читать его становится все сложнее и сложнее. Для правильного форматирования кода профессиональные программисты рекомендуют создавать свои отдельные функции для каждой небольшой задачи. Давайте и научимся это делать.

Любые функции в Processing пишутся в таком порядке:

- 1) Тип данных, который эта функция возвращает.
- 2) Название функции.
- 3) В круглых скобках типы и названия параметров, которые функция использует.
- 4) В фигурных скобках действия, которые должны выполняться в функции.

В качестве примера возьмем системную переменную:

```

void setup(){
size(displayWidth, displayHeight);
}

```

Первой частью будет слово `void`, так как эта функция при выполнении не возвращает никакого значения в программу. Потом идет название `setup`. В круглых скобках `()` ничего не написано, так как

функция не требует никаких внешних параметров для выполнения. В фигурных скобках пишутся действия, которые должны выполняться – *{size(displayWidth, displayHeight);}*.

Давайте теперь вынесем прорисовку метеорита в отдельную функцию. Такая функция не будет возвращать никакого значения, поэтому прописываем слово *void*. Назовем эту функцию *Meteor*. Параметров такой функции не требуется, поэтому круглые скобки остаются пустыми. А в фигурные скобки переносим все команды из *void draw*, которые касаются рисования метеора:

```
void Meteor(){  
  if(MeteorX<0){  
    MeteorX=displayWidth;  
    MeteorY=random(0,displayHeight-25*h);  
    MeteorSpeed+=1;  
  }  
  image(meteor,MeteorX, MeteorY,50*w,25*h);  
  MeteorX-=MeteorSpeed;  
}
```

Также переносим в отдельную функцию прорисовку корабля. Пишем *void*, потом название *Ship* с пустыми фигурными скобками и вставляем все команды прорисовки корабля из *void draw*:

```
void Ship(){  
  if(ShipY<0){  
    ShipY=0;  
  }  
  if(ShipY+50*h>displayHeight){  
    ShipY= displayHeight-50*h;  
  }  
  image(ship,10*w, ShipY, 100*w, 50*h);  
}
```

Также поступаем и для прорисовки фона:

```
void Background(){  
  image(sky,0,0,displayWidth, displayHeight);  
}
```

Выносим в отдельную функцию и кнопки:

```
void ShipMovingButtons(){  
    ellipse(60*w, displayHeight-75*h, 40*w, 50*h);  
    ellipse(displayWidth-60*w, displayHeight-75*h, 40*w, 50*h);  
}
```

В итоге *draw()* состоит всего из четырех функций, где из названия абсолютно понятно, что именно они делают:

```
void draw(){  
    Background();  
    Ship();  
    Meteor();  
    ShipMovingButtons();  
}
```

Давайте также разнесем и *void setup()*:

```
void setup(){  
    size(displayWidth, displayHeight);  
    LocalSizeUnits();  
    InitialPositions();  
    LoadImages();  
}
```

Сначала вынесем, отдельно, расчет локальных единиц измерения размеров:

```
void LocalSizeUnits(){  
    w=displayWidth/500;  
    h=displayHeight/500;  
}
```

После этого, в отдельной функции, собираем расчет начальных значений всех переменных:

```
void InitialPositions(){  
    ShipY=displayHeight/2-25*h;
```

```

MeteorX=displayWidth;
MeteorY=random(0,displayHeight-25*h);
MeteorSpeed=5;
}

```

Последним блоком в отдельную функцию собираем загрузку всех изображений из папки data:

```

void LoadImages(){
sky=loadImage(«sky.png»);
ship=loadImage(«ship.png»);
meteor=loadImage(«meteor.png»);
}

```

И, конечно, также надо оформить и *mousePressed()*:

```

void mousePressed(){
ShipMovingButtonsPresssed();
}

```

А в отдельную функцию необходимо вынести нажатие на кнопки управления кораблем:

```

void ShipMovingButtonsPresssed(){
if((sq(mouseX-60*w)/sq(20*w)+
+sq(mouseY-(displayHeight-75*h))/sq(25*h))<=1){
ShipY-=10;
}
if ((sq(mouseX - (displayWidth - 60*w)) / sq(20*w) +
+sq(mouseY - (displayHeight -75*h)) / sq(25*h)) <= 1) {
ShipY+=10;
}
}
}

```

В итоге получается программный код, разделенный на четкие логические части. И, если понадобится, что-то скорректировать, то довольно легко будет найти нужную часть и конкретное место, где эти изменения внести.

Оффтоп. Такое разделение и форматирование можно проводить все детальнее и детальнее. Код, после корректировки, будет выглядеть следующим образом:

```
final int BUTTON_WIDTH = 40;
final int BUTTON_HEIGHT = 50;
final int SHIP_SPEED = 10;
final int METEOR_START_SPEED = 5;

final String BACKGROUND_IMAGE = «sky.png»;
final String SHIP_IMAGE = «ship.png»;
final String METEOR_IMAGE = «meteor.png»;

float xRatio,yRatio;
float ShipY;
float MeteorX, MeteorY;
float MeteorSpeed;
PImage sky, ship, meteor;

void setup(){
    size(displayWidth, displayHeight);
    initSizeUnits();
    initStartPositions();
    LoadImages();
}

void draw(){
    drawBackground();
    drawShip();
    drawMeteor();
    drawButtons();
}

void mousePressed(){
    boolean isMoveUpButtonPressed =
    =(sq(mouseX-60*xRatio)/sq(20*xRatio)+
    +sq(mouseY-(displayHeight-75*yRatio))/sq(25*yRatio))<=1;
    if(isMoveUpButtonPressed){
        moveShipUp();
    }
}
```

```

    boolean isMoveDownButtonPressed = (sq(mouseX-(displayWidth-
60*xRatio))/sq(20*xRatio)+sq(mouseY-(displayHeight-75*yRatio))/
sq(25*yRatio))<=1;
    if (isMoveDownButtonPressed){
        moveShipDown();
    }
}

```

```

void initSizeUnits(){
    xRatio=displayWidth/500;
    yRatio=displayHeight/500;
}

```

```

void initStartPositions(){
    moveShipToStart();
    moveMeteorToStart();
    MeteorSpeed=METEOR_START_SPEED;
}

```

```

void LoadImages(){
    sky=loadImage(BACKGROUND_IMAGE);
    ship=loadImage(SHIP_IMAGE);
    meteor=loadImage(METEOR_IMAGE);
}

```

```

void drawBackground(){
    image(sky,0,0,displayWidth, displayHeight);
}

```

```

void drawShip(){
    if(ShipY<0){
        ShipY=0;
    }
    if(ShipY+50*yRatio>displayHeight){
        ShipY= displayHeight-50*yRatio;
    }
    image(ship,10*xRatio, ShipY, 100*xRatio, 50*yRatio);
}

```

```

void drawMeteor(){
    if(MeteorX<0){
        moveMeteorToStart();
        increaseMeteorSpeed();
    }
    image(meteor,MeteorX, MeteorY,50*xRatio,25*yRatio);
    moveMeteor();
}

void drawButtons(){
    ellipse(60*xRatio, displayHeight-75*yRatio, BUTTON_WIDTH*xRatio,
    BUTTON_HEIGHT*yRatio);
    ellipse(displayWidth-60*xRatio, displayHeight-75*yRatio, BUTTON_
WIDTH*xRatio, BUTTON_HEIGHT*yRatio);
}

void moveShipUp(){
    ShipY-=SHIP_SPEED;
}

void moveShipDown(){
    ShipY+=SHIP_SPEED;
}

void moveShipToStart() {
    ShipY=displayHeight/2-25*yRatio;
}

void moveMeteor() {
    MeteorX-=MeteorSpeed;
}

void moveMeteorToStart() {
    MeteorX=displayWidth;
    MeteorY=random(0,displayHeight-25*yRatio);
}

void increaseMeteorSpeed() {
    MeteorSpeed+=1;
}

```

Как видите, при таком форматировании намного больше отдельных функций. Кроме этого, были использованы постоянные *final*. Их декларация и использование проводится также. Два ключевых отличия: перед их декларацией обязательно необходимо писать слово *final*; после записи значения в объем памяти, изменить его уже нельзя.

Кроме того, условия для попадания по кнопкам вынесены в локальные логические переменные *boolean* под названием: *isMoveUpButtonPressed* и *isMoveDownButtonPressed*.

4.7 Добавление объектов. Создание экрана проигрыша

Теперь для полноценной игры не хватает возможности проиграть, а для этого надо: добавить условие, когда метеор попадает на корабль; и создать экран проигрыша.

Алгоритм этой части программы будет выглядеть так:

- 1) Начало.
- 2) Считывание положения корабля и метеора.
- 3) Если положение метеора совпадает с положением корабля по горизонтали, то шаг 4. Иначе, шаг 6.
- 4) Если положение метеора совпадает с положением корабля по вертикали, то шаг 5. Иначе, шаг 6.
- 5) Запуск функции проигрыша.
- 6) Конец.

Блок-схема для такого алгоритма приведена на рисунке 4.12.

Как видно, здесь идет разделение на подфункции. Алгоритм для такой функции прописывается отдельно:

- 1) Начало функции проигрыша.
- 2) Остановка движения объектов.
- 3) Прорисовка взрыва во весь экран.
- 4) Считывание нажатия на экране.
- 5) Если на экран нажали, то шаг 6. Иначе шаг 4.
- 6) Возвращение параметра корабля и метеора к начальным значениям.
- 7) Конец.

Блок-схема для функции проигрыша показана на рисунке 4.13.

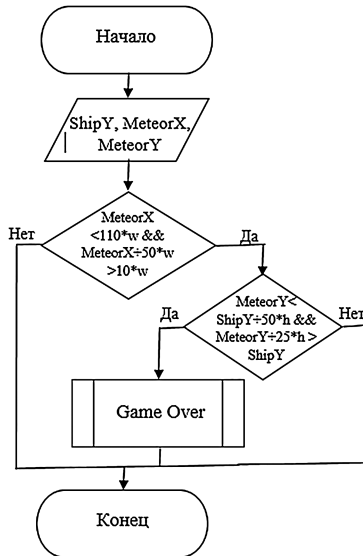


Рисунок 4.12 – Блок-схема условия проигрыша

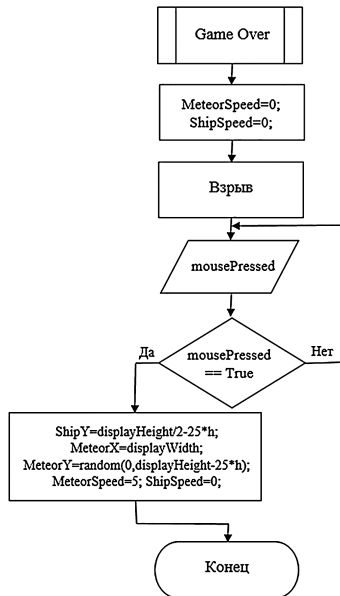


Рисунок 4.13 – Блок-схема функции проигрыша

Теперь преобразуем все это в код. Сначала прописываем условия попадания метеора в корабль по горизонтали. Здесь следуют два условия:

- 1) Чтобы левая сторона метеора *MeteorX* была левее $<$, чем правая сторона корабля $110*w$ (левая сторона корабля $10*w$ + ширина корабля $100*w$).
- 2) Чтобы правая сторона метеора $MeteorX+50*w$ (левая сторона метеора + ширина метеора) была правее $>$, чем левая сторона корабля $10*w$.

После этого условия попадания метеора в корабль по вертикали. Здесь условия также два:

- 1) Чтобы верхняя сторона метеора *MeteorY* была выше $<$, чем нижняя сторона корабля $ShipY+50*h$ (верхняя сторона корабля + высота корабля).
- 2) Чтобы нижняя сторона метеора $MeteorY+25*h$ (верхняя сторона метеора + высота метеора) была ниже $>$, чем верхняя сторона корабля *ShipY*.

Если эти условия выполняются, то запускается функция проигрыша. В итоге добавляем в draw():

```
if(MeteorX<110*w && MeteorX+50*w>10*w && MeteorY<ShipY+50*h  
&& MeteorY+25*h>ShipY){  
    GameOver();  
}
```

Для того, чтобы прописать функцию проигрыша, надо добавить и использовать переменные для скорости корабля и изображения взрыва:

```
float MeteorSpeed, ShipSpeed;  
PImage sky, ship, meteor, blow;
```

А еще подгрузить картинку со взрывом:

```
blow=loadImage(«blow.png»);
```

Теперь прописываем саму функцию:

```
void GameOver(){  
    Blow();
```

```
if(mousePressed){  
InitialPositions();  
}  
}
```

И здесь еще одна подфункция - функция взрыва:

```
void Blow(){  
MeteorSpeed=0;  
ShipSpeed=0;  
image(blow,10*w, ShipY, 200*w, 50*h);  
}
```

В итоге, при проигрыше, программа будет выглядеть как на рисунке 4.14.



Рисунок 4.14 – Экран проигрыша

Игра уже почти готова. В нее уже можно играть. Но, пока на экране только один метеор, это не так интересно. Поэтому, добавим вместо переменных для метеоров массивы:

```
float [] MeteorX=new float[3];  
float [] MeteorY=new float[3];
```

И везде, где переменные местоположения использовались, прописывается цикл для прогона каждого метеора по очереди. Для определения начальных значений:

```
for (int MeteorNum=0;MeteorNum<3;MeteorNum++){  
    MeteorX[MeteorNum]=displayWidth+MeteorNum*displayWidth/4;  
    MeteorY[MeteorNum]=random(0,displayHeight-25*h);  
}
```

Для прорисовки:

```
for (int MeteorNum=0;MeteorNum<3;MeteorNum++){  
    if(MeteorX[MeteorNum]<0){  
        MeteorX[MeteorNum]=displayWidth;  
        MeteorY[MeteorNum]=random(0,displayHeight-25*h);  
        MeteorSpeed+=1;  
    }  
    image(meteor,MeteorX[MeteorNum],  
        MeteorY[MeteorNum],50*w,25*h);  
    MeteorX[MeteorNum]-=MeteorSpeed;  
}
```

Для попадания по кораблю:

```
for (int MeteorNum=0;MeteorNum<3;MeteorNum++){  
    if(MeteorX[MeteorNum]<110*w && MeteorX[MeteorNum]+  
        +50*w>10*w && MeteorY[MeteorNum]<ShipY+  
        +50*h && MeteorY[MeteorNum]+25*h>ShipY){  
        GameOver();  
    }  
}
```

В итоге, программа будет как на рисунке 4.15.



Рисунок 4.15 – Экран проигрыша при массиве метеоров

Теперь, это уже полноценная игра. Но для ее усовершенствования можно еще добавить несколько «жизней» и количество пройденных метеоров.

4.8 Работа с текстом

Для вывода количества «жизней» и пройденных метеоров необходимо выводить текст. Ранее, команд для вывода текста нами не рассматривались. Рассмотрим команды, которые необходимы для решения возникших задач.

Поиск новых команд придется проводить на специализированном ресурсе www.processing.org. На этом сайте в правом верхнем углу есть инструмент поиска, который ограничивает зону поиска до описания типовых команд и примеров на самом сайте, и на специализированном форуме, посвященном этой среде разработки.

Так, чтобы найти команды для работы с текстом необходимо ввести слово `text`. В результате получим целый список ссылок, связанных с искомым словом. Откройте первую же ссылку про команду `text()` и перейдете на описание стандартной функции для вывода текста. Страница с описанием состоит из блоков: название, примеры использования, описание, синтаксис, параметры и другие команды, связанные с описанной командой.

Из описания для команды *text()* становится ясным, что в параметрах обязательно задаются выводимый текст (первым параметром) и расположение (от двух до пяти параметров).

Теперь применим эту команду для прорисовки количества «жизней» и очков:

```
void LivesAndScore(){  
  fill(255,0,0);  
  text(«Lives: «+Lives, 60*w, 75*h);  
  text(«Score: «+Score, displayWidth-60*w, 75*h);  
}
```

В коде введены новые переменные *Lives* и *Score*:

```
int Lives, Score;
```

Зададим им начальные значения в функции *InitialPositions()*:

```
Lives=4;  
Score=0;
```

Но при запуске полученного кода увидите (рисунок 4.16), что буквы очень мелкие и их почти не видно.



Рисунок 4.16 – Приложение с первоначальным выводом количества «жизней» и очков

Теперь обратимся к описанию команды на сайте. Там, в блоке со связанными командами, есть ссылка на команду *textSize()*. Применим ее:

```
void LivesAndScore(){  
  fill(255,0,0);  
  textSize(40);  
  text(«Lives: «+Lives, 30*w, 50*h);  
  text(«Score: «+Score, displayWidth-90*w, 50*h);  
}
```

На рисунке 4.17 видно, что надписи уже становятся читаемыми.



Рисунок 4.17 – Приложение с читаемым выводом количества «жизней» и очков

После того, как были выведены на экран количество «жизней» и счет, надо прописать логику изменения этих величин. Начнем со счета.

Количество очков будет увеличиваться на одно каждый раз, когда один из метеоров будет улетать за пределы экрана. А значит, этот код должен прописываться в условии выхода метеора за левый край:

```
if(MeteorX[MeteorNum]<0){  
  MeteorX[MeteorNum]=displayWidth;  
  MeteorY[MeteorNum]=random(0,displayHeight-25*h);  
  MeteorSpeed+=1;  
  Score+=1;  
}
```


Логика по количеству «жизней» немного сложнее. Сначала необходимо прописать уменьшение «жизней» при попадании метеора в корабль, вместо запуска экрана проигрыша. Но кроме этого, показать взрывом, что попадание произошло и перенести метеор обратно за правую часть экрана:

```
for (int MeteorNum=0;MeteorNum<3;MeteorNum++){  
if(MeteorX[MeteorNum]<110*w &&  
    MeteorX[MeteorNum]+50*w>10*w &&  
    MeteorY[MeteorNum]<ShipY+50*h &&  
    MeteorY[MeteorNum]+25*h>ShipY){  
Lives-=1;  
Blow();  
MeteorX[MeteorNum]=displayWidth;  
}  
}
```

Запуск функции проигрыша необходимо привязать к количеству «жизней». То есть, включать экран проигрыша, когда количество «жизней» становится меньше 0:

```
if (Lives==0){  
    GameOver();  
}
```

Игра готова к эксплуатации – рисунок 4.18.



Рисунок 4.18 – Полноценная игра с жизнями и очками

Теперь можно загрузить созданную игру на смартфон или планшет. Вы – разработчик!

4.9 Создание меню

Один из важных элементов игры является начальное меню. Если его прописать, то получится три типа интерфейсов: начальное меню, экран самого игрового процесса и экран проигрыша.

Для прописывания трех логических частей приложения лучше всего подходит программный инструмент «переключение вариантов» или по-английски *switch case*. Для такого переключения понадобится дополнительная переменная:

```
String MenuType;
```

В функции записи начальных значений записываем начальное значение для этой переменной:

```
MenuType=»StartMenu»;
```

Теперь в *draw()* прописываем переключение между тремя типами меню:

```
void draw(){  
  switch (MenuType){  
    case «StartMenu»:  
      StartMenu();  
      break;  
    case «Game»:  
      Game();  
      break;  
    case «GameOver»:  
      GameOver();  
      break;  
  }  
}
```

Все что было в *draw()* переносим в новую функцию *Game()*:

```

void Game(){
    Background();
    Ship();
    Meteor();
    ShipMovingButtons();
    for (int MeteorNum=0;MeteorNum<3;MeteorNum++){
        if(MeteorX[MeteorNum]<110*w &&
            MeteorX[MeteorNum]+50*w>10*w &&
            MeteorY[MeteorNum]<ShipY+50*h &&
            MeteorY[MeteorNum]+25*h>ShipY){
            Lives-=1;
            Blow();
            MeteorX[MeteorNum]=displayWidth;
        }
    }
    LivesAndScore();
    if (Lives==0){
        MenuType=>>GameOver»;
    }
}

```

Создадим функцию прорисовки меню:

```

void StartMenu(){
    background(0);
    MenuButtons();
}

void MenuButtons(){
    fill(255);
    rect(displayWidth/2-50*w,displayHeight/2-75*h, 100*w, 50*h);
    rect(displayWidth/2-50*w,displayHeight/2+75*h, 100*w, 50*h);
    fill(255,0,0);
    text(«Game»,displayWidth/2-20*w,displayHeight/2-45*h);
    text(«Exit»,displayWidth/2-15*w,displayHeight/2+108*h);
}

```

Так же разделяем на подменю и функцию *mousePressed()*:

```

void mousePressed(){
  switch (MenuType){
    case «StartMenu»:
      MenuButtonsPresssed();
      break;
    case «Game»:
      ShipMovingButtonsPresssed();
      break;
    case «GameOver»:
      InitialPositions();
      break;
  }
}

```

Кроме того нужна новая функция нажатия на кнопки меню:

```

void MenuButtonsPresssed(){
  if(mouseX>displayWidth/2-50*w && mouseX<displayWidth/2+50*w
  && mouseY>displayHeight/2-75*h && mouseY<displayHeight/2-25*h){
    MenuType=»Game»;
  }
  if(mouseX>displayWidth/2-50*w && mouseX<displayWidth/2+50*w
  && mouseY>displayHeight/2+75*h && mouseY<displayHeight/2+125*h){
    exit();
  }
}

```

Приложение теперь состоит из трех меню: начальное меню (рисунок 4.19), сама игра (рисунок 4.20) и проигрыш.

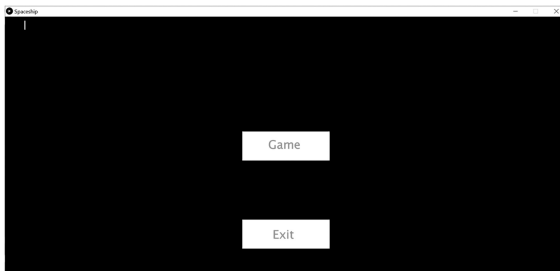


Рисунок 4.19 – Меню начала



Рисунок 4.20 – Игровой режим

Теперь игра готова полностью. Остаются только пару моментов по оформлению.

4.10 Последние шаги перед выгрузкой в PlayMarket

Прежде чем представить приложение для выпуска, необходимо установить для него иконки, поскольку Processing не будет использовать стандартные значки по умолчанию для выпуска приложения. Очень важно, чтобы было выбрано имя пакета, которое будет однозначно идентифицировать приложение в PlayMarket.

В режиме Android для приложения используется набор значков по умолчанию. Перед выгрузкой следует поставить собственные иконки, создав файлы `icon-36`, `icon-48`, `icon-72`, `icon-96`, `icon-144` и `icon-192` в формате `.PNG` и поместить их в папку с кодом. Затем проводить экспорт подписанного пакета.

Имя пакета представляет собой строку текста, которая часто выглядит как `com.example.helloworld`, и соответствует соглашению Java о наименовании пакетов (<https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html>).

Имя приложения (`helloworld`) пишется последним перед веб-сайтом компании или частного лица, разрабатывающим приложение, написанным в обратном порядке (`com.example`). Processing создает это имя пакета автоматически, путем предварительного присвоения имени `«processing.test»`.

Но можно задать собственное имя пакета, отредактировав файл манифеста, создаваемый программой Processing в папке эскиза после его первого запуска из PDE (на устройстве или в эмуляторе). Также можно задать код версии и имя версии. Например, в следующем файле манифеста имя пакета – «com.example.helloworld», код версии 1 и имя версии 0.5.4:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="10"
    android:versionName="0.5.4" package="com.example.
    helloworld">
    <uses-sdk android:minSdkVersion="16"
    android:targetSdkVersion="26"/>
    <application android:icon="@drawable/icon" android:label="">
        <activity android:name=".MainActivity"
            android:theme="@style/Theme.AppCompat.Light.
            NoActionBar.FullScreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.
                LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Обратите внимание, что название пакета приложения должно быть уникально, так как в PlayMarket не может быть двух приложений с одинаковым именем пакета.

Режим Android упрощает публикацию кода, подписывая и выравнивая приложение, поэтому можно загрузить его в консоль Google Play Developer Console без дополнительных действий. Все, что нужно сделать, это выбрать «Export Signed Package» (Экспортировать подписанный пакет) в меню Файл (рисунок 4.21).

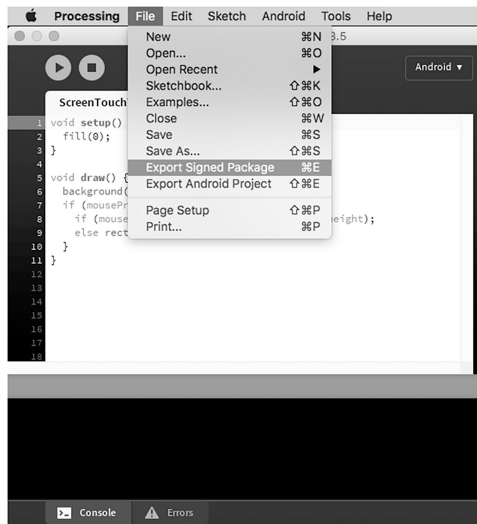


Рисунок 4.21 – Выгрузка пакета приложения

После выбора этого подменю Processing попросит создать новое хранилище ключей выпуска для подписания пакета приложения (рисунок 4.22).

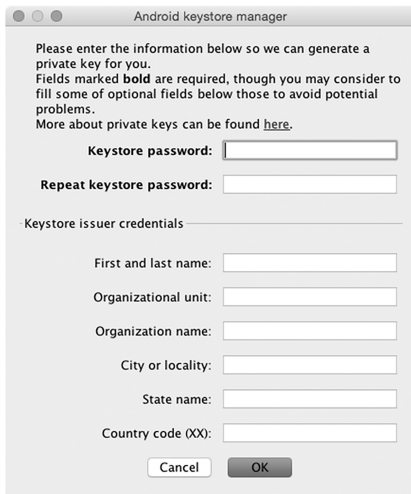


Рисунок 4.22 – Создание хранилища ключей выпуска

Необходимо запомнить этот пароль, поскольку использовать его надо каждый раз, когда будет проходить операция экспорта. Даже при том, что разработчик может перезагрузить его и создать новый ключ, пока приложение связано с тем же именем пакета у него должен быть тот же ключ выпуска.

Подписанный пакет будет сохранен в подпапке в папке с кодом, под именем [Название эскиза в строчных буквах]_release_signed_aligned.apk. Когда Вы создали этот файл, можно закончить процесс публикации приложения в PlayMarket следуя инструкциям от Google (<https://developer.android.com/distribute/googleplay/start.html>).

5 РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ В ANDROID STUDIO

5.1 Создание нового проекта

В этой главе будет краткое описание проекта на Android Studio. Причем, на самом современном языке для мобильных разработок – на **Kotlin**.

Начинается все с включения среды разработки и запуска нового проекта – рисунок 5.1. Конечно нажимаем на экране + **Start a new Android Studio project**.

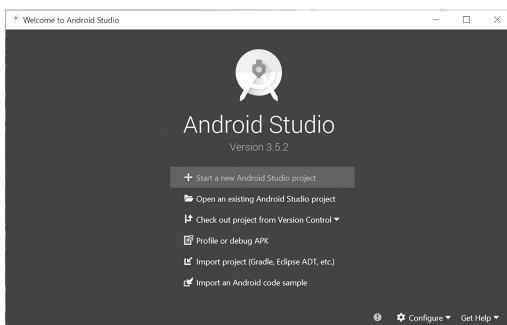


Рисунок 5.1 – Включение Android Studio

При запуске среда предлагает несколько вариантов шаблонов для приложения – рисунок 5.2. Но для этого примера надо выбрать абсолютно пустой проект или **Add No Activity**.

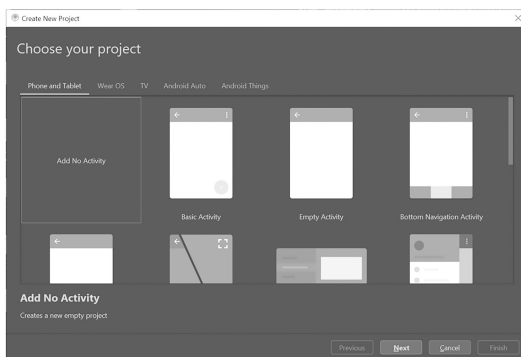


Рисунок 5.2 – Варианты начальных шаблонов

После выбора шаблона надо заполнить и выбрать параметры конфигурации проекта – рисунок 5.3.

В первую очередь, необходимо задать название. В этом примере название *SpaceShip*.

Потом создаем название пакета. В случае реального проекта это название должно быть уникальным, чтобы не конфликтовать с другими приложениями в PlayMarket. В нашей примере запишите *kz.alt.spaceship*.

Следующее поле – это адрес папки для сохранения проекта. Здесь принципиально важно, чтобы все части адреса были на латинице.

В поле, ниже, выбираем *Kotlin* как язык разработки.

Остальные поля оставляем по умолчанию.

Нажимаем кнопку *Finish* (Закончить).

***Важно!!!** Еще раз стоит отметить, что на разных этапах разработки, начиная с запуска проекта, Android Studio будет загружать из Интернета различные данные для полноценной работы. А, значит, Вам принципиально важен Интернет с хорошей скоростью и желательно без лимитный в течение всей разработки.*

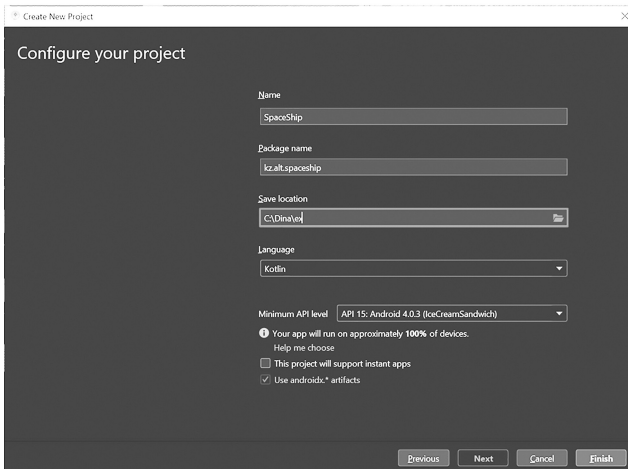


Рисунок 5.3 – Параметры конфигурации проекта

Теперь Android Studio надо дать время для загрузки всех компонентов. В результате получим окно, как на рисунок 5.4.

