

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Пензенский государственный университет»
Факультет вычислительной техники
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Программирование»
на тему: «Разработка программы сложной структуры методом нисходящего
программирования. Игра «Тамагочи»

Выполнил: ст. гр. 25BBB1
Купцова Н.М.
Проверил: к.т.н. доцент
Токарев А.Н.

0571

17.06.26

Пенза, 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

М.А. Митрохин

«___» _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Купцова Надежда Михайловна Группа 25ВВВ1

Тема проекта: *Разработка программы сложной структуры методом нисходящего программирования.*

Исходные данные (технические требования) на проектирование

*Игра "Танатоки". Ирок создает персонажа и следит за его здоровьем и развитием.
В файле сохраняется текущее состояние персонажа.*

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

Схема данных, схема ресурсов системы, схема работы системы,
иерархическая структура программы, схема взаимодействия программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания " 18 " 02 2026 г.

Дата защиты проекта " ____ " ____

Руководитель _____

Генерал А.Н.

Задание получил _____

" 18 " 02 _____

2026 г.

Студент _____

Кушова Н.М.

К

Содержание

Введение	5
1 Постановка задачи	6
2 Выбор Решения	8
3 Описание разработки программы	10
4 Отладка и тестирование	12
5 Описание программы	14
5.1 Схема функции main	14
5.2 Описание модулей	14
6 Руководство пользователя	21
6.1 Заставка	21
6.2 Список питомцев	22
6.3 Меню питомца	24
6.4 Игра Кликер	26
6.5 Игра Карточки	27
6.6 Магазин	38
Заключение	30
Список используемых материалов	32
Приложение А	33
Листинг программы	33

Введение

Концепция виртуальных питомцев, зародившаяся в конце прошлого века, прочно вошла в индустрию мобильных развлечений и продолжает развиваться с учетом современных технологий. Игра «Тамагочи» представляет собой не только цифровой аналог ухода за живым существом, но и эффективный инструмент для развития дисциплины, планирования и эмоционального интеллекта пользователя. Изучение принципов построения подобных симуляторов позволяет углубить знания в области проектирования мобильных интерфейсов, управления состоянием объектов в реальном времени и архитектуры клиентских приложений.

Актуальность работы заключается в том, что реализация мобильного «Тамагочи» идеально подходит для комплексного изучения цикла разработки под платформу Android. Проект охватывает работу с жизненным циклом Activity, обработку пользовательских событий, проектирование бизнес-логики взаимодействия с виртуальным объектом, а также организацию персистентного хранения данных. Разработка приложения формирует системное мышление — от проектирования архитектуры до оптимизации производительности и обеспечения надежности хранения прогресса.

В данной работе используется язык программирования Java, поскольку он является стандартом для Android-разработки, обеспечивает строгую типизацию, надежность и глубокую интеграцию с системными API. Для повышения скорости отклика в критически важных вычислительных блоках и более точного управления ресурсами устройства применяются ассемблерные вставки. Сохранение состояния питомца, истории действий и пользовательских настроек реализовано на основе локальной базы данных, что гарантирует автономность работы приложения и целостность информации. Проект разрабатывается в среде Android Studio, что позволяет эффективно управлять зависимостями, отлаживать код и тестировать интерфейс на различных конфигурациях устройств.

1 Постановка задачи

Целью данной работы является разработка мобильного приложения-симулятора виртуального питомца «Тамагочи» с использованием языка программирования Java и ассемблерных вставок. В ходе выполнения работы необходимо реализовать следующий функционал:

- инициализация виртуального питомца с набором базовых характеристик (голод, настроение, энергия, здоровье, гигиена);
- реализация интерактивных действий по уходу: кормление, игры, лечение, уборка и перевод в режим сна;
- непрерывная симуляция жизненных процессов с автоматическим динамическим изменением показателей питомца в реальном времени;
- визуальная и текстовая обратная связь: изменение анимации/спрайта питомца и элементов интерфейса в зависимости от текущего состояния;
- система прогрессии и развития питомца (рост, смена возрастных этапов, разблокировка новых возможностей при хорошем уходе);
- ведение журнала событий и отображение статистики взаимодействия пользователя с приложением.

Во время реализации проекта должны быть использованы объектно-ориентированные структуры данных (классы, коллекции), алгоритмы обработки состояний и симуляции временных процессов, а также принципы модульного программирования и архитектурные паттерны, принятые в Android-разработке. Оптимизация вычислительно нагруженных участков (например, расчёт динамики характеристик или обработка кадров анимации) реализуется через ассемблерные вставки.

Сохранение состояния питомца, прогресса, пользовательских настроек и истории действий.

Режим работы интерфейса – графический (GUI), адаптированный под мобильные устройства. Для начала необходимо определить структуру экранов и навигацию, затем изучить способы реализации адаптивной верстки и управления жизненным циклом Activity в Android Studio.

Устройство ввода информации – сенсорный экран мобильного устройства.

Пользовательский интерфейс должен быть построен на основе главного экрана с отображением питомца, навигационного меню и панели инструментов с кнопками действий.

2 Выбор решения

В процессе разработки данного приложения была выбрана среда разработки Android Studio. Она представляет собой официальную интегрированную среду разработки для платформы Android, предоставляющую все необходимые инструменты для комфортного и эффективного создания мобильных приложений: интеллектуальный редактор кода, мощный эмулятор устройств, инструменты профилирования производительности, отладки и тестирования.

Java выбрана в качестве основного языка программирования благодаря своей платформонезависимости, объектно-ориентированной природе и широкой поддержке в экосистеме Android. Главным преимуществом является наличие обширной стандартной библиотеки, упрощающей работу с пользовательским интерфейсом, многопоточностью и сетевыми взаимодействиями.

Ассемблерные вставки используются для оптимизации критически важных участков кода, требующих высокой производительности: расчёта динамики изменения характеристик питомца, обработки анимаций и работы с таймерами. Это позволяет снизить нагрузку на процессор мобильного устройства и продлить время автономной работы.

Проект структурирован с использованием модульного подхода и архитектурного паттерна MVVM (Model-View-ViewModel), что позволяет логически разделить код на функциональные блоки: слой представления (Activity, Fragment), слой бизнес-логики (ViewModel) и слой данных (Repository, Database). Такая архитектура упрощает поддержку, тестирование и дальнейшее развитие приложения.

Схема данных приведена на рисунке 1.

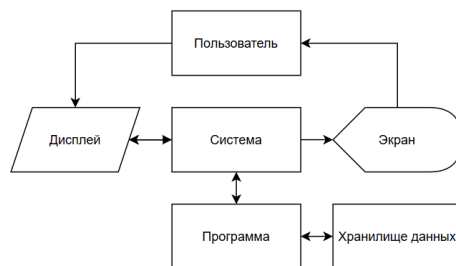


Рисунок 1 - Схема данных

Схема ресурсов системы приведена на рисунке 2.

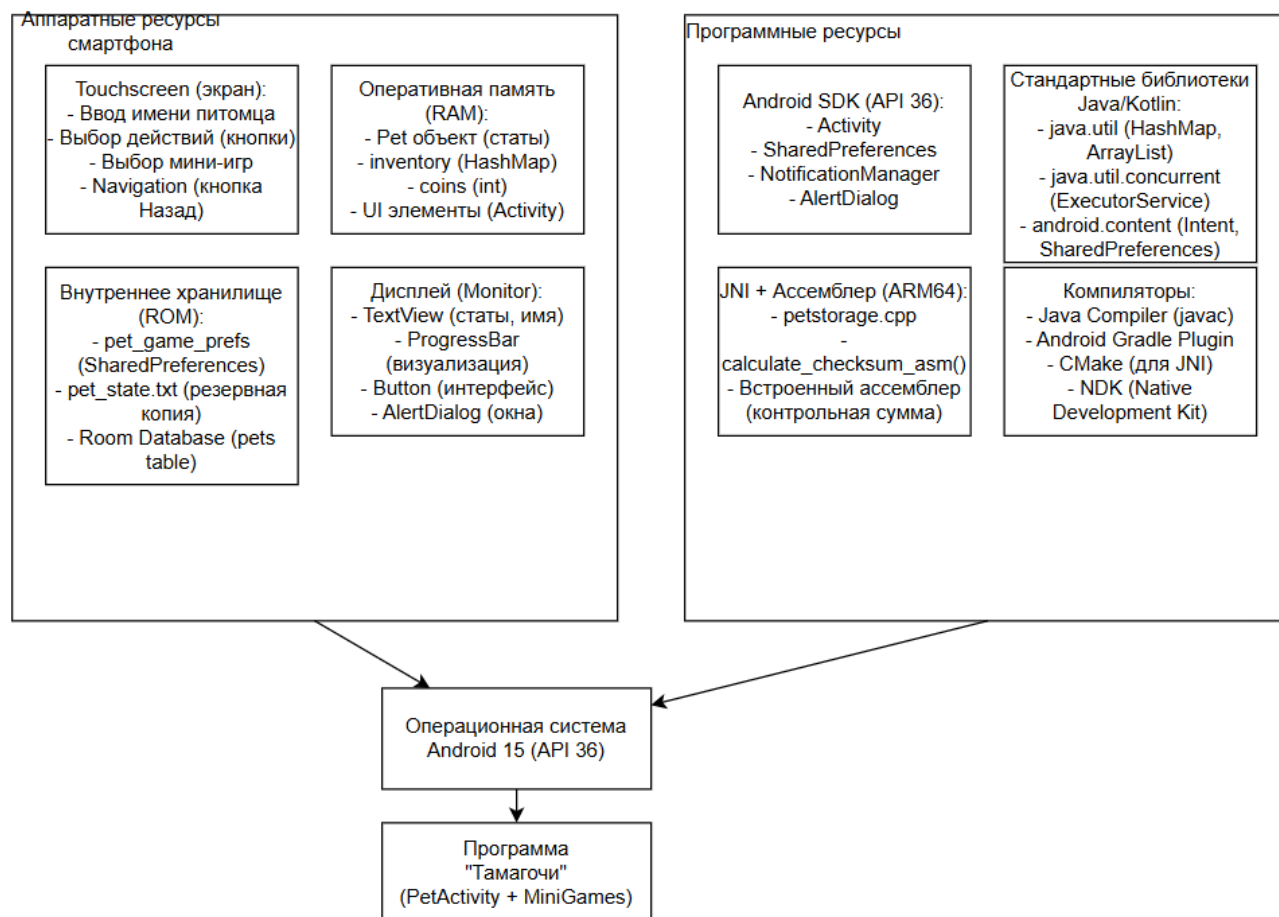


Рисунок 2 - Схема ресурсов

Результаты игры и состояние питомца сохраняются в локальную базу данных SQLite с использованием Room Persistence Library – официального компонента Android Jetpack, предоставляющего абстракцию над SQLite. При завершении сеанса или изменении состояния питомца в базу данных записываются: текущие показатели характеристик (голод, настроение, энергия, здоровье), возраст и этап развития питомца, временные метки последних действий, инвентарь и разблокированные достижения.

Использование Room обеспечивает типобезопасность работы с базой данных, поддержку миграций при изменении схемы и возможность выполнения асинхронных запросов без блокировки основного потока приложения.

3 Описание разработки программы

Для реализации был выбран объектно-ориентированный метод разработки с применением нисходящего проектирования, что позволяет сначала определить общую архитектуру приложения и основные сущности, а затем постепенно детализировать компоненты и их взаимодействие.

При запуске программы на экран выводится анимированная заставка (Splash Screen) с логотипом приложения, после чего происходит загрузка данных из локальной базы данных и инициализация виртуального питомца. Пользователю предлагается три режима сложности:

- легкий: питомец медленно теряет показатели, прощает ошибки в уходе, подходит для новичков;
- средний: сбалансированная скорость изменения характеристик, требует регулярного внимания;
- сложный: быстрое ухудшение состояния питомца, критичен к пропуску процедур ухода, для опытных игроков.

Программа разделена на 4 основных модуля, что является оптимальным вариантом, позволяющим изолировать слои ответственности и упростить поддержку кода:

- MainActivity.java;
- PetModel.java;
- PetDatabaseHelper.java;
- NativeCalculator.java.

Модуль MainActivity.java – главный файл программы, который инициализирует пользовательский интерфейс, обрабатывает навигацию между экранами, управляет жизненным циклом Activity и связывает остальные компоненты приложения.

Модуль PetModel.java отвечает за бизнес-логику: хранение и изменение характеристик питомца (голод, настроение, энергия, здоровье), реализацию игровых действий (кормление, игра, сон, лечение), проверку условий победы/поражения и систему прогрессии (рост, эволюция).

Модуль `PetDatabaseHelper.java` нужен для работы с локальной базой данных SQLite: сохранения текущего состояния питомца, загрузки прогресса, ведения журнала событий, хранения настроек пользователя и истории достижений. Использует паттерн `Repository` для абстракции работы с данными.

Модуль `NativeCalculator.java` содержит ассемблерные вставки (через JNI – Java Native Interface) для оптимизации вычислений: расчёта скорости изменения характеристик в зависимости от множества факторов, обработки временных интервалов и генерации случайных событий. Это позволяет снизить нагрузку на основной поток приложения и обеспечить плавную работу анимаций.

