

Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра алгоритмических языков

Вербин Олег

**Программная реализация обучающей игры на базе беспроводной локальной сети**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**Научный руководитель:**

м.н.с. Н.В.Баева

Москва, 2021

Оглавление

[2 Введение 3](#_Toc70891546)

[3 Обзорная глава 4](#_Toc70891547)

[4 Постановка задачи 5](#_Toc70891548)

[5 Описание программы 6](#_Toc70891549)

[5.1 Обучающее приложение 6](#_Toc70891550)

[5.2 Конструктор сценариев 7](#_Toc70891551)

[6 Реализация конструктора 10](#_Toc70891552)

[6.1 Инструменты разработки 10](#_Toc70891553)

[6.2 Описание библиотеки Tkinter 10](#_Toc70891554)

[6.3 Сохранение сценария 11](#_Toc70891555)

[6.4 Обратная польская запись 11](#_Toc70891556)

[7 Реализация обучающей игры 13](#_Toc70891557)

[7.1 Инструменты разработки 13](#_Toc70891558)

[7.2 Обзор Android Studio 13](#_Toc70891559)

[7.3 Обзор WiFi P2P 14](#_Toc70891560)

[7.4 Сокеты, handler и looper 16](#_Toc70891561)

[7.5 Структура классов и потоков 17](#_Toc70891562)

[7.6 Формат передачи данных 18](#_Toc70891563)

[7.7 Передача действий 18](#_Toc70891564)

[8 Результаты работы 20](#_Toc70891565)

[8.1 Итоговый конструктор 20](#_Toc70891566)

[8.2 Итоговое мобильное приложение 22](#_Toc70891567)

[8.3 Итог 28](#_Toc70891568)

[9 Список литературы 29](#_Toc70891569)

[Приложение A. Формат данных при сохранении сценария и передачи данных 30](#_Toc70891570)

[Приложение B. Жизненный цикл Activity 32](#_Toc70891571)

[Приложение C. Примеры обучающих игр 33](#_Toc70891572)

# Введение

В современном мире спрос на обучающие программы только растёт [3]. Это связано с тем, что скорость жизни значительно возросла, и количество окружающей нас информации повысилось в разы. Люди больше не хотят долгое время сидеть за чтением книг - для обретения нужных навыков им значительно проще получать новые знания в игровой форме.

В данной выпускной квалификационной работе рассмотрено насколько обучающие приложения на самом деле могут помогать в обучении, а также описаны некоторые примеры существующих обучающих игр.

Основная часть работы была посвящена разработке кроссплатформенного компьютерного приложения, для создания множества различных шаблонов пошаговых сетевых игр, и мобильного приложения под Android, для реализации выбранного шаблона. Был придуман гибкий способ задания шаблона, и его дальнейшего вычисления, который основан на обратной польской записи [10]

Для разработки приложений были использованы языки Java[7] и Python[5], и среды разработки Android Studio[8] и PyCharm[11].

Также в рамках работы были рассмотрены методы создания беспроводной связи на близкой дистанции и описана библиотека под Android для работы с технологией WiFi Direct[9]

Дополнительно в работе происходит краткий обзор графической библиотеки Tkinter[6] и Android Studio, а также описано каким образом можно организовывать меж-потоковое взаимодействие в Android Studio[12].

# Обзорная глава

«Игра практически с древних времён выступает как форма обучения».

Ян Коменский

Все дети любят играть. Потребность в игре одна из базовых потребностей человека. Многие считают, что игра — это прерогатива детей дошкольного возраста. Это не совсем верно. Исчезновение игры в школьном возрасте происходит не от того, что дети не хотят играть, а скорее всего из-за отсутствия объективных возможностей её осуществления. Конечно же потребность в игре сохраняется на протяжении всей жизни. Как маме легче уговорить ребёнка выполнить то или иное поручение, прибегнув к игре, так и преподавателю эффективнее и проще преподнести любой материал, используя игровой момент. Обучающая игра выступает как средство побуждения, стимулирования и реализации поставленной задачи в игровой форме.

Но насколько игра может быть эффективной в обучении? Масштабные исследования в середине 2000-х доказали: многопользовательские онлайн-игры – это эффективный тренажер для отработки навыков управления людьми [1]. В процессе других исследований [2] было установлено, что участники игры «Мафия» отличаются от своих сверстников более развитыми коммуникативными и интеллектуальными способностями, у них выше лидерский потенциал, они в большей степени склонны доверять людям и не беспокоиться из-за мелочей. Также при возникновении трудностей они предпочитают искать новые пути их решения, а не возвращаться к проверенным способам. Итог: игры – учат. Поэтому можно говорить об игре, как об **обучающем средстве**.

Стоит помнить, что игры развиваются вместе с обществом - у каждого поколения свои игры. Сейчас, когда произошла цифровизация общества - игры также перешли в цифровой вид. Мобильные и компьютерные приложения являются мощным средством повышения разносторонности личности и значительно расширяют возможности человека по реализации собственных идей путем моделирования различных ситуаций в игровой среде. Пример современных обучающих игр можно посмотреть в **Приложении C**.

Согласно исследованиям рынка технологий [3], в 2018 году потребители потратили примерно $1,9 млрд на приложения для тренировки мозга, такие как Lumosity, Peak и Elevate, — в четыре раза больше по сравнению с $475 млн в 2012 году. Это говорит о том, **что спрос на обучающие игры с каждым годом только растёт**.

# Постановка задачи

На данный момент на биологическом факультете существует обучающая игра по симуляции инфекционного заболевания. Каждому игроку выдаётся роль и сообщается, что в больницу поступило несколько пациентов со странными симптомами. С этого момента задача игроков – коммуницируя друг с другом, определить и вылечить болезнь, и сделать это с наименьшими потерями. Игра является пошаговой, и с каждым ходом количество больных увеличивается, в зависимости от действий игроков. В конце каждого хода игроки должны, пообщавшись друг с другом, определить какое следующее действие они предпримут, при этом один игрок в начале игры назначается лидером, и право последнего голоса остаётся за ним. Также игрокам доступна дополнительная информация, которая со временем может изменяться. В игре участвует ведущий, который может устраивать случайные события и давать подсказки.

**Необходимо разработать приложение и придумать его внутреннее и внешнее представление таким образом, чтобы оно было способно обрабатывать множество различных сценариев, и, чтобы среди множества принимаемых сценариев был вышеописанный.**

**Дополнительно необходимо создать конструктор сценариев, в котором можно задавать различные игры. Конструктор должен обладать высокой выразительностью, чтобы на нём можно было создать большое множество различных игр, включая вышеописанную игру. При этом конструктор должен быть достаточно простым, чтобы даже неопытный пользователь смог им воспользоваться.**

# Описание программы

## Обучающее приложение

Необходимо создать приложение, которое будет реализовывать пошаговую сетевую игру, описанную в сценарии. В игре принимает участие N человек. Каждому игроку присвоена некоторая роль. Всегда существуют роли Хоста и Лидера.

* «Хост» - игрок, создавший игру, обладающий возможностями редактировать любые параметры в игре.
* «Лидер» - все действия других игроков отправляются лидеру, и он выбирает какое из них отправить Хосту.

Для каждой роли доступны свои возможные действия. Раунд длится T минут, и в течении раунда игроки должны выбрать одно действие. При старте нового раунда:

* Выполняется выбранное игроками действие (Если оно есть)
* Обновляется внутреннее состояние игры (Увеличивается число заболевших и т.д.)
* Проверяется условие завершения игры

В каждый момент времени игрокам видны

* Значения переменных, описывающих игру (*Например, число заболевших*)
* Время до следующего раунда
* Доступные действия
* Журнал действий (Совершенные действия)
* Вкладки с полезной информацией

## Конструктор сценариев

Необходимо разработать способ по заданию различных вариаций игр. В данном случае игру можно рассматривать как набор состояний и правила перехода между этими состояниями. При этом игрок видит лишь доступную ему информацию, остальная часть от него скрыта. Хороший конструктор должен давать возможность задавать правила перехода и влиять на любые элементы, которые находятся в состоянии игры. Состояние игры описывается:

* **Переменными**
* **Вкладками и листами**
* **Ролями, присутствующими в игре**
* **Сработавшими действиями**

**Переменные**

Переменные – это объекты, которые хранят в себе текущее значение какого-либо параметра. Переменные состоят из:

* Имени
* Начального значения
* Минимального и максимального значения (при выходе за границы, переменной присваивается соответствующее предельное значение)
* Видна переменная или нет (невидимые переменные не будут отображаться игроку, но они всё-также влияют на состояние игры)
* Id

**Вкладки**

Вкладка – это массив, состоящий из **листов**. Функциональный смысл вкладок заключается в группировании доступной игроку информации. У каждой вкладки есть имя, id, а также флаг видимости (видна ли игроку при старте программы вся вкладка целиком).

**Листы**

Лист – это доступная пользователю информация, представленная в виде картинки. У каждого листа есть id, картинка и флаг видимости (виден ли лист при старте игры).

**Роли**

Роль – это роль, которую пользователь может выбрать при старте игры. Каждой роли доступны некоторые **действия**. Роль состоит из имени, id, а также описания роли (информация, доступная игрокам, для того, чтобы они могли решить, какую роль выбрать)

Правила перехода между состояниями игры описываются:

* **Функциями**
* **Действиями**
* **Условием на завершение игры (победа или проигрыш)**

Чтобы задавать правила необходимо придумать достаточно гибкий язык, который позволит проверять состояние игры и в свободной форме его изменять. В качестве такого языка используются **предикаты**.