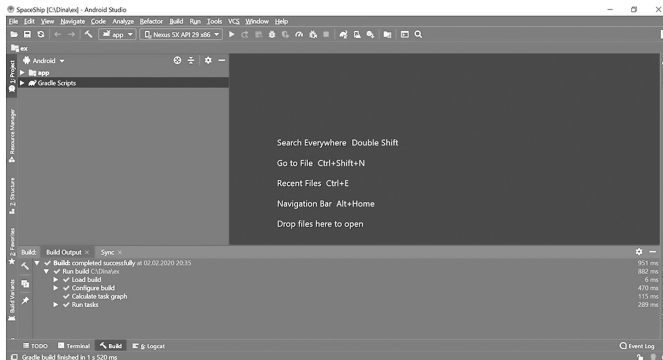


Рисунок 5.4 – Окно пустого проекта

Все готово к началу разработки. Теперь дело непосредственно за программированием.

5.2 Редактирование манифеста

В корневой папке каждого приложения должен находиться файл `AndroidManifest.xml`. Файл манифеста содержит важную информацию о приложении, которая требуется системе Android. Только получив эту информацию, система может выполнить какой-либо код приложения. Среди прочего файл манифеста выполняет следующие действия:

- 1) Он задает имя пакета Java для приложения. Это имя пакета служит уникальным идентификатором приложения.
- 2) Он описывает компоненты приложения – операции, службы, приемники широкопередаточных сообщений и поставщиков контента, из которых состоит приложение. Он содержит имена классов, которые реализуют каждый компонент, и публикует их возможности (указывает, например, какие сообщения `Intent` они могут принимать). На основании этих деклараций система Android может определить, из каких компонентов состоит приложение и при каких условиях их можно запускать.
- 3) Он определяет, в каких процессах будут размещаться компоненты приложения.
- 4) Он объявляет, какие разрешения должны быть выданы приложению, чтобы оно могло получить доступ к защищенным частям API-интерфейса и взаимодействовать с другими приложениями.

- 5) Он также объявляет разрешения, требуемые для взаимодействия с компонентами данного приложения.
- 6) Он содержит список классов Instrumentation, которые при выполнении приложения предоставляют сведения о профиле и прочую информацию. Эти объявления присутствуют в файле манифеста только во время разработки и отладки приложения и удаляются перед его публикацией.
- 7) Он объявляет минимальный уровень API-интерфейса Android, который требуется приложению.
- 8) Он содержит список библиотек, с которыми должно быть связано приложение.

Итак, файл манифеста приложения содержит важную базовую информацию о приложении, без которой система просто не сможет запустить программу на смартфоне.

Для его редактирования в левой части открытого пустого проекта найдите иерархическое описание элементов проекта и раскройте папку `app` и подпапку `manifests`. Дважды щелкните на файл `AndroidManifest.xml`. Теперь в правой части у Вас откроется окно с кодом – рисунок 5.5.

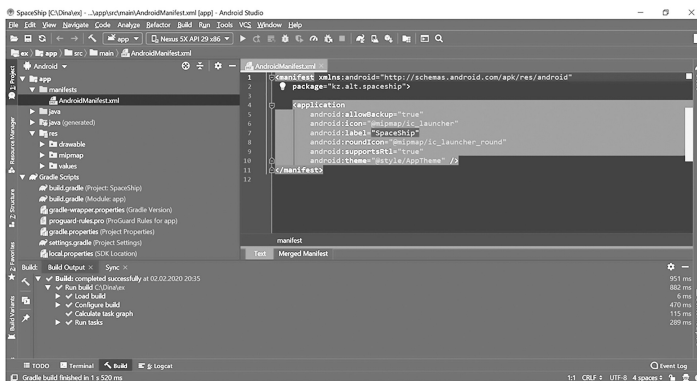


Рисунок 5.5 – Изменение файла `AndroidManifest.xml`

Этот код нужно дополнить так, чтобы получилось следующее:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kz.alt.spaceship">
    <application
        android:allowBackup="true">
```

```

android:icon=>@mipmap/ic_launcher»
android:label=>@string/app_name»
android:roundIcon=>@mipmap/ic_launcher_round»
android:supportsRtl=>true»
android:theme=>@style/AppTheme» >

```

```

<activity android:name=>.presentation.MainActivity» android:sc
reenOrientation=>landscape»>
  <intent-filter>
    <action android:name=>android.intent.action.MAIN» />
    <category android:name=>android.intent.category.
LAUNCHER» />
  </intent-filter>
</activity>
</application>
</manifest>

```

После редактирования манифеста можно перейти к файлам, выполняющим базовую логику проекта.

5.3 Добавление и редактирование файлов приложения

Перед созданием самих файлов создадим пакет для их хранения.

Для этого вначале выключим в настройках иерархического представления **Compact Middle Packages**, что позволит увидеть папки на всех уровнях – рисунок 5.6.

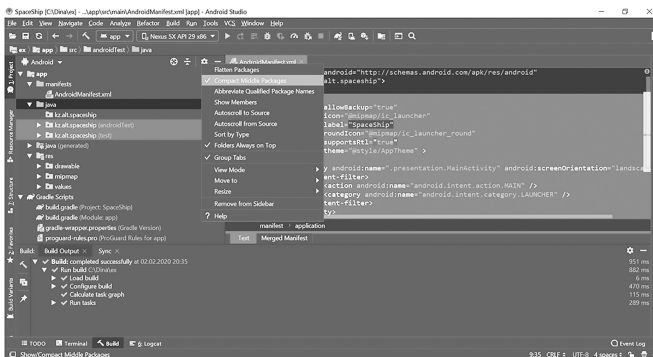


Рисунок 5.6 – Выключение Compact Middle Packages

Теперь раскрыв папки **kz**, **alt**, нажмите правой кнопкой на папку **spaceship** и выберите строку **New**, а потом **Package** – рисунок 5.7.

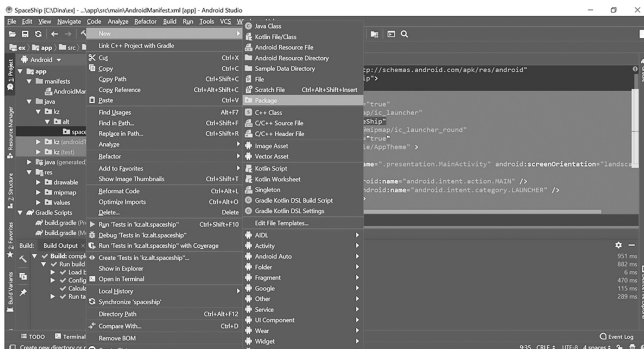


Рисунок 5.7 – Запрос на создание новой папки

Появится всплывающее окно с просьбой ввести имя папки – рисунок 5.8. Для этого примера сначала создадим папку с названием **domain**.

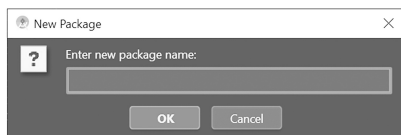


Рисунок 5.8 – Ввод имени новой папки в проекте

Уже в этой папке надо создать непосредственно файл. Для этого нажмите правой кнопкой мыши на папке **domain** и выберите **New**, а потом **Kotlin File/Class** – рисунок 5.9.

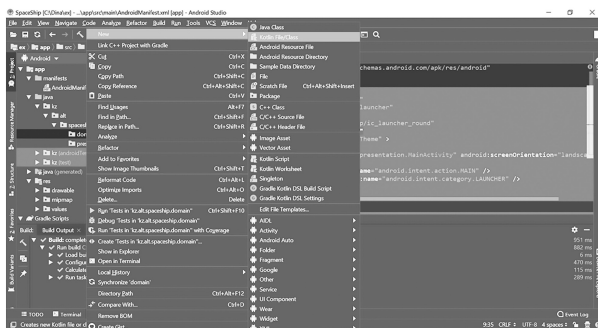


Рисунок 5.9 – Создание нового файла

Так же, как и для папки, возникнет всплывающее окно с просьбой написать название и выбрать тип файла – рисунок 5.10. Первый файл назовите **GameDrawer**. Тип оставьте по умолчанию **File**.

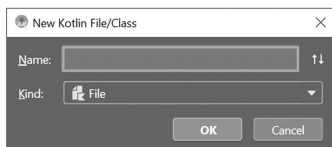


Рисунок 5.10 – Ввод имени и типа нового файла в проекте

Два раза нажав на созданный файл в иерархии Вы откроете его код в правой части среды разработки – рисунок 5.11.

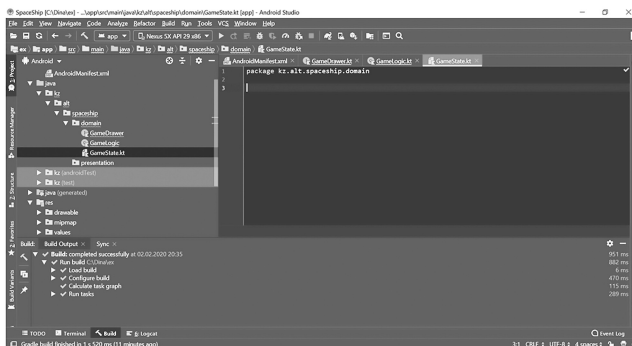


Рисунок 5.11 – Открытый новый файл

Теперь для файла **GameDrawer** введите следующий код:

```
package kz.alt.spaceship.domain
interface GameDrawer {
    fun drawGame(gameState: GameState)
}

```

Также создайте в папке **domain** файл **GameLogic** и введите следующее:

```
package kz.alt.spaceship.domain
import kotlin.random.Random

```

```

class GameLogic(
    private val displayDencity: Float
) {

    private val SPACE_SHIP_TOUCH_SPEED = 4.toPx
    private val SPACE_SHIP_WIDTH = 90.toPx
    private val SPACE_SHIP_HEIGHT = 34.toPx
    private val SPACE_SHIP_START_X = 32.toPx
    private val METEOR_WIDTH = 150.toPx
    private val METEOR_HEIGHT = 40.toPx
    private val METEOR_START_SPEED = 5.toPx
    private val METEOR_INCREASE_SPEED = 3.toPx

    private var displayWidth = 0
    private var displayHeight = 0
    private var gameState: GameState = GameState.Start
    private var isSpaceShipMovingUp: Boolean = false
    private var isSpaceShipMovingDown: Boolean = false
    private lateinit var presenter: GameStatePresenter

    fun init(
        presenter: GameStatePresenter,
        displayWidth: Int,
        displayHeight: Int
    ) {
        this.presenter = presenter
        this.displayWidth = displayWidth
        this.displayHeight = displayHeight
    }

    fun start() {
        gameState = GameState.Game(
            spaceShip = makSpaceShip(),
            meteors = makeMeteors()
        )
        updateGameState()
    }

    fun cycle() {

```

```

    val gameState = gameState
    if (gameState !is GameState.Game) return

    moveSpaceShip()
    move Meteors()
    if (isAnyMeteorHit()) {
        this.gameState = GameState.GameOver(gameState)
    }
    updateGameState()
}

fun onStartMovingSpaceShipUp() {
    isSpaceShipMovingUp = true
}

fun onStopMovingSpaceShipUp() {
    isSpaceShipMovingUp = false
}

fun onStartMovingSpaceShipDown() {
    isSpaceShipMovingDown = true
}

fun onStopMovingSpaceShipDown() {
    isSpaceShipMovingDown = false
}

private fun moveSpaceShip() {
    val gameState = gameState
    if (gameState !is GameState.Game) return

    val spaceShip = gameState.spaceShip
    if (isSpaceShipMovingDown) {
        spaceShip.yPos += SPACE_SHIP_TOUCH_SPEED
        if (spaceShip.yPos + spaceShip.height >= displayHeight) {
            spaceShip.yPos = displayHeight - spaceShip.height
        }
    }
    if (isSpaceShipMovingUp) {

```



```

        spaceShip.yPos -= SPACE_SHIP_TOUCH_SPEED
        if (spaceShip.yPos <= 0) spaceShip.yPos = 0
    }
}

private fun updateGameState() {
    presenter.onGameStateUpdate(gameState)
}

private fun makSpaceShip(): SpaceShip = SpaceShip(
    width = SPACE_SHIP_WIDTH,
    height = SPACE_SHIP_HEIGHT,
    xPos = SPACE_SHIP_START_X,
    yPos = displayHeight / 2 - SPACE_SHIP_HEIGHT / 2
)

private fun makeMeteors(): List<Meteor> {
    val meteors = mutableListOf<Meteor>()
    repeat(2) {
        meteors.add(
            Meteor(
                xPos = displayWidth,
                yPos = getMeteorStartY(),
                height = METEOR_HEIGHT,
                width = METEOR_WIDTH,
                isVisible = true,
                speed = Random.nextInt(METEOR_START_SPEED,
METEOR_START_SPEED * 2)
            )
        )
    }

    return meteors
}

private fun moveMeteorToStart(
    meteor: Meteor
) {
    meteor.xPos = displayWidth

```

```

    meteor.yPos = getMeteorStartY()
}

private fun getMeteorStartY(): Int = Random.nextInt(
    from = METEOR_HEIGHT,
    until = displayHeight - METEOR_HEIGHT
)

```

```

private fun moveMeteors() {
    val gameState = gameState
    if (gameState !is GameState.Game) return

    gameState.meteors.forEach {
        moveMeteor(
            meteor = it
        )
    }
}

```

```

private fun moveMeteor(
    meteor: Meteor
) {
    meteor.xPos -= meteor.speed
    val rightMeteorSide = meteor.xPos + meteor.width
    if (rightMeteorSide < 0) {
        meteor.speed += Random.nextInt(METEOR_INCREASE_
SPEED)
        moveMeteorToStart(meteor)
    }
}

```

```

private fun isAnyMeteorHit(): Boolean {
    val gameState = gameState
    if (gameState !is GameState.Game) return false

    return gameState.meteors.any {
        isMeteorHit(gameState.spaceShip, it)
    }
}

```

```

private fun isMeteorHit(
    spaceShip: SpaceShip,
    meteor: Meteor
): Boolean {
    val rightSpaceShipSide = spaceShip.xPos + spaceShip.width
    val leftSpaceShipSide = spaceShip.xPos
    val topSpaceShipSide = spaceShip.yPos
    val bottomSpaceShipSide = spaceShip.yPos + spaceShip.height

    val rightMeteorSide = meteor.xPos + meteor.width
    val leftMeteorSide = meteor.xPos
    val topMeteorSide = meteor.yPos
    val bottomMeteorSide = meteor.yPos + meteor.height

    val isMeteorHitByX = leftMeteorSide in (leftSpaceShipSide..
rightSpaceShipSide)
        || rightMeteorSide in (leftSpaceShipSide..rightSpaceShipSide)
    val isMeteorHitByY = topMeteorSide in (topSpaceShipSide..
bottomSpaceShipSide)
        || bottomMeteorSide in (topSpaceShipSide..
bottomSpaceShipSide)

    return isMeteorHitByX && isMeteorHitByY
}

private val Int.toPx: Int
    get() = (this * displayDencity).toInt()
}

```

Так же в папке *domain* Вам понадобится файл *GameState* со следующим кодом:

```

package kz.alt.spaceship.domain

sealed class GameState {
    data class Game(
        val spaceShip: SpaceShip,
        val meteors: List<Meteor>
    ) : GameState()
}

```

```
data class GameOver(  
    val game: Game  
) : GameState()  
  
object Start : GameState()  
}
```

А еще файл *GameStatePresenter* с такой программой:

```
package kz.alt.spaceship.domain  
  
interface GameStatePresenter {  
    fun onGameStateUpdate(gameState: GameState)  
}
```

Ну и конечно файл для данных о метеоре *Meteor*:

```
package kz.alt.spaceship.domain  
  
data class Meteor(  
    val width: Int,  
    val height: Int,  
    var speed: Int,  
    var xPos: Int,  
    var yPos: Int,  
    var isVisible: Boolean  
)
```

И данные о корабле *SpaceShip*:

```
package kz.alt.spaceship.domain  
  
data class SpaceShip(  
    val width: Int,  
    val height: Int,  
    val xPos: Int,  
    var yPos: Int  
)
```

Теперь можно перейти к коду, описывающему визуализацию логики.

5.4 Создание и редактирование файлов для визуализации

Теперь в папке *spaceship* надо создать еще одну папку с названием *presentation*. А в ней два файла.

Первый файл будет называться *MainActivity* и должен содержать такой код:

```
package kz.alt.spaceship.presentation

import android.app.Activity
import android.app.AlertDialog
import android.content.res.Resources
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.view.Window
import android.view.WindowManager
import kotlinx.coroutines.*
import kz.alt.spaceship.R
import kz.alt.spaceship.domain.GameDrawer
import kz.alt.spaceship.domain.GameLogic
import kz.alt.spaceship.domain.GameState
import kz.alt.spaceship.domain.GameStatePresenter
import kotlin.coroutines.CoroutineContext

private const val LOOP_DELAY_MS: Long = 32

class MainActivity : Activity(),
    GameStatePresenter,
    CoroutineScope
{

    private val uiContext = Dispatchers.Main
    override val coroutineContext: CoroutineContext
        get() = uiContext

    private val gameLogic: GameLogic = GameLogic(
        displayDencity = Resources.getSystem().displayMetrics.density
    )
}
```

```

private lateinit var gameDrawer: GameDrawer

private var loopJob: Job = Job()

private lateinit var leftButton: View
private lateinit var rightButton: View
private lateinit var contentLayout: View

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    requestWindowFeature(Window.FEATURE_NO_TITLE)
    window.setFlags(WindowManager.LayoutParams.FLAG_
FULLSCREEN,
    WindowManager.LayoutParams.FLAG_FULLSCREEN)
    setContentView(R.layout.activity_main)

    initView()
    contentLayout.post {
        gameLogic.init(
            presenter = this,
            displayWidth = contentLayout.width,
            displayHeight = contentLayout.height
        )
        gameDrawer = contentLayout as ViewGameDrawer
        startGame()
    }
}

override fun onGameStateUpdate(gameState: GameState) {
    when (gameState) {
        is GameState.Start -> {}
        is GameState.GameOver -> {
            handleGame(gameState.game)
            showGameOver()
            stopGame()
        }
        is GameState.Game -> {
            handleGame(gameState)
        }
    }
}

```

```

    }
}

private fun startGame() {
    gameLogic.start()
    startLoop()
}

private fun stopGame() {
    stopLoop()
}

private fun handleGame(gameState: GameState.Game) {
    gameDrawer.drawGame(gameState)
}

private fun initView() {
    leftButton = findViewById(R.id.left_button)
    leftButton.setOnTouchListener(MoveSpaceShipUpTouchListener())
    rightButton = findViewById(R.id.right_button)
    rightButton.setOnTouchListener(MoveSpaceShipDownTouchListener())
    contentLayout = findViewById(R.id.content_layout)
}

private fun startLoop() {
    loopJob.cancel()
    loopJob = launch {
        while (true) {
            delay(LOOP_DELAY_MS)

            gameLogic.cycle()
        }
    }
}

private fun stopLoop() {
    loopJob.cancel()
}

```

```

private fun showGameOver() {
    AlertDialog.Builder(this)
        .setTitle(«Вы проиграли! Начать заново?»)
        .setPositiveButton(«Да») { _, _ ->
            startGame()
        }
        .setNegativeButton(«Нет») { _, _ ->
            finish()
        }
        .setCancelable(false)
        .create()
        .show()
}

```

```

    internal inner class MoveSpaceShipUpTouchListener : View.
    OnTouchListener {

        override fun onTouch(v: View?, event: MotionEvent?): Boolean {
            when (event?.action) {
                MotionEvent.ACTION_DOWN -> gameLogic.
onStartMovingSpaceShipUp()
                MotionEvent.ACTION_UP -> gameLogic.
onStopMovingSpaceShipUp()
            }

            return true
        }
    }
}

```

```

    internal inner class MoveSpaceShipDownTouchListener : View.
    OnTouchListener {

        override fun onTouch(v: View?, event: MotionEvent?): Boolean {
            when (event?.action) {
                MotionEvent.ACTION_DOWN -> gameLogic.
onStartMovingSpaceShipDown()
                MotionEvent.ACTION_UP -> gameLogic.
onStopMovingSpaceShipDown()
            }
        }
    }
}

```



```

        return true
    }
}
}

```

Во второй файл с именем *ViewGameDrawer* надо прописать следующее:

```

package kz.alt.spaceship.presentation

import android.content.Context
import android.graphics.Canvas
import android.graphics.drawable.Drawable
import android.util.AttributeSet
import android.widget.FrameLayout
import androidx.core.content.ContextCompat
import kz.alt.spaceship.R
import kz.alt.spaceship.domain.GameDrawer
import kz.alt.spaceship.domain.GameState
import kz.alt.spaceship.domain.Meteor
import kz.alt.spaceship.domain.SpaceShip

class ViewGameDrawer @JvmOverloads constructor(
    context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0
) : FrameLayout(context, attrs, defStyleAttr),
    GameDrawer
{
    private val spaceShipDrawable: Drawable = ContextCompat.
getDrawable(context,
        R.drawable.ic_space_ship
    )!!
    private val meteorDrawable: Drawable = ContextCompat.
getDrawable(context,
        R.drawable.ic_meteor
    )!!
    private var gameState: GameState = GameState.Start

    init {
        setBackgroundResource(R.drawable.bg_main)
    }
}

```

```

override fun onDraw(canvas: Canvas) {
    val gameState = this.gameState as? GameState.Game ?: return

    drawSpaceShip(canvas, gameState.spaceShip)
    drawMeteors(canvas, gameState.meteors)
}

override fun drawGame(gameState: GameState) {
    this.gameState = gameState
    invalidate()
}

private fun drawSpaceShip(
    canvas: Canvas,
    spaceShip: SpaceShip
) {
    val drawable = spaceShipDrawable

    drawable.setBounds(
        spaceShip.xPos,
        spaceShip.yPos,
        spaceShip.xPos + spaceShip.width,
        spaceShip.yPos + spaceShip.height
    )
    drawable.draw(canvas)
}

private fun drawMeteors(
    canvas: Canvas,
    meteors: List<Meteor>
) {
    meteors.forEach {
        drawMeteor(canvas, it)
    }
}

private fun drawMeteor(
    canvas: Canvas,
    meteor: Meteor
) {

```

```
val drawable = meteorDrawable
```

```
drawable.setBounds(  
    meteor.xPos,  
    meteor.yPos,  
    meteor.xPos + meteor.width,  
    meteor.yPos + meteor.height  
)  
drawable.draw(canvas)  
}  
}
```

В итоге Вы должны получить иерархию папок и файлов как на рисунке 5.12.

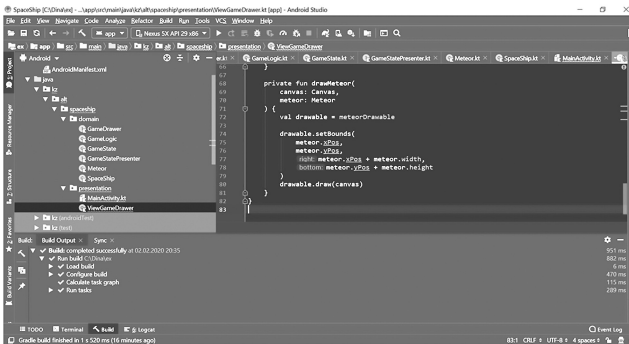


Рисунок 5.12 – Проект со всеми необходимыми функциями

Для полноценной работы приложения надо добавить ряд ресурсных файлов.

5.5 Добавление ресурсных файлов

В первую очередь необходимо добавить рисунки для звездного неба на фоне, летящего метеора и корабля. Можно использовать те же файлы, что и для проекта в среде *Processing* или найти другие. Принципиально важно назвать их *bg_main.jpg*, *ic_meteor.jpg*, *ic_space_ship.jpg* соответственно. Это важно, так как в коде, описанном выше, они обозначены именно так.

Для того, чтобы вставить их в проект, находим в иерархии в среде разработки папку *res* и, открыв ее, видим папку *drawable*. Теперь копируем по очереди каждый из рисунков из той папки, где Вы их сохранили и, нажав правой кнопкой мыши на папку *drawable*, вставляем рисунки. Всплывет окно как на рисунке 5.13. Выберем директорию по умолчанию, то есть *...\app\src\main\res\drawable*.

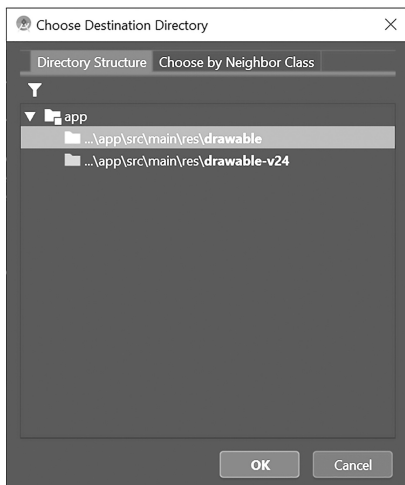


Рисунок 5.13 – Выбор директории для сохранения рисунка

Если Вы не изменили название файлов перед копированием, можете это сделать в следующем всплывающем окне – рисунок 5.14.

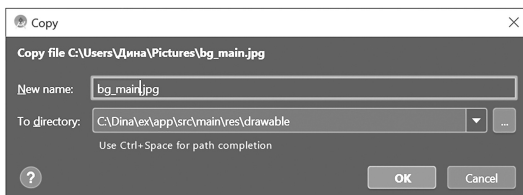


Рисунок 5.14 – Ввод названия для вставляемых изображений

Но кроме фона, метеора и корабля в приложении присутствуют две кнопки. Их надо добавить не картинкой, а файлом. Для этого необходимо нажать правой кнопкой на папку *drawable* и выбрать строку *New*, а потом *Drawable resource file* – рисунок 5.15.

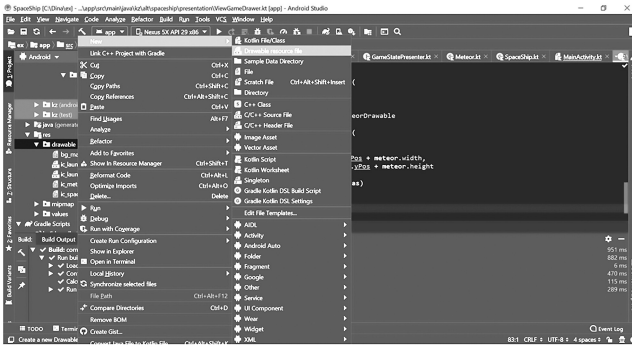


Рисунок 5.15 – Вставка ресурсного файла

Во всплывающем окне достаточно ввести имя **bg_button.xml**, а остальные параметры оставить по умолчанию – рисунок 5.16.

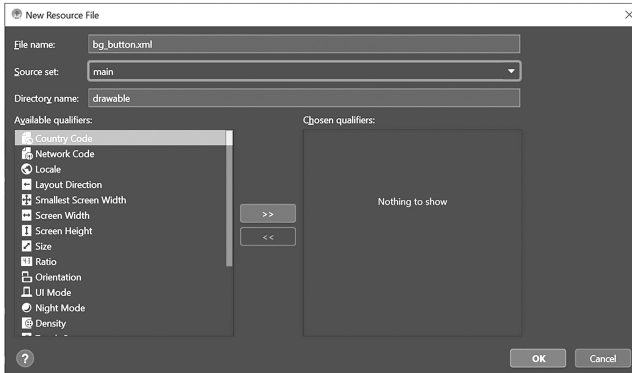


Рисунок 5.16 – Ввод параметров нового ресурсного файла

Теперь дважды нажмите на новый ресурсный файл **bg_button.xml** в правой части и введите вот такой код:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval"
    >
```

`<solid android:color=@color/button>>`

`</shape>`

В результате вставки всех изображений Вы получите проект – рисунок 5.17.

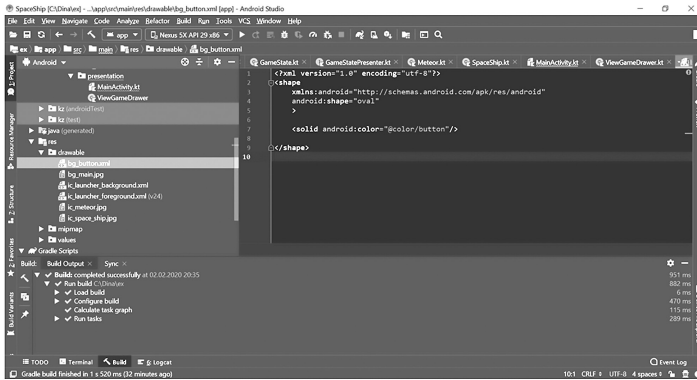


Рисунок 5.17 – Проект с добавленными изображениями

Но этого недостаточно. Надо еще прописать компоновку всех элементов. Для этого в папке `res` создаем новую Android директорию, нажав правой кнопкой мыши и выбрав **New, Android Resource Directory** – рисунок 5.18.

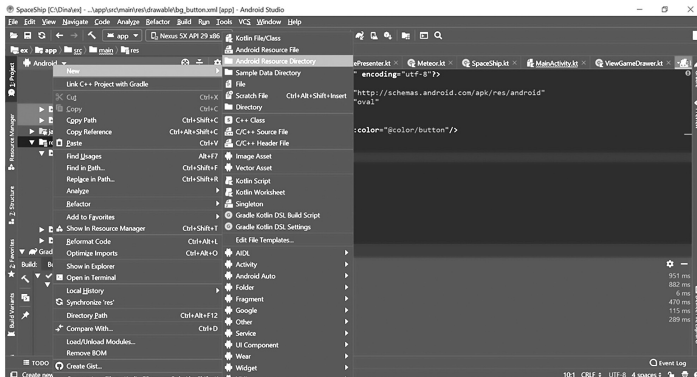


Рисунок 5.18 – Создание новой Android директории

Во всплывающем окне во второй вкладке выбираем `layout` – рисунок 5.19, что означает макет.

