Муромский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет Информационных технологий и радиоэлектроники
Кафедра Программной инженерии

КУРСОВАЯ РАБОТА

по Разработке приложений для мобильных операционных систем
(наименование дисциплины)

Тема Приложение-компаньон для настольной игры «Мафия»

Руководитель

_____ Мортин К. В.
(оценка) (фамилия, инициалы)

_____
(подпись) (дата)

Члены комиссии

Студент ПИн-119
(группа)

_____ Емельянов В. А.
(подпись) (Ф.И.О.) (фамилия, инициалы)

_____ _____
(подпись) (Ф.И.О.) (подпись) (дата)

Муром 2022

В данной курсовой работе разработано приложение-компаньон для настольной игры «Мафия». В ходе выполнения курсовой работы произведен анализ и сбор требований к разрабатываемой программе. При разработке применены языки Kotlin и Java, Android SDK, реляционная система управления базами данных SQLite и библиотека Room. Приложение протестировано и полностью работоспособно. Полученный программный продукт можно применять по его прямому назначению.

In this course work, the companion app for board game “Mafia” has been developed. In the process of creating a course work, the collection and analysis of the requirements for the projected program was made. The development used the Kotlin and the Java languages, Android SDK, the SQLite relational database management system and the Room library. The program has been tested and fully functional. The resulting product can be used for its intended purpose.

Содержание

Введение 6
1 Анализ технического задания 7
2 Разработка алгоритмов 8
2.1 Разработка моделей данных 8
2.1.1 Концептуальная модель 8
2.1.2 Логическая модель 9
2.1.3 Физическая модель 9
2.2 Описание различных элементов программы 10
3 Руководство программиста 14
4 Руководство пользователя 18
Заключение 26
Список используемой литературы 27
Приложение 1. Текст программы 28
Приложение 2. Снимки окон программы (скриншоты программы) 29

Введение

В современном мире смартфоны стали незаменимыми помощниками в повседневной жизни людей. С их помощью мы можем постоянно находиться на связи, получать любую необходимую информацию в доли секунды, выполнять различные рабочие задачи и т. д. И, так как телефоны покрывают достаточно большую часть наших потребностей, то они способны удовлетворить и потребность в досуге.

Темой курсовой работы является разработка приложения-компаньона для ведущего настольной игры «Мафия».

Приложения-компаньоны позволяют использующему их человеку облегчить тот или иной процесс, беря на себя часть функций. Так, например, телефон лучше справится с запоминанием информации, точным слежением за временем и контролем соблюдения правил. Также данное приложение имеет смысл реализовывать только для мобильной платформы, т. к. благодаря переносимости устройства оно будет наиболее удобным в использовании.

Таким образом, целью курсовой работы является создание приложения, предоставляющего своему пользователю удобный инструментарий для проведения игровой сессии, а также возможности хранения их результатов для вероятного использования в будущем, например, для контроля при проведении турниров. Главной задачей становится качественная реализация вышеперечисленного.

1 Анализ технического задания

Целью данной курсовой работы является создание приложения-компаньона для настольной игры «Мафия», следовательно, разработка должна обеспечивать следующие функциональные возможности:

– проведение игровой сессии по официальным правилам;

– возможность добавления и сохранения постоянных игроков;

– функционал для просмотра истории проведённых игр;

– всю информацию сохранять в базе данных.

Из этих требований можно сделать несколько выводов.

Во-первых, приложение, как компаньон, будет не заменять ведущего, а лишь помогать ему в проведении игровой сессии. Следовательно, интерфейс должен быть прост и понятен, а логика программы привязана к логике происходящего во время игры, чтобы не допустить критических ошибок со стороны человека.

Во-вторых, информация, связанная с непосредственным использованием программы должна храниться в базе данных. В то же время, программа является строго однопользовательской, так что для её реализации не потребуются какие-либо сетевые функции.

Таким образом, мы приходим к выводу, что для хранения будет очень удобно использовать несколько различных связанных между собой таблиц. Это приводит нас к реляционным СУБД. Лучшим выбором для разработки будет являться ПО SQLite. Эта система поддерживает все необходимые для реализации вышеперечисленного функции, является бесплатной, а также обладает хорошим быстродействием при малом размере и общей простоте своего кода. Кроме того, она является встраиваемой, что дополнительно облегчает работу с ней при отсутствии необходимости поддержки многопользовательских функций, и по умолчанию включена в Android SDK.

Приложение будет разработано на языках Kotlin и Java для операционной системы Android. Это позволит обеспечить как максимальное удобство для пользователя, так и максимальную производительность на устройстве.

2 Разработка алгоритмов

2.1 Разработка моделей данных

2.1.1 Концептуальная модель

Так как в программе будет использоваться база данных, перед тем как приступить к разработке, необходимо составить её проект. Первым делом составляется концептуальная модель. Для этого необходимо выделить сущности, которые будут встречаться в проектируемой программе, их атрибуты, а также установить связи между ними.

Базовой сущностью будет являться Игра. Её атрибутами будут Дата и время и Результат.

Игры будут сохраняться в Истории. История должна хранить в себе Номер раунда и Действие.

В играх будут участвовать Игроки. Они характеризуются Псевдонимом и опциональными Сведениями. Связь между Игроками и Игрой должна хранить Роль игрока. Кроме того, Игрок как Объект действия будет храниться в Истории.

В результате была подготовлена полная концептуальная модель. Она показана на рисунке 1.