Дополнительно реализованы вспомогательные модули:

- `PetViewAdapter.java`, это адаптер для отображения визуального состояния питомца (спрайты, анимации);

- `NotificationManager.java`, это управление push-уведомлениями о необходимости ухода.

4 Отладка и тестирование

В качестве среды разработки была выбрана программа Android Studio. Она оптимальна для разработки на языке программирования Java благодаря мощному функционалу, интеграции с Android SDK и удобству работы с мобильными приложениями.

Пошаговое выполнение в режиме отладки дает возможность отслеживать, как приложение инициализирует виртуального питомца, загружает данные из локальной базы данных, обрабатывает пользовательские действия (кормление, игры, лечение) и обновляет показатели характеристик в реальном времени.

Точки останова позволяют исследовать состояние программы, значения переменных (уровень голода, настроения, энергии, здоровья), корректность работы ассемблерных вставок при расчете динамики изменений, а также определить, как ведет себя приложение при различных сценариях взаимодействия.

Для отладки использовались следующие инструменты Android Studio:

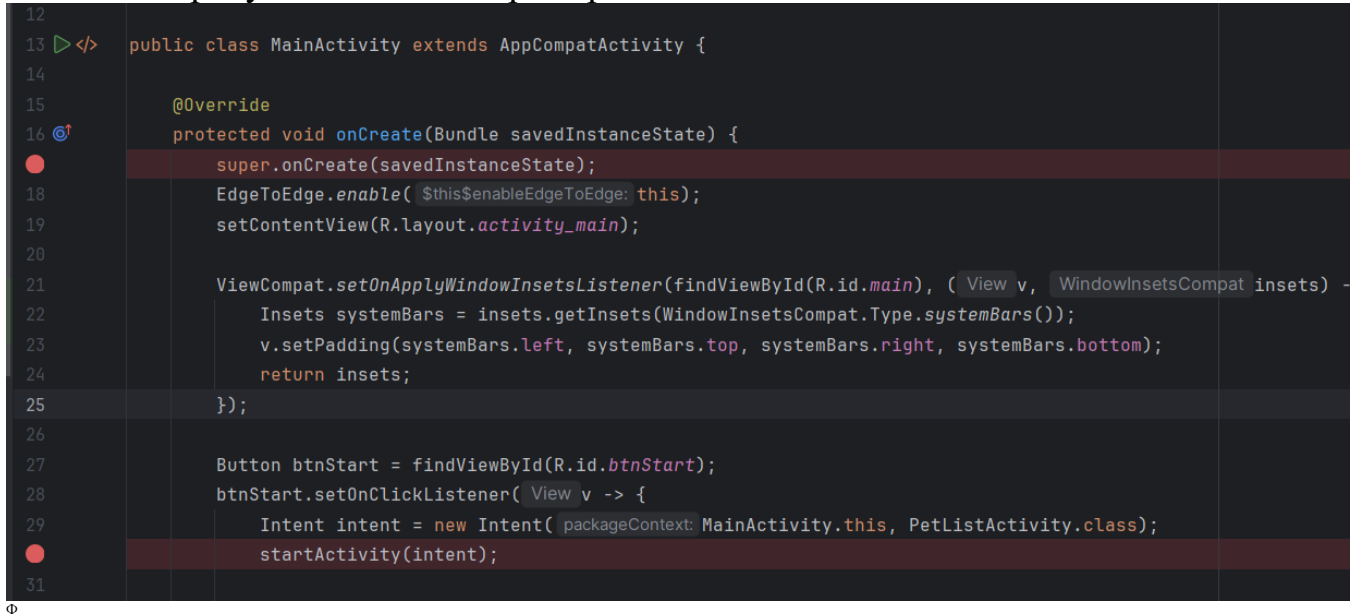
- LogCat – система логирования для отслеживания событий приложения и выявления ошибок в runtime;
- Android Emulator – тестирование на различных виртуальных устройствах с разными версиями Android и характеристиками экрана;
- Layout Inspector – проверка корректности отображения пользовательского интерфейса;
- Profiler – мониторинг использования памяти, процессора и батареи для оптимизации производительности;
- Debugger – исследование состояния объектов PetModel и корректности работы с базой данных Room.

Тестирование проходило в рабочем процессе, в процессе разработки и после завершения написания программы. Были протестированы следующие сценарии:

- корректность сохранения и загрузки состояния питомца из базы данных;
- правильность расчета изменения характеристик во времени;

- работа системы уведомлений о необходимости ухода;
- корректность переключения между уровнями сложности;
- обработка поворота экрана и сохранения состояния Activity;
- работа ассемблерных вставок на различных устройствах.

На рисунке 2 показан пример использования точек останова.



```
12
13 </> public class MainActivity extends AppCompatActivity {
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         EdgeToEdge.enable( $this$enableEdgeToEdge: this);
19         setContentView(R.layout.activity_main);
20
21         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), ( View v, WindowInsetsCompat insets) -
22             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
23             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
24             return insets;
25         });
26
27         Button btnStart = findViewById(R.id.btnStart);
28         btnStart.setOnClickListener( View v -> {
29             Intent intent = new Intent( packageContext: MainActivity.this, PetListActivity.class);
30             startActivity(intent);
31         });
32     }
33 }
```

Рисунок 3 – Использование точек останова

5 Описание программы

5.1 Схема функции main

На рисунке 3 представлена схема функции main.

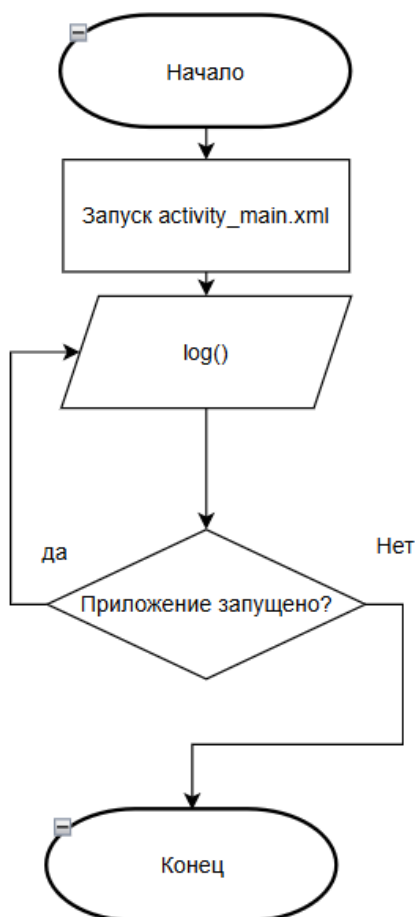


Рисунок 4 - Схема функции main

5.2 Описание модулей

Приложение построено по модульной архитектуре, что обеспечивает разделение ответственности, упрощает поддержку кода и позволяет независимо тестировать отдельные компоненты. Ниже представлено подробное описание основных модулей и их методов.

MainActivity.java – главный управляющий модуль приложения, отвечающий за точку входа и начальную навигацию.

Метод onCreate() выполняется при создании активности. В нём включается поддержка отображения во всю площадь экрана (EdgeToEdge), загружается разметка интерфейса, настраивается обработка системных отступов для

корректного отображения на устройствах с вырезами и динамическими панелями. Также здесь инициализируется кнопка запуска, на которую устанавливается обработчик нажатия: при клике создаётся намерение (Intent) для перехода к экрану списка питомцев (PetListActivity), после чего текущая активность завершается.

PetActivity.java – центральный модуль игровой логики, реализующий взаимодействие пользователя с виртуальным питомцем.

Метод onCreate() выполняет полную инициализацию экрана ухода: загружает сохранённые игровые данные (монеты, инвентарь) из SharedPreferences, получает ID питомца из намерения, инициализирует репозиторий для работы с базой данных, связывает переменные с элементами интерфейса (текстовые поля, прогресс-бары, кнопки), назначает обработчики событий для всех интерактивных элементов, запускает таймер изменения характеристик питомца, создаёт канал уведомлений и запрашивает необходимые разрешения.

Метод loadGameData() считывает из SharedPreferences сохранённые данные: количество монет и содержимое инвентаря. Инвентарь хранится в виде строки формата "предмет:количество|предмет:количество", которая разбирается и загружается в коллекцию HashMap.

Метод saveGameData() выполняет обратную операцию: собирает текущее состояние инвентаря в строку и сохраняет вместе с количеством монет в SharedPreferences, обеспечивая персистентность игрового прогресса между сеансами.

Метод loadPetData() принимает идентификатор питомца, выполняет асинхронный запрос к базе данных через репозиторий, находит соответствующую запись и, при успешном поиске, обновляет интерфейс в главном потоке: устанавливает эмодзи, имя, вид питомца и вызывает обновление статистики.

Метод updateStats() синхронизирует значения характеристик питомца (голод, жажда, счастье, здоровье) с элементами интерфейса: обновляет текстовые отображения процентов и устанавливает соответствующие значения

прогресс-баров.

Метод `startHungerTimer()` создаёт и запускает периодическую задачу (`Runnable`), которая выполняется каждую минуту. Задача уменьшает показатели голода, жажды и счастья, а при критически низких значениях – также снижает здоровье. После каждого изменения выполняется обновление базы данных и интерфейса, а также проверка состояния питомца на предмет смерти или необходимости уведомления.

Метод `checkPetStatus()` анализирует текущие значения характеристик: если здоровье упало до нуля – показывает диалог о смерти питомца и завершает активность; если голод или жажда ниже порогового значения – инициирует отправку уведомления (с учётом временного интервала между повторными оповещениями).

Метод `showFeedDialog()` формирует список доступных предметов еды из инвентаря и отображает их в диалоговом окне. Если инвентарь пуст – предлагает перейти в магазин. При выборе предмета вызывается метод применения еды.

Метод `feedPetWithItem()` применяет выбранный предмет еды к питомцу: увеличивает сытость, счастье и здоровье согласно параметрам предмета, может незначительно увеличивать жажду, уменьшает количество предмета в инвентаре, сохраняет изменения в базе данных и начисляет небольшую награду в монетах.

Метод `showDrinkDialog()` и `drinkPetWithItem()` работают аналогично методам кормления, но для категории напитков: восстанавливают жажду и счастье, не влияя на голод.

Метод `openShop()` формирует и отображает интерфейс магазина с тремя категориями товаров: еда, напитки, лекарства. Для каждого товара отображается название, цена, иконка и количество в инвентаре. При нажатии на товар: если это еда или напиток – списываются монеты, предмет добавляется в инвентарь; если это лекарство – применяется мгновенно с восстановлением здоровья.

Метод `applyMedicineImmediately()` применяет лекарство без добавления в инвентарь: увеличивает здоровье и счастье питомца, списывает монеты, показывает визуальное подтверждение эффекта.

Метод `startMiniGame()` запускает активность мини-игры через `Intent` с ожиданием результата. После завершения игры, если пользователь заработал монеты, они добавляются к общему счёту и отображаются в интерфейсе.

Метод `showInventoryDialog()` отображает список предметов в инвентаре, сгруппированных по категориям, с указанием количества каждого.

Метод `showSettingsDialog()` открывает меню настроек с опциями: переключение звука, уведомлений, темы оформления, сброс прогресса и информация о приложении. При выборе сброса прогресса запрашивается подтверждение, после чего очищаются данные инвентаря и монет.

Метод `createNotificationChannel()` создаёт канал уведомлений для Android 8.0 и выше, что является обязательным требованием для отображения push-уведомлений в современных версиях системы.

Метод `showHungerNotification()` формирует и отправляет уведомление с информацией о текущем состоянии питомца, если показатели голода или жажды опустились ниже критического уровня.

Метод `spendCoins()` проверяет достаточность средств, списывает указанную сумму монет, сохраняет изменения и обновляет отображение баланса в интерфейсе.

Метод `addCoins()` увеличивает счёт монет, сохраняет изменение и обновляет интерфейс.

Метод `performAction()` выполняет игровое действие "игра": проверяет, достаточно ли сыт и напоен питомец для игры, затем увеличивает счастье и уменьшает голод и жажду, сохраняя изменения в базе данных.

Метод `safeExecute()` обеспечивает безопасное выполнение задач в фоновом потоке через `ExecutorService`, предотвращая блокировку основного потока и гарантируя корректную работу с базой данных.

Метод `updateCoinsDisplay()` обновляет текстовое отображение количества монет в интерфейсе.

Методы `onResume()`, `onPause()`, `onDestroy()` переопределяют жизненный цикл активности: возобновляют или останавливают таймер голода, освобождают ресурсы, закрывают диалоговые окна и завершают работу пула потоков.

PetListActivity.java – модуль управления коллекцией питомцев.

Метод onCreate() инициализирует интерфейс списка, создаёт адаптер для отображения питомцев, загружает данные из базы, настраивает обработчики нажатий: короткий клик открывает детальную информацию, долгое нажатие вызывает контекстное меню с опциями переименования и удаления.

Метод loadPets() выполняет асинхронную загрузку всех питомцев из базы данных и обновляет адаптер в главном потоке.

Метод showAddPetDialog() отображает диалог создания нового питомца: поле ввода имени и выпадающий список видов (кот, собака, птица, рыбка, кролик, черепаха). При подтверждении создаётся новый объект Pet с начальными характеристиками и сохраняется в базе данных.