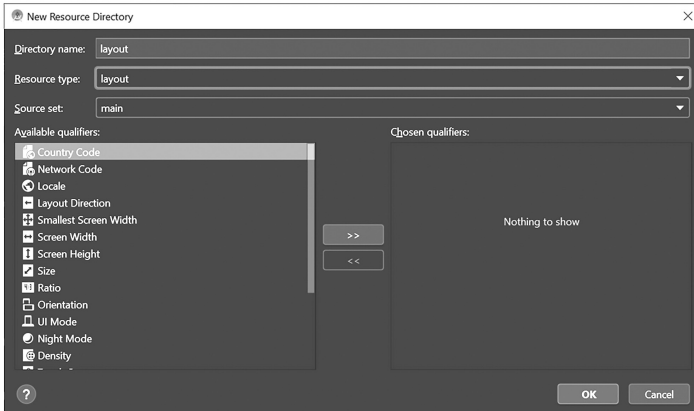


Рисунок 5.19 – Создание нового макета

Нажав, на только что созданную папку `layout`, правой кнопкой мыши запрашиваем создание ресурсного файла через `New, Layout resource file` – рисунок 5.20.

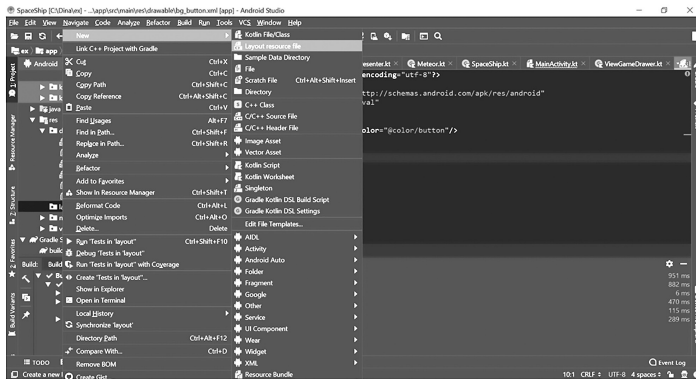


Рисунок 5.20 – Создание ресурсного файла макета

Во всплывающем окне вводим имя `activity_main.xml` – рисунок 5.21. Остальные параметры оставляем по умолчанию.

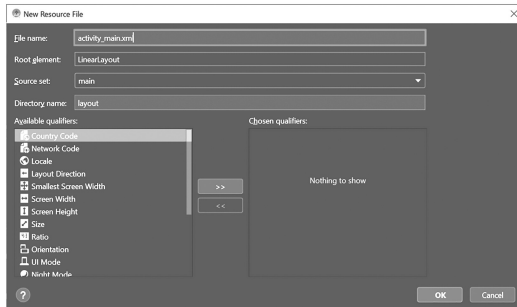


Рисунок 5.21 – Задание параметров нового ресурсного файла макета

При двойном клике на созданный файл *activity_main.xml* с правой стороны появится не код, а конструктор макета. Чуть ниже этого конструктора найдите вкладку **Text** и перейдите на нее – рисунок 5.22.

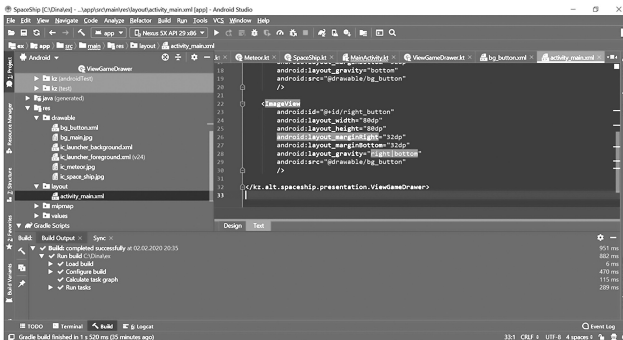


Рисунок 5.22 – Открытый ресурсный файл макета

В открывшемся поле наберите следующий код:

```
<?xml version=>1.0 encoding=>utf-8?>
<kz.alt.spaceship.presentation.ViewGameDrawer
  xmlns:android=>http://schemas.android.com/apk/res/android
  xmlns:tools=>http://schemas.android.com/tools
  android:id=>@+id/content_layout
  android:orientation=>vertical
  android:layout_width=>match_parent
  android:layout_height=>match_parent
```



```
tools:background=>@drawable/bg_main»  
>
```

<ImageView

```
android:id=>@+id/left_button»  
android:layout_width=>80dp»  
android:layout_height=>80dp»  
android:layout_marginLeft=>32dp»  
android:layout_marginBottom=>32dp»  
android:layout_gravity=>bottom»  
android:src=>@drawable/bg_button»  
</>
```

<ImageView

```
android:id=>@+id/right_button»  
android:layout_width=>80dp»  
android:layout_height=>80dp»  
android:layout_marginRight=>32dp»  
android:layout_marginBottom=>32dp»  
android:layout_gravity=>right|bottom»  
android:src=>@drawable/bg_button»  
</>
```

</kz.alt.spaceship.presentation.ViewGameDrawer>

Осталось добавить два цвета. Для этого зайдите в папку *values* и дважды щелкните на файл *colors.xml* – рисунок 5.23.

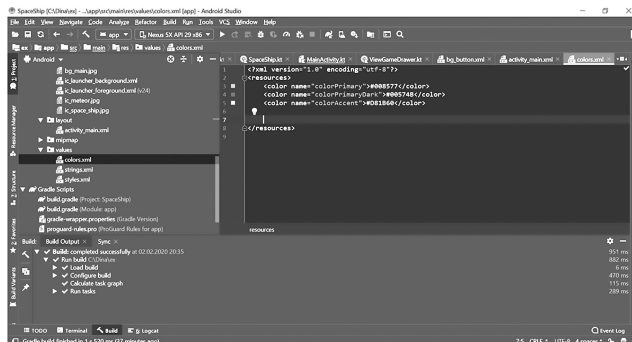


Рисунок 5.23 – Добавление цветов

В кодировку цветов добавьте две строки, чтобы код в файле выглядел так:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#008577</color>
  <color name="colorPrimaryDark">#00574B</color>
  <color name="colorAccent">#D81B60</color>

  <color name="button">#7EFFFFFF</color>
  <color name="background">#000000</color>
</resources>
```

Теперь добавлены все ресурсные элементы.

5.6 Добавление библиотек и запуск

Последним добавлением в код будут библиотеки.

Для этого открываем директорию **Gradle Scripts** и дважды нажимаем на **build.gradle (Module:app)**. В открывшемся коде находим блок **dependencies {...}** и добавляем две строки так, чтобы получилось следующее:

```
...
dependencies {
  implementation fileTree(dir: 'libs', include: ['*.jar'])
  implementation «org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version»
  implementation(«org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.3»)
  implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.3'
  implementation 'androidx.appcompat:appcompat:1.0.2'
  implementation 'androidx.core:core-ktx:1.0.2'
  testImplementation 'junit:junit:4.12'
  androidTestImplementation 'androidx.test.ext:junit:1.1.0'
  androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

В итоге, среда разработки предложит синхронизировать весь проект – возникнет синяя кнопка Sync Now выше кода в правом углу – рисунок 5.24.

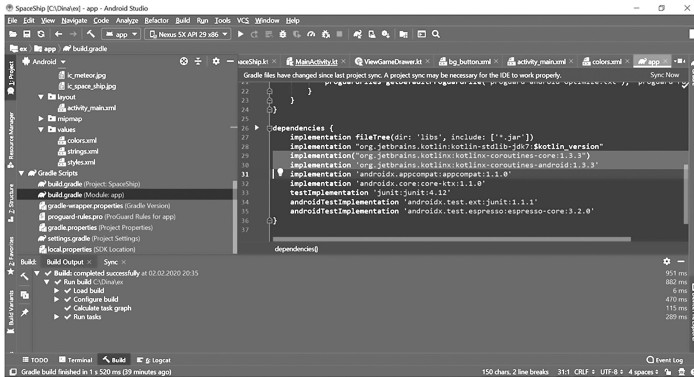


Рисунок 5.24 – Предложение синхронизировать проект

Проект готов! Можно подключать мобильный телефон, как было описано ранее для **Processing**, и нажав **Shift+F10** или зеленую кнопку Play на панели инструментов загрузить приложение на смартфон – рисунок 5.25.

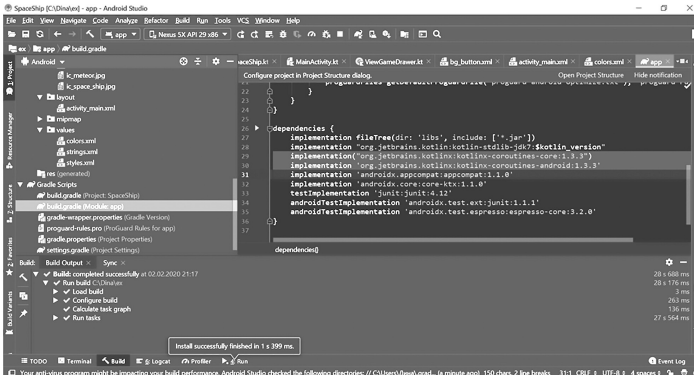


Рисунок 5.25 – Завершение загрузки на смартфон

Если Вы все сделали правильно и не пропустили ни один шаг и ни одну запятую и скобку, то в результате, приложение на смартфоне будет выглядеть и работать как на рисунке 5.26.

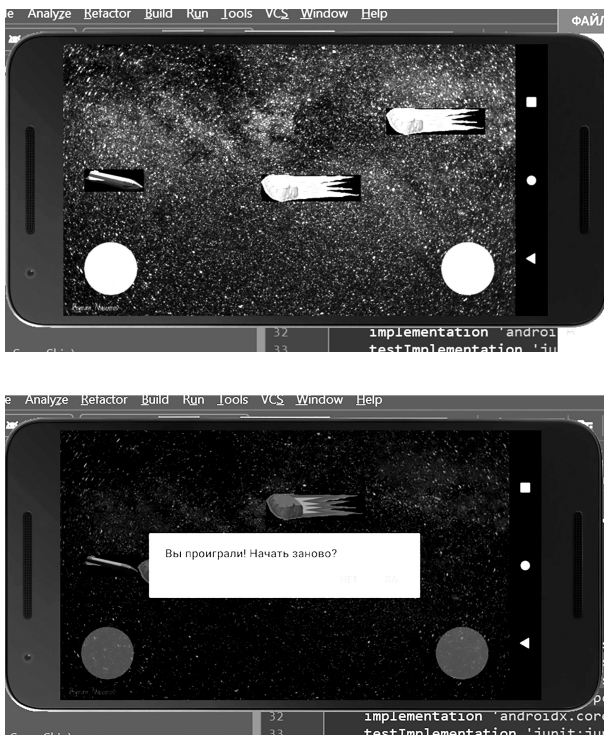


Рисунок 5.26 – Готовое приложение на смартфоне

В этой главе очень кратко рассказано про разработку мобильных приложений в среде разработки Android Studio на практическом примере небольшой игры. К сожалению, более подробное описание требует отдельного пособия только по Android Studio. Но для пытливого студента приведенный, работающий пример позволит задать вектор для дальнейшего изучения и развития.

Оффтоп. Авторы хотели бы поблагодарить за помощь в работе над этой главой и за рекомендации и комментарии ко всему пособию Чернова Олега – мобильного разработчика из казахстанской компании Kolesa.kz.

6 ПОСТРОЕНИЕ БАЗОВОГО ИНТЕРФЕЙСА ДЛЯ IOS

6.1 Создание нового проекта

В этой главе займемся разработкой простого мобильного приложения для iOS. В качестве примера рассмотрим сервис *FoodTracke*.

Для разработки приложений для iOS с использованием новейших технологий, описанных в этой главе, необходим компьютер Mac (*macOS 10.11.5* или более поздней версии) под управлением последней версии *Xcode*.

Xcode – среда, которая включает все программные элементы для того, чтобы разработчики могли спроектировать, разработать и отладить приложение. *Xcode* также содержит *SDK iOS*, который включает инструменты, компиляторы и структуры для разработки под iOS. Загрузить последнюю версию *Xcode* на Mac можно бесплатно в AppStore.

Загрузка последней версии *Xcode* выполняется в четыре шага:

- 1) Откройте приложение AppStore на компьютере Mac.
- 2) В поле поиска в правом верхнем углу введите код *Xcode* и нажмите клавишу *Return*. Приложение *Xcode* отображается первым результатом поиска.
- 3) Нажмите кнопку «Получить», а затем «Установить приложение».
- 4) При появлении запроса введите свой Apple ID и пароль. Теперь *Xcode* загружен в каталог */Applications*.

Важно!!! Практическая задача в этом учебном пособии была написана с помощью Xcode 8.1, iOS SDK 10 и Swift 3. Попробуйте использовать эти версии во время обучения. При использовании другой версии интерфейс может отличаться от экрана, показанного на рисунках. Также может потребоваться внести изменения в код для его компиляции.

Xcode включает несколько встроенных шаблонов приложений для разработки общих типов приложений iOS, таких как игры, приложения с навигацией на основе вкладок и приложения на основе табличного представления. Большинство этих шаблонов имеют предварительно настроенные файлы интерфейса и исходного кода. Начнем с самого базового шаблона – приложения *Single View*.

Для этого необходимо открыть *Xcode* в каталоге */Applications*.

Если *Xcode* запускается впервые, может потребоваться принять пользовательское соглашение и загрузить дополнительные компоненты. Следуйте указаниям на всплывающих экранах, пока *Xcode* не будет полностью настроен и готов к запуску.

После запуска *Xcode* появится окно приветствия (рисунок 6.1).

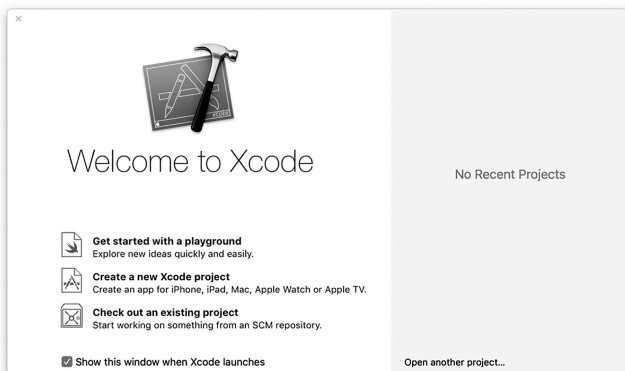


Рисунок 6.1 – Запуск Xcode

Если вместо окна приветствия появится окно проекта, это означает, что ранее уже был запущен проект в *Xcode*. Для создания проекта используйте вкладку *File*.

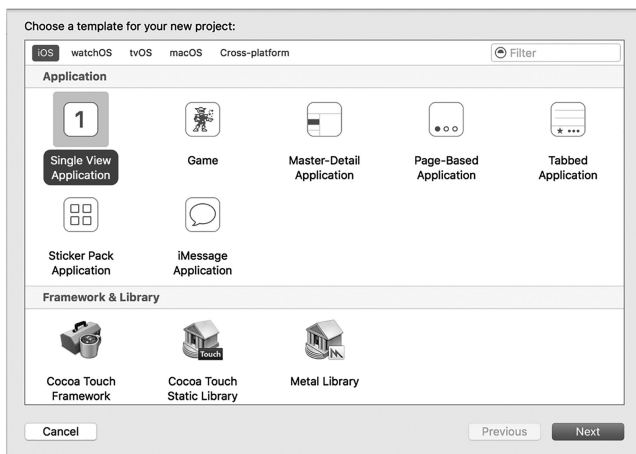


Рисунок 6.2 – Выбор шаблона приложения

Теперь в окне приветствия необходимо нажать **Create a new Xcode project** (Создать новый проект **Xcode**) (или выберите **File, New, Project**). **Xcode** откроет новое окно и выведет диалоговое окно для выбора шаблона (рисунок 6.2). Здесь выберите iOS в верхней части диалогового окна. В разделе **Application** (Приложение) выберите **Single View Application** (Приложение одного окна) и нажмите кнопку **Next** (Далее).

В появившемся диалоговом окне (рисунок 6.3) надо задать следующие значения для присвоения имени приложению. Кроме этого, необходимо выбрать дополнительные параметры для проекта:

- 1) **Название продукта:** **FoodTracker** (**Xcode** использует введенное разработчиком имя продукта, чтобы назвать проект и приложение).
- 2) **Команда:** Если она не заполнена автоматически, установить значение **None** (Нет).
- 3) **Название организации:** название организации или собственного имени разработчика. Поле можно оставить пустым.
- 4) **Идентификатор организации:** идентификатор организации, если он есть. Если нет, используйте **com.example**.
- 5) **Идентификатор пакета:** это значение автоматически создается на основе имени продукта и идентификатора организации.
- 6) **Язык:** **Swift**.

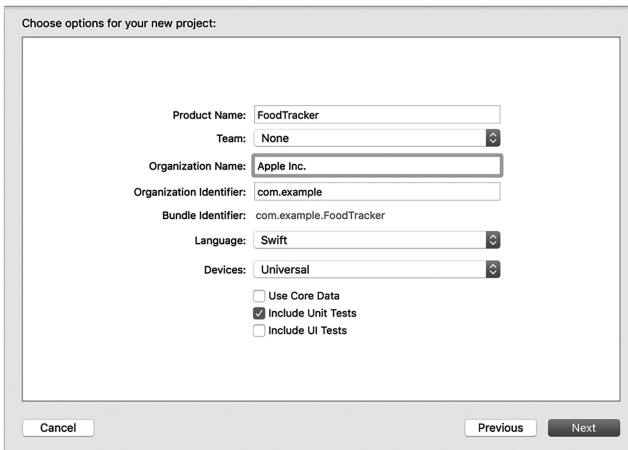


Рисунок 6.3 – Общие параметры приложения

- 7) *Устройства: **Universal*** (Универсальное приложение – это приложение, которое работает как на iPhone, так и на iPad)
- *Использовать базовые данные: **не отмечайте.***
 - *Включить единичные тесты: **отметьте.***
 - *Включить тесты пользовательского интерфейса: **не отмечайте.***

Далее необходимо нажать Next, и в появившемся диалоговом окне выбрать адрес для сохранения проекта и нажать Create (Создать). Xcode откроет новый проект в окне рабочей области (рисунок 6.4).

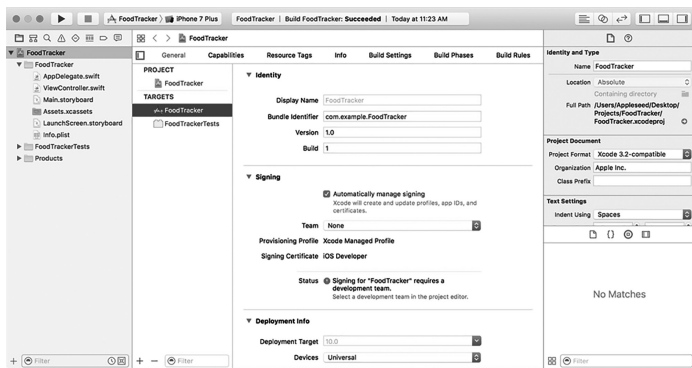


Рисунок 6.4 – Новый проект

В окне рабочей области может появиться значок ошибки с сообщением *Signing for FoodTracker requires a development team* (Подпись для **FoodTracker** требует группы разработчиков). Это предупреждение означает, что **Xcode** еще не настроен для разработки iOS, но не беспокойтесь, можно завершить приложение по этому учебному пособию и без этого.

Перед запуском приложения на устройстве iOS необходимо задать действующую группу, чтобы приложение могло быть подписано. Если вы являетесь физическим лицом или членом организации, которая является участником программы *Apple Developer Program*, можно выбрать эту группу там. В противном случае Apple ID назначается личной команде, которую можно использовать для запуска приложений на устройствах. Однако перед отправкой приложения в магазин приложений в любом случае придется присоединиться к программе *Apple Developer Program*.

Подробнее можно узнать в *Help, Xcode Help* и запросить поиск Signing workflow (Процесс подписания).

6.2 Знакомство с Xcode

Xcode включает в себя все необходимое для создания приложения. Он организует все файлы и ресурсы, которые нужны для создания приложения. Он предоставляет редакторы, как для Вашего кода, так и для Ваших пользовательских интерфейсов. Кроме того, **Xcode** позволяет создавать, запускать и отладить приложение, предоставляя симуляторы для устройств с iOS и мощный интегрированный отладчик.

Основные разделы рабочей области **Xcode** (рисунок 6.5):

Navigator area (навигатор). Обеспечивает быстрый доступ к различным частям проекта.

Editor area (редактор). Позволяет редактировать исходный код, пользовательские интерфейсы и другие ресурсы.

Utility area (информационное обеспечение). Предоставляет сведения о выбранных элементах и доступ к готовым ресурсам. Эта область разделена на две части. Верхняя – это панель инспекции, на которой можно просматривать и редактировать информацию об элементах, выбранных в навигаторе или областях редактирования. В нижней части окна находится панель библиотеки, в которой можно получить доступ к элементам пользовательского интерфейса, фрагментам кода и другим ресурсам.

Toolbar (панель инструментов). Используется для создания и запуска приложений, просмотра хода выполнения задач и настройки рабочей среды.

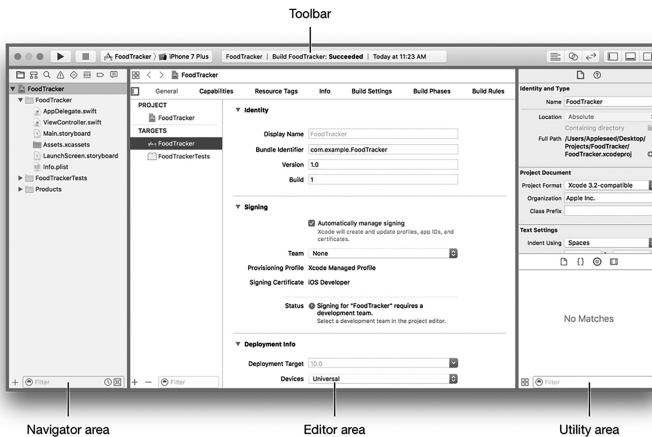


Рисунок 6.5 – Основные разделы рабочей области Xcode

Поскольку проект основан на шаблоне *Xcode*, базовая среда приложения настраивается автоматически. Можно сразу создать и запустить шаблон приложения *Single View* без дополнительной настройки.

Для создания и запуска приложения используйте приложение *iOS Simulator*, включенное в *Xcode*. Симулятор дает представление о том, как приложение будет выглядеть, и вести себя при работе на устройстве.

Симулятор может моделировать различные типы устройств - все размеры экрана и разрешения для iPad и iPhone, чтобы можно было смоделировать приложение на каждом устройстве. В этом примере используется iPhone 7.

Теперь запустим приложение в симуляторе. Во всплывающем меню *Scheme* (Схема) на панели инструментов *Xcode* необходимо выбрать iPhone 7 (рисунок 6.6). В этом всплывающем меню можно выбрать симулятор или устройство для запуска приложения. Убедитесь, что выбран симулятор iPhone 7, а не устройство iOS.

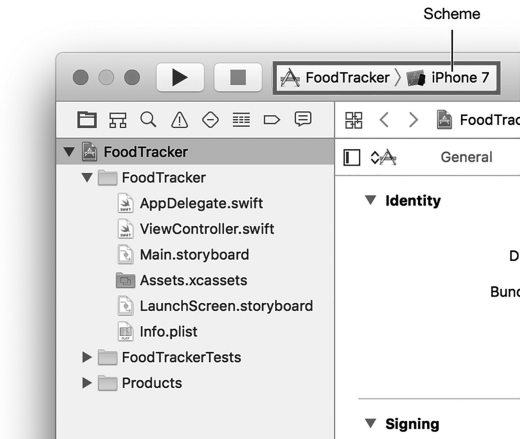


Рисунок 6.6 – Выбор iPhone 7 для симуляции

Необходимо нажать кнопку Run (Выполнить), расположенную в левом верхнем углу панели инструментов Xcode (рисунок 6.7) Либо выбрать Product, Run (или нажать Command-R).



Рисунок 6.7 – Запуск симуляции

Если мы работаем с приложением впервые, *Xcode* запросит «хотите ли Вы включить режим разработчика на компьютере Mac». Режим разработчика позволяет *Xcode* получить доступ к определенным функциям отладки, не требуя каждый раз вводить пароль. Включить режим разработчика, и далее следовать подсказкам.

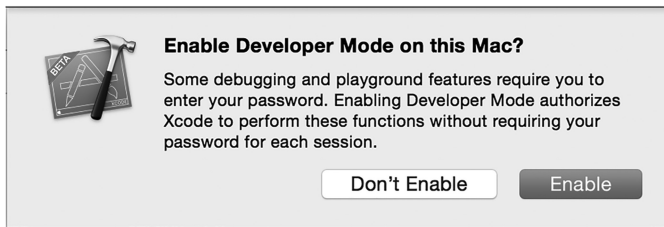


Рисунок 6.8 – Запуск режима разработчика

Если не включить режим разработчика сейчас, то возможно придется ввести пароль позже. Практическая задача в этой главе предлагают, что режим разработчика включен.

По завершению процесса построения надо просмотрите панель инструментов *Xcode*.

Xcode отображает сообщения о процессе построения в среде просмотра действий, которое находится панели инструментов.

После завершения создания проекта *Xcode* автоматически запускается симулятор. Для первого запуска может потребоваться несколько минут.

Симулятор открывается в заданном режиме iPhone, а затем запускает приложение. Сначала симулятор отображает экран запуска приложения, а затем переходит к основному интерфейсу приложения. В неизменном шаблоне приложения *Single View* экран запуска и главный интерфейс идентичны (рисунок 6.9).

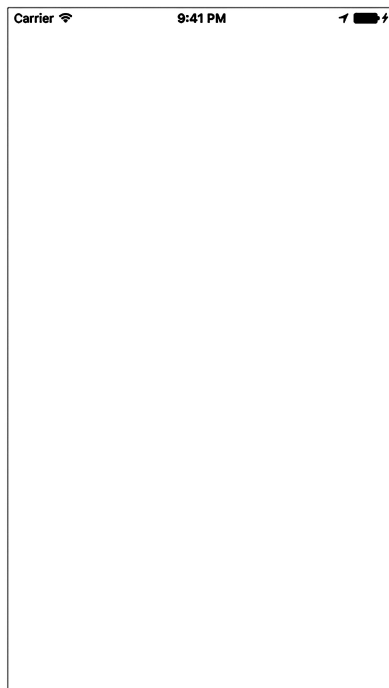


Рисунок 6.9 – Первый запуск приложения

На данный момент шаблон приложения *Single View* просто отображает белый экран. Другие шаблоны имеют более сложное поведение. Важно понимать, как работает шаблона, прежде чем его использовать, чтобы сделать свое собственное приложение. Запуск приложения в симуляторе без изменений является хорошим способом получить это понимание.

Выйти из симулятора можно выбрав *Simulator*, *Quit Simulator* (или нажав *Command-Q*).

6.3 Обзор исходного кода

Шаблон приложения *Single View* предоставляется с несколькими файлами исходного кода, которые настраивают среду приложения. Чтобы просмотреть файл *AppDelegate.swift*, необходимо выполнить следующие шаги.

Убедитесь, что навигатор проекта открыт в области навигатора. В навигаторе проекта отображаются все файлы проекта. Если навигатор проекта не открыт, нажмите крайнюю левую кнопку на панели выбора навигатора – рисунок 6.10 (Также можно выбрать *View, Navigators, Show Project Navigator*).

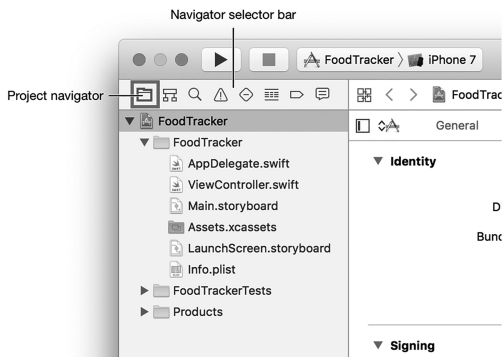


Рисунок 6.10 – Включение навигатора проекта

При необходимости откройте папку *FoodTracker* в навигаторе проекта, щелкнув рядом с ней треугольник раскрытия информации. Выберите *AppDelegate.swift*. Xcode открывает исходный файл в области главного редактора окна – рисунок 6.11. Либо дважды щелкнуть файл *AppDelegate.swift*, чтобы открыть его в отдельном окне.

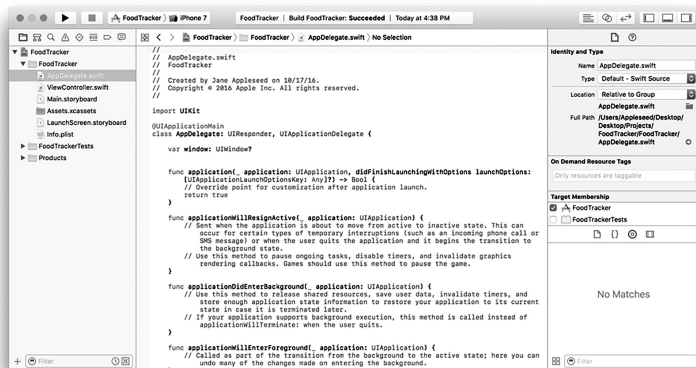


Рисунок 6.11 – Файл AppDelegate.swift

Исходный файл *AppDelegate.swift* имеет две основные функции:

- 1) Он определяет класс *AppDelegate*. Этот класс создает окно, в котором отображается содержимое приложения, и предоставляет возможность реагировать на изменение состояний в приложении.
- 2) Также он создает точку входа в приложение и цикл выполнения, который обрабатывает входные события в приложении. Эта работа выполняется атрибутом *UIApplicationMain (@UIApplicationMain)*, который находится в верхней части файла. Использование атрибута *UIApplicationMain* эквивалентно вызову функции *UIApplicationMain* и передаче имени класса *AppDelegate* в качестве имени следующего выполняющегося класса. В ответ система создает объект приложения. Объект приложения отвечает за управление циклом работы приложения. Система также создает экземпляр класса *AppDelegate* и назначает его объекту приложения. В итоге, система запускает приложение.