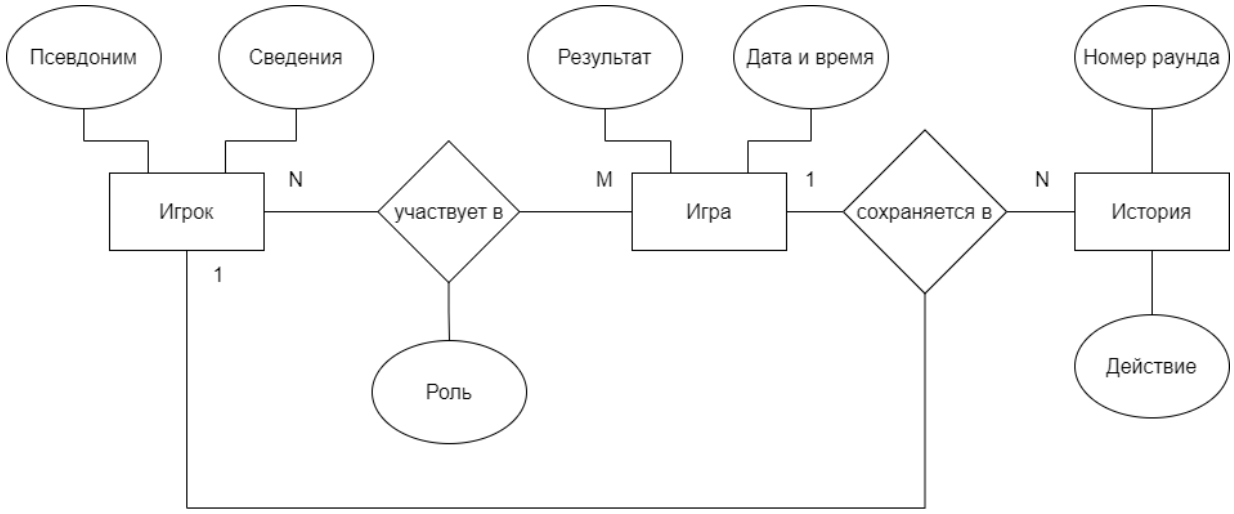

Рис. 1. Концептуальная модель

2.1.2 Логическая модель

На основе концептуальной модели данных можно построить логическую модель. Но первым шагом будет нормализация таблиц, полученных на предыдущем этапе. Для соответствия второй нормальной форме в каждую таблицу, отражающую сущность, были добавлены идентификаторы, а в каждую таблицу, отражающую связь, дополнительно, идентификаторы связанных таблиц. В остальном все таблицы изначально были созданы с учётом требований всех нормальных форм вплоть до четвёртой включительно.

Для создания логической модели данных необходимо отобразить все сущности и связи, значимые для заданной предметной области и введённые в предыдущей схеме. Модель должна отражать внутреннее устройство базы данных. Результат проектирования можно увидеть на рисунке 2.

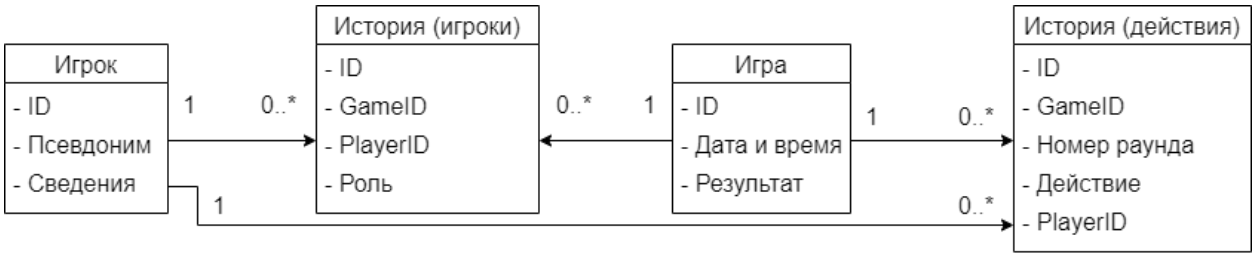

Рис. 2. Логическая модель

2.1.3 Физическая модель

Из логической модели данных напрямую произрастает физическая. Она должна увязать созданную логику и возможности СУБД SQLite. Именно здесь происходит выбор внутренних названий таблиц и полей, а также типов используемых данных. Разработанная модель отображена на рисунке 3. Курсивом на схеме выделены ключевые поля таблиц.

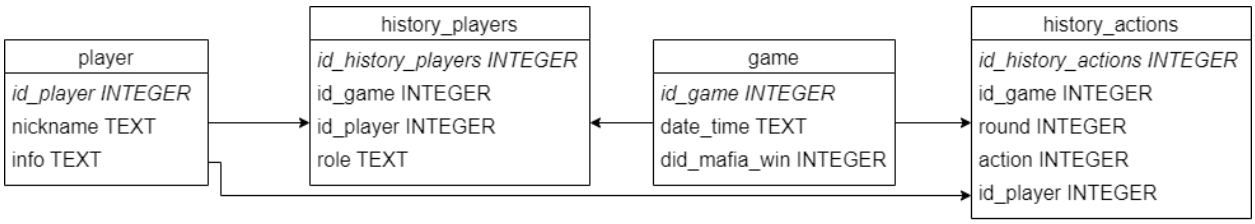

Рис. 3. Физическая модель

2.2 Описание различных элементов программы

Для отслеживания времени используется компонент CountDownTimer. Для удобства использования были написаны дополнительные функции, которые позволяют ставить таймер на паузу или целиком останавливать его при необходимости. Кроме того, таймер сохраняет свои значения между фрагментами и во время дня не переключается на следующего игрока, если пользователь находится на другом фрагменте, во избежание потери контроля за игровым процессом.