**Предикаты**

Множество булевых функций и математических выражений, объединённых через:

* Логическое и (&)
* Логическое или (|)

Внутри можно использовать (), чтобы определить нужный порядок выполнения. Используется расширенная булева алгебра, в которой любое число, кроме 0, переводится в True, 0 переводится в False, и наоборот. Среди доступных математических операций есть:

* +,-,\*,/ - соответствующие мат. операции
* // - деление нацело
* % - остаток от деления
* <,<=,=,>,>= - соответствующие операции сравнения

Среди функций есть:

* not(x) - логическое отрицание
* max(a,b) - максимум между 2 числами
* min(a,b) - минимум между 2 числами
* exp(x,y) - x в степени y
* ans(r0\_0) - возвращает индекс ответа, который был выбран в действии r0\_0
* show(n,t0,t1\_2,…) – показывает n (если n = 0, то все) листов из t0,t1\_2,.. . Можно указывать, как целиком вкладки, так и конкретные листы. Показывает только те объекты, что не были видны в данный момент игроку. Возвращает True
* eq(p0,value) – присваивает переменной p0 значение value. Возвращает p0\_min<=value AND p0\_max>=value

Также в предикатах можно использовать:

* id переменной - заменяется на её нынешнее значение
* id вкладки или листа - если вкладка/лист видна в данный момент, то возвращается 1, иначе 0
* id действия - сколько раз действие было использовано
* id роли – возвращает количество игроков, которые выбрали данную роль
* an – только внутри действия (**реакции** или **условии на реакцию**). Возвращает индекс ответа, который был выбран непосредственно во время вызова действия (или -1, если действие не подразумевает выбор ответа, или использовано снаружи действия)

Присваивание внутри eq происходит только, если функция была использована внутри **реакции**. Если eq используется внутри условия на действие, то присваивание происходит, только если целиком всё условие оказывается верным. Это нужно, чтобы можно было выставлять цену на действие

Таким образом предикаты позволяют:

* **Проверять** **любой** элемент состояния игры
* **Изменять** **любой** элемент состояния игры (если игра предусматривает такое изменение)

**Действия**

Действие – это правило доступное игроку, которое изменяет состояние игры. Действия ассоциируются с ролью и описывают, каким образом игрок может взаимодействовать с игрой. Элементы действия:

* Описание действия - основываясь на этой информации игроки должны понять, что произойдет при вызове данного действия
* Варианты ответа - при вызове действия игрокам будет необходимо дополнительно выбрать один вариант ответа из присутствующих
* **Условие** на действие - **предикат**, который описывает, в каком случае это действие можно применять
* Список из **реакций** и **условий** на реакции - предикаты. При выполнении условия на реакцию, запускается соответствующая реакция
* Максимальное количество использований - сколько раз можно использовать действие

За счёт того, что действие написано на языке предикатов, оно позволяет каким угодно образом влиять на состояние игры

**Функции**

Функции – это правила, по которым происходит обновление состояния игры в начале каждого раунда. Функции ассоциируются с переменными. Каждой переменной может соответствовать любое количество функций. Функция состоит из **условия** и **реакции**. В начале раунда для каждой переменной подряд проверяются условия функций. Далее берётся первое сработавшее условие и выполняется соответствующая реакция. Таким образом для каждой переменной срабатывает не более одной функции. Условия и реакции состоят из предикатов.

Благодаря предикатам, функции позволяют устанавливать новое состояние игры, опираясь на нынешнее состояние игры

**Условие на победу и проигрыш**

Условие на победу и проигрыш - предикаты, которые описывают в каком случае игрок побеждает или проигрывает

# Реализация конструктора

## Инструменты разработки

В конструкторе сценариев необходимо задавать большое количество элементов и набирать много текста, поэтому удобнее всего реализовать конструктор в качестве компьютерного приложения.

Существует множество инструментов разработки под компьютер. Логичнее всего выбрать те инструменты, которые позволяют создавать кроссплатформенные приложения. Одним из таких инструментов является нативно встроенная в python графическая библиотека Tkinter.

Tkinter является одной из первых графических библиотек для python, и она до сих пор поддерживается, так что её эффективность была проверена временем. *И*[*нтегрированная среда разработки*](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)*и обучения IDLE была создана с помощью библиотеки*[*Tkinter*](https://ru.wikipedia.org/wiki/Tkinter)

**Итог:**

* Операционная система – **Windows, MacOs, Linux**
* Язык программирования - **Python**
* Среда разработки - **PyCharm**
* Графическая библиотека – **Tkinter**

## Описание библиотеки Tkinter

Tkinter – это кроссплатформенная библиотека для разработки графического интерфейса на языке Python. Tkinter состоит из виджетов, которые представимы в виде экземпляров класса.

Чтобы создать графический интерфейс, достаточно инициализировать главное окно (особый виджет в tkinter, создаёт новое окно), инициализировать остальные виджеты, и разместить их на главном окне. Существует 2 самых популярных способа размещения объектов:

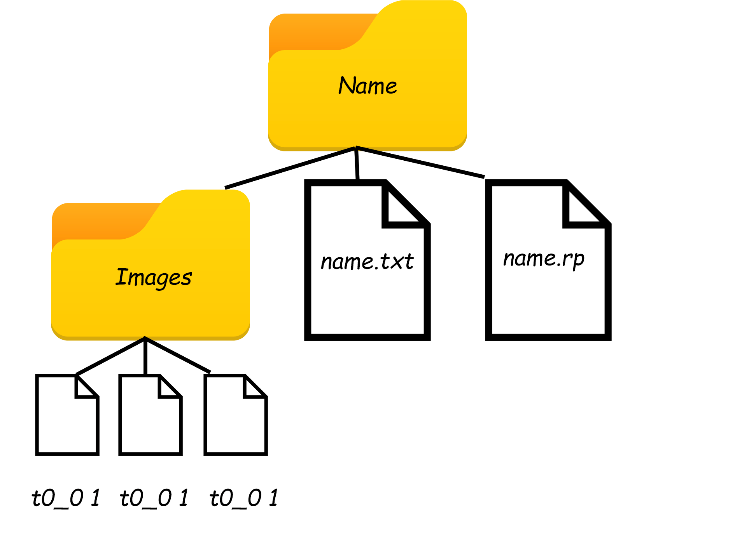
* Pack – размещает объекты последовательно друг за другом. Можно указать с какой стороны необходимо начать размещать объекты (сверху-вниз, слева-направо, и т.д.)
* Grid – размещает объекты по секте. Размер каждой строки/столбца подбирается автоматически, как размер максимального объекта внутри строки/столбца. Объект может занимать несколько строк/столбцов. Каждой строке/столбцу можно дать вес – как ей стоит расширяться, чтобы занять свободное пространство

Теперь, чтобы получить полноценное приложение, осталось лишь добавить реакцию на действие пользователя. Для этого у каждого виджета есть метод widjet.bind(Event,Func). Если для данного виджета сработает заданный эвент – произойдёт вызов указанной функции. Примеры существующих эвентов:

* Нажатие на клавишу
* Клики мышкой
* Передвижение мышки
* Изменение размера окна
* И другие

## Сохранение сценария

Чтобы использовать созданный сценарий его необходимо сохранить. Для этого нужно выработать формат, в котором будет происходить сохранение сценария. При сохранении создаются следующие файлы:



Внутри папки Images лежат файлы формата “list\_id visible.\*”, где list\_id – это id листа, visible – это 1 или 0, в зависимости от видимости листа. Формат данных в name.txt описан в **Приложении A**

## Обратная польская запись

Предикаты очень удобно представлять в виде обратной польской записи. Обратная польская запись — форма записи [математических](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и [логических](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) выражений, в которой [операнды](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4) расположены перед знаками [операций](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Преимущество такой записи в том, что это позволяет нам очень просто вычислять выражение, используя единственный стек, считывая операнды подряд, друг за другом. Можно преобразовать предикаты в обратную польскую запись на стороне приложения внутри игры, но удобнее это сделать один раз во время создания конструктора. Однако перевод предикатов изменяет исходное форматирование (например, удаляет скобки), поэтому в отдельном файле (name.txt) будет храниться информация о сценарии в исходном виде, а в другом файле (name.rp) будет храниться информация с преобразованными предикатами. Для того, чтобы привести выражение к виду обратной польской записи существует алгоритм «сортировочной станции». Его смысл в том, что нужно пройти по выражению слева-направо, и как только встречается операнд, выполнить все накопленные операнды с более высоким приоритетом, а нынешний операнд добавить в очередь. Дополнительно во время преобразование проверяется:

* Отсутствие лишних слов в предикатах (разрешены только введённые функции и операторы)
* Все указанные id существуют
* Все функции заданы от правильного количества аргументов, и аргументы функции имеют правильный тип

Такие проверки гарантируют, что сценарий всегда будет корректен для дальнейшей обработки

# Реализация обучающей игры

## Инструменты разработки

Для работы приложения необходимо, чтобы все игроки находились в одной комнате. Запускать множество ноутбуков, и тем более компьютеров, в одной комнате неудобно. Поэтому приложение должно работать на мобильных устройствах.