Класс *AppDelegate* создается автоматически при создании нового проекта. Если создается простое приложение, рекомендуется использовать именно этот класс, предоставленный *Xcode*, чтобы инициализировать приложение и реагировать на события. В классе *AppDelegate* используется протокол *UIApplicationDelegate*. Этот протокол определяет ряд методов, используемых для настройки приложения, реагирования на изменения состояния приложения и обработки других событий на уровне приложения.

Класс *AppDelegate* содержит одно свойство: *window*.

var window: UIWindow?

Это свойство сохраняет ссылку на окно приложения, представляет корень иерархии представлений приложения. Это место, где рисуется весь контент приложения. Обратите внимание, что свойство *window* является необязательным, что означает, что в какой-то момент оно может не иметь значения (быть нулевым).

Класс *AppDelegate* также содержит заглушки для следующих методов делегирования:

```
func application(_ application: UIApplication,  
didFinishLaunchingWithOptions launchOptions:  
[UIApplicationLaunchOptionsKey: Any]?) -> Bool
```

```
func applicationWillResignActive(_ application: UIApplication)  
func applicationDidEnterBackground(_ application: UIApplication)  
func applicationWillEnterForeground(_ application: UIApplication)  
func applicationDidBecomeActive(_ application: UIApplication)  
func applicationWillTerminate(_ application: UIApplication)
```

Эти методы позволяют объекту приложения взаимодействовать со следующим выполняющимся классом приложения. Во время перехода в состояние приложения (например, запуск приложения, переход в фоновый режим и завершение работы приложения) объект приложения вызывает соответствующий метод, что дает приложению возможность ответить. Не нужно делать ничего особенного, чтобы эти методы вызывались в правильное время – объект приложения обрабатывает это задание.

Каждый из методов функционирует по умолчанию. Если оставить шаблон пустым или удалить метод из класса *AppDelegate*, то при каждом вызове этого метода проявится поведение по умолчанию. Можно также добавить собственный код к работе этих методов по умолчанию, определяя не типичное поведение, которое выполняется при их вызове.

Шаблон также содержит комментарии для каждого из этих методов. Эти комментарии описывают, как методы могут быть использованы приложением. Можно использовать эти методы и комментарии в качестве базового проекта для разработки многих распространенных моделей поведения на уровне приложений.

В нашем примере будут использоваться типичные метод, поэтому не надо вносить изменения в файл *AppDelegate.swift*.

Шаблон приложения *Single View* содержит и другой файл исходного кода: *ViewController.swift*. Выберите команду *ViewController.swift* в навигаторе проекта, чтобы просмотреть его (рисунок 6.12).

Этот файл определяет пользовательский подкласс *UIViewController* с именем *ViewController*. По умолчанию этот класс просто наследует поведение, определенное *UIViewController*. Чтобы изменить это поведение можно переопределить методы, определенные в *UIViewController*.

Как видно в файле *ViewController.swift*, внедрение шаблона включает методы *viewDidLoad()* и *didReceiveMemoryWarning()*; однако их выполнение только вызывает эти методы из версии *UIViewController*.

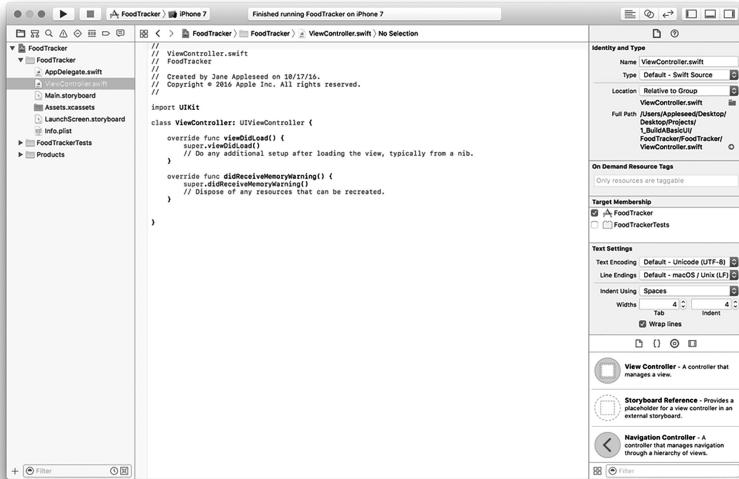


Рисунок 6.12 – Файл *ViewController.swift*

В шаблон так же входит метод *didReceiveMemoryWarning()*, но для этого примера он не нужен, поэтому удалите его.

На этом этапе код *ViewController.swift* должен выглядеть следующим образом:

```
import UIKit
```

```
class ViewController: UIViewController {
```

```
    override func viewDidLoad() {
```

```
        super.viewDidLoad()
```

```
        // Do any additional setup after loading the view, typically from a nib.
```

```
    }  
}
```

6.4 Раскадровка

Раскадровка представляет собой визуальное представление пользовательского интерфейса приложения, показывающее экраны контента и переходы между ними. Можно сразу увидеть, какой именно интер-

фейс получается, что работает, а что нет, а также мгновенно скорректировать интерфейс.

Чтобы открыть раскадровку необходимо выбрать файл *Main.storyboard* в навигаторе проекта. Xcode открывает раскадровку в *Interface Builder* – в своем визуальном редакторе интерфейса – в области редактора. Фоном раскадровки является холст (рисунок 6.13). Холст используется для добавления и упорядочивания элементов пользовательского интерфейса.

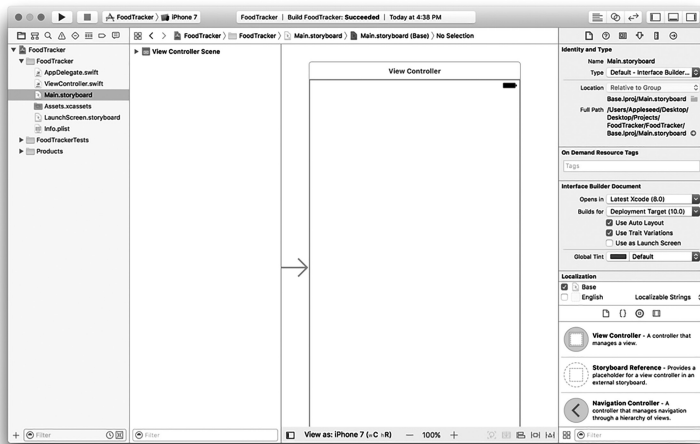


Рисунок 6.13 – Открытие раскадровки

На данный момент раскадровка в приложении содержит только один кадр, который представляет собой основной экран в приложении. Стрелка, которая указывает на левую сторону кадра на холсте, является точкой входа раскадровки, что означает, что эта сцена загружается первой при запуске приложения.

При запуске приложения в iPhone 7 Simulator в этом кадре отображается то, что вы увидите на экране устройства. Однако кадр на холсте может иметь другие размеры, чем на экране симулятора. Можно выбрать размер экрана и ориентацию в нижней части холста. В примере он установлен на iPhone 7 в портретной ориентации.

Несмотря на то, что холст показывает конкретное устройство и ориентацию, важно создать адаптивный интерфейс – интерфейс, который автоматически настраивается так, чтобы хорошо выглядеть на

любом устройстве и в любой ориентации. При разработке интерфейса можно изменить представление холста, чтобы увидеть, как интерфейс адаптируется к экранам разного размера.

6.5 Создание базового пользовательского интерфейса

Начнем с работы над пользовательским интерфейсом для кадра, в котором можно добавить новую еду в приложение *FoodTracker*.

Xcode предоставляет библиотеку объектов, которые можно добавить в файл раскадровки. Некоторые из них являются элементами пользовательского интерфейса, такими как кнопки и текстовые поля. Другие, например, контроллеры видов и распознаватели жестов, определяют поведение приложения, но не отображаются на экране.

Элементы, появляющиеся в интерфейсе пользователя, называются видами. В видах отображается содержимое для пользователя. Они являются составными элементами для построения пользовательского интерфейса. Виды имеют множество полезных встроенных моделей поведения, включая отображение на экране и реагирование на ввод данных пользователем.

Все объекты вида в iOS имеют тип *UIView* или один из его подклассов. Многие подклассы *UIView* высоко специализированы по внешнему виду и поведению. Начнем с добавления в кадр текстового поля (*UITextField*), одного из таких подклассов *UIView*. Текстовое поле позволяет пользователю ввести одну строку текста, который будет использоваться в качестве названия блюда.

Для добавления текстового поля в кадр выберите меню *Editor*, *Canvas* и убедитесь, что выбран параметр *Show Bounds Rectangles* (Показать границы прямоугольников). Этот параметр рисует синюю ограничивающую рамку вокруг всех видов холста в *Interface Builder*. Многие виды и элементы управления имеют прозрачный фон, что затрудняет просмотр их фактического размера. Ошибки макета возникают, когда система изменяет размер вида на большее или меньшее, чем ожидается. Включение этого параметра позволяет точно понять, что происходит в иерархии представлений.

Откройте *Object library* (библиотеку объектов). Библиотека появляется в нижней правой части *Xcode*. Если библиотека объектов не отображается, нажмите третью кнопку слева на панели выбора библиотеки – рисунок 6.14. (Также можно выбрать *View*, *Utilities*, *Show Object Library*).

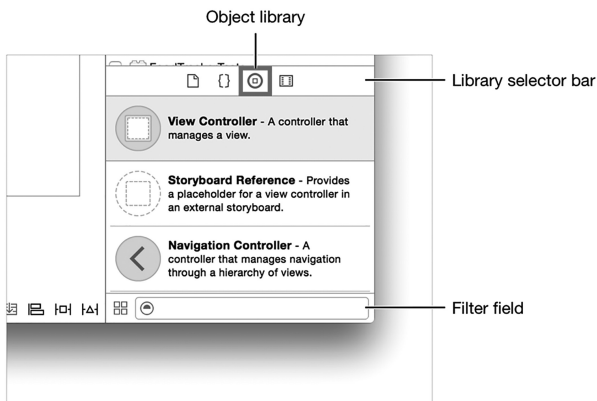


Рисунок 6.14 – Открытие библиотеки объектов

Появится список, содержащий: имя, описание и визуальное представление каждого объекта. В поле фильтра в библиотеке объектов введите *text field*, чтобы быстро найти нужный объект. Перетащите объект из библиотеки на кадр – рисунок 6.15.

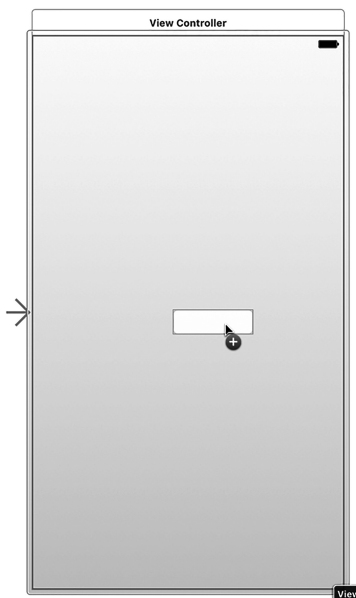


Рисунок 6.15 – Первый объект приложения

При необходимости надо увеличить изображение, выбрав *Editor, Canvas, Zoom*.

Перетащите текстовое поле так, чтобы оно было расположено в верхней половине сцены и выровнялось с левым полем сцены. Остановите перетаскивание текстового поля, когда оно будет привязано к левому краю – рисунок 6.16.

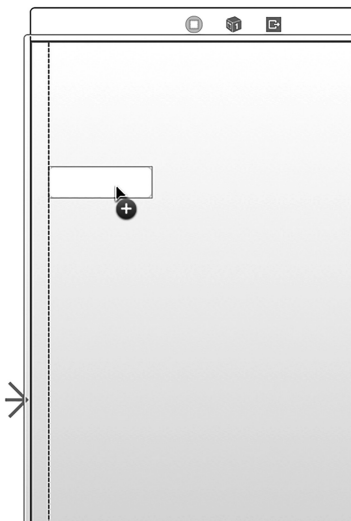


Рисунок 6.16 – Привязка текстового объекта к левому краю

Синие направляющие помогают поместить текстовое поле. Направляющие компоновки отображаются только при перетаскивании или изменении размеров объектов рядом с ними; Они исчезают, когда отпускаете текстовое поле.

При необходимости щелкните текстовое поле, чтобы отобразить маркеры изменения размера. Изменение размера элемента пользовательского интерфейса выполняется путем перетаскивания его маркеров изменения размера, которые представляют собой небольшие белые квадраты, появляющиеся на границах элемента. Для отображения маркеров изменения размера элемента выберите его. В этом случае текстовое поле уже должно быть выбрано, так как его перетаскивание прекратилось. Если текстовое поле выглядит так, как на рисунке 6.17, можно изменить его размер; Если нет, выберите его на холсте.

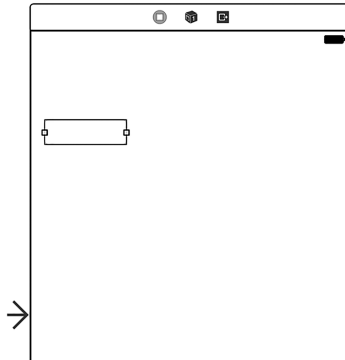


Рисунок 6.17 – Выбор текстового объекта для изменения размеров

Измените, размер левого и правого краев текстового поля, пока не появятся три вертикальные направляющие компоновки: выравнивание по левому краю, выравнивание по центру по горизонтали и выравнивание по правому краю – рисунок 6.18.

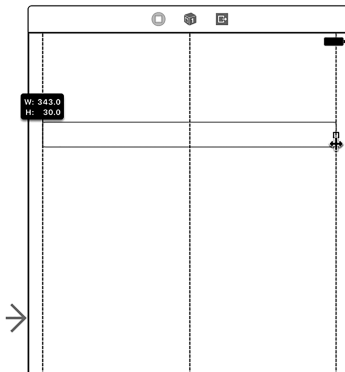



Рисунок 6.18 – Левая, центральная и правая направляющие

Несмотря на то, что в кадре имеется текстовое поле, инструкции пользователю о том, что вводить в поле, отсутствуют. Используйте **Placeholder** (место заполнитель) текстового поля, чтобы предложить пользователю ввести имя нового блюда.

Для этого выбрав текстовое поле, откройте **Attributes inspector** (Инспектор атрибутов) –  – рисунок 6.19. Инспектор атрибутов появляется при нажатии четвертой кнопки слева на панели инспектора. Она позволяет редактировать свойства объекта в раскладке.

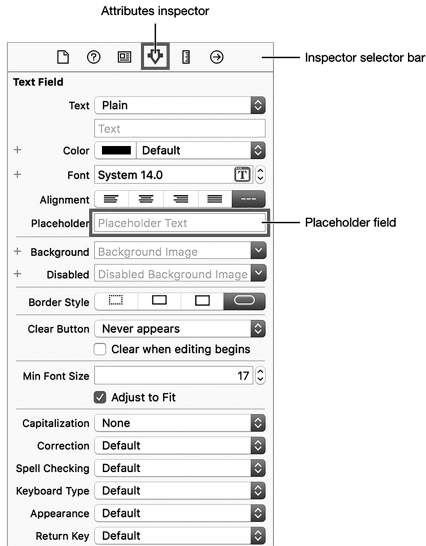


Рисунок 6.19 – Панель инспектора

В инспекторе атрибутов найдите поле с меткой **Placeholder** и наберите «*Enter meal name*».

Нажмите **Return** (Возврат), чтобы отобразить новый текст в текстовом поле – рисунок 6.20.

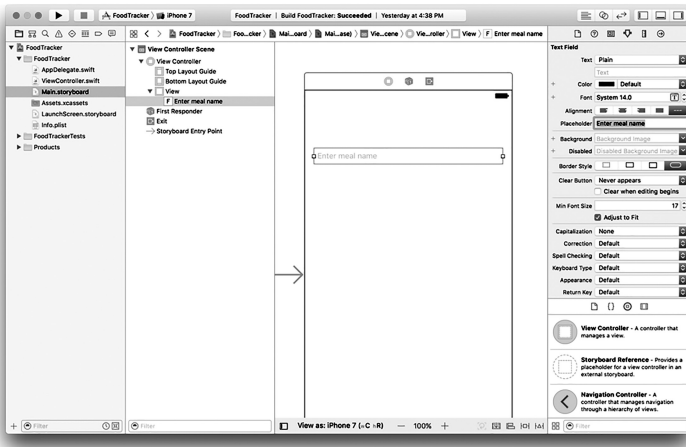


Рисунок 6.20 – Поле ввода текста с комментарием

При редактировании атрибутов текстового поля можно также редактировать атрибуты системной клавиатуры, отображаемые при выборе пользователем текстового поля.

Для настройки клавиатуры текстового поля сначала убедитесь, что текстовое поле выбрано.

В атрибутах найдите поле с меткой **Return Key** и выберите **Done** (при необходимости прокрутите вниз) – рисунок 6.21. Это изменение поменяет клавишу **Return** на клавиатуре на клавишу **Done**.

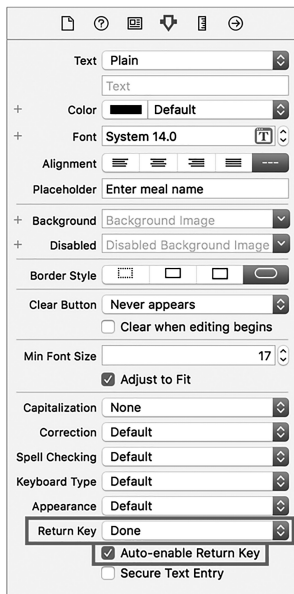


Рисунок 6.21 – Настройка клавиатуры текстового поля

В атрибутах установите флажок на **Auto-enable Return Key** (Автоматически включать ключ возврата). При необходимости снова прокрутите вниз. Это сделает невозможным, для пользователя, нажатие клавиши **Done** перед вводом текста в текстовое поле, гарантируя, что пользователи никогда не смогут вводить пустую строку в качестве названия блюда.

Затем добавьте строку (**UILabel**) в верхней части кадра. Строка не является интерактивной. Она просто отображает статический текст в интерфейсе пользователя. Настраиваем эту строку для отображения текста, вводимого пользователем в текстовое поле, чтобы проверить,

что текстовое поле принимает ввод пользователя и обрабатывает его надлежащим образом.

Для добавления строки в кадр в библиотеке объектов введите **label** в поле фильтра, чтобы быстро найти объект **Label**. Перетащите объект **Label** из библиотеки объектов в кадр. Перетащите строку так, чтобы она находилась прямо над текстовым полем и выровнялась с левым полем кадра – рисунок 6.22.

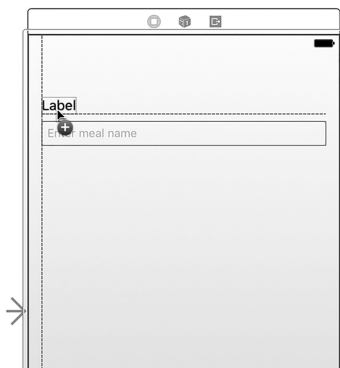


Рисунок 6.22 – Выравнивание строки к левому краю

Дважды щелкните на строке и введите **Meal Name**. Нажмите кнопку **Return** для отображения нового текста в строке – рисунок 6.23.



Рисунок 6.23 – Результат вставки строки

Теперь добавьте кнопку (***UIButton***) к кадру. Кнопка является интерактивной, поэтому пользователи могут нажать ее, чтобы запустить определенное действие. Позже будет создано действие для сброса текста метки на значение по умолчанию.

Для добавления кнопки в сцену в библиотеке объектов введите **button** в поле фильтра, чтобы быстро найти объект **Button**. Перетащите объект **Button** из библиотеки в кадр. Перетащите кнопку так, чтобы она находилась прямо под текстовым полем и выровнялась с левым полем кадра – рисунок 6.24

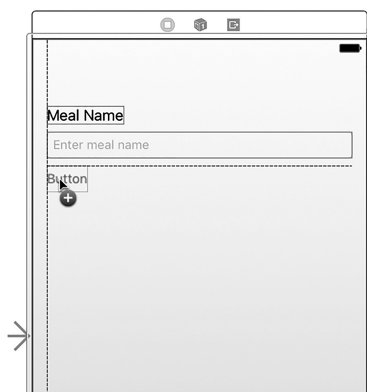
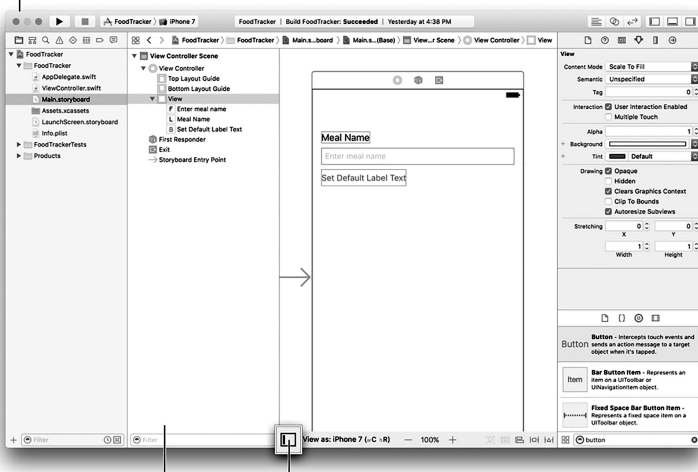


Рисунок 6.24 – Выравнивание кнопки к левому краю

Дважды щелкните кнопку и введите ***Set Default Label Text***. Нажмите кнопку ***Return*** для отображения нового текста в кнопке. При необходимости измените положение кнопки.

Стоит оценить, как добавленные элементы на самом деле организованы в кадре. Для этого просмотрите ***Outline view***, чтобы увидеть, какие элементы пользовательского интерфейса были добавлены в кадр. На раскладовке найдите переключатель ***Outline view*** – рисунок 6.25.

Если ***Outline view*** свернут, щелкните переключатель, чтобы его развернуть. Для свертывания и развертывания ***Outline view*** при необходимости можно использовать переключатель ***Outline view***. ***Outline view***, который отображается в левой части холста, обеспечивает иерархическое представление объектов в раскладовке. В просмотре можно увидеть: добавленное текстовое поле, строку и кнопку в иерархии.



Outline view Outline view toggle

Рисунок 6.25 – Переключатель Outline view

Виды не только отображаются на экране и реагируют на ввод пользователя, они могут содержать другие виды. Виды расположены в иерархической структуре, называемой иерархией видов. Иерархия видов определяет компоновку видов относительно других видов. В этой иерархии виды, заключенные в пределах вида, называются подчиненными видами, а родительский вид, который охватывает вид, называется его супервидом. Вид может иметь несколько подвидов и только один супервид.

В общем случае каждый кадр имеет свою иерархию видов. В верхней части каждой иерархии видов находится представление содержимого. В текущем кадре вид содержимого называется *View* – вид верхнего уровня в *View Controller*. Текстовое поле, строка и кнопка являются подвидами. Все другие виды, размещенные в этом кадре, будут подчиненными видами этого вида (хотя они сами могут иметь вложенные подчиненные виды).

6.6 Предварительный просмотр интерфейса

Надо периодически просматривать приложение, чтобы убедиться, что все выглядит так, как ожидали. Просмотреть интерфейс приложения можно с помощью *Assistant editor*. Для этого надо нажать кнопку

Assistant на панели инструментов *Xcode* в правом верхнем углу – рисунок 6.26



Рисунок 6.26 – Включение Assistant editor

Если требуется больше места для работы, сверните навигатор проекта и утилиты, нажав кнопки **Navigator** и **Utilities** на панели инструментов – рисунок 6.27. Можно также свернуть вид структуры.

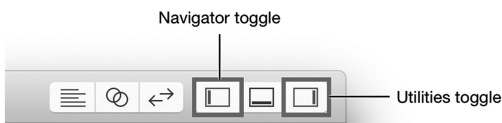


Рисунок 6.27 – Кнопки Navigator и Utilities

На панели в верхней части *Assistant editor*, переключите редактор помощника с *Automatic* на *Preview, Main.storyboard (Preview)* – рисунок 6.28.

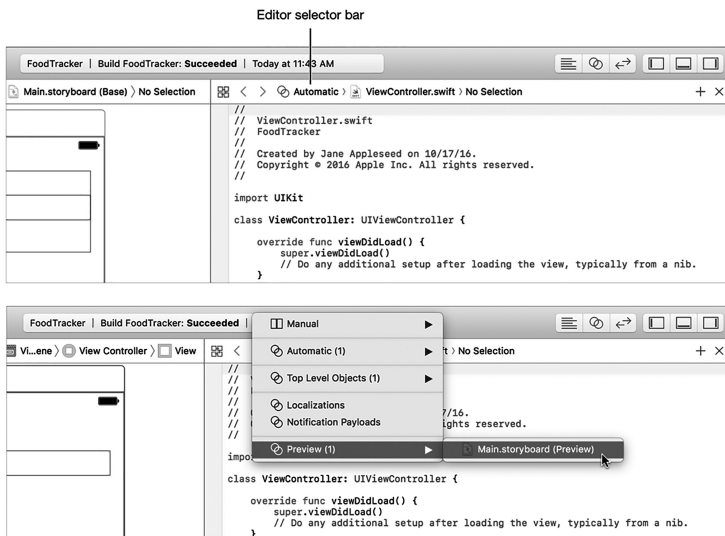


Рисунок 6.28 – Переключение на Preview

На данный момент предварительный просмотр выглядит почти одинаково с холстом – рисунок 6.29. И холст, и превью показывают одинаковый размер экрана (iPhone 7) и одинаковую ориентацию (портретную). Если требуется проверить адаптивность интерфейса, необходимо выполнить предварительный просмотр экранов различных размеров и различных ориентаций.

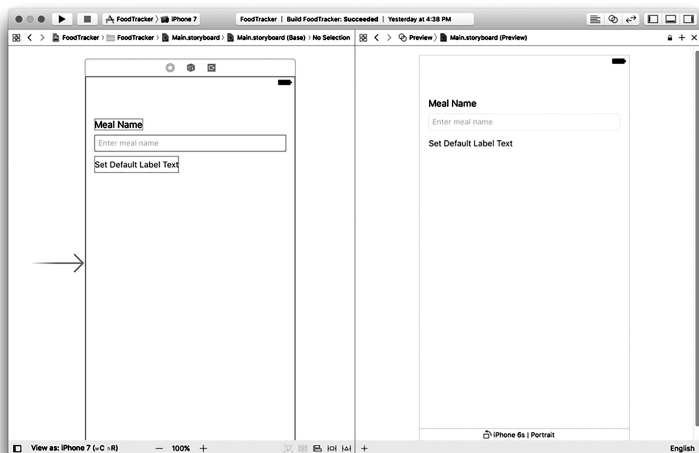


Рисунок 6.29 – Предварительный просмотр и холст

Для предварительного просмотра ориентации ландшафта нажмите кнопку **Rotate** в нижней части окна предварительного просмотра.

К сожалению, в результате все выглядит не совсем правильно. Текстовое поле, строка и кнопка сохраняют одинаковый размер и положение относительно левого верхнего угла экрана. Это означает, что текстовое поле больше не заполняет экран от края до края.

Чтобы просмотреть другой размер экрана, нажмите кнопку **Add** в нижней части редактора и выберите **iPhone SE** – рисунок 6.30

Текстовое поле, строка и кнопка сохраняют одинаковый размер и положение относительно левого верхнего угла экрана. Однако на этот раз текстовое поле выходит за правый край экрана – рисунок 6.31.

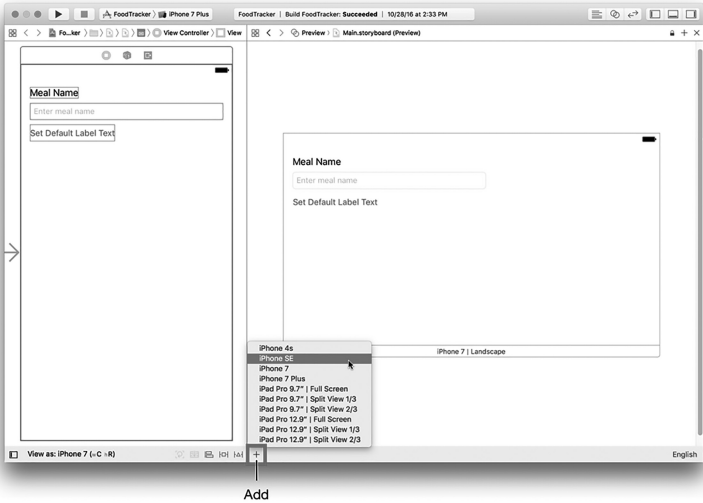


Рисунок 6.30 – Переход к эмуляции устройства другого типа

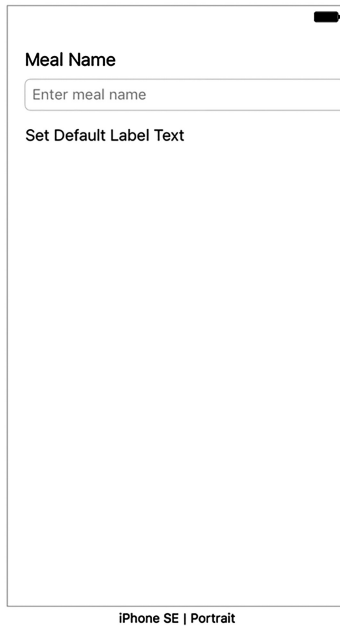


Рисунок 6.31 – Интерфейс для iPhone SE

Для создания адаптивного интерфейса необходимо указать способ настройки интерфейса в зависимости от размера экрана. Например, при повороте интерфейса в ландшафтную ориентацию текстовое поле должно увеличиваться. При отображении интерфейса на iPhone SE текстовое поле должно сокращаться. Эти типы правил отображения интерфейса можно легко задать с помощью функции *Auto Layout*.