На всех экранах перегружена функция возвращения по системной кнопке «Назад» для предотвращения неправильного функционирования программы. Так, на большинстве игровых экранов по нажатию всплывает уведомление о прекращении игры при продолжении (без этого происходил переход на предыдущий фрагмент, что могло нарушить порядок игровых действий), а на экране, оповещающем о конце игры функция отключена, чтобы пользователь не мог выйти из игры, не сохранив её результат. Например, данный функционал реализован на фрагменте создания новой игры следующим образом:

```
requireActivity().onBackPressedDispatcher.addCallback(this) {
    if (names.size>0){
        val builder = AlertDialog.Builder(context,
               R.style.AlertDialogStyle)

        builder.setTitle("Вы уверены?")
        builder.setMessage("Если вернуться, все добавленные
               игроки будут исключены из игры")
        builder.setPositiveButton("Да") { _, _ ->
               findNavController().navigate(
               R.id.action_AddPlayers_to_MainMenu) }
```

```
            builder.setNegativeButton("Нет") { dialog, _ ->
                    dialog.dismiss() }

            builder.show()
        }
        else{
            findNavController().navigate(
                    R.id.action_AddPlayers_to_MainMenu)
        }
    }
```

Данный фрагмент кода вызывается при создании фрагмента и перегружает встроенный метод onBackPressedDispatcher. Внутри происходит проверка на наличие добавленных игроков и, если таковые обнаружены, то выводится предупреждение о том, что список не сохранится. В противном случае переход происходит автоматически.

Приложение зафиксировано в портретном режиме, так как именно данный режим обеспечивает наибольшее удобство при использовании (например, больше элементов списка помещаются на экране).

При помощи интерфейсов ItemTouchHelperAdapter, OnStartDragListener и класса SimpleItemTouchHelperCallback на экране новой игры реализовано перемещение игроков в списке, созданном при помощи компонента RecyclerView, путём перетаскивания за специальный объект или долгого нажатия на весь элемент. При перетаскивании номера динамически меняются.

Для облегчения работы с базой данных используется библиотека Room. Она позволяет работать с объектами базы данных как с сущностями, а запросы реализовывать при помощи интерфейсов доступа к данным (также известных как DAO – Data Access Object), что заметно упрощает разработку и снижает вероятность ошибки. Инициализация базы данных реализована следующим образом:

```
@Database(entities = [Player::class, Game::class,
     HistoryPlayers::class, HistoryActions::class], version = 9)
abstract class AppDatabase : RoomDatabase() {
    abstract fun playerDao(): PlayerDao
    abstract fun gameDao(): GameDao
    abstract fun historyPlayersDao(): HistoryPlayersDao
```

```
    abstract fun historyActionsDao(): HistoryActionsDao

    companion object{
        private var db : AppDatabase? = null

        fun getDB(applicationContext: Context) : AppDatabase{
            if (db == null){
                db = Room.databaseBuilder(
                    applicationContext,
                    AppDatabase::class.java, "mafia-db"
                ).allowMainThreadQueries()
                .build()
            }

            return db as AppDatabase
        }
    }
}
```

В аннотации @Database происходит перечисление использующихся сущностей и текущего номера миграции. Далее, внутри класса объявляются переменные для доступа к DAO-объектам и, внутри контейнера для статических объектов, создаётся объект базы данных и функция для подключения к ней, если оно ещё не произошло, а также получения доступа к этому самому объекту. Так как приложение является однопользовательским, то постоянное подключение не повредит работоспособности, зато даст экономию ресурсов при повторном обращении при небольшом количестве занятых хранением ресурсов. Функция для закрытия не требуется, так как библиотека Room выполняет данное действие сама при закрытии приложения.

Обращение к базе данных происходит, например, так:

```
val db = AppDatabase.getDB(requireActivity().applicationContext)
for (player in db.playerDao().getAll()){
    names.add(player.nickname)
    playersDBList.adapter?.notifyItemInserted(names.size)
}
```

В приведённом выше фрагменте кода выполняется получение объекта базы данных, после чего, через обращение к методу DAO-объекта сущности Player,

получаются все записи из одноимённой таблицы и добавляются в компонент RecyclerView.

Добавление происходит путём создания нового объекта класса:

```
db.playerDao().insert(Player(0, newName, null))
```

Первым параметром передаётся идентификатор и, так как в классе данных (или классе сущности) он помечен аннотацией @PrimaryKey(autoGenerate = true), которая применяет к нему автогенерацию, то при создании он может быть инициализирован любым значением, например, нулём.

В интерфейсе DAO простые функции, такие как добавление, изменение или удаление могут быть объявлены при помощи соответствующей простой аннотации. Для реализации более сложных запросов необходимо написать следующее:

```
@Query("SELECT * FROM player WHERE nickname LIKE :nickname
     LIMIT 1")
fun findByNickname(nickname: String): Player
```

Так, данный фрагмент кода реализует поиск игрока по его псевдониму при помощи SQL-запроса, указанного в аннотации @Query.

Псевдонимы игроков уникальны. Данные о любом игроке могут быть удалены, но только в том случае, если он не участвовал ни в одной игре.

Реализация данных (и других) алгоритмов может быть найдена в полном коде программы, расположенном в Приложении 1.

3 Руководство программиста

Приложение создано на языках Kotlin и Java с использованием Android SDK. В качестве СУБД используется SQLite, для работы с ней использована библиотека Room.

Приложение, в соответствии с рекомендациями по разработке от корпорации Google, использует систему фрагментов. Все фрагменты отображаются на Activity с названием MainActivity. Также стоит отметить, что, как для Activity, так и для каждого фрагмента, существует одноимённый файл разметки, содержащий в себе описание интерфейса. Перечень фрагментов приведён в таблице 1.

