Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Отчет по проекту

**Разработка телеграмм бота для решения игры «Water Sort Puzzle»**

|  |
| --- |
| Выполнил: студент гр. 5.306М:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лаптев А. В.  Проверил: доц. каф. ВТиЭ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Грязнов А. С.  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

**Целью** работы является создание телеграмм бота для решения игры «Water Sort Puzzle».

**Задачи**, которые были выделены для достижения поставленной цели:

1. Рассмотрение аналогичных решателей для данной игры и поиск недостатков.
2. Выбор средств и библиотек для разработки.
3. Определение функциональности собственного бота и разбиение проекта на модули.
4. Разработка алгоритма для каждого модуля и их реализация.
5. Отладка и тестирование работоспособности готового программного продукта.
6. Описание бота
   1. Недостатки существующих решений

В сети существует ряд подобных проектов, которые в основном представляют из себя Web-ресурсы, которые помогают пользователю ввести свои параметры для уровня (последовательность цветов в колбах), а также найти решение, в случае если оно есть.

Но большая часть из них несет в себе ряд недостатков:

1. Необходимость вручную заполнять файл со списком цветов и колб, что не является интуитивно понятным способом ввода для большинства пользователей.
2. Большое количество операций, которые выполняет пользователь для того, чтобы воссоздать свой уровень, что сказывается на длительности воссоздания уровня и невозможности быстро решить задачу.
3. Необходимость создания новых цветов, если нужных нет в списке предложенных (требуется указать название цвета в шестнадцатеричном представлении).
4. Невозможность точечного и быстрого исправления ошибок ввода (замена одного цвета в колбе может привести к необходимости либо перезаполнять всю колбу, либо сразу несколько колб).
5. Для одного из удачных аналогов, который реализован как телеграмм бот, есть проблема с распознаванием другой палитры цветов уровня, а также формы элементов внутри колб (бот предназначен для решения уровней конкретной игры и не является универсальным).

Все описанные выше недостатки вызвали необходимость для создания собственного решателя, который был бы интуитивно понятен для большинства пользователей и не требовал бы долгого времени на воссоздание и решение уровня.

В результате данных поисков была выбрана концепция, которая была наиболее удачно реализована в одном из аналогов, а именно − создание бота, который мог бы распознавать уровень по скриншоту пользователя и осуществлять поиск решения для него с доработками. Поскольку в игре появляются уровни, в которых будут неоткрытые цвета, то появилась необходимость реализовать дозаполнение этих цветов пользователем вручную. Список недостающих цветов будет автоматически сформирован и пользователю будет предложено нажимать на соответствующие кнопки для дозаполнения пробелов.

* 1. Выбор средств и библиотек для реализации проекта

Для реализации проекта были выбраны следующие составляющие:

* + программирования: Python 3;
  + среда разработки: VS Code;
  + библиотека для написания бота: Aiogram;
  + библиотека для распознавания уровней со скриншота: OpenCV.

Python 3 был выбран по личным предпочтениям, а также по причине того, что работа с библиотеками Aiogram и OpenCV хорошо описана для этого языка программирования, в том числе и в документации, и готовый код с использованием данных библиотек выглядит читабельнее, чем на C++.

IDE VS Code была выбрана по причине легковестности и несложной настройки для использования в связке с Python 3.

Библиотека Aiogram для взаимодействия с API Telegram и написания телеграмм бота была выбрана по причине своей популярности среди разработчиков, активной поддержки и доступно описанной документации.

Библиотека машинного зрения OpenCV была выбрана по тем же причинам.

* 1. Определение функциональности и разбиение решения на модули

Как уже было описано выше, в качестве концепции для разработки собственного решателя была выбрана концепция телеграмм бота с возможностью распознавания содержимого колб со скриншота пользователя и поиска решения по распознанным цветам.