Метод showActionsDialog() открывает расширенное диалоговое окно с детальной информацией о питомце: эмодзи, имя, вид, текущие значения характеристик с прогресс-барами, а также кнопки возврата и перехода к экрану ухода.

Метод showLongClickMenu() показывает контекстное меню с опциями "Переименовать" и "Удалить" при долгом нажатии на элемент списка.

Метод showRenameDialog() позволяет изменить имя питомца: отображает поле ввода с текущим значением, при подтверждении обновляет запись в базе данных и перезагружает список.

Метод showDeleteConfirmationDialog() запрашивает подтверждение перед удалением питомца, после чего выполняет удаление через репозиторий и обновляет интерфейс.

Метод updateAddButtonState() отслеживает количество питомцев в списке: если достигнут лимит (3 питомца), блокирует кнопку добавления и изменяет её текст и прозрачность для визуального информирования пользователя.

PetRepository.java – модуль абстракции работы с базой данных.

Метод insert() добавляет нового питомца в базу данных с начальными значениями характеристик (голод, жажда, счастье, здоровье = 100%).

Метод update() обновляет существующую запись питомца: сохраняет текущие значения всех характеристик, имя и вид.

Метод `delete()` удаляет запись питомца по идентификатору.

Метод `getAll()` возвращает список всех питомцев, хранящихся в базе данных, для отображения в интерфейсе.

`Pet.java` – модель данных, описывающая сущность питомца.

Класс содержит приватные поля: уникальный идентификатор, имя, вид, эмодзи, числовые показатели характеристик. Публичные геттеры и сеттеры обеспечивают контролируемый доступ к полям. Конструкторы позволяют создавать питомца с параметрами по умолчанию или с заданными значениями.

`PetSimpleAdapter.java` – кастомный адаптер для отображения списка питомцев в `ListView`.

Метод `updatePets()` принимает новый список питомцев, очищает текущие данные, добавляет новые и уведомляет адаптер об изменении для перерисовки интерфейса.

Метод `getView()` создаёт или переиспользует представление для каждого элемента списка: устанавливает эмодзи, имя, вид питомца и визуальные индикаторы его состояния.

`MiniGameActivity.java` – модуль мини-игры для заработка монет. Активность предоставляет простой игровой процесс (например, тапалку или викторину), по завершении которого возвращает результат в `PetActivity` через `setResult()` с данными о заработанных монетах.

`db/PetDatabase.java` – модуль настройки базы данных `Room`. Содержит аннотации и конфигурацию для создания `SQLite`-базы, определения сущностей, миграций и типа данных. Обеспечивает потокобезопасный доступ к данным и автоматическое управление соединениями.

Класс `FoodItem` инкапсулирует параметры предмета еды: цена, прирост сытости, счастья, здоровья и возможное увеличение жажды. Используется для расчёта эффектов при кормлении.

Класс `DrinkItem` описывает напиток: цена, прирост гидратации и счастья.

Класс `MedicineItem` содержит параметры лекарства: цена, восстановление здоровья и бонус к счастью.

Все модули взаимодействуют через чётко определённые интерфейсы и

асинхронные механизмы, что обеспечивает отзывчивость интерфейса, целостность данных и масштабируемость архитектуры приложения.

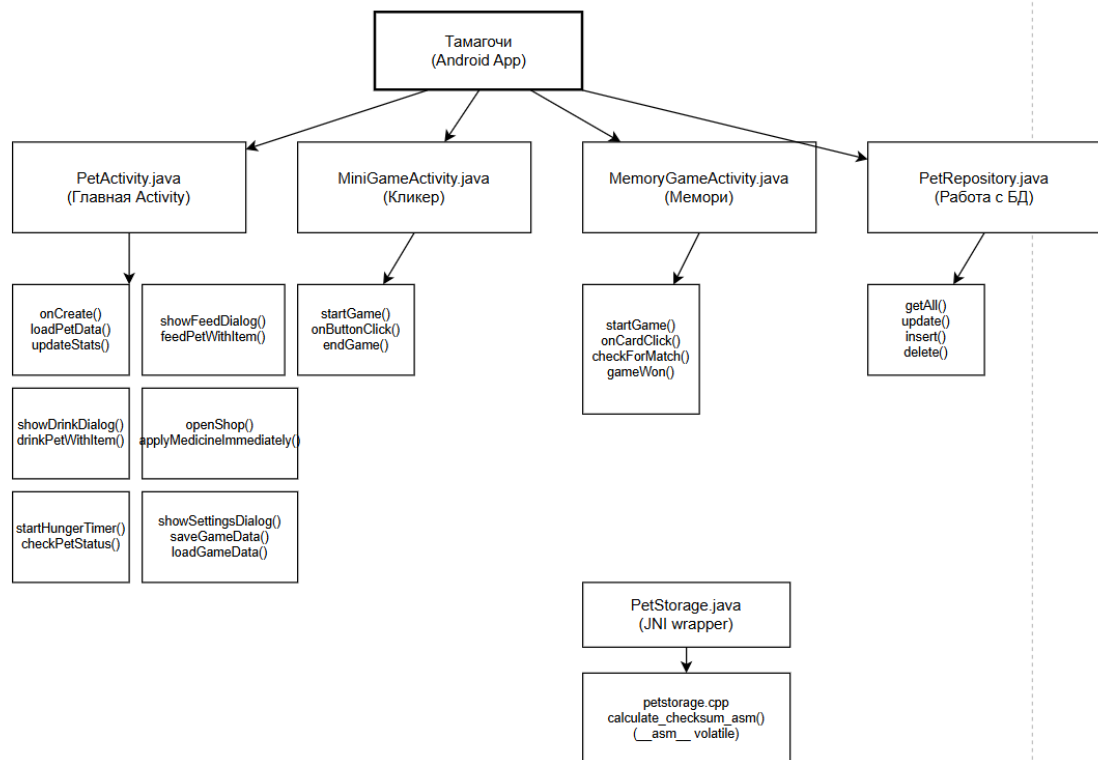


Рисунок 5 — Иерархическая структура программы

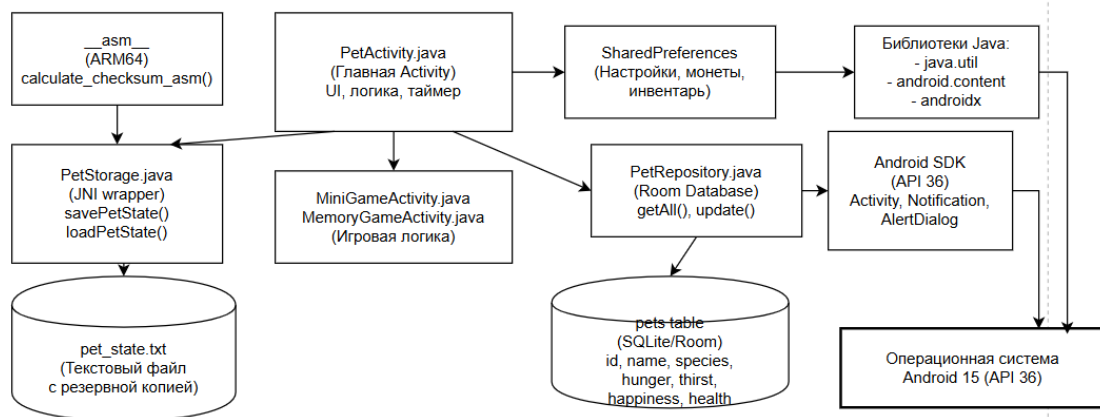


Рисунок 6 — Схема взаимодействия программы

Ассемблерная вставка находится в native-lib.cpp. Она реализует работу с сохранениями, находящимися в отдельном файле.

```

__attribute__((naked)) static void asm_xor_crypt(uint8_t* data, size_t len, uint8_t key) {
    __asm__ volatile (
        "cbz x1, xor_done\n"
        "mov x3, #0\n"
        "mov w4, w2\n"
  
```

```

        "xor_loop:\n\t"
        "ldrb w5, [x0, x3]\n"
        "eor w5, w5, w4\n"
        "strb w5, [x0, x3]\n"
        "add x3, x3, #1\n"
        "cmp x3, x1\n"
        "blt xor_loop\n"
        "xor_done:\n\t"
        "ret\n"
        ::: "memory", "x0", "x1", "x2", "x3", "x4", "x5"
    );
}

__attribute__((naked)) static long raw_write_syscall(int fd, const void* buf, size_t count)
{
    __asm__ volatile (
        "mov x8, #64\n"
        "svc #0\n"
        "ret\n"
        ::: "memory", "x8", "x0", "x1", "x2"
    );
}

__attribute__((naked)) static long raw_read_syscall(int fd, void* buf, size_t count) {
    __asm__ volatile (
        "mov x8, #63\n"
        "svc #0\n"
        "ret\n"
        ::: "memory", "x8", "x0", "x1", "x2"
    );
}

```

6 Руководство пользователя

Программа “Tamagothi” предназначена для мобильного приложения «Тамагочи». Она интуитивно понятна в управлении.

«Тамагочи» — классическая игра-симулятор, где игрок заботится о виртуальном питомце, поддерживая его характеристики на должном уровне и обеспечивая своевременный уход. Виртуальный питомец представляет собой интерактивный объект с набором изменяемых параметров: голод, настроение, энергия, здоровье и гигиена. При выполнении действий (кормление, игра, лечение, уборка, отправление ко сну) соответствующие показатели улучшаются, что визуально отображается на экране через анимацию питомца и индикаторы состояния. Если показатели опускаются до критического уровня — питомец заболевает или убегает, что приводит к завершению игры. Победа наступает, когда питомец успешно развивается, растёт и проходит все этапы эволюции благодаря заботливому уходу.

6.1 Заставка

На рисунке 4 представлена игровая заставка, которая появляется при запуске приложения.



Рисунок 7 - Заставка

6.2 Список питомцев

На рисунке 5 представлено окно списка питомцев.



Рисунок 8 - окно списка питомцев

Максимальное количество питомцев — 3 штуки. После достижения максимального количества кнопка добавления питомца становится неактивной. При длительном нажатии на питомца — открывается окно настроек (Рисунок 6), содержащие кнопки удалить и переименовать.

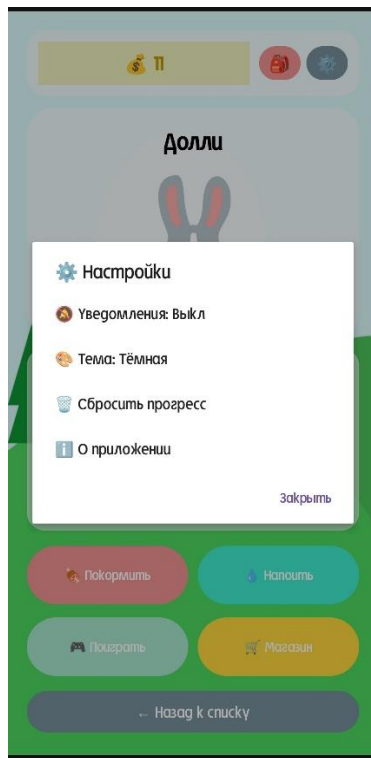


Рисунок 9 - окно списка питомцев

При нажатии на кнопку добавить нового питомца (Рисунок 7) – открывается окно добавления игрового персонажа. Окно добавления позволяет выбрать питомцу имя и выбрать вид питомца.

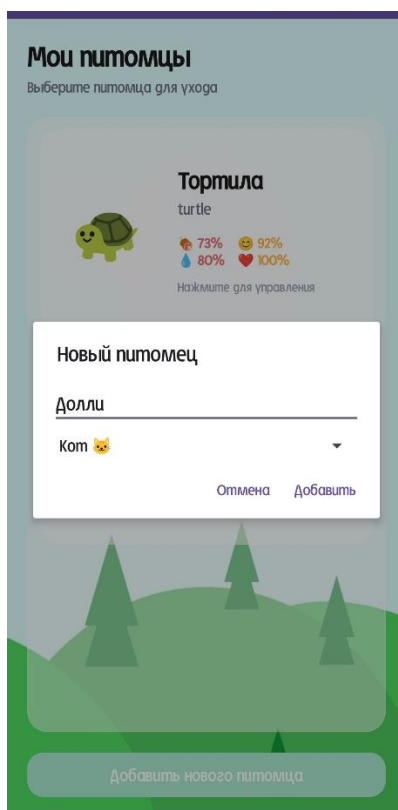


Рисунок 10₄ - Заставка

При нажатии на питомца открывается активность(Рисунок 8) позволяющая просмотреть статистику питомца и перейти на активити ухода.