Таблица 1

Перечень фрагментов

Имя фрагмента	Краткое описание
AddPlayers	Позволяет добавлять игроков к игре и запускать её
Day	Содержит реализацию основной логики происходящего днём, а также компонент для переключения между тремя нижеописанными фрагментами
DayDead	Содержит в себе список выбывших из игры игроков
DayPlayers	Содержит в себе список находящихся в игре игроков, а также таймеры их речей
DayVote	Позволяет выставлять игроков в список для голосования, а также содержит в себе этот список
EndGame	Отображает результат игры, позволяет просмотреть список игроков, записывает игру в историю
GameInfo	Отображает информацию из базы данных об игре
GetRole	Реализует раздачу случайных ролей игрокам
History	Показывает все игры, хранящиеся в базе данных

Таблица 1 (продолжение)

Имя фрагмента	Краткое описание
LastWords	Отображает имя выбывшего игрока и таймер его последней речи
MainMenu	Реализует работу главного меню
Night	Содержит реализацию логики происходящего ночью
PlayerInfo	Отображает информацию из базы данных об игроке
Players	Показывает всех игроков, хранящихся в базе данных, позволяет добавить нового
StartDayOrNight	Отображает информацию о начале дня или ночи, реализует корректные переходы между игровыми фазами

Для отображения списков используется компонент RecyclerView. Для его работы требуется создание отдельных классов – адаптеров и одноимённых файлов разметки (без постфикса «Adapter») с отдельными элементами этих списков. Их перечень приведён в таблице 2.

Таблица 2

Перечень адаптеров

Имя адаптера	Краткое описание
RecyclerAddPlayersAdapter	Список игроков для начала игры
RecyclerDBAdapter	Список информации из базы данных (используется для игроков и истории)
RecyclerDeadPlayersAdapter	Список выбывших игроков
RecyclerEveryPlayersAdapter	Справочный список всех игроков
RecyclerGameActionAdapter	Список действий, совершённых в игре
RecyclerPlayersAdapter	Список не выбывших игроков
RecyclerVotePlayersAdapter	Список игроков для голосования

Работа с базой данных посредством библиотеки Room требует создания классов сущностей, содержащих перечисления полей и их типов, интерфейсов Data Access Object, в которых указаны необходимые методы и SQL-запросы для реализации доступа к данным, и основного класса базы данных. Таким классом является класс AppDatabase, содержащий в себе подключение к самой базе данных, а также объявление переменных для работы с ней; перечень сущностей приведён в таблице 3, интерфейсы DAO обладают схожим названием, оканчивающимся на «Dao».

Таблица 3

Перечень сущностей

Имя сущности	Краткое описание
Game	Игра (дата и время проведения, результат)
HistoryActions	История действий (какое действие, в какой игре совершено, в каком раунде совершено, на какого игрока направлено)
HistoryPlayers	История игроков (какой игрок, в какой игре, какая роль)
Player	Игрок (псевдоним, дополнительная информация (может принимать значение «null»))

Помимо перечисленного выше, были созданы различные вспомогательные классы и файлы разметки. Их перечень приведён в таблицах 4 и 5.

Таблица 4

Перечень вспомогательных классов

Имя класса	Краткое описание
ItemTouchHelperAdapter	Интерфейс, используется для реализации перемещения элементов в списке на фрагменте AddPlayers, определяет функции для отслеживания факта перемещения
OnStartDragListener	Интерфейс, используется для реализации перемещения элементов в списке на фрагменте AddPlayers, определяет функцию для отслеживания начала перемещения
SimpleItemTouch HelperCallback	Класс, используется для реализации перемещения элементов в списке на фрагменте AddPlayers, определяет различные вспомогательные функции

Таблица 5

Перечень вспомогательных файлов разметки

Имя файла	Краткое описание
add_temporary_player	Содержит элемент для ввода имени нового игрока в диалоговом окне
content_main	Используется для поддержки навигации по фрагментам
show_players_list	Содержит список для отображения в диалоговом окне с показом всех игроков

Программа является полностью функциональной, однако при желании может быть доработана. Например, может быть добавлена поддержка замечаний для игроков. Возможно расширение функционала программы для поддержки не только классических правил, но и любых других вариаций. Также полезным будет портирование на iOS для большего охвата возможных пользователей.

4 Руководство пользователя

Минимальные системные требования: для запуска приложения потребуется смартфон с ОС Android версии не ниже 5.0.

На рисунке 4 показан внешний вид приложения после запуска.

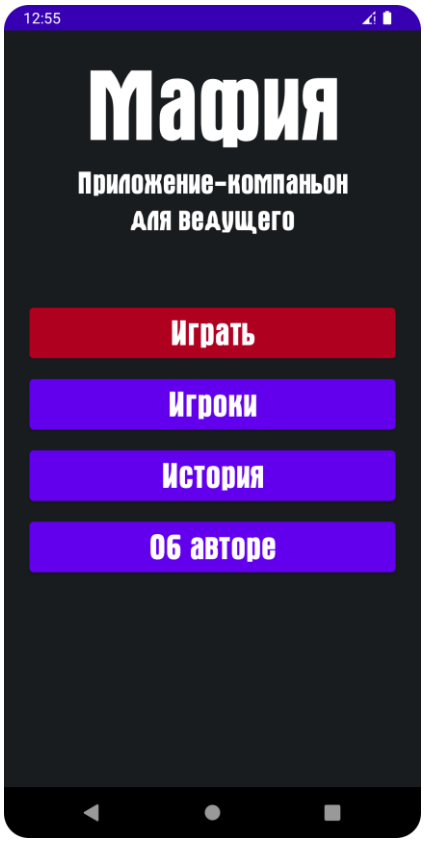

Рис. 4. Запущенное приложение

Перед началом первой игры необходимо добавить не менее 10 уникальных игроков в пункте меню «Игроки». После нажатия на соответствующую кнопку будет отображён интерфейс, показанный на рисунке 5.