6.7 Применение Auto Layout

Auto Layout (Автокомпоновка) – это мощный механизм компоновки, который помогает создавать адаптивные компоновки, динамически реагирующие на любые изменения размера кадра. Вы описываете свой макет с помощью ограничений – правил, объясняющих, где должен находиться один элемент относительно другого, или какого размера должен быть элемент. Функция автокомпоновки динамически вычисляет размер и положение каждого элемента на основе этих ограничений.

Одним из самых простых способов задания компоновки является использование стека видов (*UIStackView*). Стек видов обеспечивает упрощенный интерфейс для компоновки ряда видов. Для расчета размера и положения всех управляемых видов в стеке видов используется *Auto Layout*.

Чтобы применить *Auto Layout*, поместите существующие элементы интерфейса в стек видов, а затем добавьте зависимости, необходимые для размещения стека видов в кадре. Для этого вернитесь в стандартный редактор, нажав кнопку *Standard* – рисунок 6.32. Разверните навигатор проекта и утилиты, нажав кнопки *Navigator* и *Utilities* на панели инструментов *Xcode*.

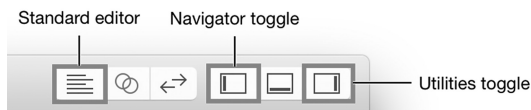


Рисунок 6.32 – Переход в стандартный редактор

Нажав клавишу *Shift* выберите текстовое поле, строку и кнопку – рисунок 6.33.

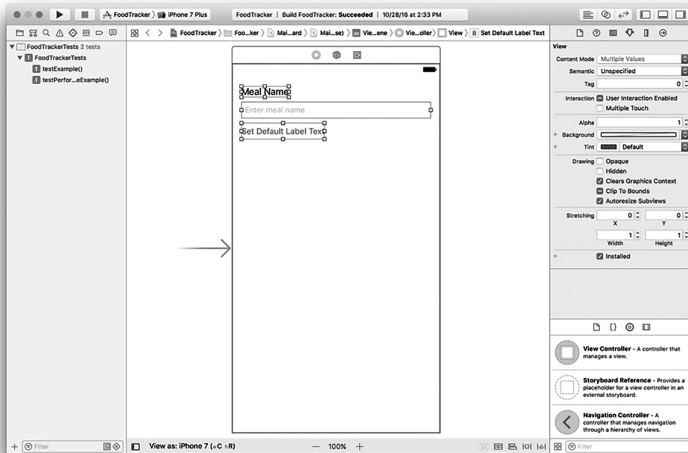


Рисунок 6.33 – Выделение текстового поля, строки и кнопки

В правом нижнем углу холста нажмите кнопку *Embed In Stack* – рисунок 6.34 (Либо выберите *Editor, Embed In, Stack View*).

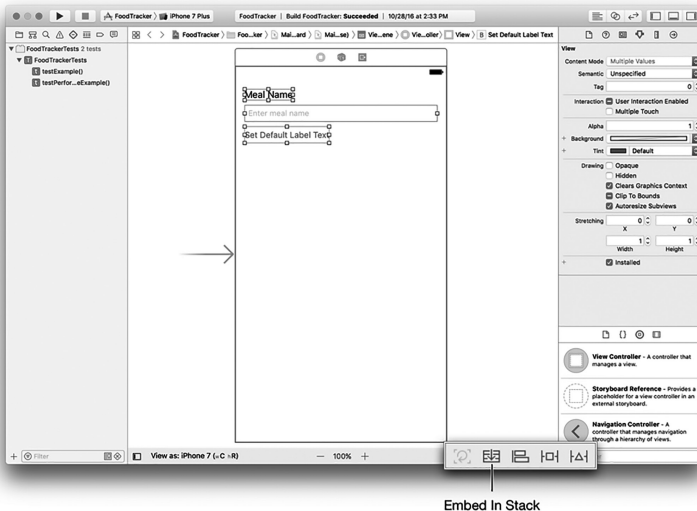


Рисунок 6.34 – Кнопка Embed In Stack

Xcode объединяет элементы пользовательского интерфейса в виде стека, сводя их воедино. *Xcode* анализирует существующую компоновку, чтобы понять, как элементы должны отображаться: вертикально или горизонтально.

При необходимости откройте структуры. Выберите объект **Stack View** – рисунок 6.35.

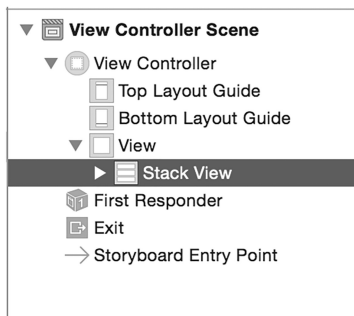


Рисунок 6.35 – Выбор объекта Stack View

В атрибутах введите 8 в поле **Spacing**. Нажмите **Return**. В правом нижнем углу холста откройте меню **Add New Constraints** – рисунок 6.36.

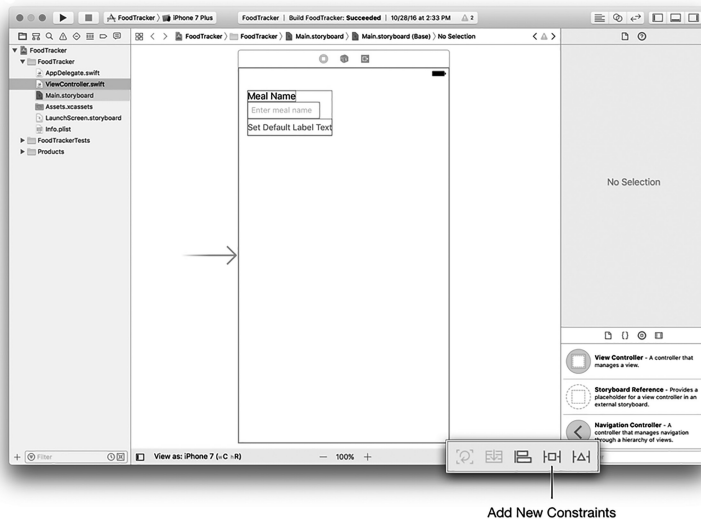


Рисунок 6.36 – Меню Add New Constraints

Над заголовком *Spacing to nearest neighbor* щелкните на двух ограничениях по горизонтали и верхнем ограничении по вертикали, чтобы выбрать их. При выборе они становятся красными – рисунок 6.37.

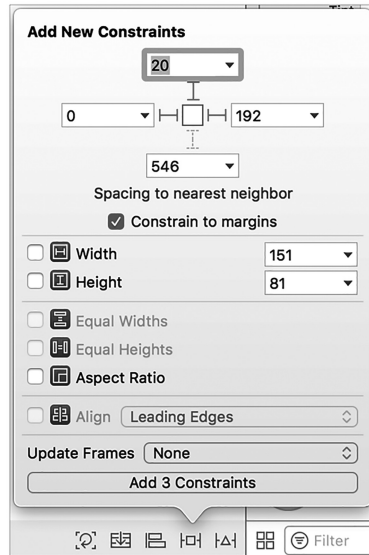


Рисунок 6.37 – Включение ограничений

Эти зависимости указывают расстояние до ближайших передних, задних и верхних объектов. В этом случае объект означает границу ближайшего элемента пользовательского интерфейса, который может быть супервидом, другим элементом пользовательского интерфейса или полем. Поскольку флажок *Constrain to margins* установлен, стек вида в этом случае будет ограничен левым и правым полями супервида. Это обеспечивает зазор между видом стека и краем кадра.

С другой стороны, верхняя часть стека ограничена относительно верхней направляющей компоновки кадра. Если строка состояния видна, верхняя направляющая компоновки располагается в нижней части строки состояния. Если нет, то он располагается в верхней части кадра. Поэтому необходимо добавить небольшое пространство между стеком вида и компоновкой.

Введите **0** в левом и правом полях и **20** в верхнем поле. Во всплывающем меню **Update Frames** выберите **Items of New Constraints**. Это приводит к автоматическому обновлению рамок изменяемых видов – рисунок 6.38.

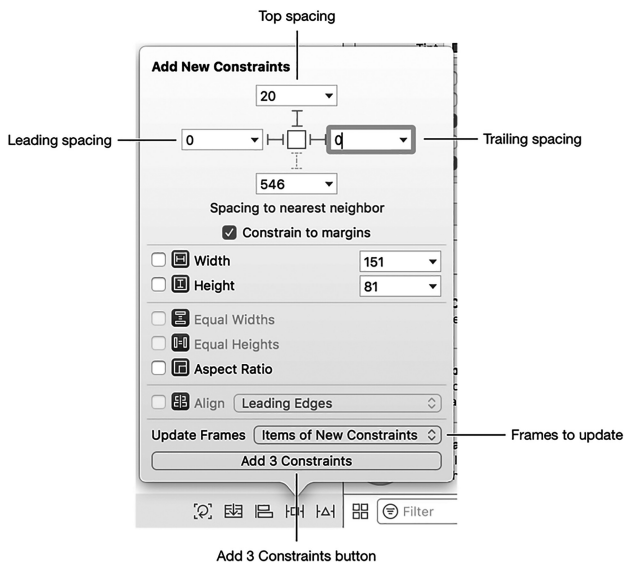


Рисунок 6.38 – Добавление ограничений между объектами

В меню **Add New Constraints** нажмите кнопку **Add 3 Constraints**.

Строка, текстовое поле и кнопка теперь выровнены по левому краю и расположены с соответствующим интервалом, но текстовое поле по-прежнему не растягивается для заполнения ширины экрана. Чтобы исправить это, необходимо добавить дополнительное ограничение.

На раскладке выберите текстовое поле. В правом нижнем углу холста снова откройте меню **Add New Constraints**. Над заголовком **Spacing to nearest neighbor** щелкните правое горизонтальное ограничение, чтобы выбрать его. При выборе оно становится красным – рисунок 6.39. Введите **0** в правом поле. Во всплывающем меню **Update Frames** выберите **Items of New Constraints**. В меню **Add New Constraints** нажмите кнопку **Add 1 Constraint**.

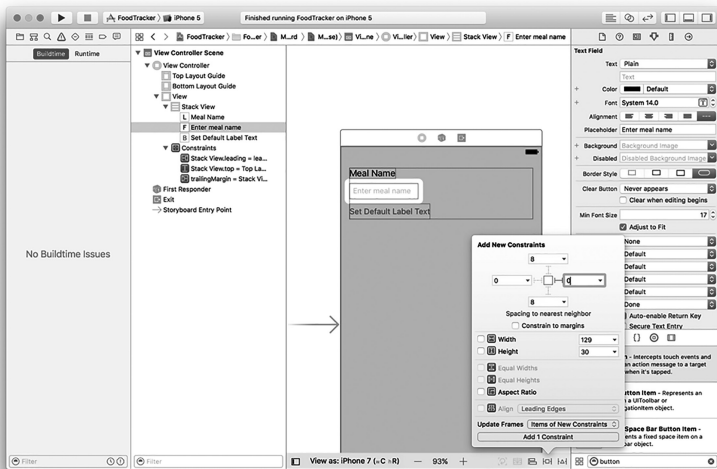


Рисунок 6.39 – Добавление ограничения для текстового поля

Для проверки запустите приложение в *iOS Simulator*. Поверните симулятор, выбрав **Hardware, Rotate Left and Hardware, Rotate Right** (или **Command-Стрелка влево** и **Command-Стрелка вправо**). Обратите внимание на увеличение и сжатие текстового поля до соответствующего размера в зависимости от ориентации устройства и размера экрана. Также обратите внимание, что строка состояния исчезает в горизонтальной ориентации.

Щелкните внутри текстового поля и введите текст с помощью экранной клавиатуры (если хотите, можно использовать клавиатуру компьютера, выбрав **Hardware, Keyboard, Connect Hardware Keyboard**).

Если ожидаемое поведение отсутствует, воспользуйтесь функциями автоматической отладки. Доступ к этим функциям можно получить с помощью кнопки **Update Frames** и меню **Resolve Auto Layout Issues** – рисунок 6.40.

При получении предупреждений о неправильном размещении видов используйте кнопку **Update Frames**. Эта кнопка обновляет рамки выбранного вида и всех его подчиненных видов. Выберите контроллер вида кадра для обновления всех видов в сцене. Можно также нажать кнопку **Update Frames**, чтобы обновить только выбранный вид.

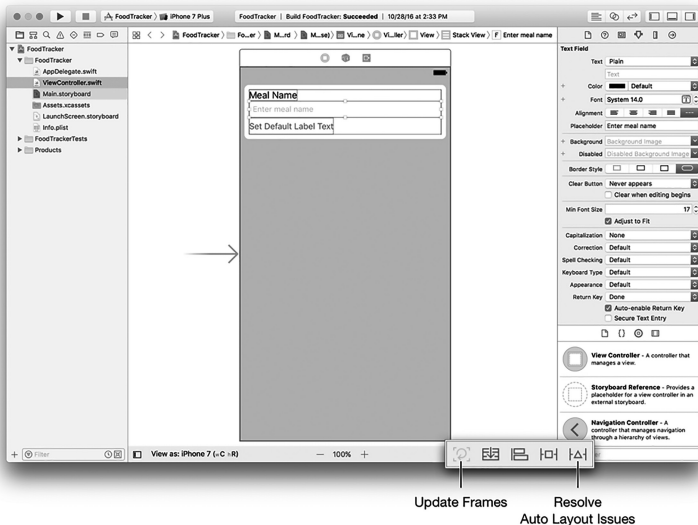


Рисунок 6.40 – Кнопка *Update Frames* и меню *Resolve Auto Layout Issues*

Если приложение работает не так, как Вы ожидаете, нажмите кнопку ***Resolve Auto Layout Issues***, чтобы открыть меню команд отладки. Все команды в этом меню имеют две формы. Один из них влияет на текущий выбранный вид. Другой режим влияет на все виды в текущем контроллере вида. Если все команды выделены серым цветом, выберите контроллер вида кадра или один из видов и снова откройте меню.

Выберите ***Reset to Suggested Constraints***, чтобы *Xcode* обновил интерфейс с помощью допустимого набора ограничений. Выберите ***Clear Constraints***, чтобы удалить все ограничения для элементов пользовательского интерфейса, а затем попробуйте выполнить предыдущие шаги для повторной установки ограничений – рисунок 6.41.

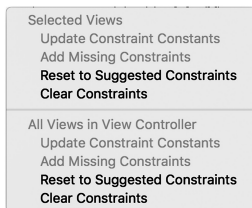


Рисунок 6.41 – Подменю *Reset to Suggested Constraints* и *Clear Constraints*

В этой главе мы познакомились с созданием проекта в *Xcode*, а также со многими инструментами, используемыми для разработки и запуска приложения iOS. Был создан простой пользовательский интерфейс.

Хотя приложение пока мало что делает, базовый пользовательский интерфейс уже есть и работает. А то, что разработанный интерфейс является надежным и расширяемым с самого начала, гарантирует, что есть прочная основа для дальнейшей разработки.

7 ДОБАВЛЕНИЕ ЛОГИКИ К БАЗОВОМУ ИНТЕРФЕЙСУ В IOS

7.1 Базовые понятия

В этой главе рассмотрим подключение базового пользовательского интерфейса приложения *FoodTracker* к коду и определим действия, которые пользователь может выполнять в этом пользовательском интерфейсе.

Элементы раскадровки связаны с исходным кодом. Важно понимать связь раскадровки с кодом, который пишется.

В раскадровке сцена представляет один экран и обычно один контроллер вида. Контроллеры видов реализуют поведение приложения. Контроллер вида управляет одним видом с его иерархией подчиненных видов. Контроллеры видов координируют поток информации между моделью данных приложения, которая инкапсулирует данные приложения, и видами, отображающими эти данные, обрабатывают изменения ориентации при повороте устройства, определяют навигацию в приложении и реализуют поведение, реагирующее на ввод данных пользователем. Все объекты контроллера вида в iOS имеют тип *UIViewController* или один из его подклассов.

Поведение контроллеров видов определяется в коде путем создания и внедрения пользовательских подклассов контроллеров видов. Затем можно создать связь между этими классами и кадрами в раскадровке, чтобы получить поведение, определенное в коде, и пользовательский интерфейс, определенный в раскадровке.

Xcode уже создал один такой класс, который мы видели ранее, *ViewController.swift*, и подключил его к кадру, над которым работали в предыдущей главе. В будущем при добавлении дополнительных сцен разработчик могут сами устанавливать такую связь в *Identity inspector* – рисунок 7.1.

Identity inspector позволяет редактировать связанные с объектом свойства в раскадровке, например, класс, к которому принадлежит объект.

Во время запуска раскадровка создает экземпляр *ViewController* – пользовательский подкласс контроллера вида. Сцена из раскадровки появляется на экране устройства, а поведение пользовательского интерфейса определяется в *ViewController.swift*.

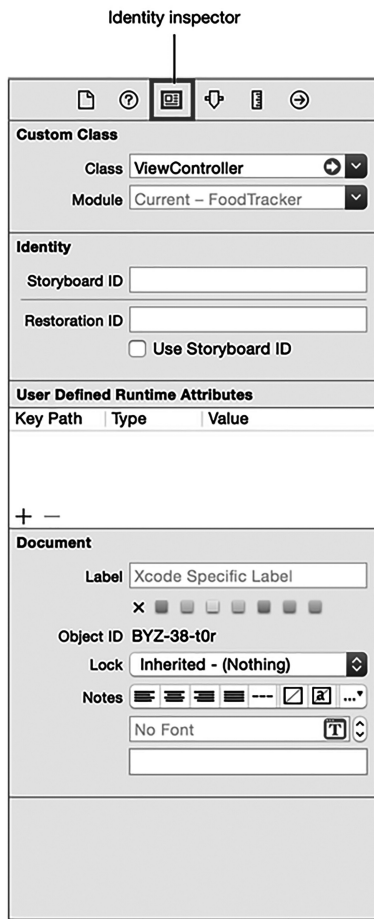


Рисунок 7.1 – Identity inspector

Хотя кадр и подключен к *ViewController.swift*, это не единственное подключение, которое нужно сделать. Чтобы прописать взаимодействия в приложении, исходный код контроллера видов должен взаимодействовать с видами на раскадровке. Это можно сделать, создав дополнительные соединения – называемые выводами и действиями – между видами в раскадровке и файлами исходного кода контроллера видов.

7.2 Создание выводов для элементов пользовательского интерфейса

Выводы обеспечивают возможность ссылки на объекты интерфейса – объекты, добавленные в раскадровку из файлов исходного кода. Чтобы создать вывод, перетащим определенный объект раскадровки в файл контроллера вида, удерживая нажатой клавишу **Control**. Это действие создаст свойство для объекта в файле контроллера видов, которое позволяет получить доступ к этому объекту и управлять им из кода во время выполнения.

Необходимо создать выводы для текстового поля и строки в интерфейсе пользователя, чтобы иметь возможность ссылаться на них. Для этого откройте раскадровку *Main.storyboard*.

Нажмите кнопку *Assistant* на панели инструментов *Xcode* в правом верхнем углу, чтобы открыть редактор – рисунок 7.2.

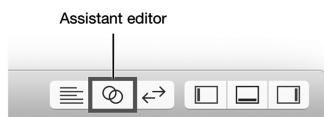


Рисунок 7.2 – Кнопка Assistant

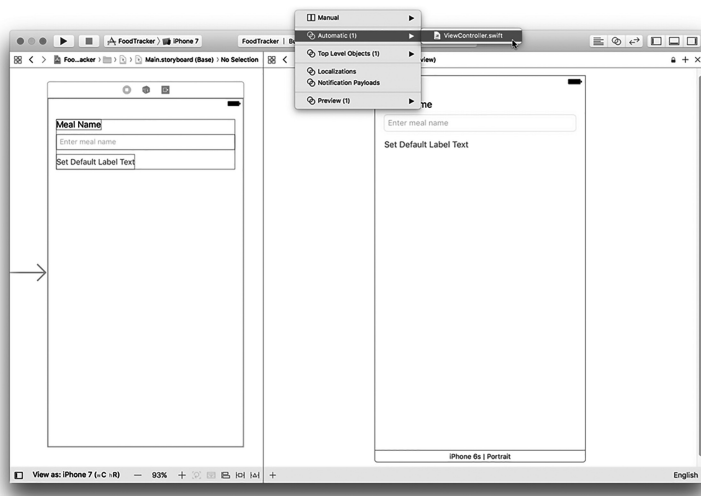


Рисунок 7.3 – Переключение на Automatic, ViewController.swift

Если требуется больше места для работы, сверните навигатор проекта и уилиты, нажав кнопки *Navigator* и *Utilities* на панели инструментов *Xcode*.

На панели выбора редактора, которая отображается в верхней части *Assistant*, измените *Preview* на *Automatic*, *ViewController.swift* – рисунок 7.3. Теперь *ViewController.swift* отображается в редакторе справа.

В *ViewController.swift* найдите строку *class*, которая должна выглядеть следующим образом:

```
class ViewController: UIViewController {
```

Под строкой класса добавьте следующее:

```
//MARK: Properties
```

Мы только что добавили комментарий к своему исходному коду. Комментарий – это фрагмент текста в файле исходного кода, который не компилируется как часть программы, но предоставляет контекстную или полезную информацию об отдельных фрагментах кода.

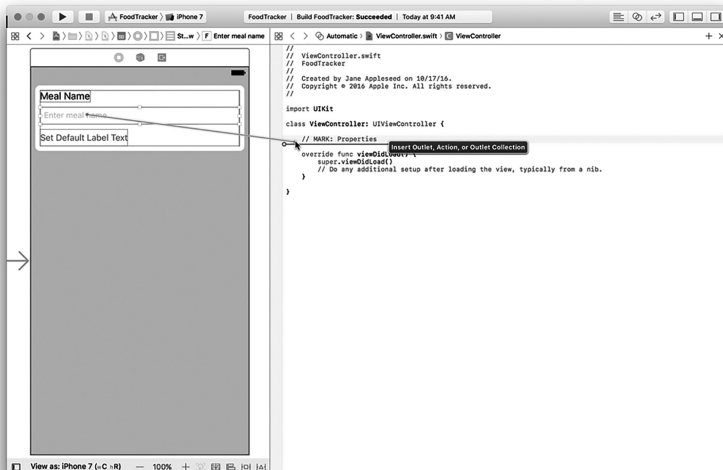


Рисунок 7.4 – Перетаскивание текстового поля в код

Комментарий, который начинается с символов **//MARK:** является специальным типом комментария, который используется для организации кода и для помощи любому, кто читает код, перемещаться по нему. В частности, добавленный комментарий указывает, что это раздел кода, в котором перечислены свойства.

В раскадровке выберите текстовое поле. Зажав клавишу **Control**, перетащите текстовое поле на холсте в код в редакторе справа, остановив перетаскивание в строке под комментарием, только что добавленным в **ViewController.swift** – рисунок 7.4.

В появившемся диалоговом окне в поле **Name** введите **nameTextField**. Оставьте остальные параметры по умолчанию – рисунок 7.5. Нажмите **Connect**.

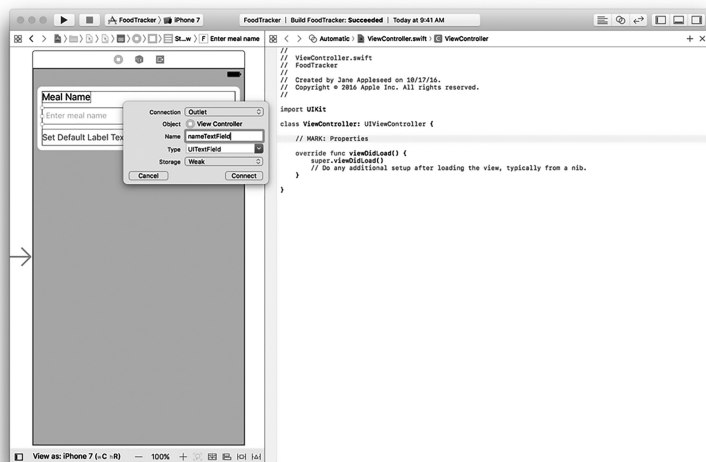


Рисунок 7.5 – Диалоговое окно для связи объекта с кодом

Xcode добавляет необходимый код в **ViewController.swift** для сохранения ссылки на текстовое поле и настраивает в раскадровке это соединение.

@IBOutlet weak var nameTextField: UITextField!

Атрибут **IBOutlet** указывает **Xcode**, что можно подключиться к свойству **nameTextField** из **Interface Builder** (поэтому атрибут имеет

префикс **IB**). Ключевое слово `weak` указывает на то, что ссылка не мешает системе отменить выделение объекта. Слабые ссылки помогают предотвратить циклы ссылок. Однако для сохранения объекта в памяти необходимо убедиться, что в какой-то другой части приложения есть сильная ссылка на объект. В данном случае это супервид текстового поля. Супервид поддерживает сильную ссылку на все подвиды. Пока супервид остается активным и в памяти, все подвиды тоже остаются активными. Аналогично, контроллер видов имеет сильную ссылку на свой главный вид – сохраняя всю иерархию видов активными и в памяти.

Остальная часть декларирования определяет неявно распакованную необязательную переменную типа `UITextField` с именем `nameTextField`. Обратите внимание на восклицательный знак в конце объявления типа. Этот восклицательный знак указывает, что тип является неявно распакованным необязательным типом, который всегда должен иметь значение после того, как он заявлен. При доступе к неявно распакованному необязательному элементу система предполагает, что оно имеет допустимое значение и автоматически распаковывает его. Обратите внимание, что это приводит к завершению работы приложения, если значение переменной еще не задано.

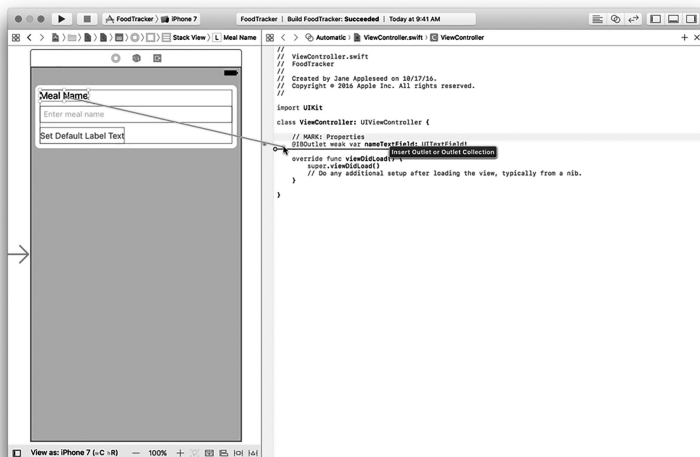


Рисунок 7.6 – Перетаскивание строки в код

При загрузке контроллера вида из раскадровки система создает иерархию видов и присваивает соответствующие значения всем выходам контроллера вида. К моменту вызова метода `viewDidLoad()` из контроллера видов система уже присвоит допустимые значения всем выводам контроллера, и можно безопасно получить доступ к их содержанию.

Теперь надо подключить строку к коду. Для этого на раскадровке выберите строку. Удерживая клавишу **Control**, перетащите строку на холсте в код в редакторе справа, остановив перетаскивание в строке непосредственно под свойством `nameTextField` в `ViewController.swift` – рисунок 7.6.

В диалоговом окне, которое появляется, в поле `Name` введите `mealNameLabel`. Оставьте остальные параметрами по умолчанию – рисунок 7.7. Нажмите **Connect**.

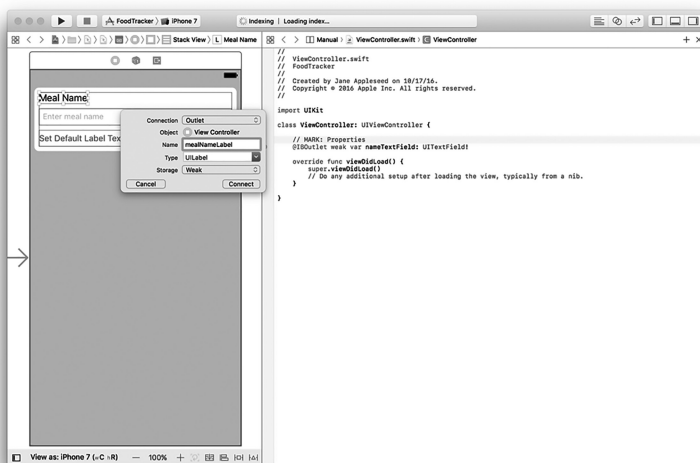


Рисунок 7.7 – Диалоговое окно для связи строки с кодом

И вновь `Xcode` добавляет необходимый код в `ViewController.swift` для сохранения ссылки на строку и настраивает в раскадровке это соединение. Этот вывод аналогичен текстовому полю, за исключением его имени и типа (`UILabel`, в соответствии с типом объекта в раскадровке).

@IBOutlet weak var mealNameLabel: UILabel!

Вывод для объекта интерфейса необходим только в том случае, если нужен доступ к значению из объекта интерфейса или планируется изменить объект интерфейса в коде. В этом случае необходимо задать свойство *delegate* текстового поля и свойство *text* строки. Мы не будем изменять кнопку, поэтому нет причин создавать для нее вывод.

Выводы позволяют ссылаться на элементы интерфейса в коде, но все равно требуется способ реагирования при каждом взаимодействии пользователя с элементами. Для этого как раз и нужны действия.

7.3 Добавление действия

Приложения iOS основаны на программировании на основе событий. То есть поток приложения определяется событиями: системными событиями и действиями пользователей. Пользователь выполняет действия в интерфейсе, которые инициируют события в приложении. Эти события приводят к выполнению логики приложения и манипуляции с его данными. Ответ приложения на действие пользователя затем отражается обратно в интерфейсе пользователя. Поскольку пользователь, а не разработчик, контролирует время выполнения определенных частей кода приложения, необходимо точно определить, какие действия пользователь может выполнить и что происходит в ответ на эти действия.