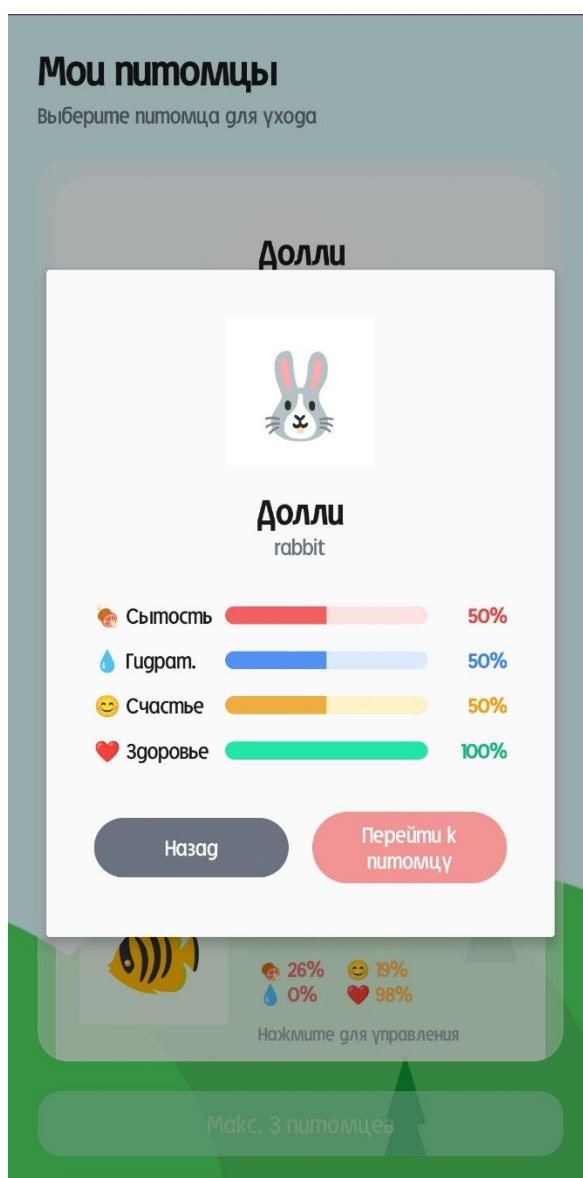


Рисунок 11 - Заставка

6.3 Меню питомца

На активности по уходу за питомцем, изображённой на Рисунке 9, находятся его основные характеристики, отображается количество денег и присутствуют кнопки: покормить, напоить, поиграть и магазин. Кнопки покормить и напоить увеличивают характеристики голода и гидротации, при условии наличия продуктов, которые можно приобрести в магазине.

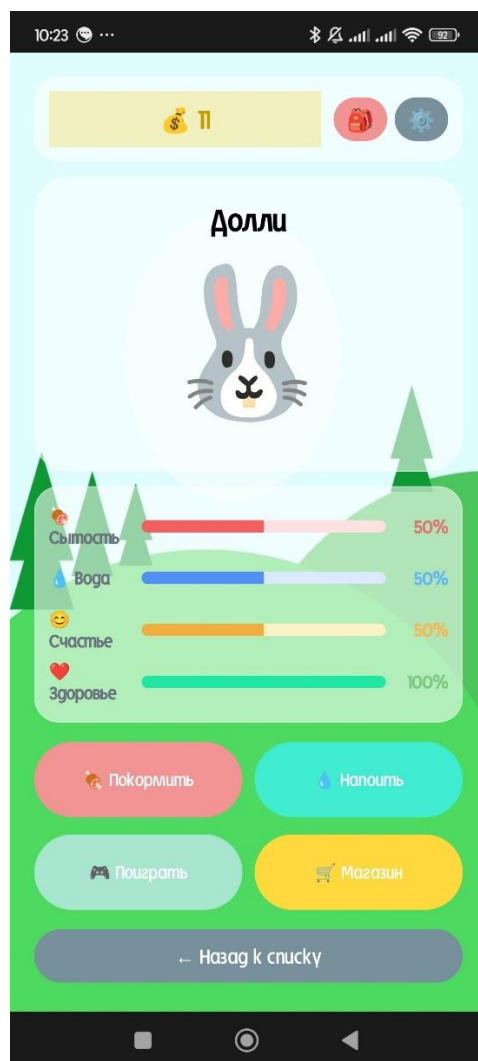


Рисунок 12 - Заставка

Нажатие кнопки поиграть вызывает окошко со списком игр(Рисунок 10). Всего имеется две игры – кликер и карточки.

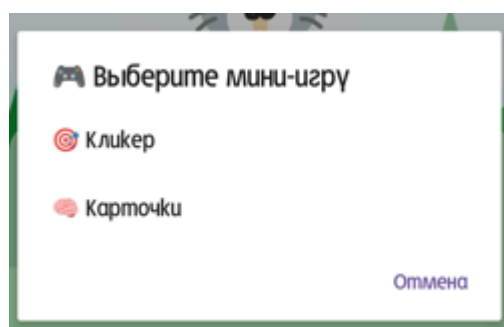


Рисунок 13 – Заставка

Также на главном окне располагается кнопка настроек(Рисунок 11), которая позволяет выбрать тему, выключить/включить уведомления, сбросить текущий прогресс и посмотреть информацию о приложении.

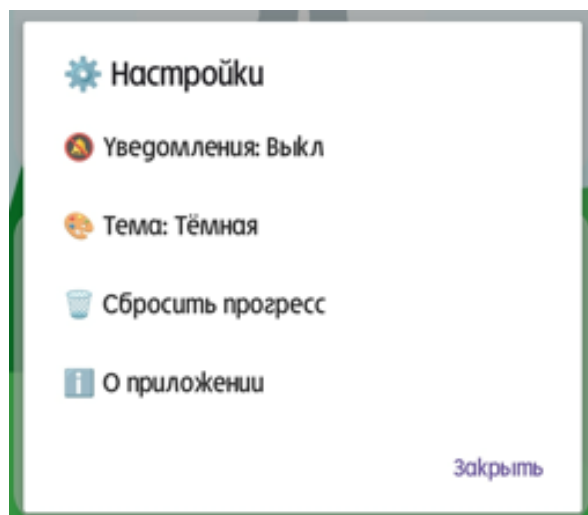


Рисунок 14 - Меню уровней сложности

6.4 Игра Кликер

После выбора игры Кликер, появляется окно с двумя счётчиками и кнопками начала игры и кнопкой Кликером(Рисунок 12).

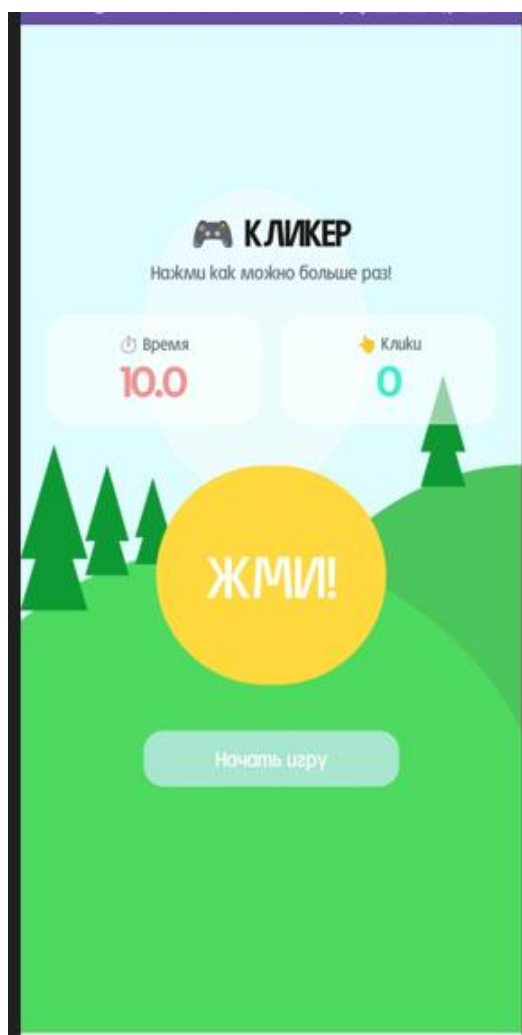


Рисунок 15 - Заставка

6.5 Игра Карточки

После выбора игры Карточки, появляется окно с кнопками-карточками, открывая их по две штуки – необходимо найти пары(Рисунок 13).

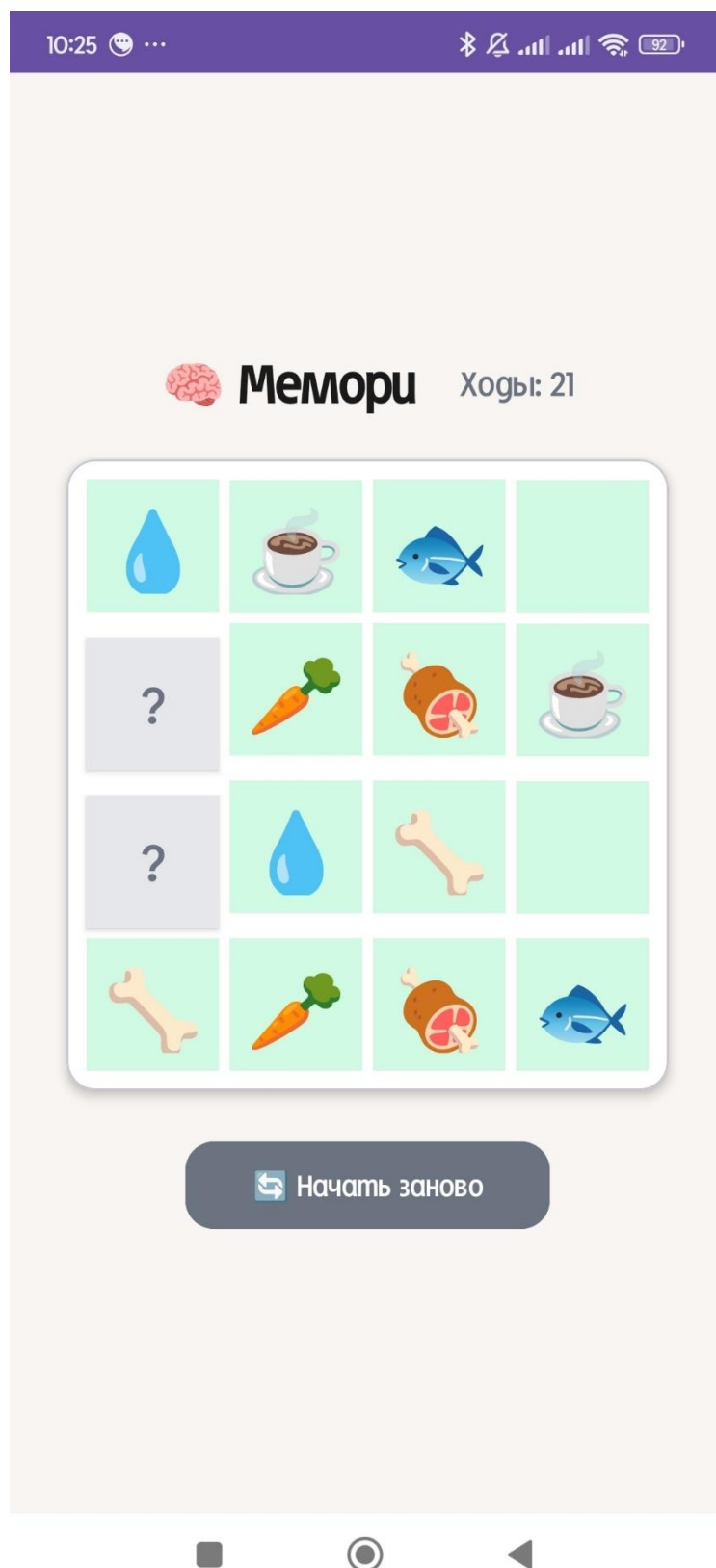


Рисунок 16 - Заставка

6.6 Магазин

Магазин, показанный на Рисунке 14, содержит в себе список продукции(еда, вода и лекарства). Каждая позиция имеет свою цену.

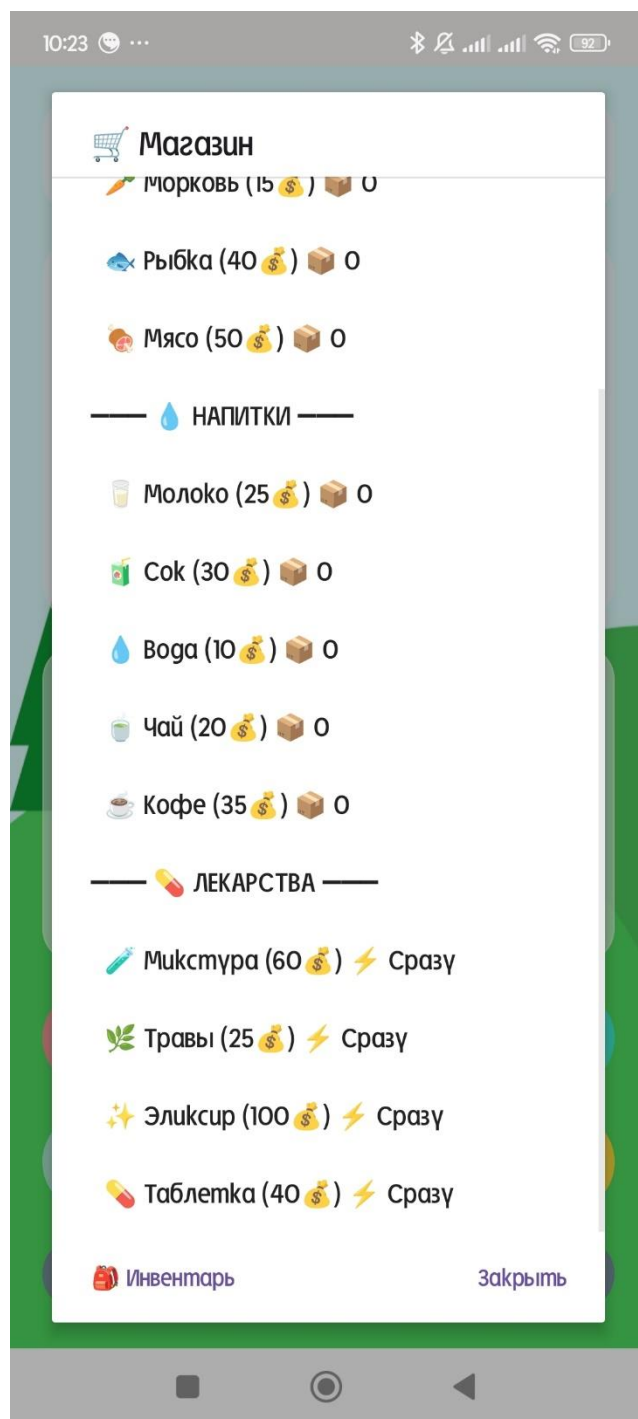


Рисунок 17 - Заставка

При нажатии на продукт происходит покупка и купленное помещается в инвентарь(Рисунок 15).