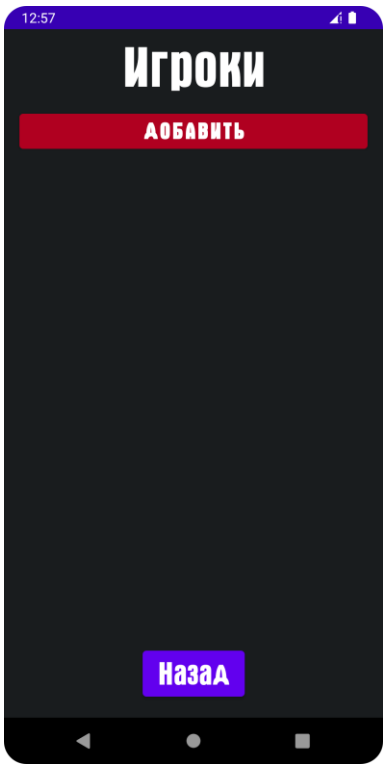

Рис. 5. Меню «Игроки»

При нажатии на кнопку «Добавить» будет отображено диалоговое окно с полем для ввода текста. Поддерживается ввод не более 25 символов русского и английского алфавитов, цифр и символа пробела. После нажатия на кнопку «ОК» игрок будет сохранён, а его имя появится в списке. Это можно увидеть на рисунке 6.

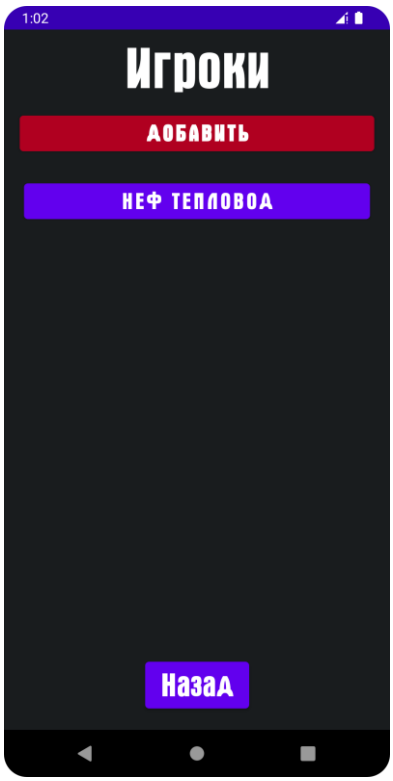

Рис. 6. Меню «Игроки» с добавленным игроком

После этого можно заходить в меню «Играть». Там, по кнопке «Добавить», необходимо прикрепить к игре ранее созданных игроков, после чего, по кнопке «Далее» запустить игру. Начнётся процесс раздачи случайных ролей игрокам. Данный экран, в отличии от остальных, можно показывать тем, кто сидит за столом. Его внешний вид показан на рисунке 7.

Рис. 7. Игрок получает роль

После раздачи ролей наступает нулевая ночь, в ходе которой мафия должна познакомиться друг с другом, а дон и шериф могут осмотреть город.

На рисунке 8 показан экран дня. На нём имеется три подменю – Игроки, Голосование и Выбывшие игроки. На первом экране отображается список игроков, который включает в себя их номера, роли, псевдонимы, таймеры их речей и кнопки для его остановки. Помимо этого, на каждом из трёх экранов над компонентом переключения между ними имеется кнопка для постановки таймера на паузу/продолжения его работы. Значения ролей следующие: «Ж» – Мирный житель, «Ш» – Шериф, «М» – Мафия, «Д» – Дон мафии.

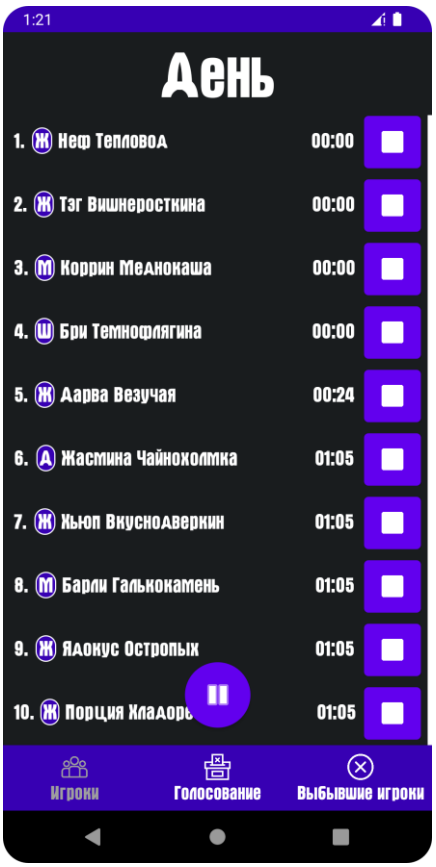

Рис. 8. День, игроки

На вкладке Голосование, показанной на рисунке 9, присутствует возможность добавить выставленного игрока в список для голосования. После завершения всех речей требуется нажать на кнопку, ранее отвечавшую за работу с таймером. Если игрок один, то он автоматически считается выбывшим по голосованию (если это не происходит в нулевой день). В ином случае присутствует возможность дать игрокам дополнительное время для речи, если голоса распределились поровну, или сразу перейти к вводу результатов голосования. После голосования выбывшим игрокам даётся право последнего слова, после чего наступает ночь.