Существует две доминирующие на мобильном пространстве операционные системы – Android и IOS. Разработка под них довольно сильно отличается, а кроссплатформенные решения не обеспечивают той гибкости, которую можно получить, разрабатывая под конкретную операционную систему. По этим причинам разработка будет вестись только под Android – так как операционная система Android на 2021 занимает около 70% мирового рынка всех операционных систем, используемых на мобильных устройствах, для России этот показатель может быть даже выше, так как устройства именно с этой операционной системой являются наиболее доступными по цене

Так-как все игроки будут находиться в одной комнате – организовывать сетевое взаимодействие через интернет излишне. Для реализации сети на близкой дистанции существует несколько вариантов:

* Bluetooth – метод сетевого взаимодействия, основанный на использовании радиоволн с частотами 2.402-2.48 ГГц
* WiFi - технология беспроводной локальной сети с устройствами на основе стандартов [IEEE 802.11](https://ru.wikipedia.org/wiki/IEEE_802.11). Работает на больших расстояниях и с большей скоростью, чем Bluetooth, но потребляет больше энергии и требует общей точки доступа.
* WiFi Direct – стандарт, позволяющий двум и более [Wi-Fi](https://ru.wikipedia.org/wiki/Wi-Fi" \o "Wi-Fi)-устройствам общаться друг с другом без [маршрутизаторов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80) и общих точек доступа

WiFi Direct быстрее Bluetooth и работает на больших расстояниях, не требует общей точки доступа, по сравнению с обычным WiFi, а так-же тратит меньше всего зарядки, за счёт умного распределения ресурсов. По этим причинам будет использоваться WiFi Direct

**Итог:**

* Операционная система – **Android**
* Язык программирования - **Java**
* Среда разработки - **Android Studio**
* Графические библиотеки – **внутренние библиотеки Android**
* Организация сети – **WiFi Direct (WiFi P2P)**

## Обзор Android Studio

В Android Studio для создания приложения необходимо работать с 3 типами файлов: **Manifest.xml, Layout.xml** и **Activity.java.**

* **Manifest** – здесь описываются: разрешения, которые могут понадобиться чтобы приложение заработало; технические и программные составляющие, которые необходимы для корректного запуска приложения; название пакета, в котором содержится весь код приложения; все компоненты из которых состоит приложение (все activity и прочее)
* **Layout.xml** – здесь описывается внешний вид приложения. Существует множество объектов (кнопки, текстовые поля, списки с объектами и т.д.), которые можно здесь использовать. Для размещения в пространстве существует несколько базовых слоёв, например, линейные слои (вертикальные и горизонтальные, всё размещается линейно друг за другом), constraint слои (необходимо связывать объекты горизонтальными и вертикальными линиями к другим объектам. Можно привязывать объекты к краям экрана) и др. Ещё каждому объекту здесь задаётся уникальный id.
* **Activity.java** – с точки зрения Java, это просто класс, унаследованный от AppCompatActivity. Activity является ключевым классом для приложения. Всё взаимодействие с пользователем происходит именно здесь.

У activity есть жизненный цикл, и чтобы понять, что происходит, необходимо его разобрать:

* **onCreate**() вызывается, как только activity было создано, или при повторном обращении к activity, при его уничтожении.
* **onStart**() вызывается всегда после onCreate() или после onRestart().
* **onResume**() вызывается всегда после onStart() или после onPause(), в случае, если пользователь вернулся к activity, и оно ещё не было уничтожено.
* **onPause**() вызывается после onResume(), если другое activity вышло на передний план.
* **onStop**() вызывается после onPause(), если activity больше не видно пользователю.
* **onRestart**() вызывается после onStop(), если пользователь вернулся к activity, и оно ещё не было уничтожено.
* **onDestroy**() вызывается после onStop(), если activity завершено или убивается системой.

Таким образом onResume() будет вызываться очень часто, поэтому здесь нежелательно размещать тяжёлые элементы, с другой стороны onCreate() вызывается редко, поэтому всю тяжёлую инициализацию необходимо выполнять именно здесь. Также в onCreate() необходимо позаботиться о визуальном виде приложения, то есть отобразить окно с одним из существующих Layout.xml. В **Приложении B** приведена схема, для лучшего понимания жизненного цикла activity.

Чтобы отобразить layout нужно внутри activity обратиться к функции setContentView(). Чтобы найти объект внутри layout нужно вызвать findViewById(). Далее для объектов можно изменить свойства (например, выставить текст) и зарегистрировать **callback** – действие, которое будет происходить при взаимодействии с объектом (например, при нажатии на кнопку)

## Обзор WiFi P2P

В приложении, любой пользователь может создать комнату, указав имя комнаты и нужный сценарий. В таком случае все другие пользователи увидят эту комнату и могут к ней подключиться. Для создания сети используется встроенная в android библиотека - WiFi Direct (WiFi P2P). В WiFi Direct следует разобрать следующие объекты: **слушатели, менеджер, канал, broadcast receiver, фильтр интентов и группа.**

* **Слушатели** – это классы, которые извещают нас в случае успеха или провала запрашиваемой функции. Практически всё взаимодействие в WiFi Direct происходит именно через слушателей. *Например, при вызове функции подключения к другому устройству, необходимо указать слушателя*. Эта функция просто зарегистрирует слушателя и сразу же вернёт пустое значение, а когда подключение действительно инициализируется (или произойдёт ошибка) – слушатель нас об этом известит (при создании слушателя нужно самим указать, что он будет делать в том или ином случае).
* **Менеджер** – это ключевой класс в WiFi Direct. В нём определенны все основные функции. Через него осуществляется вся необходимая работа.
* **Канал** – это объект, который ассоциируется с менеджером и обеспечивает передачу данных, а также необходимую функциональность. До создания канала никакие действия с WiFi Direct производить нельзя. В программном плане просто некоторые функции будут требовать канал в качестве своего аргумента, больше он никак не используется
* **Broadcast receiver** – это объект, который будет извещать нас о различных изменениях связанных с сетью. Для того, чтобы его использовать, необходимо создать свой класс, унаследованный от BroadcastReceiver и вызвать функцию registerReceiver(receiver, фильтр интентов).
* **Фильтр интентов** – это объект, который показывает, какие события необходимо отлавливать. Внутри broadcast receiver нужно самим определить, что делать после каждого из этих событий. В приложении используются следующие события:
  + **WifiP2pManager.*WIFI\_P2P\_STATE\_CHANGED\_ACTION*** - состояние P2P сети изменилось. Например, пользователь выключил WiFi.
  + **WifiP2pManager.*WIFI\_P2P\_PEERS\_CHANGED\_ACTION*** - список доступных (к которым можно подключиться, используя WiFi P2) устройств изменился
  + **WifiP2pManager.*WIFI\_P2P\_CONNECTION\_CHANGED\_ACTION*** – изменилась конфигурация сети. Например, новое устройство к нам подключилось, или наоборот отсоединилось от нас.
  + **WifiP2pManager.*WIFI\_P2P\_THIS\_DEVICE\_CHANGED\_ACTION*** – изменения, связанные с нынешним устройством. Например, изменилось имя устройства или его IP адрес.
* **Группа** – множество устройств, которые подключены друг к другу посредством WiFi Direct. В группе всегда есть только один лидер. Каждое устройство может состоять не более чем в одной группе. WiFi Direct постоянно анализирует состояние сети и пытается оптимизировать расход трафика и энергии внутри группы. Таким образом, если в группе есть устройство, которое мощнее всех остальных и у него больше всех заряда, то все тяжёлые операции будут проходить через это устройство.

Если устройства A, B и C состоят в одной группе, но A и C не видят друг друга, но видят B, то они всё равно могут обмениваться информацией, используя B в качестве посредника, и WiFi Direct сам об этом позаботится.

Если устройства K (которое не является лидером группы) пытается подключиться к устройству D, то он отправляет запрос лидеру, и лидер пытается добавить D в свою группу. Таким образом всегда вопросом о подключении занимается сам лидер группы. На практике лидер может не видеть D, тогда K отсоединится от своей группы и подключится напрямую к D. Поэтому лучше все подключения самому организовывать через лидера

## Сокеты, handler и looper

Передача данных между устройствами реализована через **сокеты**. Как только устройства наладят связь через WiFi Direct и окажутся в одной группе, необходимо обеспечить обмен информацией посредством сокетов

Общение между сокетами организовано классическим образом: хост в вечном цикле принимает новые подключения, и на каждого клиента создаются два потока - один для чтения из сокета, один для записи в сокет.

Также необходима возможность действовать на эти потоки со стороны activity и наоборот. В Android, если объект был создан каким-то activity, то только этот activity может внести какие-либо изменения в объект. *Например, сторонний поток не сможет изменить какой-либо объект в layout.*

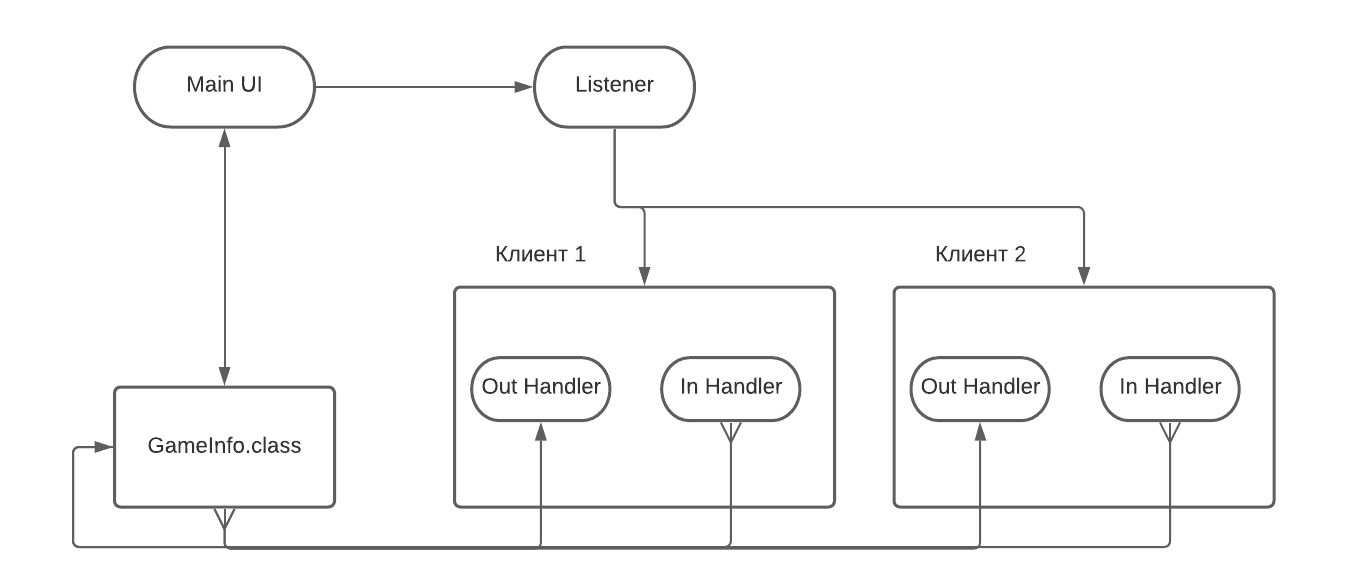
Для решений этой проблем в android существует **handler** и **looper**.