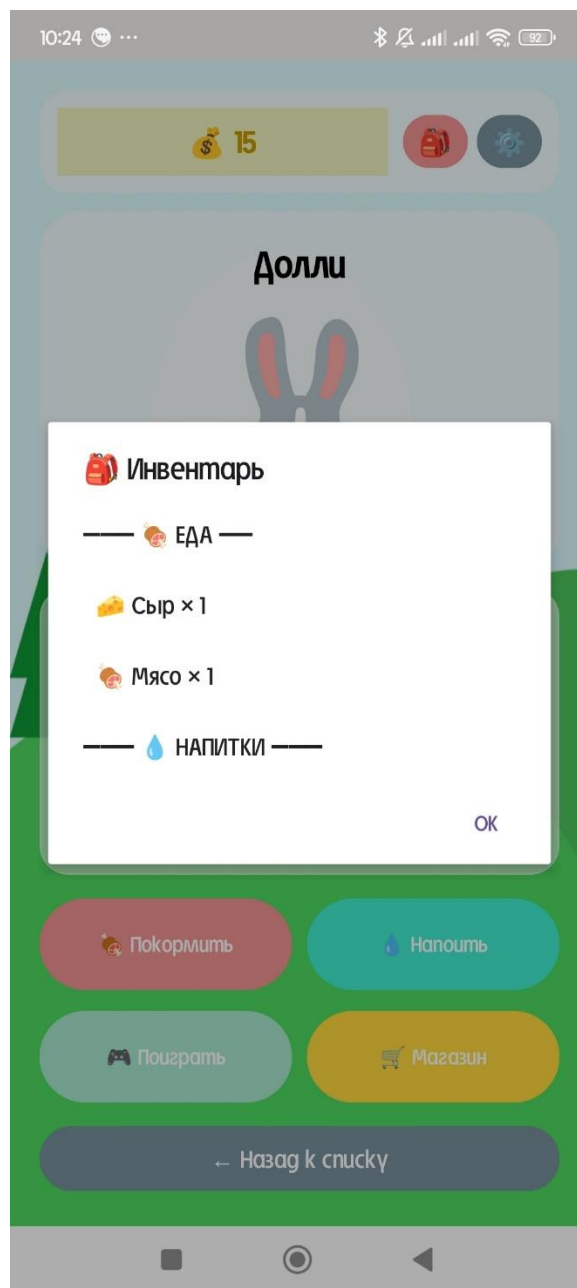


Рисунок 18 - Заставка

Заключение

В ходе выполнения курсовой работы была разработана мобильная игра «Тамагочи» для платформы Android, реализованная на языке программирования Java с использованием ассемблерных вставок. Данный проект позволил закрепить и углубить ключевые навыки мобильной разработки, а также продемонстрировал эффективное применение возможностей Java и низкоуровневой оптимизации для решения практических задач.

Была успешно применена работа с объектно-ориентированными структурами данных, включая:

- создание и управление жизненным циклом объектов виртуального питомца;
- оптимизацию использования коллекций и структур данных;
- предотвращение утечек памяти через корректное управление ресурсами Android.

Реализована сложная логика игры, включающая:

- динамическую симуляцию изменения характеристик питомца в реальном времени;
- систему прогрессии и развития (рост, эволюция, разблокировка контента);
- механизм обработки пользовательских действий (кормление, игры, лечение, сон);
- систему уведомлений и напоминаний о необходимости ухода.

Применены низкоуровневые оптимизации:

- использование ассемблерных вставок через JNI для критически важных вычислений;
- оптимизация работы с таймерами и расчёта временных интервалов;
- улучшение производительности алгоритмов симуляции и генерации событий.

Разработан удобный пользовательский интерфейс:

- адаптивный дизайн под различные размеры экранов Android-

устройств;

- система выбора уровня сложности;
- наглядное отображение состояния питомца через анимации и визуальные индикаторы;
- интуитивная навигация и панель инструментов.

Реализована надёжная система хранения данных:

- работа с локальной базой данных SQLite через Room Persistence Library;
- автосохранение состояния питомца и прогресса игры;
- ведение журнала событий и статистики взаимодействия.

Выполненная работа продемонстрировала эффективность языка Java для разработки сложных мобильных приложений. Программа успешно реализует все основные функции классической игры «Тамагочи», демонстрируя при этом высокую производительность и надёжность. Интеграция ассемблерных вставок позволила оптимизировать ресурсоёмкие операции без потери читаемости основного кода.

Приобретённые в ходе работы навыки имеют важное значение для дальнейшего профессионального роста в области мобильной разработки, изучения архитектурных паттернов и принципов создания производительных Android-приложений.

Список используемых материалов

1. Android Developers – Official Documentation:
<https://developer.android.com/>
2. Oracle Java Documentation: <https://docs.oracle.com/javase/>
3. Room Persistence Library – Android Developers:
<https://developer.android.com/training/data-storage/room>
4. Java Native Interface (JNI) Specification:
<https://docs.oracle.com/javase/8/docs/technotes/guides/jni/>
5. Stack Overflow – Android Development Tag:
<https://stackoverflow.com/questions/tagged/android>
6. GitHub – Android Architecture Components:
<https://github.com/android/architecture-components-samples>
7. X86 Assembly Language Reference:
<https://www.intel.com/content/www/us/en/develop/documentation/>
8. MVVM Architecture Pattern – Android Developers:
<https://developer.android.com/topic/architecture/ui-layer>
9. SQLite Official Documentation: <https://www.sqlite.org/docs.html>
10. GeeksforGeeks – Java Programming: <https://www.geeksforgeeks.org/java/>
11. Шилдт, Г. Java: полное руководство / Г. Шилдт. – 9-е изд. – Москва: Вильямс, 2020. – 1248 с.
12. Андре, К. Android. Программирование для профессионалов / К. Андре, Б. Филлипс, Б. Харди. – 4-е изд. – Санкт-Петербург: Питер, 2021. – 832 с.
13. Гаск, Д. SQLite. Карманный справочник / Д. Гаск. – Москва: Символ-Плюс, 2019. – 176 с.
14. Зюльков, Д.А. Ассемблер и объектно-ориентированное программирование / Д.А. Зюльков. – Москва: БИНОМ, 2018. – 352 с.
15. Иртегов, Д. Введение в системное программирование / Д. Иртегов. – Санкт-Петербург: Питер, 2020. – 528 с.
16. Брукс, Ф. Мифический человеко-месяц, или Как создаются программные системы / Ф. Брукс. – 3-е изд. – Москва: Вильямс, 2019. – 304 с.
17. Лафоре, Р. Объектно-ориентированное программирование в Java / Р. Лафоре. – 4-е изд. – Москва: Вильямс, 2017. – 912 с.

Приложение А

MainActivity.java

```
package com.example.kursovoi;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity extends AppCompatActivity {

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

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        Button btnStart = findViewById(R.id.btnStart);
        btnStart.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this, PetListActivity.class);
            startActivity(intent);

            finish();
        });
    }
}
```

Pet.java

```
package com.example.kursovoi;

public class Pet {
    private long id;
    private int thirst;
    private String name;
    private String species;
    private String emoji;
```

```

private int hunger;
private int happiness;
private int health;

public Pet() {}

public Pet(String name, String species, String emoji) {
    this.name = name;
    this.species = species;
    this.emoji = emoji;
    this.hunger = 50;
    this.happiness = 50;
    this.health = 100;
    this.thirst = 50;
}

public long getId() { return id; }
public void setId(long id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public String getSpecies() { return species; }
public void setSpecies(String species) { this.species = species; }
public String getEmoji() { return emoji; }
public void setEmoji(String emoji) { this.emoji = emoji; }
public int getHunger() { return hunger; }
public void setHunger(int hunger) { this.hunger = hunger; }
public int getHappiness() { return happiness; }
public void setHappiness(int happiness) { this.happiness = happiness; }
public int getHealth() { return health; }
public void setHealth(int health) { this.health = health; }
public int getThirst() { return thirst; }
public void setThirst(int thirst) { this.thirst = thirst; }
}

```

PetListActivity.java

```

package com.example.kursovoi;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import com.example.kursovoi.db.PetRepository;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class PetListActivity extends AppCompatActivity {

    private ListView lvPets;
    private Button btnAddPet;
    private PetRepository repository;
    private PetSimpleAdapter adapter;

```

```

private ExecutorService executor = Executors.newSingleThreadExecutor();

private static final int MAX_PETS = 3;

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

    lvPets = findViewById(R.id.lvPets);
    btnAddPet = findViewById(R.id.btnAddPet);
    repository = new PetRepository(this);

    adapter = new PetSimpleAdapter(this, repository);
    lvPets.setAdapter(adapter);

    loadPets();
    updateAddButtonState();

    lvPets.setOnItemClickListener((parent, view, position, id) -> {
        Pet selectedPet = repository.getAll().get(position);
        showActionsDialog(selectedPet);
    });

    lvPets.setOnItemLongClickListener((parent, view, position, id) -> {
        Pet selectedPet = repository.getAll().get(position);
        showLongClickMenu(selectedPet);
        return true;
    });

    btnAddPet.setOnClickListener(v -> {
        if (adapter.getCount() >= MAX_PETS) {
            new AlertDialog.Builder(this)
                .setTitle("🔒 Лимит достигнут")
                .setMessage("Можно завести максимум " + MAX_PETS + " питомцев.")
                .setPositiveButton("ОК", null)
                .show();
            return;
        }
        showAddPetDialog();
    });
}

private void loadPets() {
    executor.execute(() -> {
        List<Pet> pets = repository.getAll();
        runOnUiThread(() -> {
            adapter.updatePets(pets);
            updateAddButtonState();
        });
    });
}

```

```

}

private void updateAddButtonState() {
    if (adapter.getCount() >= MAX_PETS) {
        btnAddPet.setEnabled(false);
        btnAddPet.setAlpha(0.5f);
        btnAddPet.setText("Макс. " + MAX_PETS + " питомцев");
    } else {
        btnAddPet.setEnabled(true);
        btnAddPet.setAlpha(1.0f);
        btnAddPet.setText("Добавить нового питомца");
    }
}

private void showAddPetDialog() {
    if (adapter.getCount() >= MAX_PETS) {
        Toast.makeText(this, "Достигнут лимит питомцев!", Toast.LENGTH_SHORT).show();
        return;
    }

    LinearLayout layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    layout.setPadding(50, 40, 50, 10);

    EditText inputName = new EditText(this);
    inputName.setHint("Имя питомца");
    inputName.setLayoutParams(new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT));
    layout.addView(inputName);

    Spinner spinnerSpecies = new Spinner(this);
    String[] speciesList = {"Кот 🐱", "Собака 🐶", "Птица 🐦", "Рыбка 🐠", "Кролик 🐰", "Черепаша 🐢"};
    ArrayAdapter<String> speciesAdapter = new ArrayAdapter<>(this,
        android.R.layout.simple_spinner_item, speciesList);
    speciesAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinnerSpecies.setAdapter(speciesAdapter);
    spinnerSpecies.setLayoutParams(new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT));

    LinearLayout.LayoutParams spinnerParams = (LinearLayout.LayoutParams)
    spinnerSpecies.getLayoutParams();
    spinnerParams.topMargin = 20;
    spinnerSpecies.setLayoutParams(spinnerParams);
    layout.addView(spinnerSpecies);

    new AlertDialog.Builder(this)
        .setTitle("Новый питомец")