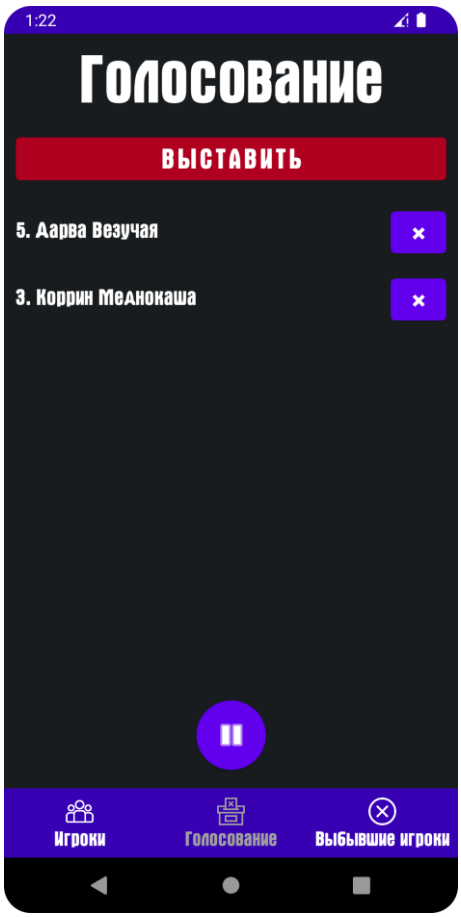

Рис. 9. День, голосование

Ночью последовательно ходят мафия, дон мафии и шериф. После хода мафии необходимо ввести результаты попытки отстрела. Так как мафия имеет возможность промахнуться, то помимо игроков в списке присутствует пункт «Промах». Данный функционал показан на рисунке 10. После ввода появляется возможность перейти к следующему действию.

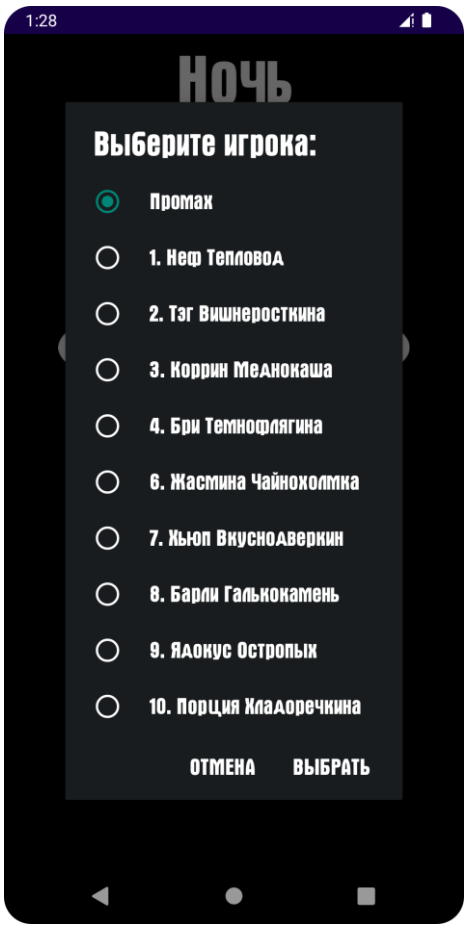

Рис. 10. Ввод результата действий мафии

Во время хода дона и шерифа они пытаются разыскать членов противоборствующей команды. Для этого по их запросу необходимо открыть список игроков по кнопке «Поиск» и, при помощи установленных символов, сообщить им результат проверки. В качестве справочной информации живые игроки выделяются зелёным цветом. Это можно увидеть на рисунке 11.

Рис. 11. Список игроков во время хода шерифа

После ночи наступает день, игра продолжается по ранее описанному принципу.

В конце игры выводится экран с указанием победившей команды и возможностью просмотра списка игроков, который аналогичен списку, показываемому ночью во время хода дона и шерифа. По нажатию на кнопку «Завершить» происходит возврат в главное меню.

После завершения игра отображается на экране «История». На данном экране можно увидеть список участвовавших игроков, а также совершённые во время игры действия.

Помимо добавления игрока, о нём можно добавить какую-либо заметку, а также поменять его псевдоним при необходимости. Для этого в меню «Игроки» необходимо нажать на кнопку с именем изменяемого игрока, после чего нажать на кнопку изменения необходимой информации и сохранить её. Имена игроков должны быть уникальны, в то время как дополнительные сведения могут содержать любой текст. Также возможно удаление игрока, не участвовавшего ни в одной игре.

Заключение

В ходе данной курсовой работы была выполнена разработка приложения-компаньона для настольной игры «Мафия».

В ходе выполнения были решены следующие задачи:

– проведение игровой сессии по официальным правилам;

– возможность добавления и сохранения постоянных игроков;

– функционал для просмотра истории проведённых игр;

– всю информацию сохранять в базе данных.

Разработанная программа может быть усовершенствована. Однако это не обязательно, ведь на выходе имеется полностью рабочий продукт, который уже можно использовать.

Таким образом, в курсовой работе были реализованы все пункты технического задания.

Список используемой литературы

1. Android: Теория: сайт. – URL: http://developer.alexanderklimov.ru/android/theory/ (дата обращения: 16.09.2022). – Текст: электронный.

2. Раздел языка Kotlin на Metanit.com: сайт. – URL: https://metanit.com/kotlin/ (дата обращения: 20.10.2022). – Текст: электронный.

3. Documentation | Android Developers: сайт. – URL: https://developer.android.com/docs (дата обращения: 09.11.2022). – Текст: электронный.