Действие (или метод действия) – это часть кода, связанная с событием, которое может произойти в приложении. Можно определить метод действия для выполнения любых действий от манипулирования фрагментом данных до обновления пользовательского интерфейса. Действия используются для управления потоком приложения в ответ на события пользователя или системы.

Действие создается таким же образом, как и при создании вывода: перетаскивание определенного объекта из раскладки в файл контроллера вида при нажатой клавише *Control*. Эта операция создает метод в файле контроллера вида, который запускается при взаимодействии пользователя с объектом, к которому присоединен метод действия.

Начнем с создания простого действия, в котором строка будет иметь значение *Default Text*, если пользователь нажимает кнопку *Set*

Default Text. Для этого в *ViewController.swift*, непосредственно над последней фигурной скобкой *}*, добавьте следующее:

```
//MARK: Actions
```

Этот комментарий указывает, что это раздел кода, в котором перечислены действия.

На раскладке нажмите кнопку *Set Default Label Text*.

Перетащите курсор с кнопки *Set Default Label Text* на холсте на отображение кода в редакторе справа, остановив перетаскивание в строке под комментарием, только что добавленным в *ViewController.swift* – рисунок 7.8.

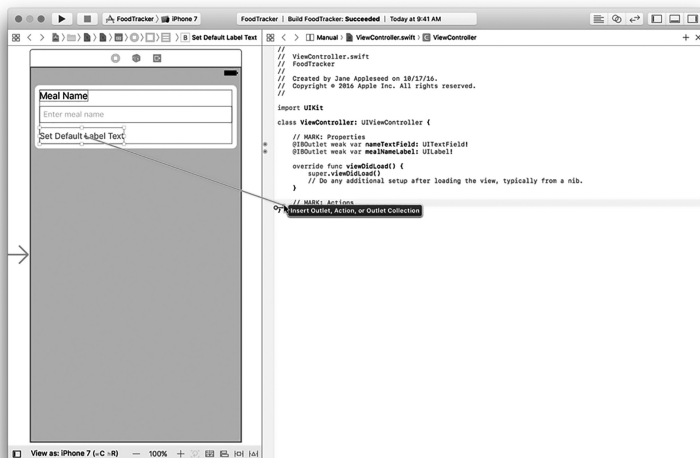


Рисунок 7.8 – Добавление действия

В появившемся диалоговом окне для параметра *Connection* выберите *Action*. Для *Name* напечатайте *setDefaultLabelText*. Для параметра *Type* выберите значение *UIButton*. Значение поля *Type* по умолчанию равно *AnyObject*. В *Swift* *AnyObject* – это тип, используемый для описания объекта, который может принадлежать любому классу. Указание типа этого метода действия как *UIButton* означает, что только объекты кнопок могут подключаться к этому действию. Важно помнить об этом при разработке. Оставьте остальные параметры по умолчанию – рисунок 7.9. Нажмите *Connect*.

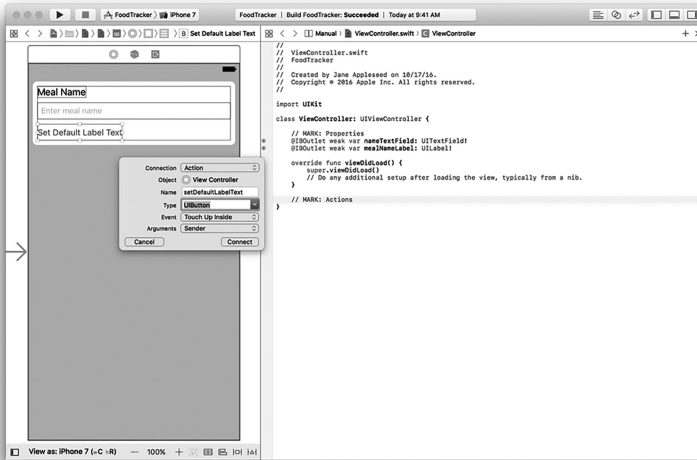


Рисунок 7.9 – Задание параметров действия

Xcode добавляет необходимый код в *ViewController.swift* для настройки метода действия.

```
@IBAction func setDefaultLabelText(_ sender: UIButton) {
}
```

Параметр *sender* относится к объекту, ответственному за инициирование действия – в данном случае кнопке. Атрибут *IBAction* указывает, что метод является действием, к которому можно подключиться из раскадровки в *Interface Builder*. Остальная часть декларации объявляет метод под названием *setDefaultLabelText* (*_* :).

На данный момент декларация метода пустая. Но код для сброса значения строки достаточно прост. Так, для выполнения такого сброса в *ViewController.swift* найдите метод действия *setDefaultLabelText*, который мы только что добавили. В реализации метода между фигурными скобками *}* добавьте следующую строку кода:

```
mealNameLabel.text = «Default Text»
```

Как можно догадаться, этот код задает для параметра *text* строки значение *Default Text*.

IOS обрабатывает весь код перерисовки, так что это на самом деле весь код, который нужно написать на данный момент. Метод действия `setDefaultLabelText(_:)` должен выглядеть следующим образом:

```
@IBAction func setDefaultLabelText(_ sender: UIButton) {  
    mealNameLabel.text = «Default Text»  
}
```

Теперь проверьте изменения, запустив симулятор. При нажатии кнопки *Set Default Label Text*, вызывается `setDefaultLabelText (_:)` метод, и `text` значение `mealNameLabel` объекта изменяется с *Meal Name* (значение при запуске приложения) на *Default Text* (значение в результате действия). Изменения должны отображаться в интерфейсе пользователя – рисунок 7.10.

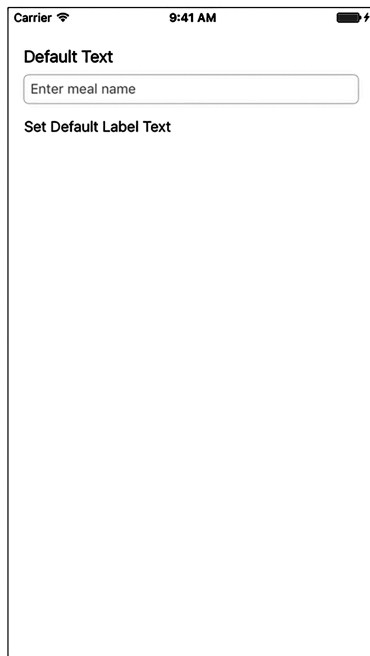


Рисунок 7.10 – Интерфейс с добавленным действием

Хотя изменение названия блюда на *Default Text* не особенно полезно, оно иллюстрирует важный момент. Только что реализованное поведение является примером модели *Target-action* (целевого действия) в дизайне приложения iOS. *Target-action* – это шаблон проектирования, в котором один объект отправляет сообщение другому объекту при возникновении определенного события.

В этом случае:

- 1) Событие – это нажатие пользователем кнопки *Set Default Text*.
- 2) Действие – *setDefaultLabelText (_)*.
- 3) Целевой объект – *ViewController* (где определен метод действия).
- 4) Отправителем является кнопка *Set Default Label Text*.

Система отправляет сообщение, вызывая метод действия на целевой объект и передавая в объект отправителя. Отправителем обычно является элемент управления (например, кнопка, ползунок или переключатель), который может инициировать событие в ответ на взаимодействие пользователя, например, касание, перетаскивание или изменение значения. Эта модель чрезвычайно распространена в программировании приложений для iOS.

7.4 Процесс ввода данных пользователем

Теперь пользователи могут сбросить строку с именем блюда на значение по умолчанию, но необходимо дать возможность пользователю вводить собственные имена блюд с помощью текстового поля. Программа будет просто обновлять *text* значение *mealNameLabel* объекта каждый раз, когда пользователь вводит текст в текстовое поле и нажимает *Return*.

При работе с пользовательским вводом из текстового поля требуется помощь делегата текстового поля. Делегат – это объект, действующий от имени или в координации с другим объектом. Делегирующий объект – в данном случае текстовое поле сохраняет ссылку на другой объект. Делегат в соответствующее время делегирующий объект посылает сообщение делегату. Сообщение сообщает делегату о событии, которое объект делегирования собирается обработать или только что обработал. Делегат может ответить, например, обновляя внешний вид или состояние самого себя или других объектов в приложении или возвращая значение, которое влияет на обработку предстоящего события.

Делегат текстового поля взаимодействует с текстовым полем во время редактирования текста пользователем и знает, когда, например, пользователь начинает или прекращает редактирование текста. Делегат может использовать эту информацию для сохранения или очистки данных в нужное время, закрытия клавиатуры и так далее.

Любой объект может служить делегатом для другого объекта, если он соответствует протоколу. Протокол, определяющий делегат текстового поля, называется *UITextFieldDelegate*. Очень часто контроллер вида становится делегатом для управляемых им объектов. В рассматриваемом примере экземпляр *ViewController* станет делегатом текстового поля.

Для начала *ViewController* должен принять протокол *UITextFieldDelegate*. Протокол принимается, если его перечислить как часть строки декларирования класса. Для этого вернитесь в стандартный редактор, нажав кнопку *Standard*. Разверните навигатор проекта и утилиты, нажав кнопки *Navigator* и *Utilities* на панели инструментов *Xcode*. В навигаторе проекта выберите *ViewController.swift*.

В *ViewController.swift* найдите строку *class*, которая должна выглядеть следующим образом:

```
class ViewController: UIViewController {
```

После *UIViewController* добавьте запятую , и *UITextFieldDelegate* для принятия протокола:

```
class ViewController: UIViewController, UITextFieldDelegate {
```

Протокол *UITextFieldDelegate* сообщает компилятору, что класс *ViewController* может действовать как допустимый делегат текстового поля. Это означает, что можно реализовать методы протокола для обработки ввода текста и назначить экземпляры класса *ViewController* в качестве делегата текстового поля.

Для установки объекта *ViewController* в качестве делегата его свойства *nameTextField* в *ViewController.swift* найдите метод *viewDidLoad ()*, который должен выглядеть следующим образом:

```
override func viewDidLoad() {  
    super.viewDidLoad()
```

```
// Do any additional setup after loading the view, typically from a nib.  
}
```

Реализация шаблона этого метода включает комментарий. Этот комментарий не нужен в реализации метода, поэтому удалите его.

Под линией *super.viewDidLoad ()* добавьте пустую строку и следующее:

```
// Handle the text field's user input through delegate callbacks.  
nameTextField.delegate = self
```

self ссылается на класс *ViewController*, поскольку на него имеется ссылка в области определения класса *ViewController*. Можно также добавить собственные комментарии, которые помогут понять, что происходит в коде.

Метод *viewDidLoad ()* должен выглядеть следующим образом:

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    // Handle the text field's user input through delegate callbacks.  
    nameTextField.delegate = self  
}
```

При загрузке *ViewController* он устанавливает себя в качестве делегата свойства *nameTextField*.

Протокол *UITextFieldDelegate* определяет восемь возможных методов. Просто реализуйте те, которые нужны, чтобы получить желаемое поведение. На данный момент необходимо внедрить два из этих методов:

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool  
func textFieldDidEndEditing(_ textField: UITextField)
```

Чтобы понять, когда эти методы вызываются и что делают, важно знать, как текстовые поля реагируют на события пользователя. Когда пользователь нажимает на текстовое поле, оно автоматически становится первым исполнителем. В приложении первый исполнитель это –

объект, который первым получает информацию о многих видах событий приложения, включая ключевые события, события движения и сообщения о действиях. Другими словами, многие события, генерируемые пользователем, первоначально направляются первому исполнителю.

В результате того, что текстовое поле становится первым исполнителем, iOS отображает клавиатуру и начинает сеанс редактирования для этого текстового поля. То, что пользователь набирает с помощью этой клавиатуры, вставляется в текстовое поле.

Когда пользователь хочет закончить редактирование текстового поля, текстовое поле должно отменить свой статус первого исполнителя. Поскольку текстовое поле больше не будет активным объектом в приложении, события должны быть направлены на более подходящий объект.

На этом этапе осуществляется внедрение методов *UITextFieldDelegate*. Необходимо указать, что текстовое поле должно отменить свой статус первого исполнителя, когда пользователь нажимает кнопку, чтобы завершить редактирование в текстовом поле. Это делается в *textFieldShouldReturn* (:) метод, который вызывается, когда пользователь нажимает *Return* (или Вашем случае, *Done*) на клавиатуре.

Чтобы осуществить метод *textFieldShouldReturn* (:) протокола *UITextFieldDelegate* в *ViewController.swift*, прямо над **//MARK: Actions**, добавьте следующее:

```
//MARK: UITextFieldDelegate
```

Этот комментарий используется для организации кода и помогает любому пользователю, читающему код перемещаться по нему.

Мы уже добавили несколько комментариев. *Xcode* перечисляет каждый из этих комментариев как заголовок раздела в *Functions menu* исходного кода, которое появляется, если щелкнуть имя файла в верхней части редактора – рисунок 7.11.

Functions menu позволяет быстро перейти к разделу кода. Можно увидеть разделы, обозначенные как **//MARK:** написанные ранее. Можно щелкнуть один из заголовков раздела, чтобы перейти к этому разделу в файле.

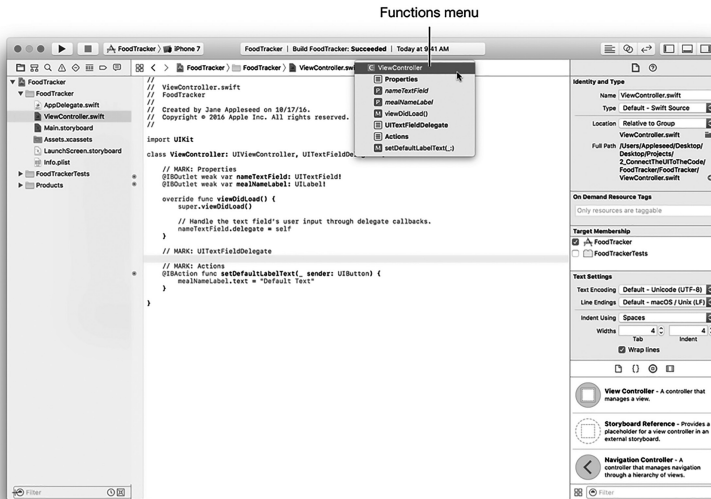


Рисунок 7.11 – Открытие Functions menu

Под комментарием **//MARK: UITextFieldDelegate** добавьте следующий метод:

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
}
```

В этом методе добавьте следующий код для отмены статуса текстового поля как первого исполнителя и комментарий для описания действий кода:

```
// Hide the keyboard.
textField.resignFirstResponder()
```

Попробуйте не копировать и вставить, а ввести вторую строку. Вы увидите, что завершение кода является одной из лучших функций **Xcode**, экономящих время. Когда **Xcode** выведет список потенциальных завершений – рисунок 7.12, прокручивайте список до тех пор, пока не найдете нужный, а затем нажмите кнопку **Return**. **Xcode** вставит всю строку.

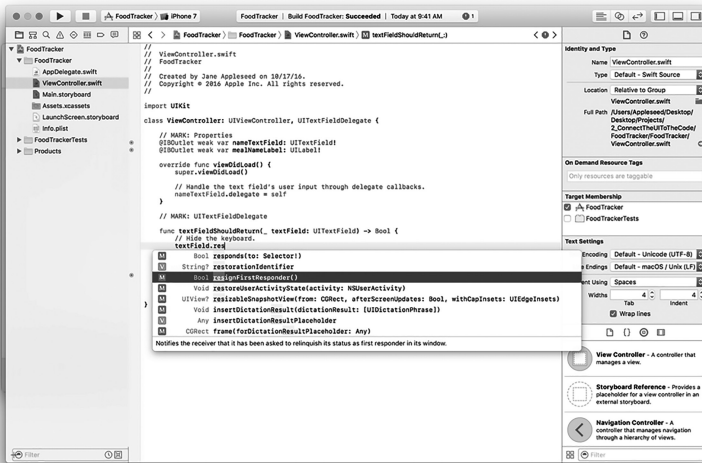


Рисунок 7.12 – Подсказка возможных завершений в Xcode

В этом методе добавьте следующую строку кода:

```
return true
```

Этот метод возвращает логическое значение, указывающее, должна ли система обрабатывать нажатие клавиши **Return**. В описываемом примере всегда требуется действовать при нажатии пользователем **Return**, поэтому надо вернуть значение true.

Теперь **textFieldShouldReturn** (:) метод должен выглядеть так:

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    // Hide the keyboard.
    textField.resignFirstResponder()
    return true
}
```

Второй метод, который необходимо реализовать, **textFieldDidEndEditing**(:), вызывается после того, как текстовое поле отдает свой статус первого исполнителя. Поскольку Вы оставляете статус первого исполнителя в **textFieldShouldReturn**, система вызывает этот метод сразу после запроса **textFieldShouldReturn**.

Метод *textFieldDidEndEditing* (:) дает шанс прочитать информацию, введенную в текстовое поле, и что-то с ней сделать. В приложении будете брать текст, который находится в текстовом поле, и использовать его для изменения значения строки.

Чтобы осуществить метод *textFieldDidEndEditing* (:) протокола *UITextFieldDelegate* в *ViewController.swift*, после метода *textFieldShouldReturn* (:) , добавьте следующий метод:

```
func textFieldDidEndEditing(_ textField: UITextField) {  
}
```

В этот метод добавьте следующую строку кода:

```
mealNameLabel.text = textField.text
```

Это все, что нужно сделать. В итоге метод *textFieldDidEndEditing* (:) должен быть таким:

```
func textFieldDidEndEditing(_ textField: UITextField) {  
    mealNameLabel.text = textField.text  
}
```

Проверьте изменения, запустив симулятор. Можно выделить текстовое поле и ввести в него текст. При нажатии кнопки **Done** клавиатура прерывается, и текст строки изменяется на текст из текстового поля. При нажатии кнопки **Set Default Label Text** строка меняется на **Default Text** (значение, заданное действием ранее).

Для дальнейшего усовершенствования этого приложения можно воспользоваться уроками по ссылке:

https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/WorkWithViewControllers.html#apple_ref/doc/uid/TP40015214-CH6-SW1

8 СОЗДАНИЕ МОБИЛЬНОГО WEB-ПРИЛОЖЕНИЯ

8.1 Web-разработка как основа мобильного приложения

На сегодняшний день сфера программирования расширилась настолько, что большинство программистов еще на стадии обучения специализируются на каком-то определенном направлении разработки. И, соответственно, компаниям приходится для различных стоящих перед ними задач нанимать одного или целую команду узких специалистов.

Если говорить о мобильной разработке, то компаниям, разрабатывающим свои мобильные приложения, приходится иметь в штате как минимум двух специалистов: специалиста по Android и iOS. А часто, это целые команды «Android'еров» и «iOS'еров». Конечно же содержание большого штата программистов весьма накладно для компании. Поэтому появилась альтернативная возможность для разработки мобильных приложений: мобильные Web-приложения. А, учитывая, что у большинства ИТ-отделов и компаний есть группа Web-разработчиков, это существенно оптимизирует затраты на разработку.

Последовательность разработки мобильного Web-приложения такая:

- 1) Создается Web-приложение (в том числе и как часть сайта для компании).
- 2) Web-приложение преобразуется в мобильное приложение под определенную операционную систему (Android или iOS).

Из минусов такой разработки мобильных приложений можно назвать меньшую нативность и отчасти ограниченный функционал таких мобильных приложений. Но, стоит заметить, мало кто из рядовых пользователей может вообще догадаться, что это именно мобильное Web-приложение. Особенно если оно хорошо написано и отлажено. В этой главе Вы сможете повторить разработку мобильного Web-приложения для iOS на примере все той же игры про космический корабль летящий между метеоров.

Для удобства создания самого Web-приложения с этой игрой установим *VS Code* с официального сайта (<https://code.visualstudio.com/>) – рисунок 8.1.

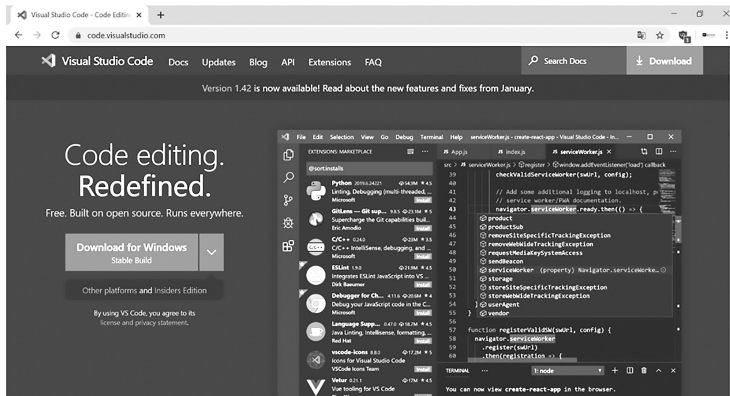


Рисунок 8.1 – Официальный сайт VS Code

Стоит отметить, что это не обязательно и для разработки Web-приложения можно использовать любой редактор кода. Кроме *VS Code*, это еще и *Notepad++*, *Sublime Text*, *TextMate* *Notepad++* и другие.

Но вернемся к установке. В первую очередь, выбираем папку для установки – рисунок 8.2.

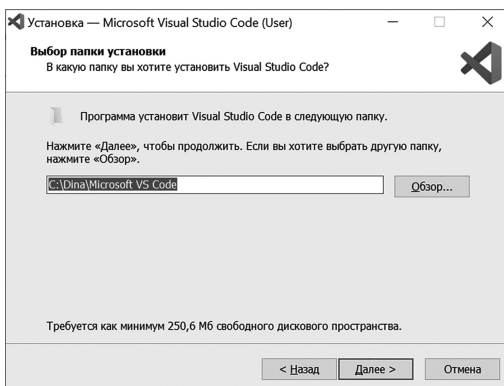


Рисунок 8.2 – Выбор папки для установки VS Code

Всегда старайтесь устанавливать компоненты и системы для программирования по адресам, не содержащим кириллицу. Для многих систем – это становится существенным препятствием для полноценной работы. Особенно при разработке мобильных приложений.

Теперь задаем название папки в меню **Пуск** – рисунок 8.3. Либо отказываемся от такой папки убрвав галочку в нижней строке.

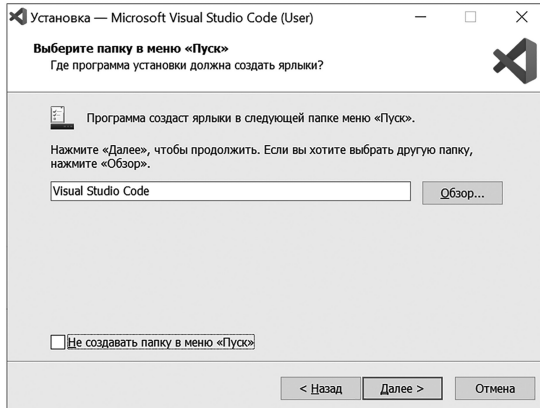


Рисунок 8.3 – Создание папки в меню «Пуск» для VS Code

После этого рекомендуется добавить **VS Code** в **PATH** для Windows по умолчанию – рисунок 8.4. Это позволит дополнительным системам разработки легко находить этот редактор кода без Вашего участия. Также очень удобно открывать файлы с Web-разработками, если заранее добавить возможность открытия файлов в контекстное меню проводника Windows.

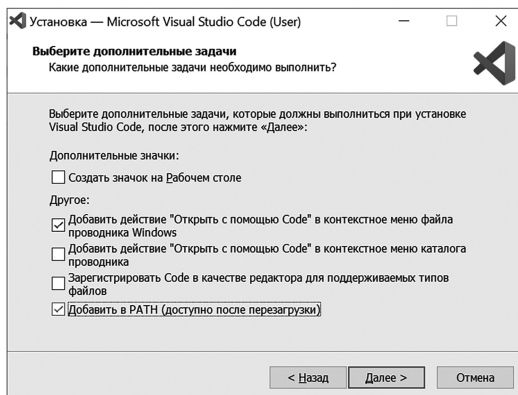


Рисунок 8.4 – Включение дополнительных задач для VS Code

Осталось проверить правильно ли все введено и запустить установку – рисунок 8.5.

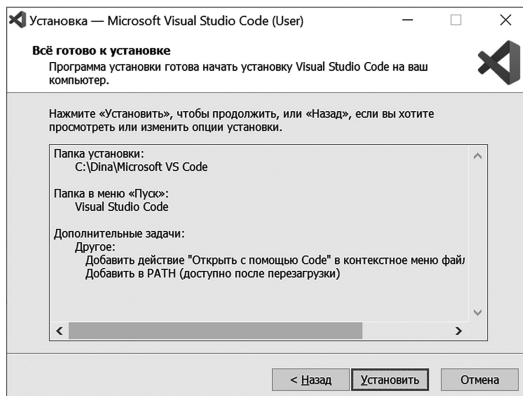


Рисунок 8.5 – Запуск установки VS Code

Теперь редактор кода установлен и готов к работе – рисунок 8.6.

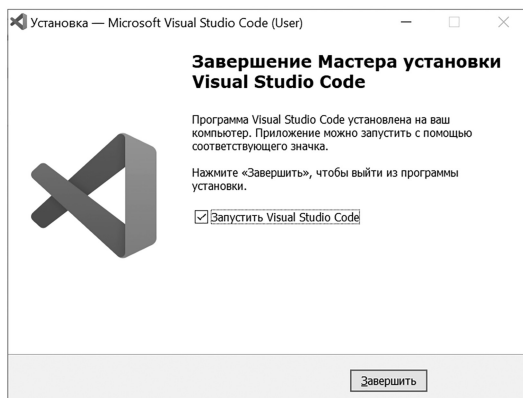


Рисунок 8.6 – Завершение установки VS Code

Приступим к разработке самого приложения.

8.2 Разработка игры как Web-приложения

Найдите **VS Code** в меню пуск и запустите. После открытия в левом верхнем углу откройте меню **File** и выберите **New File** – рисунок 8.7.