```

```

        .setView(layout)
        .setPositiveButton("Добавить", (dialog, which) -> {
            String rawName = inputName.getText().toString().trim();
            String petName = rawName.isEmpty() ? "Безымянный" : rawName;

            int selectedPosition = spinnerSpecies.getSelectedItemId();
            String[] speciesKeys = {"cat", "dog", "bird", "fish", "rabbit", "turtle"};
            String[] speciesEmojis = {"🐱", "🐶", "🐦", "🐟", "🐰", "🐢"};

            String selectedSpecies = speciesKeys[selectedPosition];
            String selectedEmoji = speciesEmojis[selectedPosition];

            executor.execute(() -> {
                repository.insert(new Pet(petName, selectedSpecies, selectedEmoji));
                runOnUiThread(() -> {
                    loadPets();
                    Toast.makeText(PetListActivity.this, petName + " добавлен!",
                        Toast.LENGTH_SHORT).show();
                });
            });
        })
        .setNegativeButton("Отмена", null)
        .show();
    }

```

```

private void showActionsDialog(Pet pet) {
    // 🍷 Инфлейтим готовый XML-дизайн
    View dialogView = getLayoutInflater().inflate(R.layout.dialog_pet_info, null);

    TextView tvEmoji = dialogView.findViewById(R.id.tvDialogEmoji);
    TextView tvName = dialogView.findViewById(R.id.tvDialogName);
    TextView tvSpecies = dialogView.findViewById(R.id.tvDialogSpecies);
    TextView tvHunger = dialogView.findViewById(R.id.tvDialogHunger);
    TextView tvThirst = dialogView.findViewById(R.id.tvDialogThirst);
    TextView tvHappiness = dialogView.findViewById(R.id.tvDialogHappiness);
    TextView tvHealth = dialogView.findViewById(R.id.tvDialogHealth);

    ProgressBar pbHunger = dialogView.findViewById(R.id.pbHunger);
    ProgressBar pbThirst = dialogView.findViewById(R.id.pbThirst);
    ProgressBar pbHappiness = dialogView.findViewById(R.id.pbHappiness);
    ProgressBar pbHealth = dialogView.findViewById(R.id.pbHealth);

    Button btnBack = dialogView.findViewById(R.id.btnDialogBack);
    Button btnGoToPet = dialogView.findViewById(R.id.btnDialogGoToPet);

    tvEmoji.setText(pet.getEmoji());
    tvName.setText(pet.getName());
    tvSpecies.setText(pet.getSpecies());
    tvHunger.setText(pet.getHunger() + "%");
    tvThirst.setText(pet.getThirst() + "%");
}

```

```

tvHappiness.setText(pet.getHappiness() + "%");
tvHealth.setText(pet.getHealth() + "%");

pbHunger.setProgress(pet.getHunger());
pbThirst.setProgress(pet.getThirst());
pbHappiness.setProgress(pet.getHappiness());
pbHealth.setProgress(pet.getHealth());

AlertDialog dialog = new AlertDialog.Builder(this)
    .setView(dialogView)
    .create();

btnBack.setOnClickListener(v -> dialog.dismiss());
btnGoToPet.setOnClickListener(v -> {
    dialog.dismiss();
    Intent intent = new Intent(PetListActivity.this, PetActivity.class);
    intent.putExtra("PET_ID", pet.getId());
    startActivity(intent);
});

dialog.show();
}

private void showLongClickMenu(Pet pet) {
    String[] actions = {" Переименовать", " Удалить"};
    new AlertDialog.Builder(this)
        .setTitle("Действия с " + pet.getName())
        .setItems(actions, (dialog, which) -> {
            if (which == 0) showRenameDialog(pet);
            else showDeleteConfirmationDialog(pet);
        })
        .show();
}

private void showRenameDialog(Pet pet) {
    EditText inputName = new EditText(this);
    inputName.setHint("Новое имя");
    inputName.setText(pet.getName());
    inputName.setSelection(pet.getName().length());

    new AlertDialog.Builder(this)
        .setTitle("Переименовать питомца")
        .setView(inputName)
        .setPositiveButton("Сохранить", (dialog, which) -> {
            String newName = inputName.getText().toString().trim();
            if (!newName.isEmpty()) {
                executor.execute() -> {
                    pet.setName(newName);
                    repository.update(pet);
                    runOnUiThread() -> {
                        loadPets();

```

```

        Toast.makeText(this, "Имя изменено!", Toast.LENGTH_SHORT).show();
    });
}
})
.setNegativeButton("Отмена", null)
.show();
}

private void showDeleteConfirmationDialog(Pet pet) {
    new AlertDialog.Builder(this)
        .setTitle("Удалить питомца?")
        .setMessage("Вы точно хотите удалить \'" + pet.getName() + "\"?")
        .setIcon(android.R.drawable.ic_dialog_alert)
        .setPositiveButton("✅ Да, удалить", (dialog, which) -> {
            executor.execute() -> {
                repository.delete(pet.getId());
                runOnUiThread() -> {
                    loadPets();
                    Toast.makeText(this, pet.getName() + " удалён", Toast.LENGTH_SHORT).show();
                }
            });
        })
        .setNegativeButton("❌ Отмена", null)
        .setNeutralButton("🏠 Назад", (dialog, which) -> {
            dialog.dismiss();
            showLongClickMenu(pet);
        })
        .show();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    executor.shutdown();
}
}

```

PetActivity.java

```

package com.example.kursovoi;

import android.Manifest;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import com.example.kursovoi.db.PetRepository;
import org.json.JSONObject;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

```

```

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class PetActivity extends AppCompatActivity {

    private Pet pet;
    private PetRepository repository;
    private final ExecutorService executor = Executors.newSingleThreadExecutor();
    private Map<String, Integer> inventory = new HashMap<>();

    private TextView tvEmoji, tvName, tvHunger, tvHappiness, tvHealth, tvThirst, tvPetSpecies, tvCoins;
    private ProgressBar pbHunger, pbThirst, pbHappiness, pbHealth;
    private Button btnFeed, btnPlay, btnDrink, btnShop, btnBack, btnInventory, btnSettings;

    private Handler hungerHandler;
    private Runnable hungerTask;
    private static final long HUNGER_INTERVAL = 60000;
    private NotificationManager notificationManager;
    private static final String CHANNEL_ID = "pet_care_channel";
    private static final int NOTIFICATION_ID = 1;
    private static final int NOTIFICATION_PERMISSION_CODE = 100;

    private int coins = 100;
    private long lastStatusNotificationTime = 0;
    private static final long STATUS_NOTIFICATION_COOLDOWN = 2 * 60 * 1000;

    private boolean soundEnabled = true, notificationsEnabled = true, themeDark = false;

    private AlertDialog shopDialog;
    private static final int REQUEST_MINI_GAME = 1001;
    private static final int REQUEST_MEMORY_GAME = 1002;

    private AssemblySave saver;

    private static final Map<String, FoodItem> FOOD_ITEMS = new HashMap<>();
    private static final Map<String, DrinkItem> DRINK_ITEMS = new HashMap<>();
    private static final Map<String, MedicineItem> MEDICINE_ITEMS = new HashMap<>();

    static {
        FOOD_ITEMS.put("🍏 Яблоко", new FoodItem(20, 15, 8, 0, -5));
        FOOD_ITEMS.put("🍖 Мясо", new FoodItem(50, 40, 5, 5, -10));
        FOOD_ITEMS.put("🥕 Морковь", new FoodItem(15, 20, 3, 2, -3));
        FOOD_ITEMS.put("🧀 Сыр", new FoodItem(35, 25, 10, 0, -8));
        FOOD_ITEMS.put("🐟 Рыбка", new FoodItem(40, 30, 7, 3, -12));
    }

    static {
        DRINK_ITEMS.put("💧 Вода", new DrinkItem(10, 30, 5));
        DRINK_ITEMS.put("🥛 Молоко", new DrinkItem(25, 45, 10));
        DRINK_ITEMS.put("🍹 Сок", new DrinkItem(30, 40, 15));
    }

```

```

    DRINK_ITEMS.put("🍵 Чай", new DrinkItem(20, 35, 8));
    DRINK_ITEMS.put("☕ Кофе", new DrinkItem(35, 25, 20));
}

static {
    MEDICINE_ITEMS.put("💊 Таблетка", new MedicineItem(40, 40, 5));
    MEDICINE_ITEMS.put("💧 Микстура", new MedicineItem(60, 70, 10));
    MEDICINE_ITEMS.put("🌟 Эликсир", new MedicineItem(100, 100, 20));
    MEDICINE_ITEMS.put("🌿 Травы", new MedicineItem(25, 25, 3));
}

private static class FoodItem {
    int price, satietyBoost, happinessBoost, healthBoost, thirstPenalty;
    FoodItem(int price, int s, int h, int hp, int t) {
        this.price = price; this.satietyBoost = s; this.happinessBoost = h;
        this.healthBoost = hp; this.thirstPenalty = t;
    }
}

private static class DrinkItem {
    int price, hydrationBoost, happinessBoost;
    DrinkItem(int price, int hydration, int happiness) {
        this.price = price; this.hydrationBoost = hydration; this.happinessBoost = happiness;
    }
}

private static class MedicineItem {
    int price, healthBoost, happinessBoost;
    MedicineItem(int price, int health, int happiness) {
        this.price = price; this.healthBoost = health; this.happinessBoost = happiness;
    }
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    saver = new AssemblySave(this);
    loadProgress();
    applyTheme();

    setContentView(R.layout.activity_pet);

    long petId = getIntent().getLongExtra("PET_ID", -1);
    repository = new PetRepository(this);

    tvEmoji = findViewById(R.id.tvPetEmoji);
    tvName = findViewById(R.id.tvPetName);
    tvPetSpecies = findViewById(R.id.tvPetSpecies);
}

```

```

tvHunger = findViewById(R.id.tvHunger);
tvThirst = findViewById(R.id.tvThirst);
tvHappiness = findViewById(R.id.tvHappiness);
tvHealth = findViewById(R.id.tvHealth);
tvCoins = findViewById(R.id.tvCoins);

pbHunger = findViewById(R.id.pbHunger);
pbThirst = findViewById(R.id.pbThirst);
pbHappiness = findViewById(R.id.pbHappiness);
pbHealth = findViewById(R.id.pbHealth);

btnFeed = findViewById(R.id.btnFeed);
btnPlay = findViewById(R.id.btnPlay);
btnDrink = findViewById(R.id.btnDrink);
btnShop = findViewById(R.id.btnShop);
btnBack = findViewById(R.id.btnBack);
btnInventory = findViewById(R.id.btnInventory);
btnSettings = findViewById(R.id.btnSettings);

btnFeed.setOnClickListener(v -> showFeedDialog());
btnPlay.setOnClickListener(v -> showGameSelectionDialog());
btnDrink.setOnClickListener(v -> showDrinkDialog());
btnShop.setOnClickListener(v -> openShop());
btnBack.setOnClickListener(v -> finish());
btnInventory.setOnClickListener(v -> showInventoryDialog());
btnSettings.setOnClickListener(v -> showSettingsDialog());

loadPetData(petId);
hungerHandler = new Handler(Looper.getMainLooper());
startHungerTimer();

createNotificationChannel();
requestNotificationPermission();
updateCoinsDisplay();
}

private void loadProgress() {
    JSONObject data = saver.load();
    if (data == null) {
        coins = 100;
        inventory.clear();
        soundEnabled = true;
        notificationsEnabled = true;
        themeDark = false;
        return;
    }
    try {
        coins = data.optInt("coins", 100);
        soundEnabled = data.optBoolean("sound", true);
        notificationsEnabled = data.optBoolean("notify", true);
        themeDark = data.optBoolean("dark", false);
    }
}

```

```

String invJson = data.optString("inventory", "{}");
JSONObject invObj = new JSONObject(invJson);
inventory.clear();
Iterator<String> keys = invObj.keys();
while (keys.hasNext()) {
    String key = keys.next();
    inventory.put(key, invObj.optInt(key, 0));
}
} catch (Exception e) {
    e.printStackTrace();
    coins = 100; inventory.clear();
}
}

private void saveProgress() {
    try {
        JSONObject data = new JSONObject();
        data.put("coins", coins);
        data.put("sound", soundEnabled);
        data.put("notify", notificationsEnabled);
        data.put("dark", themeDark);

        JSONObject invObj = new JSONObject();
        for (Map.Entry<String, Integer> e : inventory.entrySet()) {
            invObj.put(e.getKey(), e.getValue());
        }
        data.put("inventory", invObj);

        saver.save(data);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void testAssemblyFunction() {
    try {
        byte[] testData1 = "Hello".getBytes();
        int checksum1 = saver.computeChecksum(testData1, testData1.length);

        int checksum2 = saver.computeChecksum(testData1, testData1.length);

        byte[] testData3 = "World".getBytes();
        int checksum3 = saver.computeChecksum(testData3, testData3.length);

        byte[] emptyData = new byte[0];
        int checksumEmpty = saver.computeChecksum(emptyData, 0);

        android.util.Log.d("ASM_TEST", "===== TECT ACCЕМБЛЕРА =====");
        android.util.Log.d("ASM_TEST", "Test 1 'Hello': 0x" + Integer.toHexString(checksum1));
        android.util.Log.d("ASM_TEST", "Test 2 'Hello': 0x" + Integer.toHexString(checksum2));
        android.util.Log.d("ASM_TEST", "Test 3 'World': 0x" + Integer.toHexString(checksum3));
    }
}