4. Stack Overflow: сайт. – URL: https://stackoverflow.com/ (дата обращения: 12.12.2022). – Текст: электронный.

Приложение 1. Текст программы

Текст программы доступен по ссылке:
https://github.com/FireBoT-er/MafiaCompanion

Приложение 2. Снимки окон программы (скриншоты программы)

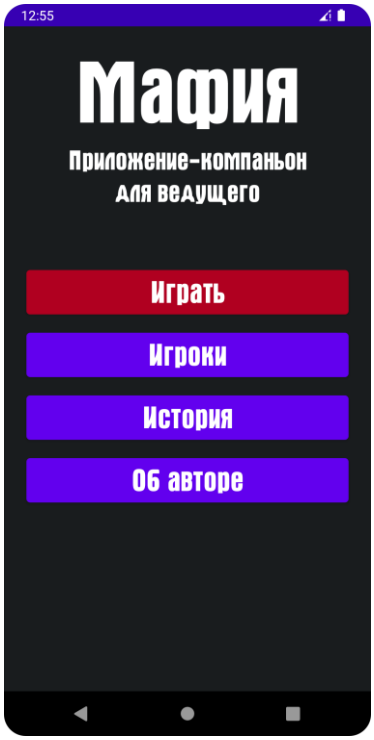

Рис. 1. Главное окно приложения после запуска

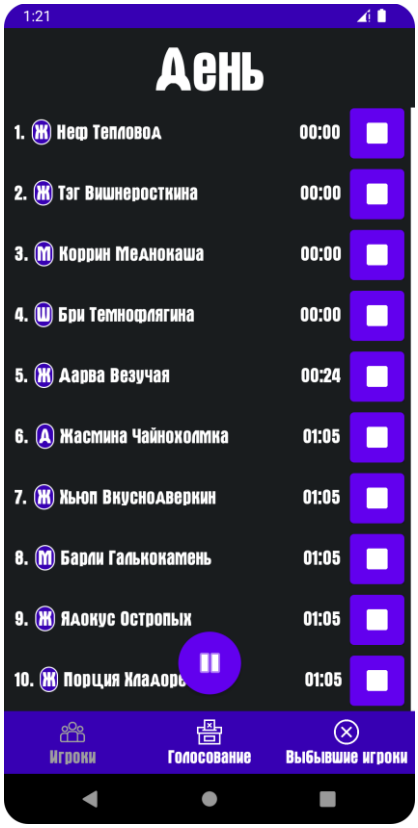

Рис. 2. День, игроки

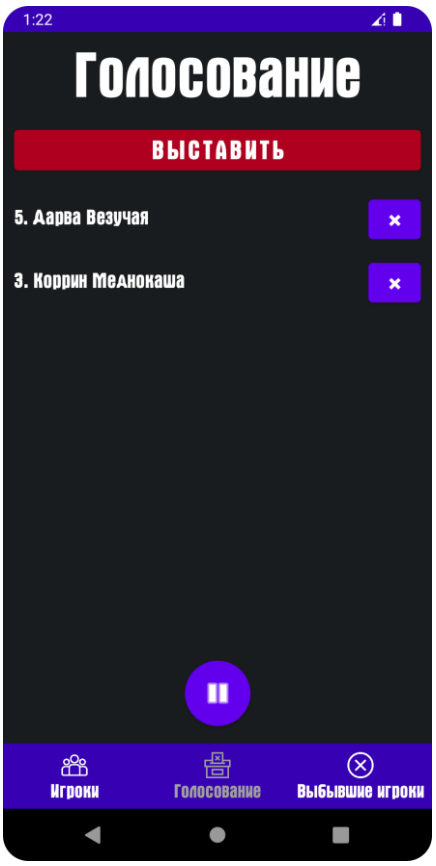

Рис. 3. День, голосование