Для реализации данного функционала было решено разбить проект на следующие модули:

1. Модуль бота − непосредственно реализация телеграмм бота и его взаимодействия с пользователем.
2. Модуль распознавания содержимого колб − этот модуль отвечает за все, что касается обработки скриншота отправленного пользователем для его преобразования и дальнейших манипуляций в ходе решения.
3. Модуль решения уровня − в этом модуле происходит поиск решения для обработанных данных с изображения пользователя.

Структура модуля бота представляет собой дерево файлов и каталогов, в которых реализована вся логика взаимодействия бота и пользователя.

Этот модуль состоит из следующих составляющих:

основной файл бота − файл в котором реализована инициализация и запуск бота, а также меню команд;

* директория classes включает в себя файл с инициализацией классов, которые использовались при создании бота (класс для логгирования и класс для хранения текущего состояния пользователя);
* директория keyboards содержит в себе файл, в котором реализованы все клавиатуры, которые используются для взаимодействия с пользователем (в том числе и динамически генерируемые);
* директория handlers содержит в себе набор файлов, в которых прописана основная логика взаимодействия с пользователем: нажатия кнопок, ввод текста или отправка скриншотов.

Модуль распознавания содержимого колб представляет собой скрипт, в котором прописана основная логика для предобработки исходного изображения, выделения колб и цветов внутри них, серилизации распознанных объектов в JSON файл. В этом модуле помимо OpenCV используется библиотека для работы с JSON объектами и библиотека линейной алгебры numpy.

В модуле для решения уровня используется только библиотека для работы с JSON объектами и в качестве метода решения используется собственная реализация итеративного метода для обхода графов в глубину с использованием стека. Этот метод был выбран за относительную простоту реализации и достаточную скорость поиска решения. В данной задаче было принято решение не использовать рекурсивный обход в качестве решения, поскольку для сложных игровых уровней глубина рекурсии превышала размер стека и решение не могло быть найдено, а также рекурсия даже на простых уровнях негативно сказывалась на скорости поиска решения (для некоторых уровней итерационный подход давал выигрыш по времени более чем в 5 раз). Подробнее алгоритм поиска решения будет описан ниже.

1. Алгоритм работы бота

Алгоритм работы бота представляет собой последовательную проверку действий пользователя в хэндлерах для правильной реакции на эти действия.

Последовательность действий, реализованная в боте, см. Приложение В:

* 1. Приветственное сообщение о назначении бота (доступна только кнопка старт), рис. 1.
  2. Сообщение с правилами использования бота и кнопкой «Start solving», рис. 2.
  3. Предложение пользователю загрузить фото и ожидание его загрузки пользователем, рис. 3.
  4. Попытка распознавания цветов с фото и вывод ответного сообщения с результатами распознавания (включая кнопки для заполнения неизвестных цветов вручную) для случаев перезагрузки изображения и после загрузки изображения, рис. 4, рис. 5.
  5. Ответное сообщение от бота с полностью распознанным изображением, рис. 6.
  6. В случае неудачного поиска решения выводится соответствующее сообщение и варианты действий, рис. 7.
  7. В случае удачного поиска выводится сообщение с решением для уровня и дальнейшие варианты действий, рис. 10.
  8. В случае если пользователь выбрал действие по загрузке нового изображения выводится сообщение с предложением загрузить фото, рис. 8.
  9. В случае желания пользователя добавить пустую колбу в ответное изображение будет добавлен пустой прямоугольник, ответное сообщение показано на рис. 9.

Кроме того, на рис. 11 показано всплывающее меню, которое было реализовано для удобства сброса генерации решения.