* **Looper** - при запуске приложения создаётся главный поток и главный looper (По сути - очередь с кодом). Главный поток в вечном цикле выполняет код из looper-а (в порядке очереди). Не рекомендуется занимать главный поток тяжёлыми операциями, так как он отвечает за интерфейс. Необходимо, чтобы пользователь мог взаимодействовать с приложением, даже если на фоне производятся вычисления.
* **Handler** - это объект, который можно ассоциировать с конкретным looper-ом и он позволяет отправлять в looper свой код на выполнение.

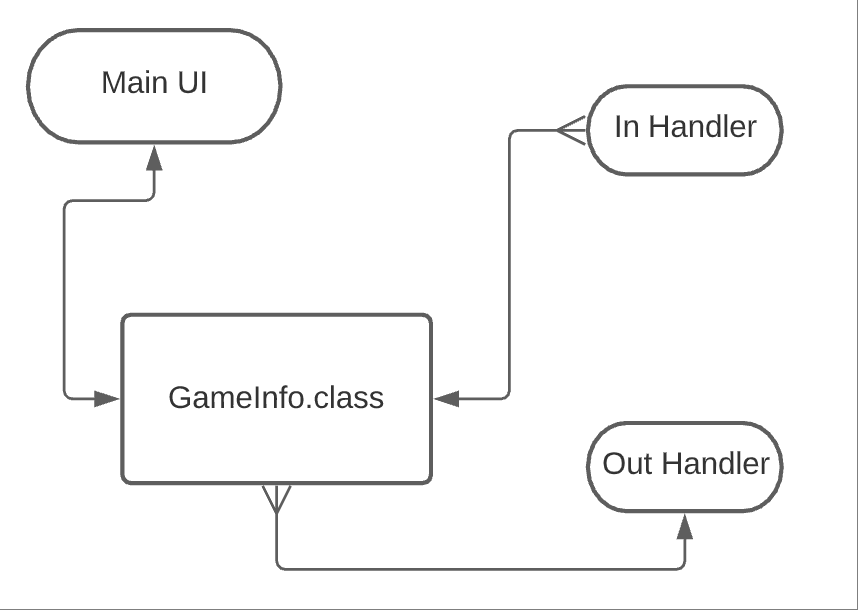
Таким образом каждому потоку нужно ассоциировать handler и работать, через него. То есть, если нужно изменить какой-то объект внутри layout из внешнего потока – нужно отправить соответствующий код на handler, ассоциируемый с главным looper-ом.

## Структура классов и потоков

**Хост**



**Клиент**



**Main UI** – основной поток. Создаётся при запуске приложения. Отвечает за отображение всего интерфейса.

**Listener** – поток, который принимает новые соединения на сокете. При налаживании нового соединения, создаёт In Handler и Out Handler.

**In Handler** – поток, который используется, чтобы прочитать данные с сокета. Читает данные с сокета построчно и вызывает метод GameInfo.parseLine(string)

**Out Handler** – поток, который используется, чтобы записать данные в сокет. Если главный поток хочет что-то передать по сети, он вызывает метод GameInfo.print(string,to)

**GameInfo.class** – класс, который хранит в себе всё состояние игры, а также методы, необходимые для взаимодействия между потоками. Метод parseLine(string) обрабатывает строку и делает соответствующие изменения в главном потоке. Метод print(string,to) заставляет соответствующий (to) Out Handler записать строку в сокет

Для переменных, функций, вкладок, изображений, ролей и действий существует свой класс. Во всех этих классах есть метод перевода в строку – он представляет объект в виде строки, готовой к передаче по сокету.

Такой подход значительно упрощает взаимодействие между потоками. Теперь, чтобы передать объект, достаточно вызвать его метод, для получения правильного строкового представления, и затем передать эту строку в метод GameInfo.print. Чтобы принять объект, достаточно написать соответствующий код в GameInfo.parseLine.

## Формат передачи данных

Хосту необходимо считать сценарий, и затем обмениваться частью этого сценария с другими игроками. Для этого необходимо выработать формат передачи данных. При сохранении сценария в конструкторе сценариев, такой формат уже был выработан, сейчас лишь необходимо его дополнить.

Расширение формата передачи данных можно найти в **Приложении A**.

## Передача действий

С передачей действий могут быть некоторые проблемы. Только хост знает, какие действия доступны в данный момент, а какие недоступны. Поэтому хост информирует игроков о том, доступны действия, или нет. В таком подходе кроется опасность – если пользователь отправил действие хосту, а в этот момент как-то изменилось состояние игры (например, просто начался новый раунд) – отправленное действие теперь может быть недоступно. Можно просто каждый раз при получении действия проверять доступно ли оно. Также нужно извещать клиента о том, что его действие сработало, иначе он никогда не будет знать – отправлено действие, или нет. В таком подходе тоже есть проблема – если клиент отправил действие в нулевом раунде, пока действие дойдёт, может начаться первый раунд. Тогда, если действие всё ещё будет доступно, хост выполнит действие и начнёт второй раунд – а это является ошибкой. Игрок собирался завершить нулевой раунд, а не первый. Поэтому дополнительно нужно отправлять номер раунда, и на стороне хоста смотреть, совпадает ли принятый номер с текущим. При этом нужно следить, чтобы номер раунда отправлялся корректно. Простой пример ошибки:

1. Клиент проверяет доступно ли действие
2. Клиент считывает нынешнее значение номера раунда
3. Клиент отправляет действие и номер раунда хосту

Если между 1) и 2) на стороннем потоке произойдёт обновление номера раунда – произойдёт ошибка.

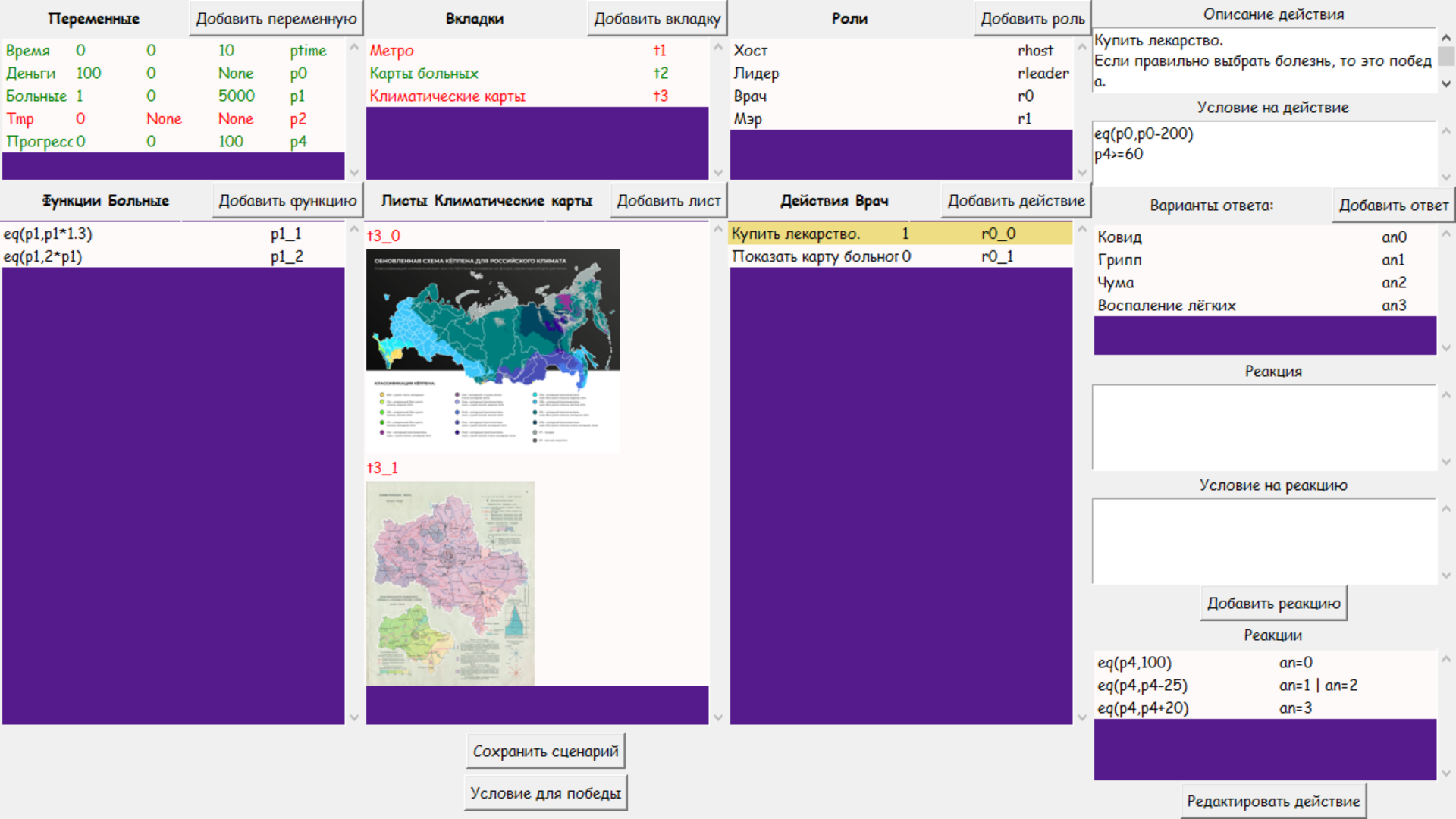
При этом на самом деле действия от клиента будут отправляться лидеру (через хоста), и лишь действия отправленные лидером будут сразу-же выполняться. Поэтому нужно дополнительно следить, чтобы у лидера всегда были корректные действия от игроков.

Расширение формата передачи данных можно найти в **Приложении A**.