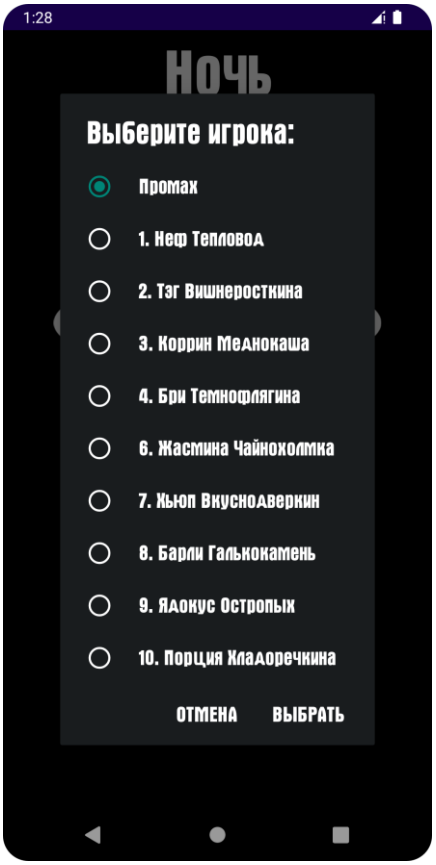

Рис. 4. Ночь, ввод результата действий мафии

Рис. 5. Ночь, список игроков во время хода шерифа

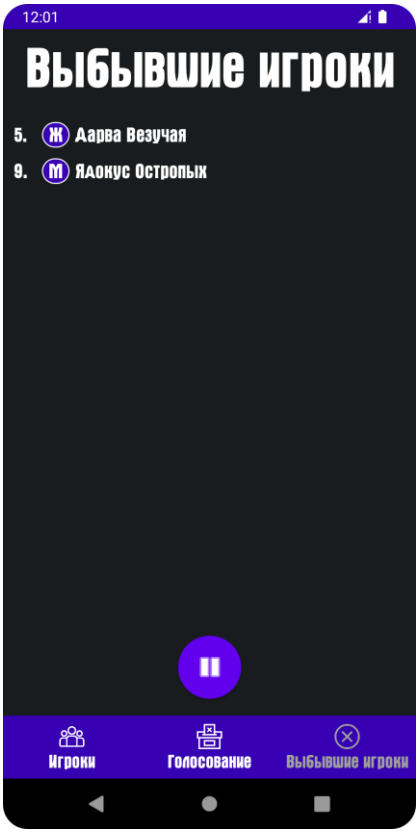

Рис. 6. День, выбывшие игроки

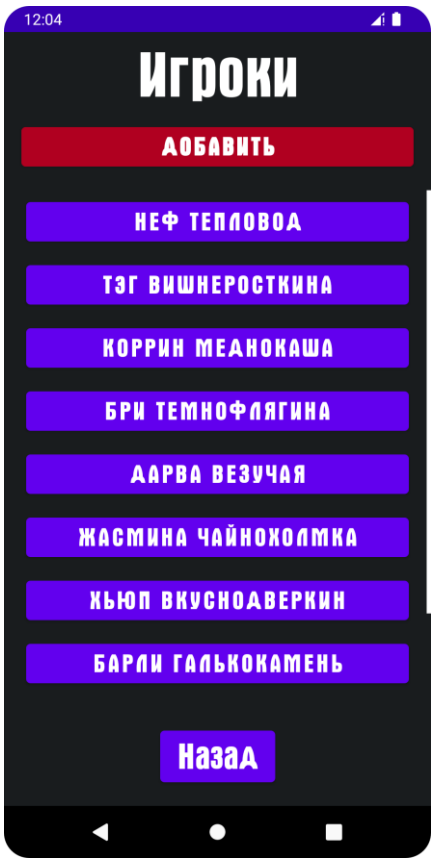

Рис. 7. Меню «Игроки»

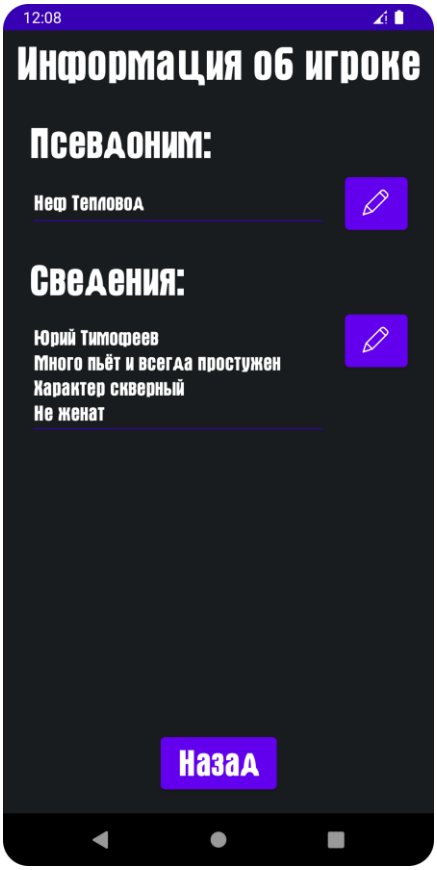

Рис. 8. Информация об игроке

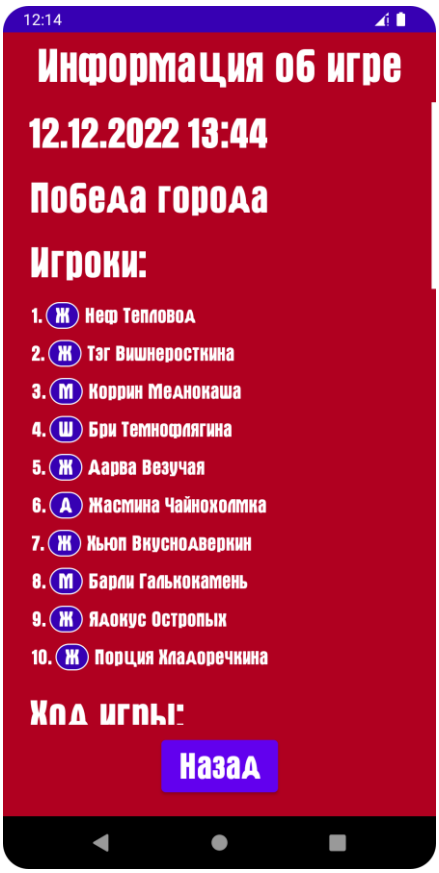

Рис. 9. Информация об игре, начало

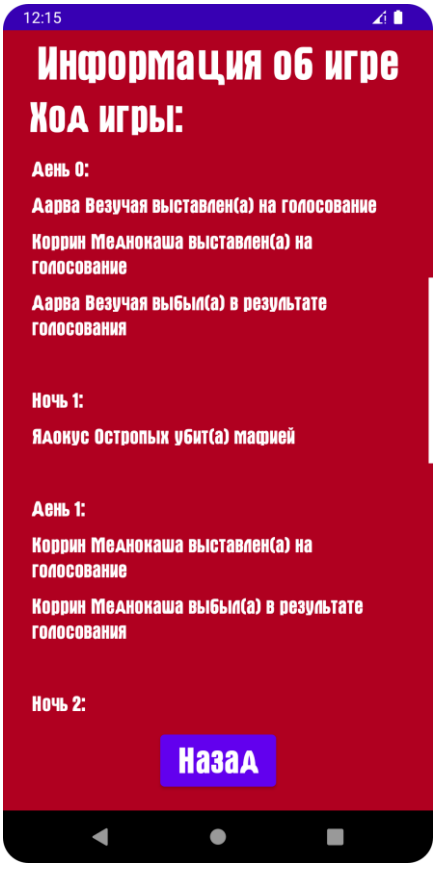

Рис. 10. Информация об игре, ход игры