1. Алгоритм распознавания цветов

Алгоритм для распознавания цветов внутри колб работает в несколько этапов:

* 1. В основной функции осуществляется чтение исходного файла для распознавания, после чего происходит предобработка изображения для того, чтобы повысить точность распознавания с использованием машинного зрения (изображение обрезается, преобразуется в градации серого, накладывается размытие и осуществляется пороговая обработка);
  2. С помощью алгоритмов машинного зрения осуществляется поиск всех границ колб с цветами (пустые колбы добавляются позднее, так что в случае определения границ пустой колбы такая колба игнорируется в дальнейшем) и происходит выделение контура и сохранение колбы как отдельного временного файла;
  3. Алгоритм перебирает все найденные колбы (временные файлы) и определяет контуры границ цветов внутри них;
  4. Порядок цветов сортируется по убыванию координат по оси ординат, а непосредственно название цвета определяется путем сравнения кодировки цвета внутри контура с интервалом кодировок, полученным с использованием фильтра (для более точного распознавания цвет каждый файл с колбой проходит через наложение фильтра эррозии);
  5. Применяется пользовательская сортировка для того, чтобы расставить колбы в том же порядке, в каком они идут на изображении (счет колб идет слева направо и сверху вниз);
  6. После всех манипуляций добавляются две пустых колбы и удаляются все временные файлы (колбы по отдельности);
  7. Полученная последовательность списков с колбами сериализуется в JSON файл.

Помимо всего описанного выше в данном модуле реализованы еще несколько функций, которые требуются для взаимодействия с пользователем, а именно:

* 1. Функция замены неопределенного элемента в JSON файле выбранным пользователем цветом.
  2. Функция для отображения результата распознавания или замены неопределенных цветов (в виде изображения).
  3. Функция добавление в JSON еще одной пустой колбы, если пользователь выберет данный пункт меню в боте.

На выходе работы модуля получается JSON файл со списком цветов в колбах и генерируется изображение, которое является результатом распознавания скриншота пользователя.

1. Алгоритм решения уровня

Модуль для нахождения решения уровня состоит из 3 составляющих:

* 1. Чтение из JSON файла и десериализация структуры файла в список;
  2. Запись решения (если оно было найдено) в текстовый файл для последующей отправки пользователю;
  3. Поиск решения с использованием бэктрекинга (метод обхода графа в глубину).

Поиск решения осуществляется по следующему алгоритму:

Начало

* 1. В основной функции модуля вызывается функция десериализации и полученный список берется за стартовую позицию.
  2. Стартовая позиция передается в непосредственно функцию поиска решения. Результатом выполнения функции будет являться флаг решения (True − если решение найдено и False в противном случае), а также список шагов решения, если решение найдено.
  3. Если флаг решения − True, то вызывается функция для записи результата в файл.
  4. В функции решения инициализируется стек для хранения состояний которые нужно пройти и списка шагов, в который добавляется стартовая позиция, и множество для хранения всех посещенных состояний.

Цикл

* 1. Пока в стеке есть элементы поиск решения продолжается, если элементов не осталось, то решение не было найдено.
  2. Если в ходе решения повторно было получено стартовое состояние, то решения не было найдено.
  3. Происходит добавление состояния в множество посещенных.
  4. Вызывается функция для проверки решения задачи. Если задача решена, то возвращается истинный флаг решения и список шагов, который передается в основную функцию (переход к п. 3).
  5. Вызывается функция в которой осуществляется поиск всех возможных шагов из текущего положения согласно правилам игры. Если список шагов пустой, то вызывается функция для отмены текущего движения (переход к предыдущей позиции).

Цикл

* 1. Для текущей позиции применяем перемещение.
  2. Проверяем новую позицию среди уже посещенных. Если позиция была посещена и движение было последним в списке, то удаляется позиция из стека и отменятся движение, иначе только отменяется действие.
  3. Добавление текущей позиции в множество посещенных и добавление шага в список шагов.
  4. Прерывание обхода по движениям, чтобы продвигаться в стеке далее в глубину.