# Результаты работы

## Итоговый конструктор

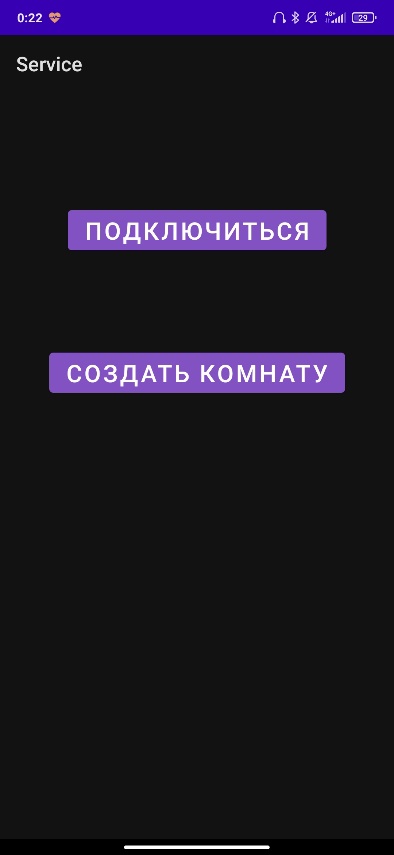
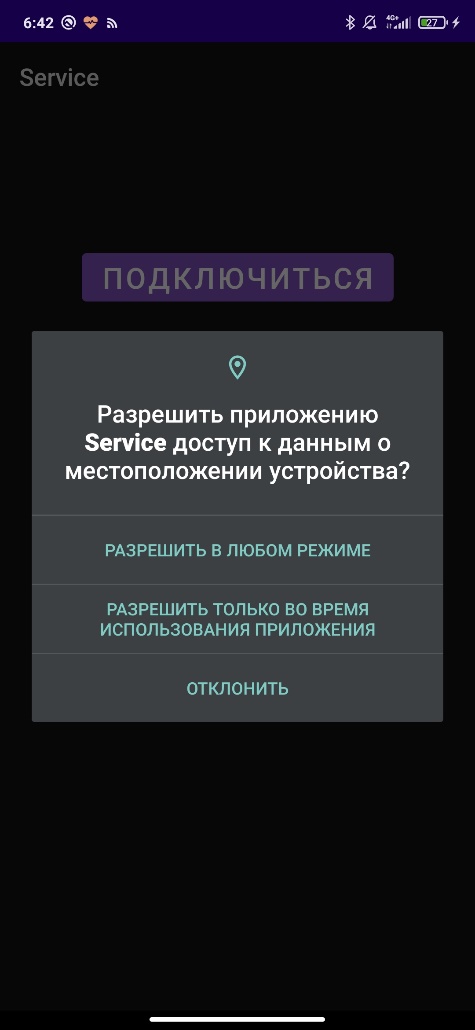
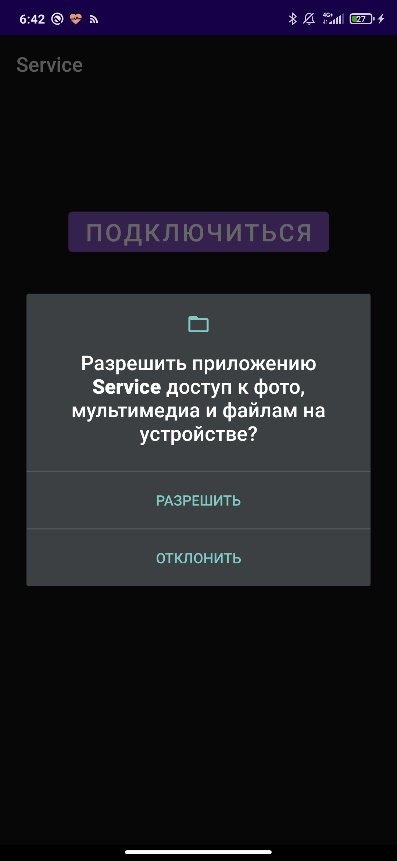
Было разработано компьютерное приложение для создания и редактирования сценария



* При старте программы можно либо выбрать существующий сценарий и отредактировать его, либо создать новый сценарий
* При клике на предмет – он выделяется специальным цветом.
* Двойной клик открывает редактирование.
* Двойной клик на id изменяет видимость объекта. Видимые объекты выделяются зелёным цветом, невидимые – красным.
* Del – удаляет выделенный объект.
* Esc – снять выделение.
* При клике на переменные/вкладки/роли в соответствующем окне показываются соответствующие функции/листы/действия.
* Внутри функций можно выделить кликом предмет, а затем использовать стрелки (Up и Down) чтобы поменять местоположение объекта (функции выполняются сверху вниз, при этом выполнение обрывается при первом срабатывании, поэтому порядок важен).
* Двойной клик на картинку внутри листа открывает её в новом окне (здесь можно изменять размер окна, чтобы лучше разглядеть картинку).
* Минимальное/максимально значение переменной может быть только числом или none, что означает отсутствие предельного значения
* Начальное значение в переменной с индексом ptime показывает продолжительность раунда в секундах (0 – значит не ограничено по времени). Максимальное значение показывает количество раундов (0 или None – значит нет ограничения по раундам)
* Проверяет корректность предикатов и указывает на ошибки

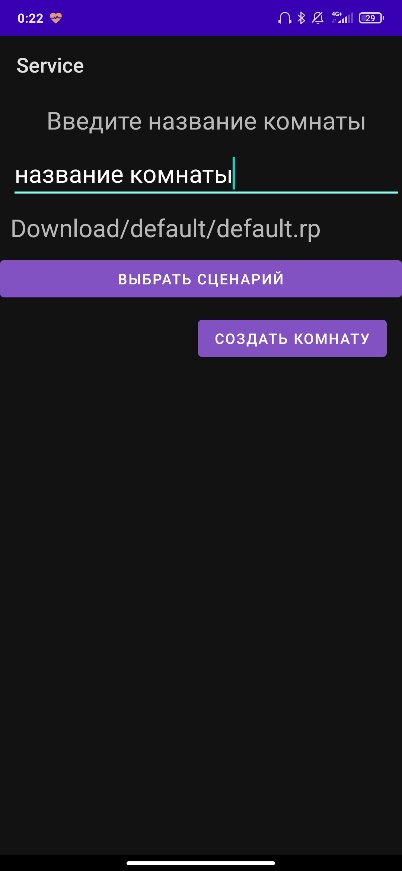
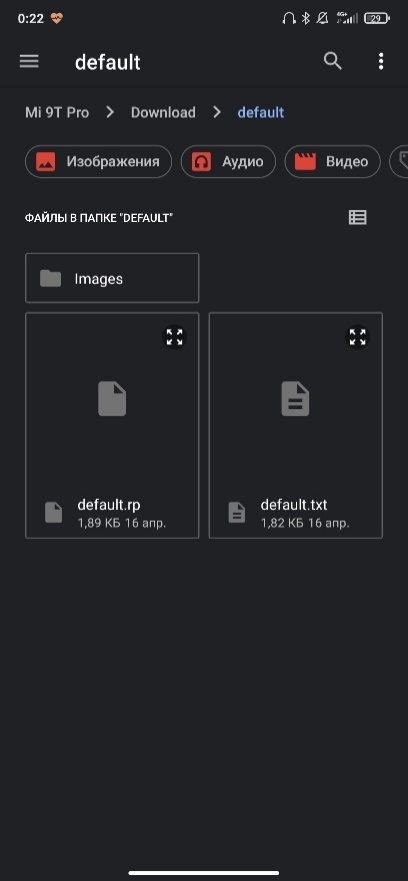
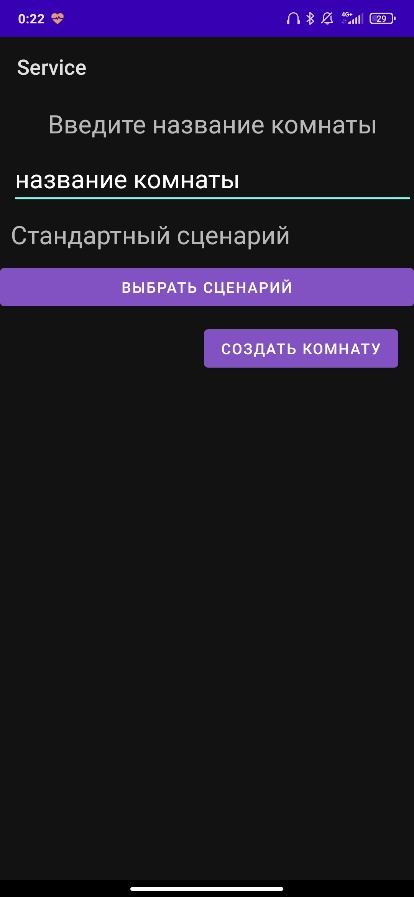
## Итоговое мобильное приложение

**Запуск**



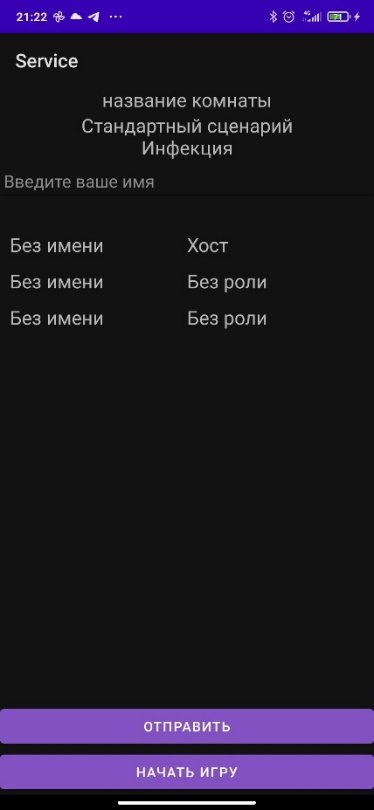
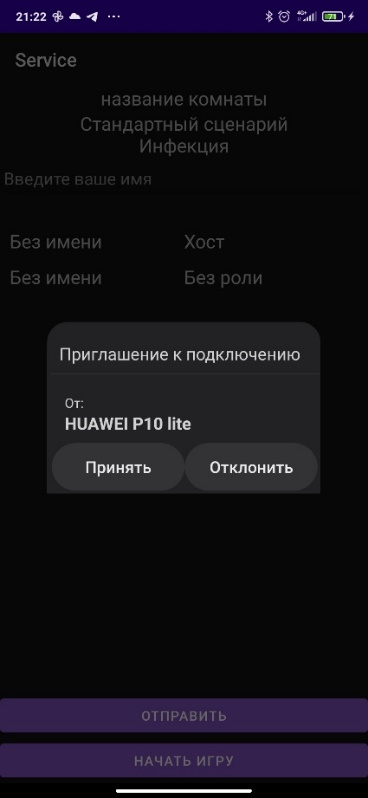
Приложению, для корректной работы необходим доступ к файлам на устройсте (для загрузки сценария) и доступ к геоданным (для работы WiFi Direct). При запуске приложение потребует эти разрешения. Так-же необходимо включить WiFi и геоданные, иначе сеть будет невозможно создать и приложение будет сигнализировать о выключенном WiFI.