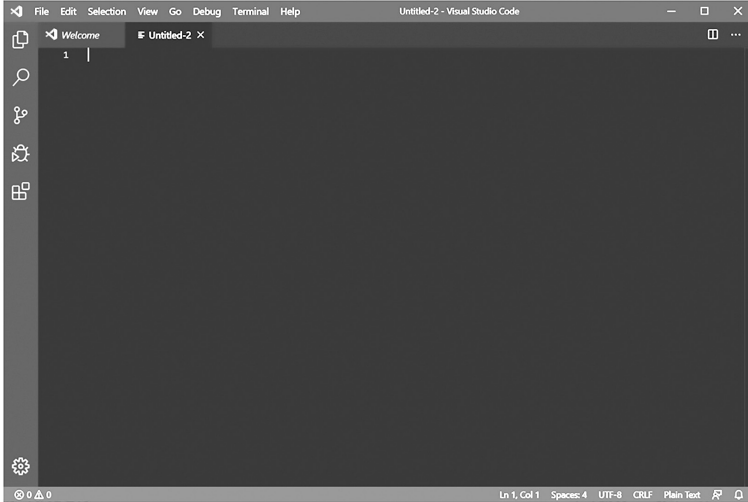


Рисунок 8.7 – Открытие нового файла в VS Code

Теперь введите следующий код:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Spaceship</title>  
    <link rel=»stylesheet» type=»text/css» href=»style.css»>  
  </head>  
  
  <body>  
    <span class=»title»>  
      Spaceship  
    </span>  
  
    <a class=»button» href=»game.html»>  
      Play
```

```
</a>
</body>
</html>
```

Это основной файл Вашего мобильного приложения. Сохраните его как HTML-файл под названием *index.html* в папке, где Вы будете собирать и хранить свой проект. Не забудьте, что папка хранения проекта тоже желательно должна быть без букв кириллицы в адресе.

Теперь, там же создаем новый файл в формате *.css* для хранения цветов и стилей текста. Называем его *style.css* и сохраняем в папке проекта. Код внутри должен быть следующим:

```
html, body {
    margin: 0;
    overflow: hidden;
    height: 100%;
    width: 100%;

    background-color: #000000;

    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}

.title {
    font-family: monospace;
    font-size: 6vw;
    color: antiquewhite;
}

.button {
    font-family: monospace;
    font-size: 3vw;
    color: bisque;
}
```

Теперь основная страница Web-приложения будет выглядеть как на рисунке 8.8.

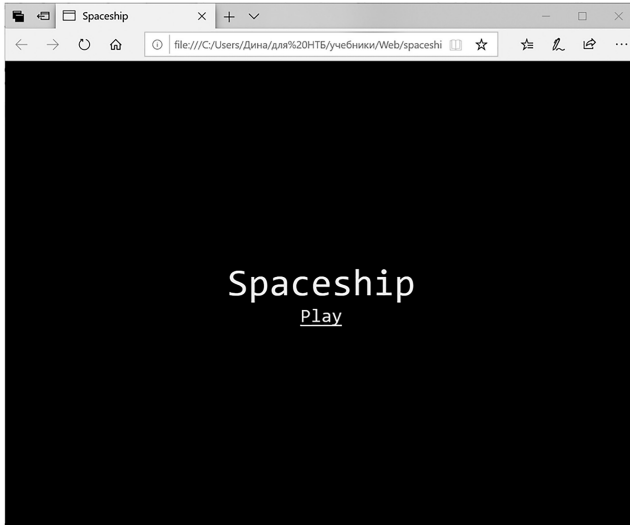


Рисунок 8.8 – Основная страница Web-приложения

Создайте так же файл под названием **game.html** с основной логикой игры:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Spaceship</title>  
    <style>  
      html, body {  
        margin: 0;  
        overflow: hidden;  
      }  
    </style>  
  
    <script>  
      const FRAME_RATE = 60  
  
      let ship = {  
        x: 0,  
        y: 0,
```

```
width: w(100),  
height: h(50),  
speed: 20,
```

```
draw(ctx) {  
  let shipImg = document.getElementById('ship')  
  ctx.drawImage(shipImg, this.x, this.y, this.width, this.height)  
}
```

```
moveUp() {  
  this.y -= this.speed  
  if (this.y < 0) {  
    this.y = 0  
  }  
}
```

```
moveDown() {  
  this.y += this.speed  
  if (this.y > window.innerHeight - this.height) {  
    this.y = window.innerHeight - this.height  
  }  
}
```

```
let buttonUp = {  
  x: w(60),  
  y: h(425),  
  width: w(25),  
  height: w(25),
```

```
draw(ctx) {  
  ctx.fillStyle = 'white'  
  ctx.beginPath();  
  ctx.ellipse(this.x, this.y, this.width, this.height, 0, 0, Math.PI * 2)  
  ctx.fill();  
}
```

```
let buttonDown = {  
  x: w(460),
```

```
y: h(425),  
width: w(25),  
height: w(25),
```

```
draw(ctx) {  
  ctx.fillStyle = 'white'  
  ctx.beginPath();  
  ctx.ellipse(this.x, this.y, this.width, this.height, 0, 0, Math.PI * 2)  
  ctx.fill();  
}  
}
```

```
let meteor = {  
  x: window.innerWidth,  
  y: 0,  
  width: w(50),  
  height: w(25),  
  speed: 10,
```

```
reset() {  
  this.x = window.innerWidth  
  this.y = Math.random() * window.innerHeight  
  this.speed = Math.random() * 3 + 5  
},
```

```
draw(ctx) {  
  this.x -= this.speed  
  if (this.x < 0) {  
    score++  
    this.reset()  
  }  
}
```

```
let meteorImg = document.getElementById('meteor')  
ctx.drawImage(meteorImg, this.x, this.y, this.width, this.height)  
},
```

```
blow() {  
  let blow = {  
    x: this.x,  
    y: this.y,  
    width: w(200),
```



```

        height: w(50),

        draw(ctx) {
            let meteorImg = document.getElementById('blow')
            ctx.drawImage(meteorImg, this.x, this.y, this.width,
this.height)
        }
    }

    blows.push(boom)

    window.setTimeout(() => blows.shift(), 500)
}

let meteors = []
let blows = []

let lives = 3
let score = 0

function setup() {
    // Растягиваем холст на всю страничку
    const canvas = document.getElementById('canvas')
    canvas.width = window.innerWidth
    canvas.height = window.innerHeight

    // Добавляем метеоры
    for (let i = 0; i < 3; i++) {
        let m = Object.assign({}, meteor) // копируем метеор
        m.reset()
        meteors.push(m)
    }

    canvas.addEventListener('click', onClick)
    window.setInterval(draw, 1000/FRAME_RATE)
}

function draw() {

```

```

const canvas = document.getElementById('canvas')
const ctx = canvas.getContext('2d')

let backgroundImg = document.getElementById('sky')
  ctx.drawImage(backgroundImg, 0, 0, window.innerWidth,
window.innerHeight)

ship.draw(ctx)

buttonUp.draw(ctx)
buttonDown.draw(ctx)

for (let m of meteors) {
  m.draw(ctx)
  if (rectCollision(ship, m)) {
    console.log('BOOM')
    lives--

    if (lives <= 0) {
      window.location.replace('gameover.html')
    }

    m.blow()
    m.reset()
  }
}

for (let b of blows) {
  b.draw(ctx)
}

ctx.fillStyle = 'white'
ctx.font = '4vw monospaced'
ctx.fillText(`Lives: ${lives}`, w(10), h(40))
ctx.fillText(`Score: ${score}`, w(10), h(90))
}

function onClick(event) {
  if (isPointInEllipse(buttonUp, event.clientX, event.clientY)) {

```

```

        ship.moveUp()
    }

    if (isPointInEllipse(buttonDown, event.clientX, event.clientY)) {
        ship.moveDown()
    }
}

function rectCollision(rect1, rect2) {
    return rect1.x < rect2.x + rect2.width &&
        rect1.x + rect1.width > rect2.x &&
        rect1.y < rect2.y + rect2.height &&
        rect1.y + rect1.height > rect2.y
}

function isPointInEllipse(ellipse, x, y) {
    let val = Math.pow(x - ellipse.x, 2) / Math.pow(ellipse.width, 2) +
        Math.pow(y - ellipse.y, 2) / Math.pow(ellipse.height, 2)

    return val >= 1
}

function w(size) {
    return size * (window.innerWidth / 500)
}

function h(size) {
    return size * (window.innerHeight / 500)
}

</script>
</head>
<body onload=>setup()>
    <canvas id=>canvas>
        <img src=>assets/blow.png id=>blow />
        <img src=>assets/meteor.png id=>meteor />
        <img src=>assets/ship.png id=>ship />
    </canvas>
</body>
</html>

```

```

    <img src=»assets/sky.png» id=»sky» />
  </canvas>
</body>
</html>

```

Результат работы такой страницы будет выглядеть как на рисунке 8.9.

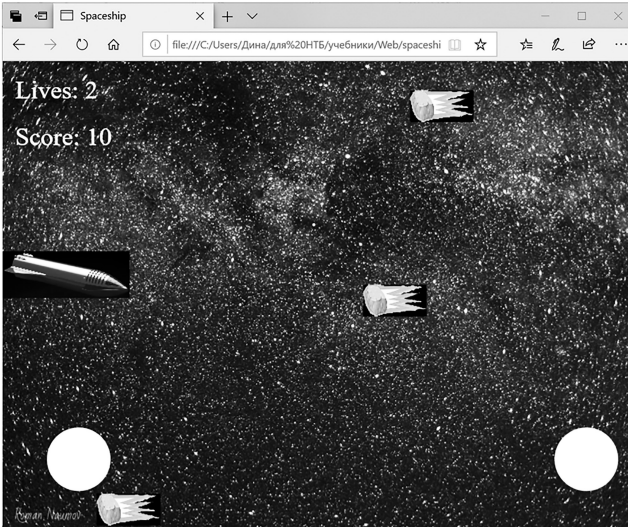


Рисунок 8.9 – Игра как Web-приложение

Не хватает только экрана проигрыша. Создаем новый файл под названием **gameover.html** с кодом:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Spaceship</title>
    <link rel=»stylesheet» type=»text/css» href=»style.css»>
  </head>

  <body>
    <span class=»title»>

```

```

        Game Over!
    </span>

    <a class=»button» href=»game.html»>
        Try again
    </a>
</body>
</html>

```

В результате получается экран как на рисунке 8.10.



Рисунок 8.10 – Экран проигрыша Web-приложения

Но чтобы получить Web-приложение в таком виде, как показано выше, не хватает рисунков для вставки. Для их хранения создайте в папке проекта папку под названием `assets` и вставьте туда изображения взрыва, метеора, корабля и неба под названиями ***blow.png, meteor.png, ship.png u sky.png*** соответственно.

Web-приложение готово к работе. При желании его можно выложить в общий доступ на сервер как часть сайта.

Но перед Вами сейчас стоит немного другая задача. Необходимо теперь преобразовать это Web-приложение в мобильное приложение.

8.3 Средства для разработки мобильных Web-приложений

В данном примере для преобразования Web-приложения в мобильное Вы будете использовать *Apache Cordova*.

Apache Cordova – это фреймворк для разработки мобильных устройств с открытым исходным кодом. Он позволяет использовать стандартные Web-технологии – *HTML5*, *CSS3* и *JavaScript* для разработки под разные платформы. Приложения выполняются в оболочках, предназначенных для каждой платформы, и используют соответствующие стандартам привязки API для доступа к возможностям каждого устройства, таким как датчики, данные, состояние сети и так далее.

Apache Cordova можно использовать, если Вы:

- 1) Хотите расширить приложение на несколько платформ без необходимости его повторного внедрения с использованием языка и набора инструментов каждой отдельной платформы.
- 2) Хотите развернуть Web-приложение, упакованное для распространения в различных магазинах приложений.

Cordova предоставляет два основных потока операций для создания мобильного приложения.

1) *Межплатформенный рабочий процесс*: Используйте этот рабочий процесс, если Вы хотите, чтобы приложение выполнялось на максимально возможном количестве мобильных операционных систем. При этом не требуется разработка для конкретной платформы. Этот рабочий процесс разворачивается вокруг интерфейса командной строки *Cordova*. Интерфейс командной строки – это инструмент высокого уровня, который позволяет создавать проекты сразу для многих платформ, забывая о большей части функциональности скриптов оболочки более низкого уровня. Интерфейс командной строки копирует общий набор Web-ресурсов во вложенные папки для каждой мобильной платформы, вносит необходимые изменения в конфигурацию для каждой из них, запускает сценарии сборки для создания двоичных файлов приложений. Интерфейс командной строки также предоставляет общий интерфейс для применения плагинов к приложению. Если у Вас нет необходимости в платформенном рабочем процессе, рекомендуется использовать межплатформенный рабочий процесс.

2) *Рабочий процесс, ориентированный на платформу*: Используйте этот рабочий процесс, если Вы хотите сосредоточиться на создании приложения для одной платформы и должны будете изменять его на более низком уровне. Вы должны использовать этот подход, например,

если Вы хотите, чтобы приложение использовало пользовательские нативные компоненты с Web-компонентами *Cordova*. Как правило, этот рабочий процесс используется при необходимости изменения проекта в *SDK*. Этот рабочий процесс основан на наборе скриптов оболочки нижнего уровня, адаптированных для каждой поддерживаемой платформы, и отдельной утилите *Plugman*, позволяющей применять плагины. Хотя можно использовать этот рабочий процесс для создания кроссплатформенных приложений, это, как правило, сложнее, потому что отсутствие инструмента более высокого уровня означает отдельные циклы сборки и модификации плагинов для каждой платформы.

При первом запуске проще всего использовать межплатформенный рабочий процесс для создания приложения. А уже после этого можно переключиться на процесс, ориентированный на платформу, если требуется больший контроль со стороны *SDK*.

После перехода от рабочего процесса на основе интерфейса командной строки к рабочему процессу, расположенному вокруг пакетов *SDK* и инструментов оболочки для конкретной платформы, нельзя вернуться назад. Интерфейс командной строки поддерживает общий набор кроссплатформенного исходного кода, который в каждом построении использует для записи специфичный для платформы исходный код. Чтобы сохранить любые изменения, внесенные в ресурсы платформы, необходимо переключиться на инструменты оболочки, ориентированные на платформу, которые игнорируют исходный код кросс-платформы и вместо этого полагаются на исходный код платформы.

Фреймворк *Apache Cordova* для полноценной работы требует установки *Node.js*.

Как асинхронная среда выполнения *JavaScript* на основе событий, *Node.js* предназначена для создания масштабируемых сетевых приложений. В коде *Node.js* многие соединения могут обрабатываться одновременно. После каждого соединения инициируется обратный вызов, но если нет работы, то *Node.js* уйдет в спящий режим.

Это сильно отличается от современной модели параллелизма, в которой используются потоки ОС. Сеть на основе потоков является относительно неэффективной и очень трудной в использовании. Кроме того, *Node.js* свободен от полной блокировки процесса, так как блокировок нет вообще. В *Node.js* почти нет функций непосредственно выполняющих операций ввода-вывода, поэтому процесс никогда не

блокируется. Поскольку ничто не блокирует, *Node.js* очень разумно использовать для масштабируемых систем.

Node.js разрабатывался под влиянием таких систем, как *Event Machine* на *Ruby* и *Twisted* на *Python*, поэтому аналогичен им по дизайну. Но, в отличие от них, в *Node.js* принцип модели события идет немного дальше. Он представляет цикл событий как конструкцию среды выполнения, а не как библиотеку. В других системах всегда существует блокирующий вызов для запуска цикла событий. Как правило, поведение определяется посредством обратных вызовов в начале сценария, а в конце сервер запускается посредством блокирующего вызова, такого как *Event Machine: run ()*. В *Node.js* нет такого вызова как *start-the-event-loop*. *Node.js* просто входит в цикл событий после выполнения сценария ввода. *Node.js* выходит из цикла событий, когда больше нет обратных вызовов для выполнения.

HTTP является частью *Node.js*, разработанной с учетом потоковой передачи и низкой задержки. Это делает *Node.js* хорошо подходящим для основания Web-библиотеки или инфраструктуры.

Node.js как без потоковая среда не означает, что Вы не можете использовать преимущества нескольких ядер. Дочерние процессы могут быть созданы с помощью *API child_process.fork ()* и разработаны так, чтобы к ним было легко общаться.



Рисунок 8.11 – Запуск установки *Node.js*

А теперь возьмемся непосредственно за установку *Node.js*. Для этого заходим на сайт <https://nodejs.org/en/download/> и выберем версию под свой тип операционной системы (как выяснить 32- или 64-бит-

ная ли у Вас операционная система смотрите в главах выше). После загрузки установочных файлов запускаем установку – рисунок 8.11.

После запуска установки требуется согласие на условия лицензирования Node.js – рисунок 8.12. Авторы рекомендуют все же тратить немного времени на то, чтобы хотя бы просматривать такие условия перед согласием.

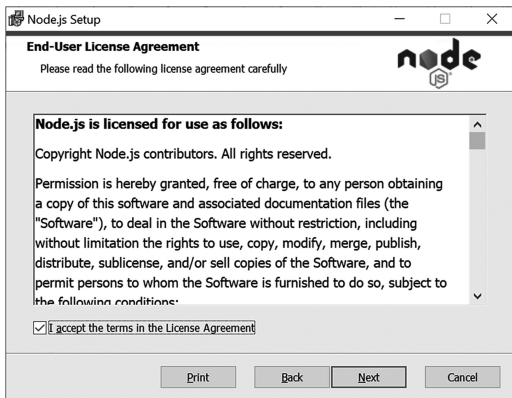


Рисунок 8.12 – Согласие с условиями лицензирования Node.js

Следующий шаг – рисунок 8.13 – выбор папки для установки. Здесь также стоит указывать директорию без букв кириллицы даже если Вы захотите установить среду не в Program Files, которая предлагается по умолчанию.

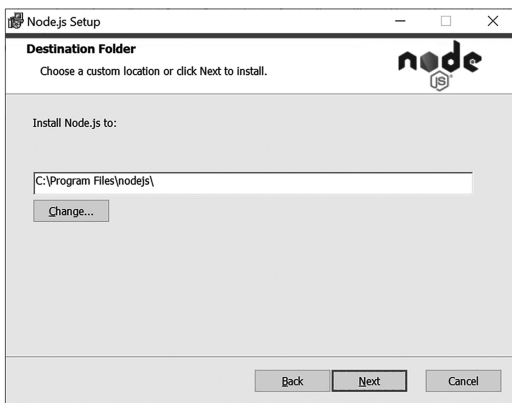


Рисунок 8.13 – Выбор места для установки Node.js

Дальше выбираются дополнительные задачи для установщика Node.js – рисунок 8.14. В том числе и прописывание среды в PATH.

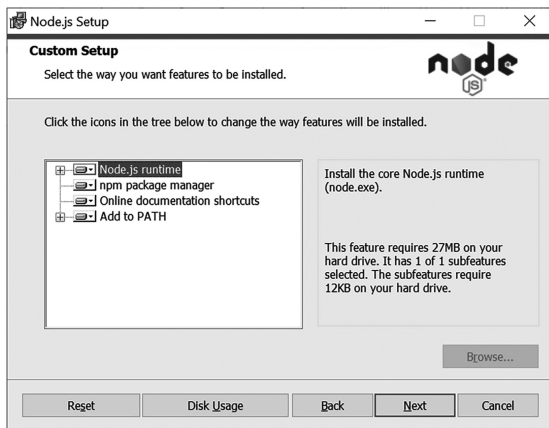


Рисунок 8.14 – Дополнительные задачи для установки Node.js

Предпоследний шаг установки предлагает автоматически установить необходимые инструменты для полноценной работы среды – рисунок 8.15. Стоит поставить там галочку, если Вы не собираетесь подробнее разбираться с необходимыми инструментами и устанавливать их частями самостоятельно.

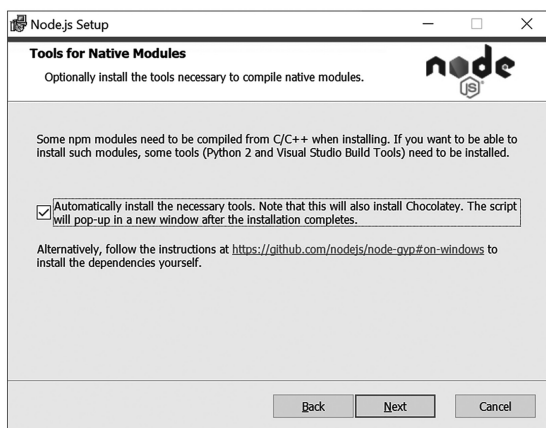


Рисунок 8.15 – Согласие на автоматическую загрузку необходимых инструментов для Node.js

Осталось только запустить установку – рисунок 8.16.

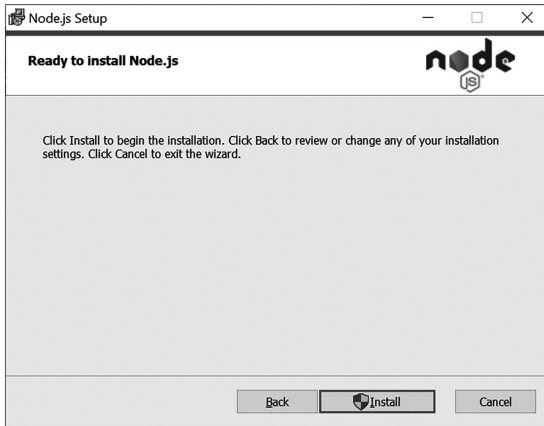


Рисунок 8.16 – Установка Node.js

Если Вы выбрали автоматическую установку необходимых инструментов, то появится следующее окно – рисунок 8.17.

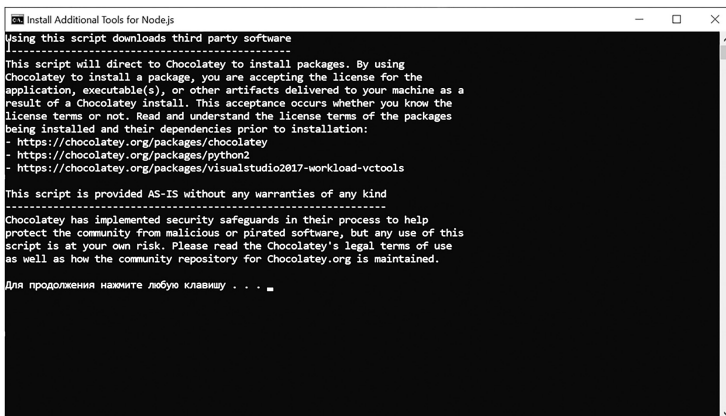


Рисунок 8.17 – Установка необходимых инструментов

Установка *Node.js* завершена – рисунок 8.18.

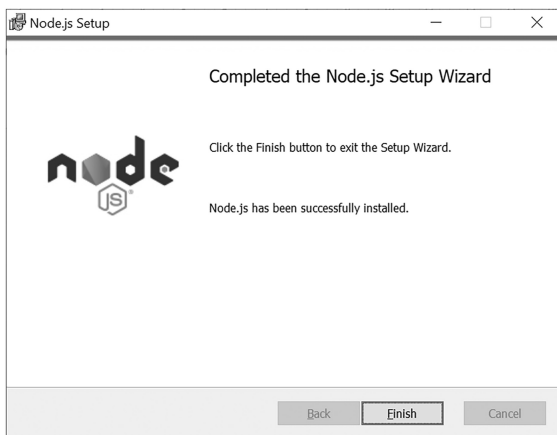


Рисунок 8.18 – Завершение установки Node.js

Теперь можно переходить к работе непосредственно с *Apache Cordova*.

8.4 Создание мобильного Web-приложения

Теперь рассмотрим, как создать приложение JS/HTML Cordova и развернуть его на различных собственных мобильных платформах с помощью интерфейса командной строки *Cordova*.

Для начала, интерфейс командной строки *Cordova* необходимо установить. Для этого загрузите и установите файл *Node.js* – предыдущая подглава. Так как при установке необходимо иметь возможность вызова *node* и *npm* в командной строке.

Установите модуль Cordova с помощью утилиты *Node.js* – рисунок 8.19. Модуль *Cordova* будет автоматически загружен утилитой npm. В Windows введите в командной строке:

```
C:>npm install -g cordova
```

Флаг *-g* указывает npm установить *Cordova* глобально. В противном случае, он будет установлен в *node_modules* подпапке текущей рабочей папки.

После установки можно запустить *Cordova* в командной строке без аргументов и распечатать текст справки.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.592]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.
C:\Users\Дина>npm install -g cordova
C:\Users\Дина\AppData\Roaming\npm\cordova -> C:\Users\Дина\AppData\Roaming\npm\node_modules\cordova\bin\cordova
+ cordova@9.0.0
added 469 packages from 364 contributors in 34.325s
C:\Users\Дина>
```

Рисунок 8.19 – Установка Cordova

Оффтоп. Запустить командную строку можно нажав сочетание клавиш **WIN+R**, введя **cmd** и нажав кнопку **OK**. Или нажав кнопку **Пуск**, введя в поле поиска **командная строка** и щелкнув в результатах поиска **Командная строка** (можно ввести **cmd** и выбрать в результатах **cmd**).

Теперь возьмемся непосредственно за создание приложения.

Для этого перейдите в каталог, в котором ведется исходный код (введите **cd** и адрес папки), и создайте проект **Cordova** – рисунок 8.20 – с помощью ввода:

cordova create SpaceShip com.example.SpaceSip SpaceShip

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.592]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.
C:\Users\Дина>cd C:\Users\Дина\для НТБ\учебники\Web\spaceship-master
C:\Users\Дина\для НТБ\учебники\Web\spaceship-master>cordova create SpaceShip com.example.SpaceSip SpaceShip
```

Рисунок 8.20 – Создание проекта Cordova

При этом создается необходимая структура каталогов для приложения **Cordova**. По умолчанию сценарий создания **Cordova** создает скелетное Web-приложение, домашней страницей которого является файл проекта **index.html** в папке **www**.

Все последующие команды должны выполняться в каталоге проекта или в любых вложенных папках:

cd SpaceShip

Добавим платформы iOS и Android и обеспечим их сохранение в файле **config.xml** и **package.json**:

```
cordova platform add ios  
cordova platform add android
```

Выполнение команд добавления или удаления платформ влияет на содержимое каталога платформ проекта, где каждая указанная платформа отображается в качестве подпапки.

При использовании интерфейса командной строки для создания приложения не следует изменять файлы в каталоге **/platforms/**. Файлы в этом каталоге обычно перезаписываются при подготовке приложений к построению или при повторной установке плагинов.

Для создания и запуска приложений необходимо установить **SDK** для каждой целевой платформы. Кроме того, при использовании браузера для разработки можно использовать платформу браузера, которая не требует каких-либо **SDK** платформ.

Для проверки соответствия требованиям к созданию платформы введите:

cordova requirements

В результате Вы получите информацию на подобии этой:

Requirements check results for android:

Java JDK: installed.

Android SDK: installed

Android target: installed android-19,android-21,android-22,android-23,Google Inc.:Google APIs:19,Google Inc.:Google APIs (x86 System Image):19,Google Inc.:Google APIs:23

Gradle: installed

Requirements check results for ios:

Apple OS X: not installed

Cordova tooling for iOS requires Apple OS X Error: Some of requirements check failed

В тексте выше видно, что требования по Android выполнены, а по iOS не все инструменты установлены.

Если для нужной Вам мобильной платформы не хватает, каких-то служб и инструментов, установите их.

По умолчанию **Cordova** создает скелетное Web-приложение, начальная страница которого является файлом **index.html** проекта в папке **www**.

Теперь для загрузки Web-приложения написанного ранее открываем папку с проектом созданную **Cordova** и находим там папку **www**. Копируем созданные ранее файлы с расширением **.html** в эту папку, **style.css** помещаем в подпапку **css**, а картинки помещаем в подпапку **img**.

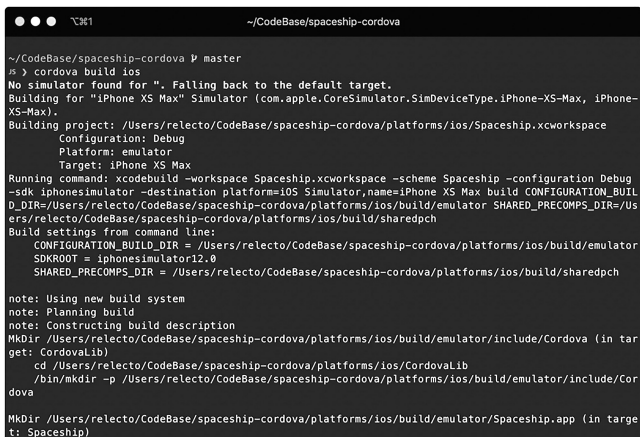
Теперь для создания проекта для всех платформ выполните следующую команду:

cordova build

При необходимости, можно ограничить область применения каждой сборки конкретными платформами. Например, iOS:

cordova build ios

В результате, в командной строке появиться текст, как на рисунке 8.21.



```
~/CodeBase/spaceship-cordova P master
# cordova build ios
No simulator found for ". Falling back to the default target.
Building for "iPhone XS Max" Simulator (com.apple.CoreSimulator.SimDeviceType.iPhone-XS-Max, iPhone-XS-Max).
Building project: /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/Spaceship.xcworkspace
Configuration: Debug
Platform: emulator
Target: iPhone XS Max
Running command: xcodebuild -workspace Spaceship.xcworkspace -scheme Spaceship -configuration Debug
-sdk iphonesimulator -destination platform=iOS Simulator,name=iPhone XS Max build CONFIGURATION_BUILD
D_DIR=/Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator SHARED_PRECOMPS_DIR=/Us
ers/relecto/CodeBase/spaceship-cordova/platforms/ios/build/sharedpch
Build settings from command line:
CONFIGURATION_BUILD_DIR = /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator
SDKROOT = iphonesimulator12.0
SHARED_PRECOMPS_DIR = /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/sharedpch

note: Using new build system
note: Planning build
note: Constructing build description
MKDIR /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator/include/Cordova (in tar
get: CordovaLib)
cd /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/CordovaLib
/bin/mkdir -p /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator/include/Cor
dova
MKDIR /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator/Spaceship.app (in targ
et: Spaceship)
```

Рисунок 8.21 – Результат выполнения cordova build ios

Теперь можно проверить работу приложения.

SDK для мобильных платформ часто поставляются в комплекте с эмуляторами, выполняющими образ устройства, чтобы можно было запустить приложение с главного экрана и посмотреть, как оно взаимодействует со многими функциями платформы. Выполните следующую команду, чтобы перестроить приложение и посмотреть его в эмуляторе конкретной платформы:

cordova emulate ios

После выполнения этой команды обновляет изображение эмулятора для отображения последнего приложения, которое теперь доступно для запуска с главного экрана – рисунок 8.22-8.25.