Конец цикла

Конец цикла

Конец

Приложение А (программный код)

Листинг 1. Программный код основного файла бота.

from aiogram import Bot, Dispatcher  # Подключение библиотек

from aiogram.types import BotCommand

from aiogram.exceptions import TelegramNetworkError

from handlers import send\_welcome, start\_solving, get\_photo, fill\_undef\_values

import config

import classes.all\_my\_classes as amc

import asyncio

import shutil

import os

"""

Список поддерживаемых команд (желательно все сделать в виде кнопок):

/start - начало работы с ботом

"""

async def clue(bot: Bot):

    # Реализация меню команд

    bot\_commands = [

        BotCommand(command='/start', description='Restarting me and receiving instructions for working with me')

    ]

    await bot.set\_my\_commands(bot\_commands)

    # Реализация описания бота

    await bot.set\_my\_description("Hello, I'm the Alchemist!🧪🧑‍🔬🧪🧑‍🔬🧪🧑‍🔬\nI can help you transfer the different colored liquids into your flasks so that you get flasks with liquids filtered by color.\nI can work with pictures so you don't have to fill the flasks completely by hand, and I can also change the level a little by adding an empty flask if your level cannot be solved with two empty flasks😊")

    await bot.set\_my\_short\_description("Alchemist - telegram bot for solving your levels for games with transfusion of colored liquids")

async def main():

    """Главная функция с инициализацией бота"""

    # Определяем количество свободного пространства на диске в Гб

    if os.name == 'nt':

        free\_space = shutil.disk\_usage('C:/').free / 10\*\*9

    elif os.name == 'posix':

        free\_space = shutil.disk\_usage('/dev/sda').free / 10\*\*9

    logger = amc.ConfigLogger(\_\_name\_\_)

    # Логгируем предупреждение, если свободного места меньше 5 Гб

    if free\_space < 5:

        logger.log\_warning(f'Заканчивается свободное место на диске, осталось свободно: {free\_space} Гб')

    # Инициализация диспетчера

    dp = Dispatcher()

    bot = Bot(token=config.API\_TOKEN)

    dp.startup.register(clue)

    dp.include\_routers(send\_welcome.rtr, start\_solving.rtr, get\_photo.rtr, fill\_undef\_values.rtr)

    try:

        await bot.delete\_webhook(drop\_pending\_updates=True)

        await dp.start\_polling(bot)

    except TelegramNetworkError:

        logger.log\_error('Ошибка подключения клиента к API')

if \_\_name\_\_ == '\_\_main\_\_':

    """Запуск"""

    asyncio.run(main())

Листинг 2. Программный код приветственного хэндлера.

from aiogram import Router  # Подключение библиотек

from aiogram.filters import CommandStart

from aiogram.types import Message

from aiogram.fsm.context import FSMContext

import classes.all\_my\_classes as amc

from keyboards.all\_my\_keyboards import start\_keyboard

import os

rtr = Router()

logger = amc.ConfigLogger(\_\_name\_\_)

@rtr.message(CommandStart())  # Команда для начала работы с ботом

async def send\_welcome(message: Message,  state: FSMContext):

    """Приветственная функция"""

    await state.clear()

    logger.log\_info(f'Пользователем {message.from\_user.id} был запущен или перезапущен бот')

    # Создаем папку с именем пользователя, который использует бота, если ее не существовало

    if not os.path.isdir(f'./{message.from\_user.id}'):

        os.mkdir(f'./{message.from\_user.id}')

    await message.answer(

        f"Hello, <b>{message.from\_user.first\_name}</b>!😁\nNow I'll tell you a little more about myself so that you know how to interact with me correctly. This is very important, because if you follow these few simple rules, you will get more accurate recognition of the colors inside the flasks, as well as the correct solutions for your specific level\nNow a few words about the functionality:\n✔️When you upload a screenshot, you need to upload it as a picture, not as a file, that is, in a chat with me, I should see your message as a full-fledged picture, about half the screen\n✔️The screenshot does not need to be cropped or compressed in any way, I will do it myself, so just send me the original image\n✔️<u><b>IF YOU USUALLY USE PATTERN MODE, PLEASE TURN IT OFF BEFORE TAKING A SCREENSHOT</b></u> and sending it to me, I'm not good at recognizing shapes inside colored rectangles, so I won't be able to recognize your level accurately (maybe I'll learn this in the future!)\n✔️Upload me an image with the initial position of the colors in the flasks (that is, 2 empty flasks and the rest are completely filled), this is the only way I can find a solution\nThat's probably all the subtleties that I wanted to tell you about working with me, good luck!🤞🤞🤞\n\nTo restart me, you can enter the /start command.",

        parse\_mode='HTML',

        reply\_markup=start\_keyboard()