**Создание комнаты**



Здесь необходимо указать название комнаты и указать путь к файлу со сценарием (\*.rp). При нажатии на кнопку «выбрать сценарий» открывается файловый менеджер, в котором можно выбрать нужный файл.

**Комната ожидания (Хост)**

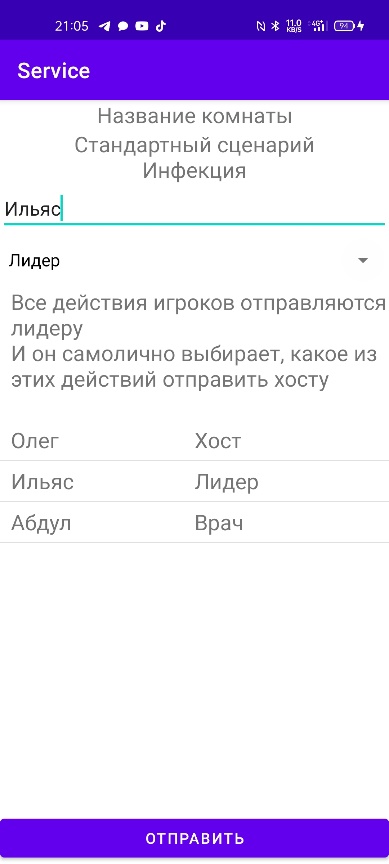
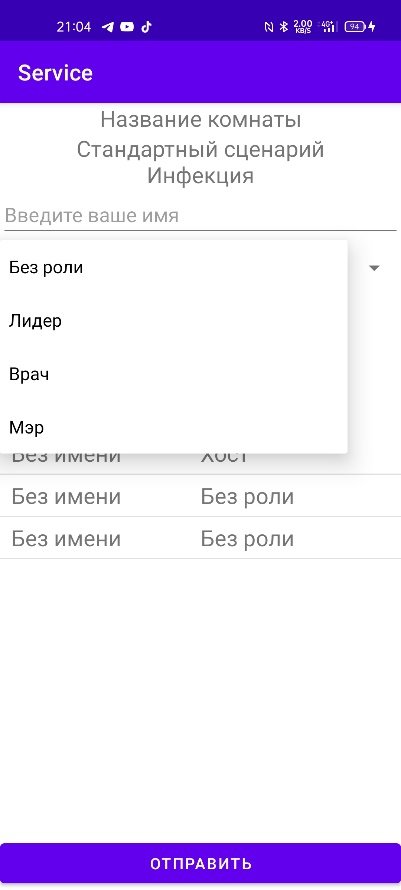
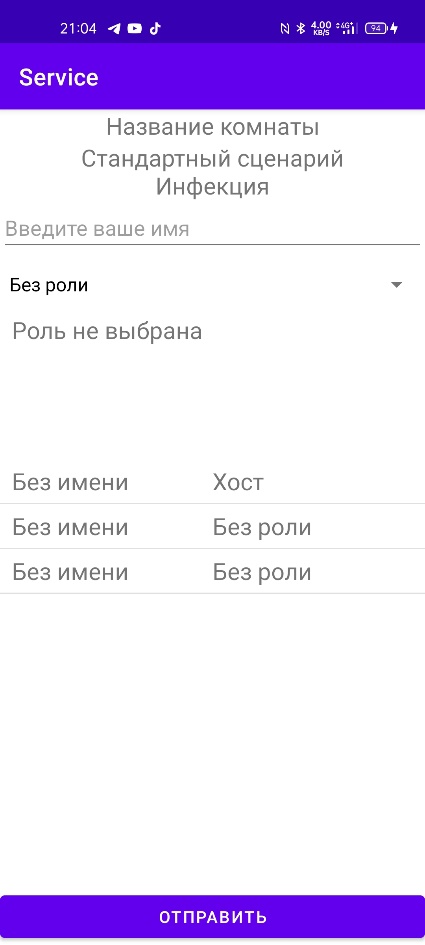
****

При создании комнаты открывается данное окно. Здесь хост может видеть подключённых игроков (их роль и имя), а также может переименоваться сам. При подключении каждого игрока всплывает соответствующее окно. Когда наберётся достаточное количество игроков хост может начать игру, к этому моменту все игроки должны выбрать роль, среди которых должен быть хотя-бы один лидер.

**Подключение**

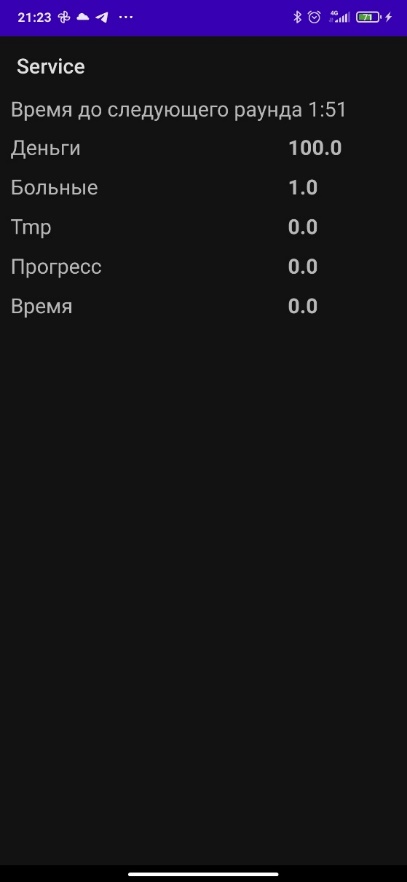
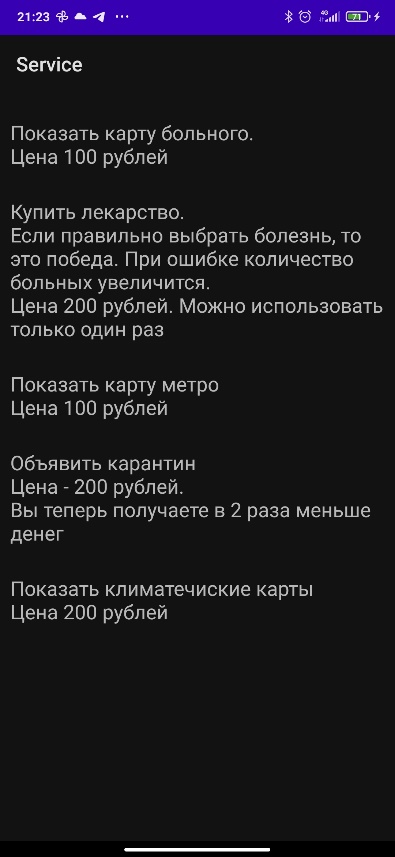
Здесь отображают все доступные на данный момент комнаты. При нажатии на комнату, происходит подключение к ней.

**Комната ожидания (Клиент)**

****

При подключении к комнате открывается это окно. Здесь пользователь видит всех игроков в комнате и может установить себе роль и имя.

**Игра**

****

При старте игры перед пользователем открываются окно с действиями, окно с основной информацией и окно с вкладками. Между окнами можно переключаться через свайпы влево-вправо.

**Действия**

В окне с действиями отображаются все действия для данной роли. Хост видит все свои действия + действия всех остальных ролей. Действие может быть недоступно, в этом случае всплывёт соответствующее окно. Хосту доступны все действия.

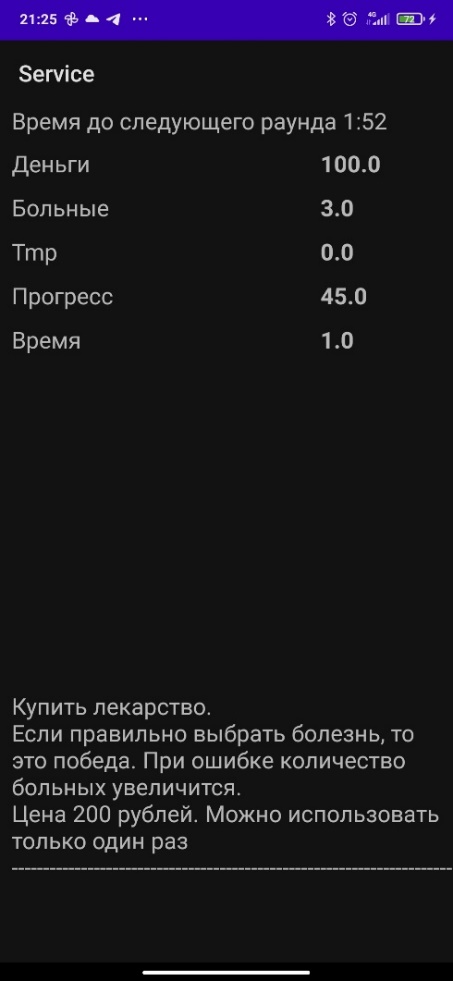
При клике на действие, если оно доступно, открывается окно, в котором можно выбрать один из вариантов ответа, если действие предполагает несколько вариантов ответа. Кнопка «отправить действие не прерывая раунд» есть только у хоста.