```

```

        android.util.Log.d("ASM_TEST", "Test 4 Empty: 0x" + Integer.toHexString(checksumEmpty));

        boolean test1Pass = checksum1 == checksum2;
        boolean test2Pass = checksum1 != checksum3;
        boolean test3Pass = checksumEmpty == 0;

        android.util.Log.d("ASM_TEST", "===== РЕЗУЛЬТАТЫ =====");
        android.util.Log.d("ASM_TEST", "✓ Test 1 (Consistency): " + (test1Pass ? "PASS" : "FAIL"));
        android.util.Log.d("ASM_TEST", "✓ Test 2 (Different data): " + (test2Pass ? "PASS" : "FAIL"));
        android.util.Log.d("ASM_TEST", "✓ Test 3 (Empty array): " + (test3Pass ? "PASS" : "FAIL"));

        if (test1Pass && test2Pass) {
            android.util.Log.d("ASM_TEST", "🎉 ВСЕ ТЕСТЫ ПРОЙДЕНЫ! Ассемблер работает!");
            runOnUiThread() -> Toast.makeText(this, "✅ Ассемблер работает!",
Toast.LENGTH_SHORT).show());
        } else {
            android.util.Log.d("ASM_TEST", "❌ ОШИБКА В АССЕМБЛЕРЕ!");
            runOnUiThread() -> Toast.makeText(this, "❌ Ошибка ассемблера!",
Toast.LENGTH_LONG).show());
        }

    } catch (Exception e) {
        android.util.Log.e("ASM_TEST", "КРАШ: " + e.getMessage());
        e.printStackTrace();
    }
}

private void applyTheme() {
    if (themeDark) {
        getWindow().setStatusBarColor(0xFF1A1A1A);
        getWindow().setNavigationBarColor(0xFF1A1A1A);
    } else {
        getWindow().setStatusBarColor(0xFFF8F5F2);
        getWindow().setNavigationBarColor(0xFFFFFFFF);
    }
}

private void showGameSelectionDialog() {
    String[] games = {"🎮 Кликер ", "🧠 Карточки"};
    new AlertDialog.Builder(this)
        .setTitle("🎮 Выберите мини-игру")
        .setItems(games, (dialog, which) -> {
            Intent intent;
            int requestCode;
            if (which == 0) {
                intent = new Intent(PetActivity.this, MiniGameActivity.class);
                requestCode = REQUEST_MINI_GAME;
            } else {
                intent = new Intent(PetActivity.this, MemoryGameActivity.class);

```

```

        requestCode = REQUEST_MEMORY_GAME;
    }
    startActivityResult(intent, requestCode);
})
.setNegativeButton("Отмена", null)
.show();
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK && data != null) {
        int reward = data.getIntExtra("REWARD", 0);
        if (reward > 0) {
            coins += reward;
            saveProgress();
            updateCoinsDisplay();
            String gameName = (requestCode == REQUEST_MINI_GAME) ? "кликер" : "мемори";
            Toast.makeText(this, "💰 +" + reward + " монет за " + gameName + "!",
Toast.LENGTH_LONG).show();
        }
    }
}

private void requestNotificationPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.POST_NOTIFICATIONS)
            != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.POST_NOTIFICATIONS},
                NOTIFICATION_PERMISSION_CODE);
        }
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == NOTIFICATION_PERMISSION_CODE && grantResults.length > 0
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        Toast.makeText(this, "✅ Уведомления включены", Toast.LENGTH_SHORT).show();
    }
}

private void checkPetStatus() {
    if (pet.getHealth() <= 0) {
        if (hungerHandler != null && hungerTask != null) hungerHandler.removeCallbacks(hungerTask);
        if (isFinishing() || isDestroyed()) return;
        new AlertDialog.Builder(this).setTitle("💀 Питомец умер")
            .setMessage(pet.getName() + " погиб от голода и жажды...")

```

```

        .setPositiveButton("OK", (d, w) -> finish()).setCancelable(false).show();
    } else if (pet.getHunger() <= 20 || pet.getThirst() <= 20) {
        long now = System.currentTimeMillis();
        if (now - lastStatusNotificationTime >= STATUS_NOTIFICATION_COOLDOWN) {
            lastStatusNotificationTime = now;
            showHungerNotification();
        }
    }
}

private void loadPetData(long petId) {
    safeExecute() -> {
        List<Pet> allPets = repository.getAll();
        Pet found = null;
        for (Pet p : allPets) if (p.getId() == petId) { found = p; break; }
        if (found != null) {
            pet = found;
            runOnUiThread() -> {
                tvEmoji.setText(pet.getEmoji());
                tvName.setText(pet.getName());
                updateStats();
            };
        } else {
            runOnUiThread() -> { Toast.makeText(this, "Питомец не найден",
Toast.LENGTH_SHORT).show(); finish(); });
        }
    };
}

private void updateStats() {
    if (pet == null) return;
    if (tvHunger != null) tvHunger.setText(pet.getHunger() + "%");
    if (tvThirst != null) tvThirst.setText(pet.getThirst() + "%");
    if (tvHappiness != null) tvHappiness.setText(pet.getHappiness() + "%");
    if (tvHealth != null) tvHealth.setText(pet.getHealth() + "%");
    if (tvPetSpecies != null) tvPetSpecies.setText(pet.getSpecies());
    if (pbHunger != null) pbHunger.setProgress(pet.getHunger());
    if (pbThirst != null) pbThirst.setProgress(pet.getThirst());
    if (pbHappiness != null) pbHappiness.setProgress(pet.getHappiness());
    if (pbHealth != null) pbHealth.setProgress(pet.getHealth());
}

private void updateCoinsDisplay() {
    if (tvCoins != null) tvCoins.setText(String.valueOf(coins));
}

private boolean spendCoins(int amount) {
    if (coins >= amount) {
        coins -= amount;
        saveProgress();
        runOnUiThread(this::updateCoinsDisplay);
    }
}

```

```

        return true;
    }
    return false;
}

private void addCoins(int amount) {
    coins += amount;
    saveProgress();
    runOnUiThread(this::updateCoinsDisplay);
}

private void openShop() {
    if (pet == null) { Toast.makeText(this, "Питомец не загружен!", Toast.LENGTH_SHORT).show();
return; }
    if (shopDialog != null && shopDialog.isShowing()) shopDialog.dismiss();

    List<String> items = new ArrayList<>();
    items.add("—— 🍖 ЕДА ——");
    for (Map.Entry<String, FoodItem> e : FOOD_ITEMS.entrySet()) {
        int q = inventory.getDefault(e.getKey(), 0);
        items.add(" " + e.getKey() + " (" + e.getValue().price + " 💰) 📦 " + q);
    }
    items.add("—— 💧 НАПИТКИ ——");
    for (Map.Entry<String, DrinkItem> e : DRINK_ITEMS.entrySet()) {
        int q = inventory.getDefault(e.getKey(), 0);
        items.add(" " + e.getKey() + " (" + e.getValue().price + " 💰) 📦 " + q);
    }
    items.add("—— 🧴 ЛЕКАРСТВА ——");
    for (Map.Entry<String, MedicineItem> e : MEDICINE_ITEMS.entrySet()) {
        int q = inventory.getDefault(e.getKey(), 0);
        items.add(" " + e.getKey() + " (" + e.getValue().price + " 💰) ⚡ Сразу");
    }

    String[] arr = items.toArray(new String[0]);
    shopDialog = new AlertDialog.Builder(this).setTitle("🛒 Магазин")
        .setItems(arr, null)
        .setNeutralButton("📦 Инвентарь", (d, w) -> showInventoryDialog())
        .setNegativeButton("Заккрыть", (d, w) -> shopDialog = null).create();
    shopDialog.show();

    shopDialog.getListView().setOnItemClickListener((p, v, pos, id) -> {
        String sel = arr[pos].trim();
        if (sel.startsWith("——")) return;
        String name = sel.split(" \\(")[0];

        if (FOOD_ITEMS.containsKey(name)) {
            FoodItem f = FOOD_ITEMS.get(name);
            if (!spendCoins(f.price)) { Toast.makeText(this, "❌ Недостаточно 💰",
Toast.LENGTH_SHORT).show(); return; }

```

```

        inventory.put(name, inventory.getDefault(name, 0) + 1);
        refreshShop();
    } else if (DRINK_ITEMS.containsKey(name)) {
        DrinkItem d = DRINK_ITEMS.get(name);
        if (!spendCoins(d.price)) { Toast.makeText(this, "❌ Недостаточно 💰",
Toast.LENGTH_SHORT).show(); return; }
        inventory.put(name, inventory.getDefault(name, 0) + 1);
        refreshShop();
    } else if (MEDICINE_ITEMS.containsKey(name)) {
        MedicineItem m = MEDICINE_ITEMS.get(name);
        if (!spendCoins(m.price)) { Toast.makeText(this, "❌ Недостаточно 💰",
Toast.LENGTH_SHORT).show(); return; }
        applyMedicineImmediately(name, m);
    }
});
}

private void refreshShop() {
    if (shopDialog != null && shopDialog.isShowing()) shopDialog.dismiss();
    openShop();
}

private void showInventoryDialog() {
    if (inventory.isEmpty()) { Toast.makeText(this, "Инвентарь пуст!", Toast.LENGTH_SHORT).show();
return; }
    List<String> items = new ArrayList<>();
    items.add("—— 🍷 ЕДА ——");
    for (Map.Entry<String, Integer> e : inventory.entrySet()) {
        if (e.getValue() > 0 && FOOD_ITEMS.containsKey(e.getKey()))
            items.add(" " + e.getKey() + " × " + e.getValue());
    }
    items.add("—— 💧 НАПИТКИ ——");
    for (Map.Entry<String, Integer> e : inventory.entrySet()) {
        if (e.getValue() > 0 && DRINK_ITEMS.containsKey(e.getKey()))
            items.add(" " + e.getKey() + " × " + e.getValue());
    }
    if (items.size() <= 2) { Toast.makeText(this, "Инвентарь пуст!", Toast.LENGTH_SHORT).show();
return; }
    new AlertDialog.Builder(this).setTitle("📦 Инвентарь")
        .setItems(items.toArray(new String[0]), null).setPositiveButton("OK", null).show();
}

private void showFeedDialog() {
    if (pet == null) return;
    List<String> avail = new ArrayList<>();
    for (Map.Entry<String, Integer> e : inventory.entrySet()) {
        if (e.getValue() > 0 && FOOD_ITEMS.containsKey(e.getKey()))
            avail.add(e.getKey() + " × " + e.getValue());
    }
    if (avail.isEmpty()) {

```

```

        new AlertDialog.Builder(this).setTitle("🍲 Нет еды!")
            .setMessage("Купить в магазине?").setPositiveButton("🛒 В магазин", (d, w) -> openShop())
            .setNegativeButton("Отмена", null).show();
        return;
    }
    new AlertDialog.Builder(this).setTitle("🍲 Чем покормить?")
        .setItems(avail.toArray(new String[0]), (d, w) -> feedPetWithItem(avail.get(w).split(" × ")[0]))
        .setNegativeButton("Отмена", null).show();
}

private void feedPetWithItem(String name) {
    final Pet cp = pet;
    if (cp == null || !FOOD_ITEMS.containsKey(name)) return;
    FoodItem f = FOOD_ITEMS.get(name);
    safeExecute() -> {
        if (cp.getHunger() >= 100) {
            runOnUiThread() -> Toast.makeText(this, "Уже сыт!", Toast.LENGTH_SHORT).show();
            return;
        }
        cp.setHunger(Math.min(100, cp.getHunger() + f.satietyBoost));
        cp.setHappiness(Math.min(100, cp.getHappiness() + f.happinessBoost));
        cp.setHealth(Math.min(100, cp.getHealth() + f.healthBoost));
        cp.setThirst(Math.max(0, cp.getThirst() + f.thirstPenalty));
        inventory.put(name, inventory.get(name) - 1);
        if (inventory.get(name) <= 0) inventory.remove(name);
        repository.update(cp);
        runOnUiThread() -> { updateStats(); addCoins(3); };
    });
}

private void showDrinkDialog() {
    if (pet == null) return;
    List<String> avail = new ArrayList<>();
    for (Map.Entry<String, Integer> e : inventory.entrySet()) {
        if (e.getValue() > 0 && DRINK_ITEMS.containsKey(e.getKey()))
            avail.add(e.getKey() + " × " + e.getValue());
    }
    if (avail.isEmpty()) {
        new AlertDialog.Builder(this).setTitle("💧 Нет напитков!")
            .setMessage("Купить в магазине?").setPositiveButton("🛒 В магазин", (d, w) -> openShop())
            .setNegativeButton("Отмена", null).show();
        return;
    }
    new AlertDialog.Builder(this).setTitle("💧 Чем напоить?")
        .setItems(avail.toArray(new String[0]), (d, w) -> drinkPetWithItem(avail.get(w).split(" × ")[0]))
        .setNegativeButton("Отмена", null).show();
}

private void drinkPetWithItem(String name) {
    final Pet cp = pet;