```
~/CodeBase/spaceship-cordova
$ cordova run --emulator --target "iPhone-7"
Building for "iPhone 7" Simulator (com.apple.CoreSimulator.SimDeviceType.iPhone-7, iPhone-7).
Building project: /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/Spaceship.xcworkspace
Configuration: Debug
Platform: emulator
Target: iPhone 7
Running command: xcodebuild -workspace Spaceship.xcworkspace -scheme Spaceship -configuration Debug
-sdk iphonesimulator -destination platform=iOS Simulator,name=iPhone 7 build CONFIGURATION_BUILD_DIR
=/Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator SHARED_PRECOMPS_DIR=/Users/r
electo/CodeBase/spaceship-cordova/platforms/ios/build/sharedpch
Build settings from command line:
CONFIGURATION_BUILD_DIR = /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator
SDKROOT = iphonesimulator12.0
SHARED_PRECOMPS_DIR = /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/sharedpch
note: Using new build system
note: Planning build
note: Constructing build description
mkdir /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator/include/Cordova (in tar
get: CordovaLib)
cd /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/CordovaLib
/bin/mkdir -p /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/build/emulator/include/Cor
dova
WriteAuxiliaryFile /Users/relecto/Library/Developer/Xcode/DerivedData/Spaceship-abehnrpvoellxairpxi
mtsxugh/Build/Intermediates.noindex/CordovaLib.build/Debug-iphonesimulator/CordovaLib.build/Cordova
-all-non-framework-target-headers.hmap (in target: CordovaLib)
cd /Users/relecto/CodeBase/spaceship-cordova/platforms/ios/CordovaLib
```

Рисунок 8.22 – Результат выполнения *cordova emulate ios* в командной строке



Рисунок 8.23 – Открытие рабочего приложения в эмуляторе

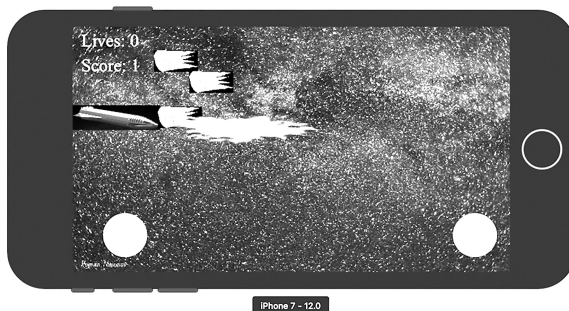


Рисунок 8.24 – Игра в эмуляторе



Рисунок 8.25 – Экран проигрыша в эмуляторе

Кроме того, можно подключить смартфон к компьютеру и непосредственно протестировать приложение:

cordova run ios

Перед выполнением этой команды необходимо настроить устройство для тестирования, следуя процедурам, которые различаются для каждой платформы.

Оффтоп. Авторы хотели бы поблагодарить за помощь в работе над этой главой и за рекомендации и комментарии ко всему пособию Дюсебаева Эмиля – DevOps-инженера казахстанской компании Kaspi bank и тренера клуба робототехники, кибернетики и программирования «Альтернатива».

ГЛОССАРИЙ

Аккаунт (учётная запись) – личный кабинет, собственное пространство человека в социальных сетях, онлайн-магазинах, электронной почте и других интернет-сервисах, а также в компьютерных и мобильных приложениях.

Активности (Activities) – видимая часть приложения (экран, окно, форма), отвечает за *отображение* графического интерфейса пользователя. При этом *приложение* может иметь несколько активностей, например, в приложении, предназначенном для работы с электронной почтой, одна *активность* может использоваться для отображения списка новых писем, другая *активность* – для написания, и еще одна – для чтения писем. Несмотря на то, что для пользователя *приложение* представляется единым целым, все активности приложения не зависят друг от друга. В связи с этим любая из этих активностей может быть запущена из другого приложения, имеющего *доступ* к активностям данного приложения. Например, *приложение* камеры может запустить *активность*, создающую новые письма, чтобы отправить только что сделанную фотографию адресату, указанному пользователем.

Антивирус – программа (приложение), которая защищает компьютер от вирусов и других атак интернет-преступников. Блокирует вирусы и восстанавливает («лечит») заражённые файлы.

Аватарка – фотография или другая картинка, которую человек устанавливает как главную в своем аккаунте в социальных сетях, блогах и других онлайн-сервисах.

Браузер – приложение, с помощью которого можно просмотреть сайты (Internet Explorer, Google Chrome, Safari, Mozilla Firefox).

Быстрое прототипирование (Rapid Prototyping) – технология быстрого «макетирования», быстрого создания опытных образцов или работающей модели системы для демонстрации заказчику или проверки возможности реализации. Прототип позже уточняется для создания макета дизайна и получения конечного продукта.

Блог – регулярно обновляющийся онлайн-дневник человека или компании. Располагается на специальных сайтах (LiveJournal, Tumblr) или в социальных сетях.

Вкладки браузера – функция, которая позволяет открывать новую веб-страницу, не покидая предыдущие. Кликая по разным вкладкам, человек в любом порядке (но не одновременно) просматривает все открытые им страницы.

Видеоблогер – человек, который ведёт блог с помощью регулярных видеозаписей. Такие блоги называют влогами: сокращённое от «видеоблог».

Галерея (приложение) – приложение для просмотра и хранения фотографий и видео. Здесь можно сортировать файлы по папкам и дате съёмки, а также делиться ими в мессенджерах, по электронной почте и в соцсетях. Стандартная галерея по умолчанию установлена в мобильных устройствах, но есть и дополнительные, например A+ Gallery, F-Stop Gallery и Focus.

Гаджет – электронное устройство относительно небольших размеров, которое упрощает повседневную жизнь человека или развлекает его. Смартфон, «умные» часы, планшеты, очки виртуальной реальности – всё это гаджеты.

Дизайн веб-сайта – уникальные для конкретного веб-сайта структура, графическое оформление и способы представления информации.

Закладки – функция браузера для сохранения ссылок на веб-страницы, чтобы потом открывать их в один клик. В закладки обычно добавляют либо страницы, которые посещают чаще остальных, либо те, о существовании которых не хотят забыть (непрочитанная статья, новый портал, необычный сайт и так далее).

Интерфейс пользователя (User Interface) – разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и тому подобное), представленные пользователю на дисплее, исполнены в виде графических изображений. В графическом интерфейсе исполнение действий чаще всего происходит через непосредственное манипулирование графическими элементами.

Иконка (ярлык) – пиктограмма, маленькая картинка на сайтах, а также в телефонах и компьютерах. Обозначает конкретное приложение, веб-страницу, файл или папку и открывает его. Skype, Word, Excel.

Интернет-магазин (онлайн-магазин) – сайт и/или приложение, через которые продают товары: одежду, продукты, технику, автомобили и т.д.

Концепция дизайна (эскиз, макет, Mockup) – графический файл, который состоит из наиболее мелких картинок-слоев элементов общего рисунка.

Клик – нажатие на ссылку или иконку кнопкой мыши (если речь о компьютере) или пальцем – на смартфоне или планшете.

Крэш-лог (Crash Log) – файл, в котором хранится вся информация по ошибке.

Content providers) – контент-провайдер управляет распределенным множеством данных приложения. Данные могут храниться в файловой системе, в базе данных SQLite, в сети, в любом другом доступном для приложения месте. Контент-провайдер позволяет другим приложениям при наличии у них соответствующих прав делать запросы или даже менять данные. Например, в системе *Android* есть контент-провайдер, который управляет информацией о контактах пользователя. В связи с этим, любое *приложение* с соответствующими правами может сделать *запрос* на чтение и *запись* информации какого-либо контакта. Контент-провайдер можно использовать для чтения и записи частных данных приложения, не предназначенных для доступа извне.

Логин – имя пользователя, псевдоним человека в сети. Используют вместе с паролем для входа в аккаунты соцсетей, электронной почты, форумов, онлайн-магазинов и других сервисов.

Лог-файл (журнал событий, Log) – это файл, содержащий системную информацию работы сервера или компьютера, в который вносятся определенные действия пользователя или программы.

Лайк – отметка «нравится» к постам, лайкнуть – отметить пост, как понравившийся. Лайки используют преимущественно в социальных сетях.

Мессенджер – мобильное и компьютерное приложение для онлайн-обмена сообщениями разного формата: текст, файлы (картинки, видео, документы), голосовые и видеозвонки. Популярные мессенджеры: WhatsApp, Viber, Skype.

Мобильное приложение (Mobile Application) – это специально разработанное приложение под конкретную мобильную платформу (iOS, Android, Windows Phone).

Мобильный интернет – беспроводной доступ к интернету с помощью технологий мобильной связи. Позволяет быть онлайн с телефона и планшета там, где есть сеть, то есть практически везде: на улице, в магазине, дома, в метро и в путешествии. Пользоваться мессенджерами, просматривать сайты, общаться в соцсетях, находить и слушать музыку или смотреть видео.

Мобильный web-сайт (Mobile Website) – специализированный сайт, адаптированный для просмотра и функционирования на мобильном устройстве.

Нативное мобильное приложение – приложение можно назвать «родным» для операционных систем – Android, IOS, WinPhone. Такие мобильные приложения пишутся на языках программирования, утвержденных разработчиками программного обеспечения под каждую конкретную платформу, а потому органично встраиваются в сами операционные системы. Приложения загружаются через магазины приложений (App Store, Google Play и т.д.) и соответствуют требованиям этих магазинов.

Сайт – набор веб-страниц, расположенных по одному основному адресу и связанных ссылками.

Сервисы (Services) – *компонент*, который работает в фоновом режиме, выполняет длительные по времени *операции* или работу для удаленных процессов. Сервис не предоставляет пользовательского интерфейса. Например, сервис может проигрывать музыку в фоновом режиме, пока *пользователь* использует другое *приложение*, может загружать данные из сети, не блокируя взаимодействие пользователя с активностью. Сервис может быть запущен другим компонентом и после этого работать самостоятельно, а может остаться связанным с этим компонентом и взаимодействовать с ним.

Смартфон (Smartphone) – в данную категорию включают мобильные аппараты, оснащенные собственной операционной системой. Кроме того, практически обязательными элементами оснащения для современного смартфона являются сенсорный экран, модули Wi-Fi, Bluetooth и GPS.

Сенсорный экран (Touchscreen) – устройство ввода информации, представляющее собой экран, реагирующий на прикосновения к нему.

Портал – сайт с набором разных сервисов. Например, yandex.ru - это и поисковик, и электронная почта, и прогноз погоды, и анализатор пробок, и агрегатор новостей.

Пароль – секретный набор букв, цифр и других символов; код доступа к закрытой информации. Благодаря паролю люди защищают данные в своих аккаунтах от всех остальных.

Персональные данные – любая информация, которая помогает определить личность человека: от номера телефона до ФИО. Адрес регистрации, серия и номер паспорта, дата рождения, место работы, уровень дохода, номер ИНН.

Приложение (мобильное) – компьютерная программа для телефонов, планшетов, «умных» часов и других мобильных

Приемники широковещательных сообщений (Broadcast Receivers) – компонент, который реагирует на широковещательные извещения. Большинство таких извещений порождаются системой, например, извещение о том, что экран отключился или низкий заряд батареи. Приложения также могут инициировать *широковещание*, например, разослать другим приложениям сообщение о том, что некоторые данные загружены и доступны для использования. Хотя приемники не отображают пользовательского интерфейса, они могут создавать уведомление на панели состояний, чтобы предупредить пользователя о появлении сообщения. Такой приемник служит проводником к другим компонентам и предназначен для выполнения небольшого объема *работ*, например, он может запустить соответствующий событию сервис.

Поисковик (адресно-поисковая строка) – сервис для поиска информации в интернете: сайтов, музыки, статей, отзывов, товаров, страниц людей в социальных сетях, видео, фото. Yandex, Google, Mail.ru – это поисковики.

Пост – любая публикация в блоге или социальной сети. Это может быть фотография, видео, текст или всё сразу.

Профиль (в социальной сети) – личная страница человека в социальной сети, где хранится информация о нём: имя, дата рождения, место жительства, фото и видео, посты и комментарии к ним, список «друзей» и контакты.

Собака @ – обязательный символ адреса электронной почты. Отделяет логин (имя пользователя) от адреса сайта или сервиса, на котором зарегистрирован ящик.

Ссылка (гиперссылка) – адрес веб-страницы в интернете. Чтобы сэкономить место, ссылку обычно встраивают в текст, изображение или видео. При нажатии на неё открывается определённый сайт, запускается конкретное приложение или скачивается файл. Чаще всего используют именно текстовые ссылки.

Социальная сеть – сервис (обычно сайт и приложение) для общения в интернете, имитирующий офлайн-жизнь: с системой «друзей», возможностью делиться моментами из жизни (текст, фото, видео), а также общаться публично и приватно. Как правило, люди регистрируются в соцсетях под реальными именами, чтобы находить знакомых и быть найденными.

Сториз (истории) – короткие видео в Instagram, которые автоматически удаляются через сутки. Обычно с помощью сториз люди

делятся друг с другом малозначительными событиями повседневной жизни: поездка на работу, ужин, шопинг и так далее.

Фишинг – вид интернет-мошенничества. Цель злоумышленников – получить секретную информацию человека: данные банковской карты, а также логины и пароли от электронной почты, аккаунтов в социальных сетях и онлайн-банка.

Хэштег – отметка из букв, цифр и других символов, которая помогает искать и группировать посты на одну тему. Хэштеги отображаются как гиперссылки и всегда начинаются с символа #. Например #iPhone8, #работа, #Алматы, #отдыхаем, #пятница. Хэштег всегда пишется без пробелов, поэтому может объединять сразу несколько слов: #жизньпрекрасна, #любовь_морковь.

Чат – мгновенный обмен сообщениями между двумя или несколькими людьми. Беседы в Viber, WhatsApp, Skype или в личных сообщениях в соцсетях – всё это чаты.

Электронный почтовый ящик (e-mail) – сервис обмена электронными сообщениями (письмами). Позволяет человеку отправлять и получать тексты, ссылки на сайты и веб-страницы, а также вложения: картинки, видео, документы и вообще любые файлы.

Эмулятор (Emulator) – программа, копирующая функционал и поведение другой программы.

Activity – платформа с открытым исходным кодом для управления бизнес-процессами.

ADT – установка и настройка среды программирования.

ADB (Android Debug Bridge, отладочный мост Андроид) – компонент Android SDK, который устанавливает связь между устройством и компьютером и позволяет прямо на компьютере выполнять различные манипуляции с системой Android.

Apps2Sm – метод сохранения приложений и кэш-данных на microSD карте устройства.

ADB – интерфейс для отладки Android (дословно: отладочный мост андроида) это многофункциональная утилита командной строки, которая позволяет общаться с экземпляром эмулятора или подключенным устройством с ОС Android. Это клиент-серверная утилита, которая включает три компонента:

- Клиент, который запущен на компьютере разработчика. Запустить клиента из консоли можно использовать команду adb. Дру-

гие утилиты Android, такие как ADT-plugin, и DDMS тоже создают adb клиентов.

- Сервер, который запущен как процесс фонового исполнения на компьютере разработчика. Сервер обслуживает коммуникации между клиентом и adb-демоном, запущенном на эмуляторе или устройстве.
- Домен, который запущен как процесс фонового исполнения на каждом экземпляре эмулятора или устройства.

Android – основанная на Linux операционная система для мобильных устройств, таких например, как HTC EVO.

AMOLED – активная матрица на органических светодиодах. Обычно это очень яркий дисплей с хорошей цветопередачей, который есть в некоторых смартфонах.

APK – файл пакета приложения для Android. Каждое приложение Android компилируется и упаковывается в один файл, который включает в себя весь байткод приложения (.dex файлы), ресурсы, вложения, и файл манифеста. Пакет приложения может иметь любое имя, но должен использовать расширение .apk. Например: myExampleAppname.apk. Для удобства, файл пакета приложения часто обозначают как «.apk».

API – (интерфейс прикладного программирования) представляет собой интерфейс или протокол связи между различными частями программы, предназначенной для упрощения внедрения и эксплуатации программного обеспечения.

App – приложение для реализации тестирования и отладки на мобильном устройстве.

AVD – виртуальное, эмулирование работы реального смартфона.

Alpha – альфа стадия жизненного цикла релиза, это первая фаза начала тестирования (альфа – первая буква греческого алфавита, используемая как номер).

Boot Animation – термин (загрузочная анимация), обозначающий графическое отображение процесса загрузки операционной системы. Загрузочная анимация, может быть, простой визуализацией бегущих загрузочных сообщений в консоли, но также она может представлять из себя графику либо комбинацию текста и графики.

Bootloop – описывает состояние, когда система перезагружается раз за разом без входа в основную операционную систему.

Beta – стадия разработки ПО. следующая за альфа.

Bundle – программное приложение для запуска простейшего приложения на эмуляторе и мобильном устройстве.

CDMA – стандарт мобильной связи, называемый cdmaOne, CDMA2000 (3G эволюционирование cdmaOne) и WCDMA (3G стандарт использующий GSM связь), на который часто ссылаются как просто на CDMA, и использующий CDMA как нижестоящий канал доступа к данным.

CIQ – расшифровывается как Carrier IQ. Часть предустановленного программного обеспечения, которое запущено с повышенными правами в фоновом режиме портативного устройства по умолчанию и записывающего все. Потенциально может использоваться для кражи информации.

Content – список содержимого.

Dalvik – виртуальная машина платформы Android. Dalvik VM – это только интерпретирующая виртуальная машина, которая исполняет файлы формата Dalvik Executable (.dex); формата, оптимизированного для эффективного хранения и распределения памяти.

Dalvik Cache – записываемый кэш, который содержит оптимизированный байткод для всех арк-файлов (приложений) на устройстве Android. Хранение информации в собственном кэше делает загрузку приложений быстрее и исполнение лучше.

Default – ресурсы, которые будут использоваться независимо от конфигурации устройства или в том случае, когда под конфигурацию нет подходящих альтернативных ресурсов.

Device – устройство, компонент ПК, аппаратного обеспечения.

Download – загрузка дистрибутива из Интернета.

Example – имя приложения.

Fastboot – диагностический протокол, использующийся в основном для модифицирования flash-файловой системы в Android-смартфонах, с другого компьютера через USB соединение. Это часть Android Debug Bridge библиотеки. После активации протокола на устройстве, оно будет принимать любую команду, пришедшую к нему через USB из консоли. Некоторые из наиболее используемых команд включают в себя:

- *Flash* – перезаписывает раздел в flash памяти бинарным образом, который находится на компьютере-хосте.
- *Erase* – стирает раздел в flash.
- *Reboot* – перезагружает устройство в основную операционную систему или в раздел восстановления системы.

– *Devices* – отображает список всех устройств (с серийным номером), присоединенных к компьютеру.

Flashing – энергонезависимая память, используемая в смартфонах, планшетах и т.д. Часто то же, что и флэш-память в SD и USB flash дисках, просто оптимизированная для лучшей скорости и производительности во время работы операционной системы.

Google play – онлайн-магазин, с помощью которого пользователи телефонов на операционной системе Android находят и устанавливают приложения, скачивают книги, музыку и фильмы. Аналог Google play для телефонов на базе iOS (Apple) – App Store.

Gesture – приложение для процесса создания набора жестов в Android.

Global positioning system – система глобального позиционирования (GPS).

GPU – графический процессор, предназначен для быстрого манипулирования и при создании изображения в буфере кадра.

Hotspot – точка доступа, которая предоставляет доступ в Internet через беспроводную сеть с использованием роутера, присоединенного к сети провайдера Internet. Hotspot обычно использует Wi-Fi технологию.

HDMI – компактный аудио/видео для передачи зашифрованных несжатых цифровых данных. Это цифровая альтернатива потребительским аналоговым стандартам.

Hboot – отвечает за проверку и инициализацию аппаратного обеспечения и телефонного ПО. Он также может использоваться для записи официальных релизов ПО, так же, как и еще некоторых вещей. HBoot может быть сравним с BIOS в компьютере.

HAVS – система контроля, которая динамически корректирует вольтаж основываясь на загрузке CPU, что уменьшает расход батареи.

iTunes – программа для синхронизации устройств на базе iOS.

iTools – аналог iTunes, программа для просмотра и управления информацией на устройстве, снятия логов, установки билдов и снятия видео/скриншотов на базе iOS.

JIT – The Just-in-Time Compiler. Появился с версии Android 2.2, это метод значительного ускорения приложений в Android на стороне программного обеспечения.

Java – язык программирования.

JDK – сред разработки в Eclipse.

Kernel – слой программного обеспечения, который позволяет операционной системе и приложениям взаимодействовать с аппаратным обеспечением мобильного телефона.

Launcher – часть интерфейса пользователя Android, которая позволяет запускать приложения, делать телефонные звонки и т.д. Она встроена в Android, или может быть получена на Android Market.

LCD Density – измерение разрешения экрана в разных контекстах, обычно дисплеев компьютера, сканеров, цифровых камер.

Nandroid – используется для резервного копирования и восстановления.

OpenGL ES – предоставляет библиотеки OpenGL ES, которые могут использоваться для быстрых и сложных 3D изображений.

Partition – внутренняя память телефона (не SD карта), это постоянная(флеш) память, NAND. Она может быть разделена на разделы как обычный жесткий диск.

Preview – предварительный просмотр.

Provider – провайдер услуг.

PRL – Preferred Roaming List (предпочитаемый список роуминга), обычно, способ сообщить телефону, к каким вышкам подсоединиться в первую очередь.

Recovery – загрузочный режим для телефона, который позволяет удалить настройки с раздела данных на телефоне (жесткое удаление), или сделать обновление используя update.zip файл в корне microSD карты. Rom/Firmware – относится к внутреннему хранилищу устройства, которое предназначено для хранения инструкций операционной системы, которые не должны быть модифицированы во время нормальной работы устройства.

Rooting – процесс предоставления пользователям мобильных телефонов, планшетов, и других устройств запущенных в ОС Android привилегий управления.

Sideloadng – установка приложений минуя официальный Android Market.

Splash Screen – изображение, которое появляется пока загружается android. Splash screens покрывает весь экран или просто прямоугольник в центре экрана.

Superuser/SU – специальная учетная запись пользователя, используемая системным администратором. В зависимости от операционной системы имя этого аккаунта может быть: root, administrator или

supervisor. В организациях, административные привилегии часто зарезервированы для отдельно определенных персон.

Script – слой скриптов для Android (The Scripting Layer for Android) (аббревиатура SL4A, и ранее называлась Android Scripting Environment or ASE) это библиотека, которая позволяет создавать и запускать скрипты, созданные на большом количестве различных языков непосредственно на устройствах Android. Эти скрипты имеют доступ к многим API доступным Java приложениям Android, но с упрощенным интерфейсом. Скрипты могут интерактивно запускаться в терминале, в фоновом режиме, или через Locale.

SDK – это набор разработчика программного обеспечения, который позволяет создавать приложения для определенно пакета, фреймворка, игровой консоли, операционной системы или платформы.

SOD – состояние, когда устройство «засыпает» и не просыпается.

S-On – режим включенной безопасности (Security-on), обозначает отсутствие доступа к операционной системе телефона.

S-Off – нарушение безопасности доступа к операционной системе.

Tethering – распространение Internet соединения, имеющим доступ в Интернет мобильного телефона с другими устройствами. Это распространение может быть предоставлено через беспроводную сеть (Wi-Fi), Bluetooth, или физическое соединение используя кабель. В случае тетеринга через беспроводную сеть, есть возможность пометить устройство как мобильный hotspot. Телефон, имеющий соединение в интернет, работает как портативный роутер, предоставляющий услуги тетеринга для других.

Twitter – американский микроблог социальных сетей, сервис, на котором пользователь может публиковать и взаимодействовать с другими пользователями.

UDID (Unique Device Identifier) – это уникальный идентификатор устройства, состоящий из 40 символов (для устройств: iPad, iPhone или iPod Touch).

Userspace (Governor) – планировщик, который позволяет любой программе, запущенной пользователем, выставить операционную частоту CPU.

Underclock – уменьшение быстродействия CPU.

Undervolt – означает некоторое уменьшение напряжения питания CPU, которое позволяет продлить работу батареи и снизить температуру во время интенсивного использования CPU.

USB – предназначено для обозначения Universal Serial Bus. Это метод соединения устройств к компьютеру. Большинство смартфонов сейчас используют микро-usb для зарядки и синхронизации.

Updater Script – когда устройство Android устанавливает обновления через ‘update.zip’ файлы, используемые в режиме обновления должны осуществить широкий набор действий с файлами и правами. Вместо использования командной строки, такой как {b,d,c} sh, разработчики Android решили создать маленький функциональный язык, который может быть расширен производителями устройств при необходимости. С релиза Android «Donut»(v1.6) скриптовый язык называется Edify, и определен обычно в bootable/recovery/{edify,edifyscripting,updater} папках дерева исходного кода Android.

User eXperience (UX) – это восприятие и ответные действия пользователя, возникающие в результате использования и/или предстоящего использования продукции, системы или услуги.

Wireless N – технология увеличивает скорость беспроводного интернет соединения. Роутеры Wireless ‘N’ также работают с Wireless ‘G’ и ‘B’ беспроводными адаптерами.

WiiMax – коммуникационная технология, для высокоскоростного беспроводного распространения интернет сигнала на широких территориях.

Widget – универсальный тип программного приложения, содержащий переносимый код, предназначенный для одной или нескольких различных программных платформ.

YAFFS – версия файловой системы, работающей на NAND чипах. Эти старые чипы также имели 2 или 3 цикла записи на страницу, с которыми YAFFS имела дело – так называемые грязные страницы помечались записыванием в специальные зарезервированные ячейки.

Zipalign – утилита архивирования представленная в первый раз с версией 1.6 Android SDK. Она оптимизирует пути при создании APK. Это действие позволяет операционной системе Android взаимодействовать с приложением более эффективно, и, следовательно, потенциально делает работу приложений и системы в целом значительно быстрее. Время исполнения для приложений, которые обработаны zipalign быстрее, что дает меньший объем потребления оперативной памяти во время запуска приложения.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

- 1) История появления мобильных устройств и их архитектура.
- 2) Операционные системы для мобильных устройств (обзор).
- 3) Возможности современных ОС для мобильных устройств.
- 4) Классификация мобильных приложений.
- 5) Мобильные приложения для бизнеса.
- 6) Мобильные приложения для игр.
- 7) Мобильные социальные сети.
- 8) Мобильные облачные решения.
- 9) Классификация мобильных устройств.
- 10) Коммуникационные технологии.
- 11) Стандарты GSM.
- 12) Технология Wi-Fi. Стандарты.
- 13) Технология Bluetooth.
- 14) Проблемы безопасности мобильной связи.
- 15) Основные разновидности мобильных устройств.
- 16) Основные особенности ОС для мобильных устройств.
- 17) Основные форматы файлов, необходимые для обеспечения совместимости в ОС для мобильных устройств.
- 18) Коммуникационные технологии поддержки в ОС для мобильных устройств.
- 19) Широко используемые ОС для мобильных устройств.
- 20) Windows Mobile.
- 21) Symbian OS.
- 22) BlackBerry OS.
- 23) Перспективы и направления дальнейшего развития ОС для мобильных устройств.
- 24) Реализация мобильных приложений на платформе Java 2 Micro Edition.
- 25) Конструкторы мобильных приложений.
- 26) Особенности разработки интерфейсов для смартфонов. Принципы юзабилити.
- 27) Разработка мобильных приложений в среде Processing.
- 28) Мобильные устройства на примере устройств для ОС iOS, особенности.
- 29) Мобильные устройства на примере устройств для ОС Android, особенности.

- 30) Мобильные устройства на примере устройств для ОС WindowsMobile, особенности.
- 31) Java для мобильных устройств, архитектура и возможности.
- 32) Недостатки и преимущества Java при программировании для мобильных устройств.
- 33) Клиент-серверное взаимодействие мобильных приложений.
- 34) Виртуальная машина Java в Android, ее особенности.
- 35) Создание приложений под ОС Android: способы разработки приложений.
- 36) Android SDK и Android NDK. Назначение и особенности.
- 37) Принципы работы с ОС Android: Activity и Intents. Определения, пример.
- 38) Принципы работы с ОС Android: Views, Services. Назначение, пример.
- 39) Принципы работы с ОС Android: ContentProvider, BroadcastReceiver. Назначение.
- 40) Инструментарий элементов управления Android.
- 41) Модель обработки событий ОС Android. Пример обработчиков событий.
- 42) Пример доступа к оборудованию в ОС Android: акселерометры и гироскопы.
- 43) Анимация и жесты в ОС Android.
- 44) Концепция закрытой экосистемы Apple.
- 45) Способы распространения приложений iOS. Особенности разных версий iOS. Концепции пользовательского интерфейса iOS.
- 46) Классификация защищенности ОС по международным стандартам.
- 47) Мобильное программирование, платформы для разработки.
- 48) Архитектура приложений для Android. Ресурсы приложения. Пользовательский интерфейс. Инструментарий разработки приложений для Android
- 49) Основные средства разработки под Android.
- 50) Этапы и тенденции развития программирования, способы применения ИТ при разработке мобильных приложений.
- 51) Сервисные программы и оболочки при разработке мобильных приложений.
- 52) Публикация Android-приложения на Google Play.
- 53) Обоснование создания мобильного приложения.

- 54) Разработка структуры мобильного приложения: техническое задание
- 55) Реализация мобильного приложения.
- 56) Мобильное приложение, как новый канал коммуникации с целевой аудиторией.
- 57) Шаблон приложения.
- 58) Визуальное редактирование пользовательского интерфейса.
- 59) Дизайн или проектирование интерфейса для графических дизайнеров
- 60) Набор средств программирования, который содержит инструменты, необходимые для создания, компиляции и сборки мобильного приложения.
- 61) Язык разметки для описания иерархии компонентов графического пользовательского интерфейса Android-приложения.
- 62) Бизнес-анализ целевого рынка мобильных приложений.
- 63) Прототипирование мобильных приложений.
- 64) Написание кода и внедрение технологий мобильных приложений.
- 65) Тестирование мобильных приложений.
- 66) Добавление приложения в магазин.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://studref.com/384493/informatika/klassifikatsiya_mobilnyh_prilozheniy
2. <https://lifehacker.ru/business-apps/> 36б
3. <https://www.mediaplans.net/ru/uslugi/dizajn/dizajn-mobilnyh-prilozhenij/>
4. <https://vc.ru/flood/3607-apps-9> 6
5. <https://www.ibm.com/ru-ru/cloud/learn/what-is-mobile-cloud-computing>
<http://1234g.ru/1g/chto-takoe-pokolenie-setej-sotovoj-svyazi>
6. <http://bourabai.kz/einf/bluetooth.htm>
7. <https://www.web-canape.ru/business/vsya-statistika-interneta-na-2019-god-v-mire-i-v-rossii/>
8. <http://bourabai.kz/os/mobile.htm>
9. <https://appsstudio.ru/etapy-razrabotki-mobilnogo-prilozheniya>
10. http://window.edu.ru/resource/077/79077/files/MAD_coursebook_ru.pdf
11. <http://app-global.ru/blog/konstruktor-mobilnyih-prilozheniy-v-chem-plyusyi-dlya-razrabotchikov/>
12. <https://lifehacker.ru/business-apps/>
13. <https://polygant.net/ru/mobiledev/razrabotka-mobilnykh-igr/>
14. <https://vc.ru/flood/3607-apps-9>
15. <https://www.osp.ru/pcworld/2011/10/13010968/>
16. <https://www.ibm.com/developerworks/ru/library/cl-mobilecloudcomputing/index.html>
17. <https://www.crn.ru/numbers/spec-numbers/detail.php?ID=79648>
18. Кронин Д., Купер А., Рейман Р. Алан Купер об интерфейсе. Основы проектирования взаимодействия Пер. с англ. – СПб.: Символ-Плюс, 2009. – 688 с
19. <https://www.intuit.ru/studies/courses/12786/1219/lecture/22484?page=3>
20. <https://processing.org/>
21. <https://developer.android.com/studio>
22. <https://help.apple.com/xcode/mac/11.4/>
23. <https://code.visualstudio.com/>
24. <https://cordova.apache.org/>
25. <https://nodejs.org/en/>

**Г.Б. Нурпеисова, Т.Б. Нурпеисова,
И.Н. Кайдаш, Д.В. Панюкова**

РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

ISBN 978-601-7991-42-5

Компьютерная верстка – **Любовицкая Ольга**
Дизайн обложки – **Любовицкая Елизавета**

Подписано в печать в 2020 г.
Формат 60x84 1/16. Объем 20,25 печ.л.
Гарнитура Times New Roman. Печать офсетная.
Заказ № _____. Тираж 500 экз.

Издательство «Бастау».
Гос. лицензия № 0000036
Министерства образования и науки РК.
Сертификат Национальной государственной
книжной палаты РК №155 о присвоении
международного регистрационного кода 978-601-281.
Национальный сертификат «Лидер отрасли-2018»
Национального бизнес-рейтинга Республики Казахстан.
г. Алматы, пр. Сейфуллина, 458/460-95.
Тел.: 279 49 53, 279 97 32.

Отпечатано в типографии
«Полиграфсервис» (тел.: 233 32 53).
г. Алматы, ул. Зеленая, 13а.