**Вкладки**

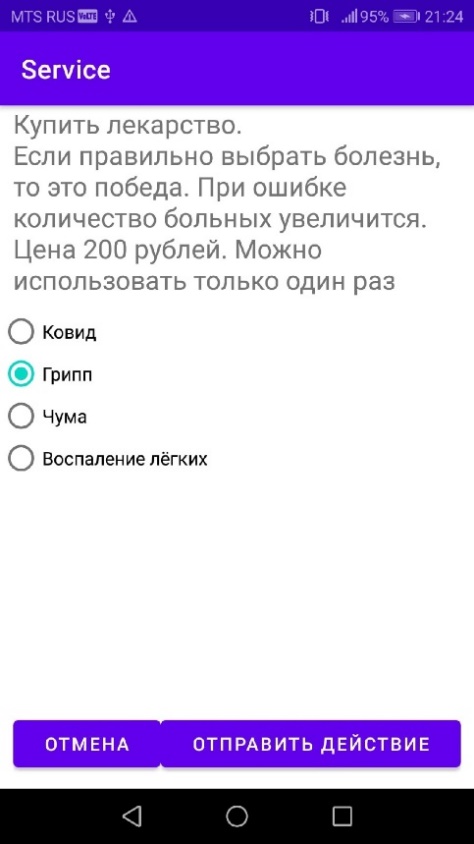
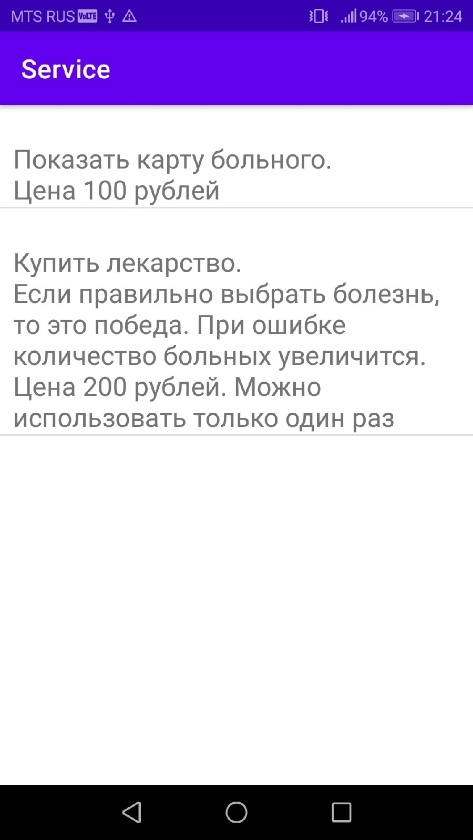
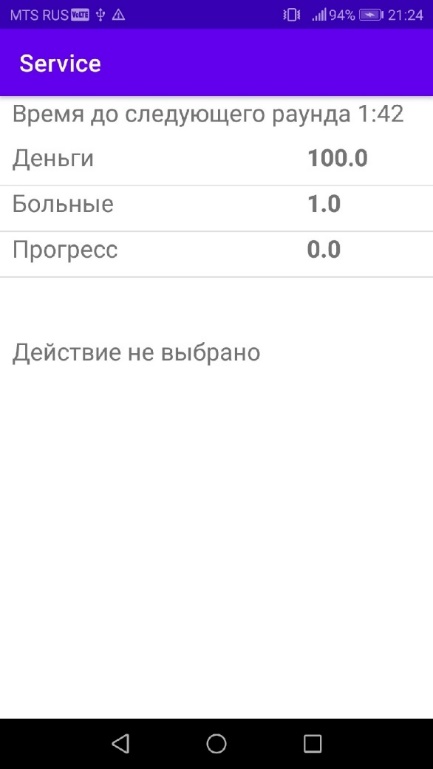
****В окне с вкладками отображаются все видимые игроку вкладки. Хосту отображаются все вкладки в игре.

При клике на вкладку открывается её содержимое. Свайпами влево-вправо можно переключаться между картинками. Кнопка «отправить игрокам» есть только у хоста, и отображается, только если данного изображения нет у игроков.

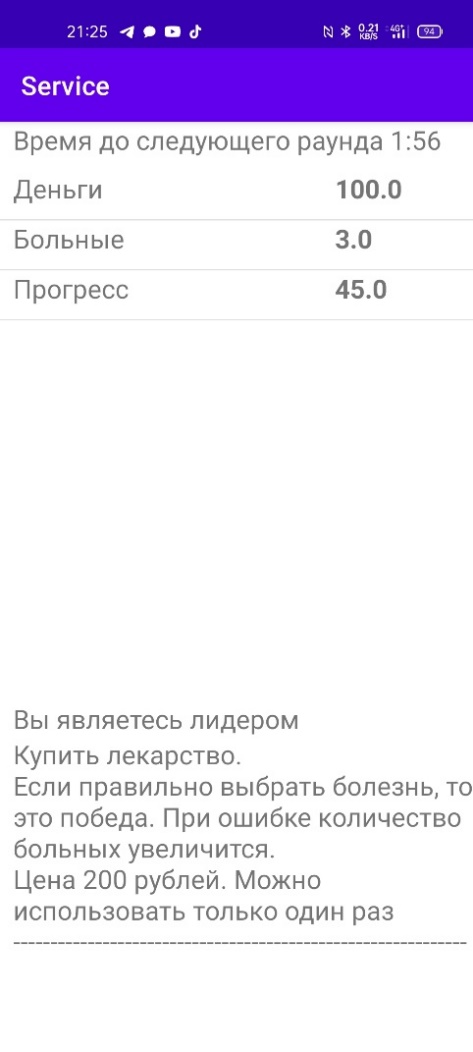
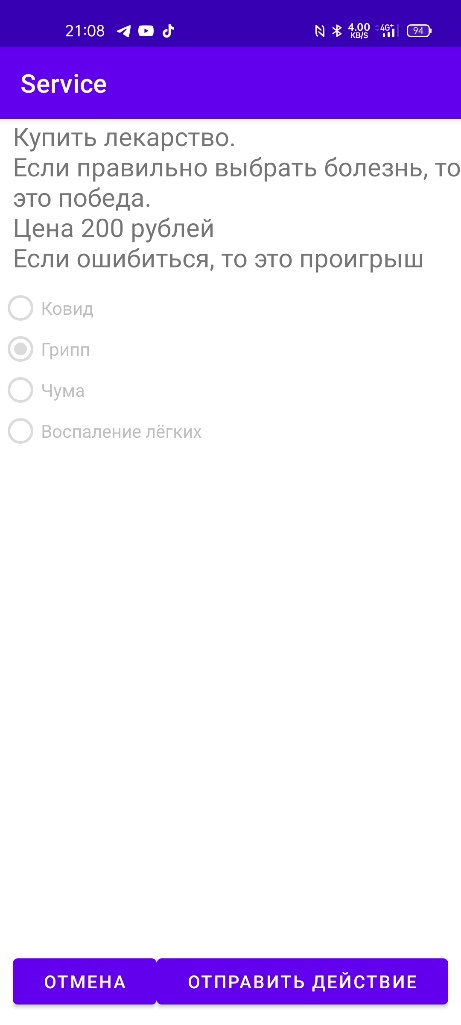
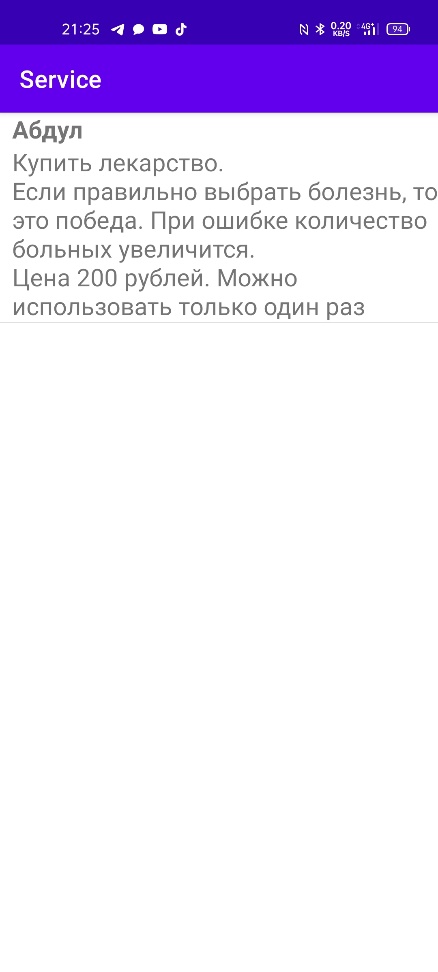
**Переменные**

В центральном окне отображаются значения переменных и время до следующего раунда. Внизу отображаются все действия, которые были совершены к данному моменту. Хост может кликать на переменные, чтобы изменять их

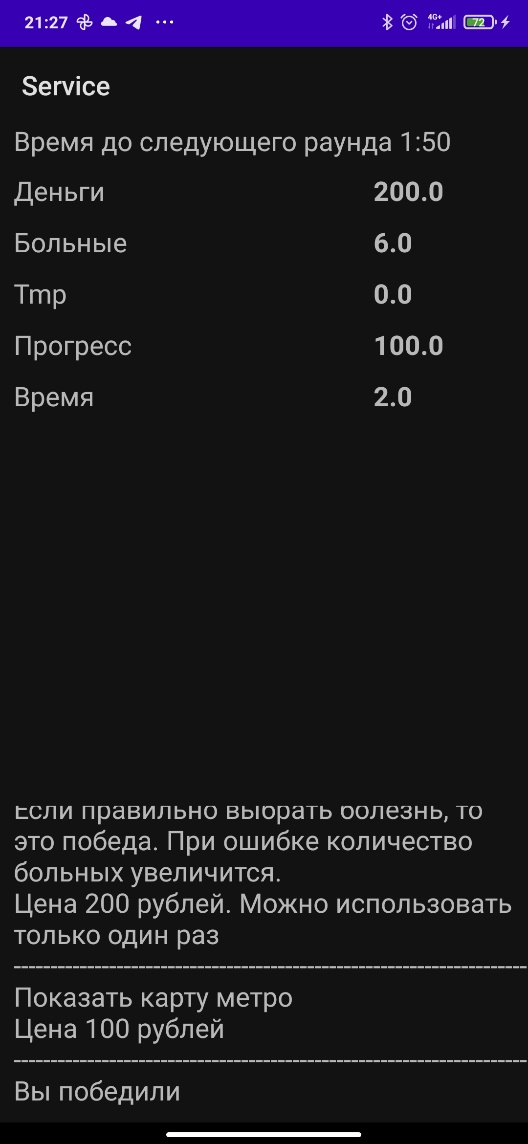
**Отправка действий**



Если пользователь (не лидер) отправляет действие, то оно передаётся лидеру. Теперь лидер может это действие подтвердить, и тогда оно отправится хосту и начнётся новый раунд.