```

```

if (cp == null || !DRINK_ITEMS.containsKey(name)) return;
DrinkItem d = DRINK_ITEMS.get(name);
safeExecute() -> {
    if (cp.getThirst() >= 100) {
        runOnUiThread() -> Toast.makeText(this, "Не хочет пить!", Toast.LENGTH_SHORT).show();
        return;
    }
    cp.setThirst(Math.min(100, cp.getThirst() + d.hydrationBoost));
    cp.setHappiness(Math.min(100, cp.getHappiness() + d.happinessBoost));
    inventory.put(name, inventory.get(name) - 1);
    if (inventory.get(name) <= 0) inventory.remove(name);
    repository.update(cp);
    runOnUiThread() -> { updateStats(); addCoins(2); });
});
}

```

```

private void applyMedicineImmediately(String name, MedicineItem med) {
    final Pet cp = pet;
    if (cp == null) return;
    safeExecute() -> {
        if (cp.getHealth() >= 100) {
            runOnUiThread() -> Toast.makeText(this, "Уже здоров!", Toast.LENGTH_SHORT).show();
            return;
        }
        int old = cp.getHealth();
        cp.setHealth(Math.min(100, cp.getHealth() + med.healthBoost));
        cp.setHappiness(Math.min(100, cp.getHappiness() + med.happinessBoost));
        repository.update(cp);
        runOnUiThread() -> {
            updateStats();
            Toast.makeText(this, name + "! ❤️ +" + (cp.getHealth() - old), Toast.LENGTH_LONG).show();
        });
    });
}

```

```

private void startHungerTimer() {
    if (pet == null || hungerHandler == null) return;
    hungerHandler.removeCallbacks(hungerTask);

    hungerTask = () -> {
        if (pet == null) return;
        safeExecute() -> {
            pet.setHunger(Math.max(0, pet.getHunger() - 3));
            pet.setThirst(Math.max(0, pet.getThirst() - 4));
            pet.setHappiness(Math.max(0, pet.getHappiness() - 1));

            if (pet.getHunger() <= 10 || pet.getThirst() <= 10) {
                pet.setHealth(Math.max(0, pet.getHealth() - 2));
            }
            repository.update(pet);
            runOnUiThread() -> { updateStats(); checkPetStatus(); });
        }
    };
}

```

```

    });
    hungerHandler.postDelayed(hungerTask, HUNGER_INTERVAL);
};

    hungerHandler.postDelayed(hungerTask, HUNGER_INTERVAL);
}

private void safeExecute(Runnable task) {
    if (executor != null && !executor.isShutdown() && !executor.isTerminated())
        executor.execute(task);
}

private void createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel ch = new NotificationChannel(CHANNEL_ID, "Уход за питомцем",
NotificationManager.IMPORTANCE_HIGH);
        ch.setDescription("Напоминания");
        notificationManager = getSystemService(NotificationManager.class);
        if (notificationManager != null) notificationManager.createNotificationChannel(ch);
    }
}

private void showHungerNotification() {
    if (!notificationsEnabled) return;
    if (notificationManager == null) return;
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU &&
ContextCompat.checkSelfPermission(this, Manifest.permission.POST_NOTIFICATIONS) !=
PackageManager.PERMISSION_GRANTED)
        return;
    NotificationCompat.Builder b = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(android.R.drawable.ic_popup_reminder)
        .setContentTitle("🐾 " + pet.getName() + " нуждается в заботе!")
        .setContentText("Сытость: " + pet.getHunger() + "% | Гидратация: " + pet.getThirst() + "%")
        .setPriority(NotificationCompat.PRIORITY_HIGH).setAutoCancel(true);
    notificationManager.notify(NOTIFICATION_ID, b.build());
}

private void showSettingsDialog() {
    String notifText = notificationsEnabled ? "🔔 Уведомления: Вкл" : "🔕 Уведомления: Выкл";
    String themeText = themeDark ? "🌑 Тема: Тёмная" : "🌞 Тема: Светлая";
    String[] options = { notifText, themeText, "🗑 Сбросить прогресс", "ℹ О приложении" };

    new AlertDialog.Builder(this)
        .setTitle("⚙ Настройки")
        .setItems(options, (dialog, which) -> {
            switch (which) {
                case 0:
                    notificationsEnabled = !notificationsEnabled;
                    saveProgress(); // 🔥 Сохранение
                    Toast.makeText(this, notificationsEnabled ? "🔔 Уведомления включены" : "🔕

```

```

Уведомления выключены", Toast.LENGTH_SHORT).show();
        showSettingsDialog();
        break;
    case 1:
        themeDark = !themeDark;
        saveProgress(); // 🔥 Сохранение
        applyTheme();
        Toast.makeText(this, themeDark ? "🌙 Тёмная тема" : "☀️ Светлая тема",
Toast.LENGTH_SHORT).show();
        recreate();
        break;
    case 2:
        new AlertDialog.Builder(this)
            .setTitle("🗑 Сбросить всё?")
            .setMessage("Все данные питомца и деньги будут удалены!")
            .setPositiveButton("✅ Да, сбросить", (d, w) -> {
                coins = 100; inventory.clear();
                soundEnabled = true; notificationsEnabled = true; themeDark = false;
                saveProgress(); // 🔥 Сохранение чистого состояния
                applyTheme(); updateCoinsDisplay();
                Toast.makeText(this, "Прогресс сброшен!", Toast.LENGTH_LONG).show();
                recreate();
            })
            .setNegativeButton("❌ Отмена", null)
            .show();
        break;
    case 3:
        new AlertDialog.Builder(this)
            .setTitle("ℹ О приложении")
            .setMessage("Tamagotchi v1.0\n\nУхаживайте за питомцем:\n🍖 Кормите, 💧
поите, 🎮 играйте!\n\nРазработано для курсового проекта.")
            .setPositiveButton("OK", null)
            .show();
        break;
    }
    })
    .setNegativeButton("Закреть", null)
    .show();
}

@Override
protected void onResume() {
    super.onResume();
    if (pet != null) {
        startHungerTimer();
    }
}

@Override
protected void onPause() {
    super.onPause();

```

```

        saveProgress();
        if (hungerHandler != null && hungerTask != null) {
            hungerHandler.removeCallbacks(hungerTask);
            hungerTask = null;
        }
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (shopDialog != null && shopDialog.isShowing()) {
            shopDialog.dismiss();
            shopDialog = null;
        }
        if (hungerHandler != null) {
            hungerHandler.removeCallbacksAndMessages(null);
            hungerHandler = null;
        }
        if (executor != null && !executor.isShutdown())
            executor.shutdown();
    }
}

```

MiniGameActivity.java

```

package com.example.kursovoi;

import android.content.Intent;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MiniGameActivity extends AppCompatActivity {

    private TextView tvTimer, tvClicks, tvResult;
    private Button btnClick, btnStart;

    private int clicks = 0;
    private boolean isPlaying = false;
    private static final long GAME_DURATION = 10000;
    private static final long TIMER_INTERVAL = 100;

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

        tvTimer = findViewById(R.id.tvTimer);
        tvClicks = findViewById(R.id.tvClicks);
    }
}

```

```

tvResult = findViewById(R.id.tvResult);
btnClick = findViewById(R.id.btnClick);
btnStart = findViewById(R.id.btnStart);

btnStart.setOnClickListener(v -> startGame());
btnClick.setOnClickListener(v -> onButtonClick());
}

private void startGame() {

    clicks = 0;
    tvClicks.setText("0");
    tvResult.setText("");
    btnStart.setEnabled(false);
    btnClick.setEnabled(true);
    isPlaying = true;

    new CountDownTimer(GAME_DURATION, TIMER_INTERVAL) {
        @Override
        public void onTick(long millisUntilFinished) {
            tvTimer.setText(String.format("%.1f", millisUntilFinished / 1000.0));
        }

        @Override
        public void onFinish() {
            endGame();
        }
    }.start();
}

private void onButtonClick() {
    if (!isPlaying) return;

    clicks++;
    tvClicks.setText(String.valueOf(clicks));

    btnClick.setScaleX(0.95f);
    btnClick.setScaleY(0.95f);
    btnClick.postDelayed(() -> {
        btnClick.setScaleX(1.0f);
        btnClick.setScaleY(1.0f);
    }, 50);
}

private void endGame() {
    isPlaying = false;
    btnClick.setEnabled(false);
    btnStart.setEnabled(true);
    tvTimer.setText("0.0");

    int reward = clicks / 5;

```

```

if (reward < 1 && clicks > 0) reward = 1;

String resultText;
if (clicks == 0) {
    resultText = "😞 Ни одного клика...\nПопробуй ещё раз!";
} else if (clicks < 20) {
    resultText = String.format("👍 Неплохо!\nКликов: %d\n💰 Награда: %d", clicks, reward);
} else if (clicks < 40) {
    resultText = String.format("🎉 Отлично!\nКликов: %d\n💰 Награда: %d", clicks, reward);
} else {
    resultText = String.format("🏆 РЕКОРД!\nКликов: %d\n💰 Награда: %d", clicks, reward);
}

tvResult.setText(resultText);

if (reward > 0) {
    Intent resultIntent = new Intent();
    resultIntent.putExtra("REWARD", reward);
    setResult(RESULT_OK, resultIntent);

    btnStart.postDelayed(() -> finish(), 2000);
} else {
    btnStart.setText("🔄 Ещё раз");
}
}

@Override
protected void onDestroy() {
    super.onDestroy();
    isPlaying = false;
}
}

```

MemoryGameActivity.java

```

package com.example.kursovoi;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.GridLayout;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MemoryGameActivity extends AppCompatActivity {

```

```

private GridLayout gridGame;
private TextView tvMoves;
private Button btnRestart;

private List<Button> buttons = new ArrayList<>();
private List<String> emojis = new ArrayList<>();

private Button firstButton = null;
private Button secondButton = null;
private boolean isClickable = true;
private int matchedPairs = 0;
private int moves = 0;

private final String[] EMOJI_SET = {"🍎", "🍌", "🧊", "🦴", "💧", "🍵", "", "🥕"};

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

    gridGame = findViewById(R.id.gridGame);
    tvMoves = findViewById(R.id.tvMoves);
    btnRestart = findViewById(R.id.btnRestart);

    btnRestart.setOnClickListener(v -> startGame());
    startGame();
}

private void startGame() {
    gridGame.removeAllViews();
    buttons.clear();
    emojis.clear();

    firstButton = null;
    secondButton = null;
    matchedPairs = 0;
    moves = 0;
    isClickable = true;
    updateMovesText();

    for (String emoji : EMOJI_SET) {
        emojis.add(emoji);
        emojis.add(emoji);
    }
    Collections.shuffle(emojis);

    for (String emoji : emojis) {
        Button btn = new Button(this);
        btn.setText("?");
        btn.setTextSize(28);
    }
}

```

```

        btn.setTextColor(0xFF6B7280);
        btn.setBackgroundColor(0xFFE5E7EB);

        GridLayout.LayoutParams params = new GridLayout.LayoutParams();
        params.width = 200;
        params.height = 200;
        params.setMargins(8, 8, 8, 8);
        btn.setLayoutParams(params);
        btn.setTag(emoji);
        btn.setOnClickListener(v -> onCardClick(btn));

        buttons.add(btn);
        gridGame.addView(btn);
    }
}

private void onCardClick(Button clickedBtn) {
    if (!isClickable) return;
    if (clickedBtn == firstButton) return;

    clickedBtn.setText(clickedBtn.getTag().toString());
    clickedBtn.setTextSize(44);
    clickedBtn.setTextColor(0xFF1A1A1A);
    clickedBtn.setBackgroundColor(0xFFFFFFFF);

    if (firstButton == null) {
        firstButton = clickedBtn;
    } else {
        secondButton = clickedBtn;
        moves++;
        updateMovesText();
        checkForMatch();
    }
}

private void checkForMatch() {
    isClickable = false;
    String firstEmoji = firstButton.getTag().toString();
    String secondEmoji = secondButton.getTag().toString();

    if (firstEmoji.equals(secondEmoji)) {
        firstButton.setEnabled(false);
        secondButton.setEnabled(false);
        firstButton.setBackgroundColor(0xFFD1FAE5);
        secondButton.setBackgroundColor(0xFFD1FAE5);
        resetSelection();
        matchedPairs++;

        if (matchedPairs == EMOJI_SET.length) gameWon();
    } else {
        gridGame.postDelayed(() -> {

```

```

        firstButton.setText("?");
        firstButton.setTextSize(28);
        firstButton.setTextColor(0xFF6B7280);
        firstButton.setBackgroundColor(0xFFE5E7EB);

        secondButton.setText("?");
        secondButton.setTextSize(28);
        secondButton.setTextColor(0xFF6B7280);
        secondButton.setBackgroundColor(0xFFE5E7EB);
        resetSelection();
    }, 800);
}
}

private void resetSelection() {
    firstButton = null;
    secondButton = null;
    isClickable = true;
}

private void updateMovesText() {
    tvMoves.setText("Ходы: " + moves);
}

private void gameWon() {
    int reward = Math.max(5, 50 - (moves - 8));
    Toast.makeText(this, "🏆 Победа! Награда: " + reward + " монет", Toast.LENGTH_LONG).show();

    Intent resultIntent = new Intent();
    resultIntent.putExtra("REWARD", reward);
    setResult(RESULT_OK, resultIntent);
    finish();
}
}

```