**Конец игры**



Когда таймер отсчитывает нужное время – начинается новый раунд. Если игроки не успеют победить за отведённое им время (в раундах), то они проиграют.

При победе или поражении соответствующая информация записывается в журнал действий, и игра перестаёт отвечать на действия игрока.

## Итог

Был придуман способ перевода уже существующей игры в цифровой вид.

На основе этого цифрового вида было разработано на языке Java мобильное приложение по реализации различных сценариев (~2000 строк кода), при этом была решена проблема создания и поддерживания сетевого взаимодействия между устройствами.

Также было разработано на языке Python кроссплатформенное компьютерное приложение для создания сценариев (~1700 строк кода), при этом был придуман способ, основанный на предикатах, по описыванию и выполнению этих самых сценариев.

Мобильное приложение было протестировано на 7-ми телефонах, на одном из которых оно не заработало. Компьютерное приложение было протестировано на 5-ти ноутбуках с Windows и одним ноутбуком с MacOs, проблем не обнаружено.

# Список литературы

1. Ривз БайронМалоун ТомасО’Дрисколл Тони. Виртуальный мир как кузница руководящих кадров [Электронный ресурс].- Электрон.дан. - URL: <https://hbr-russia.ru/liderstvo/psikhologiya/a9572>. (дата обращения: 03.04.2021).
2. О. Ю. Гроголева, В. С. Анкудинова. Типологические особенности личности участников интеллекутально-психологической игры "Мафия"// Вестник Омского университета. Серия "Психология".-2014.№2. C.4-12
3. Trend: Consumers spend significantly more on digital brain health and neurotechnology apps [Электронный ресурс].- Электрон.дан. - URL: <https://sharpbrains.com/blog/2019/05/24/trend-consumers-spend-significantly-more-on-digital-brain-health-and-neurotechnology-apps/>. (дата обращения: 03.04.2021).
4. Roumen Vesselinov, John Grego. [Изучение эффективности Дуолинго](http://static.duolingo.com/s3/DuolingoReport_Final.pdf) (англ.) = Duolingo Effectiveness Study. — [Нью-Йорк](https://ru.wikipedia.org/wiki/%D0%9D%D1%8C%D1%8E-%D0%99%D0%BE%D1%80%D0%BA), [Колумбия (Южная Каролина)](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%83%D0%BC%D0%B1%D0%B8%D1%8F_(%D0%AE%D0%B6%D0%BD%D0%B0%D1%8F_%D0%9A%D0%B0%D1%80%D0%BE%D0%BB%D0%B8%D0%BD%D0%B0)), 2012. — С. 25.
5. Марк Лутц. Изучаем Python. :5-е изд. Том 1. - СПб.: Символ-Плюс, 2011
6. [Alan Moore](https://www.google.com/search?sa=X&rlz=1C1GCEB_enRU910RU910&biw=1280&bih=610&sxsrf=ALeKk01CcpYe5UCaBqbPBCkHIvK8s-0_8w:1619920759540&q=Alan+Moore&stick=H4sIAAAAAAAAAOPgE-LVT9c3NEzPTsoxKLOMV2LXz9U3qKzI0ZLJTrbST8rPz9YvL8osKUnNiy_PL8q2SiwtycgvWsTK5ZiTmKfgm59flLqDlREAifIvO0gAAAA&ved=2ahUKEwjszL6L86nwAhUoposKHah7CAQQmxMoATAYegQIExAD). Python GUI Programming with Tkinter: Develop Responsive and Powerful GUI Applications with Tkinter.- Бирмингем: Packt Publishing,2018.- С. 454
7. Joshua Bloch. Effective Java.- Boston : Addison-Wesley, 2018.-С. 392
8. Android Studio [Электронный ресурс].- Электрон.дан. - URL: <https://developer.android.com/studio/intro>. (дата обращения: 08.12.2020).
9. WiFi Direct [Электронный ресурс].- Электрон.дан. - URL: <https://developer.android.com/training/connect-devices-wirelessly>. (дата обращения: 08.12.2020).
10. Обратная польская запись [Электронный ресурс].- Электрон.дан. - URL: <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C>. (дата обращения: 15.03.2021).
11. PyCharm [Электронный ресурс].- Электрон.дан. - URL: <https://www.jetbrains.com/pycharm/>. (дата обращения: 26.03.2021).
12. Handler [Электронный ресурс].- Электрон.дан. - URL: https://betterprogramming.pub/a-detailed-story-about-handler-thread-looper-message-queue-ac2cd9be0d78. (дата обращения: 08.12.2020).

# Формат данных при сохранении сценария и передачи данных

Перед сохранением/отправкой данных все переносы строк заменяются на {. # используется в качестве разделителя

**Формат одной строки в name.txt:**

* -v#id#value#min\_value#max\_value#visible#name – информация о переменной
* -f#id#reaction#condition – информация о функции. Reaction – реакция. Condition – условие на реакцию. Функции идут в том-же порядке, что они объявлены в конструкторе (сверху вниз)
* -t#id#visible#name – информация о вкладке
* -r#id#descr#name – информация о роли. Descr – описание роли.
* -a#id#count#descr#req#ans0,ans1,..#ans\_id0,ans\_id1,…#N – информация о действии. Count – максимальное количество использований действия. Descr – описание действия. Req – условие на действие. Ans0,ans1… - варианты ответов, через запятую.Ans\_id0,Ans\_id1… - индексы ответов, через запятую. N – количество реакций в действии. Следующие N строк будут описывать эти реакции и будут иметь формат reaction#condition.
* -n#name – название сценария
* -w#cond – условие на победу
* -l#cond – условие на проигрыш

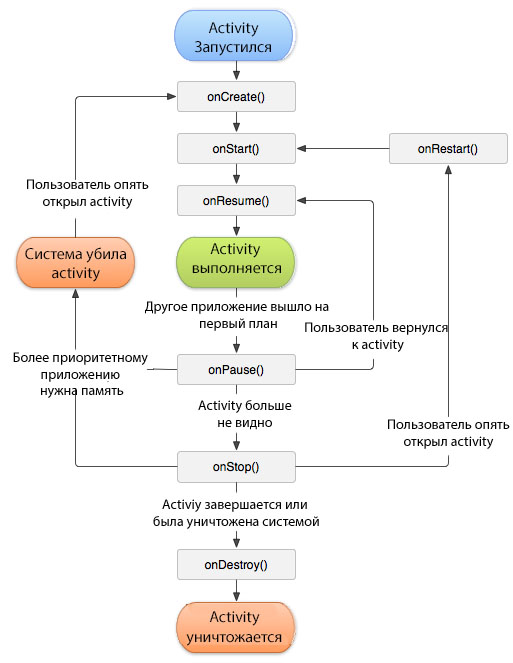
**При обмене данными поддерживаются все строки из name.txt, а так-же дополнительные:**

* -v#id#value – укороченная версия передачи переменной. В процессе игры значение переменной будет часто изменяться, хотя другие параметры переменной всегда остаются неизменными
* -p#id#role\_id#name – игрок с таким id выбрал такую роль и такое имя. (В начале игры все игроки должны определиться с ролью и именем)
* -b – игра началась
* -d#cur\_round– блок данных передан, нужно обновить все значения. В начале раунда происходит передача большого количества изменений, удобнее обновлять интерфейс, после принятия всего блока целиком. Cur\_round – номер раунда
* -i#tab\_id#list\_id#visible#N – передаётся соответствующий лист. Следующие N байт содержат изображение в специальном формате
* start\_time#begin#length – обновить таймер раунда. Begin – время начала таймера. Length – продолжительность раунда
* log\_journal#string – добавить в журнал действий строку

**Расширение формата передачи данных, для отправки действий:**

* send#action\_id#ans\_id#cur\_round – клиент передал действие хосту. Action\_id – номер действия. Ans\_id – номер ответа
* send#actin\_id#ans\_id#cur\_round#id#name#descr#ans#ans\_id – хост передал действие лидеру. Id – номер передавшего игрока. Name – имя передавшего игрока. Descr – описание действия. Ans – ответы действия. Ans\_id – номера ответов.
* -ka#id – убрать у лидера действие от игрока под номером id

# Жизненный цикл Activity



# Примеры обучающих игр

**Duolingo**

Duolingo – это приложение, которое позволяет в игровой форме учить иностранный язык. Исследование, проведённое профессорами [Городского университета Нью-Йорка](https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D1%80%D0%BE%D0%B4%D1%81%D0%BA%D0%BE%D0%B9_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82_%D0%9D%D1%8C%D1%8E-%D0%99%D0%BE%D1%80%D0%BA%D0%B0) и [Университета Южной Каролины](https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82_%D0%AE%D0%B6%D0%BD%D0%BE%D0%B9_%D0%9A%D0%B0%D1%80%D0%BE%D0%BB%D0%B8%D0%BD%D1%8B) [4], показало, что 34 часа на Duolingo дают столько же навыков чтения и письма, сколько даёт начальный [семестровый](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B5%D1%81%D1%82%D1%80) курс в американском высшем учебном заведении, занимающий около 130 часов

**Lumosity**

Lumosity – это приложение, которое представляет из себя набор «тренажёров», каждый из которых должен развивать какую-то часть мозга. В основном Lumosity состоит из игр, которые должны развивать память, концентрацию, аналитическое мышление и языковые возможности.

**Among Us**

Among Us – групповая игра на 4-10 игроков, которая является современным аналогом игры «Мафия. Игроки делятся на 2 команды: товарищи по команде и самозванцы. Задача товарищей по команде — выполнять задания по обслуживанию и ремонту космического корабля, а цель Самозванца — саботировать всю миссию до того, как шкала задач заполнится. В процессе игры игроки будут общаться друг с другом, голосовать за подозрительных игроков и выгонять их из игры.