    )

await state.set\_state(amc.SolveFlasks.start\_solving)

Листинг 3. Программный код хэндлера для кнопки Start solving.

from aiogram import Router, F  # Подключение библиотек

from aiogram.types import Message, CallbackQuery

from aiogram.fsm.context import FSMContext

import classes.all\_my\_classes as amc

import asyncio

rtr = Router()

logger = amc.ConfigLogger(\_\_name\_\_)

@rtr.callback\_query(

    amc.SolveFlasks.start\_solving,

    F.data.in\_(

        [

            'start\_solving', 'upload\_new\_image'

        ]

    )

)    #   Команды выбора режима распознавания

async def start\_solving(callback: CallbackQuery, state: FSMContext):

    """Функция загрузки изображения"""

    if callback.data == 'start\_solving':

        logger.log\_info(f'Пользователь {callback.from\_user.id} приступил к загрузке первого изображения')

        await callback.message.answer("So let's get started😎\nUpload the screenshot as an image, please")

        await callback.answer()

        await state.set\_state(amc.SolveFlasks.send\_photo)

    elif callback.data == 'upload\_new\_image':

        logger.log\_info(f'Пользователь {callback.from\_user.id} приступил к загрузке нового изображения')

        await callback.message.answer('Upload a new screenshot as an image, please')

        await callback.answer()

        await state.set\_state(amc.SolveFlasks.send\_photo)

@rtr.message(amc.SolveFlasks.start\_solving)

async def start\_solving\_incorrectly(message: Message):

    '''Функция для отслеживания любых действий кроме нажатия кнопки'''

    logger.log\_info(f'Пользователь {message.from\_user.id} ввел неверную команду перед загрузкой изображения')

    msg = await message.answer('To get started, click the "🚀Start solving" button, please')

    await asyncio.sleep(10)

    await message.delete()

await msg.delete()

Листинг 4. Программный код для хэндлера получающего скриншот.

from aiogram import Bot, Router, F  # Подключение библиотек

from aiogram.types import Message, BufferedInputFile

from aiogram.fsm.context import FSMContext

from aiogram.exceptions import TelegramBadRequest

from transfusion\_of\_liquids import transfusion\_manage

from found\_colors import found\_colors\_in\_flasks, create\_image\_for\_replace

import classes.all\_my\_classes as amc

from keyboards.all\_my\_keyboards import error\_image, colors, feedback, upload\_new, no\_result

import asyncio

import os

rtr = Router()

logger = amc.ConfigLogger(\_\_name\_\_)

@rtr.message(

    amc.SolveFlasks.send\_photo,

    F.photo

)

async def get\_photo(message: Message, bot: Bot, state: FSMContext):

    '''Функция получения и обработки фотографий'''

    # Создание папки со скриншотами и папки для временных изображений в папке с id пользователя

    this\_path = f'./{message.from\_user.id}'

    if not os.path.isdir(f'{this\_path}/images'):

        os.mkdir(f'{this\_path}/images')

    if not os.path.isdir(f'{this\_path}/tmp'):

        os.mkdir(f'{this\_path}/tmp')

    image\_for\_load = f'{this\_path}/images/{message.photo[-1].file\_id}.jpg'   # Сохраняем на всякий случай путь к картинке

    in\_file, out\_file = f"{this\_path}/tmp/start\_level\_{message.from\_user.id}.json", f"{this\_path}/tmp/result\_level\_{message.from\_user.id}.txt"

    lvl\_file = f'{this\_path}/tmp/level\_for\_{message.from\_user.id}.jpg'

    # Сохраняем пути в машину состояний

    await state.update\_data(original\_image=image\_for\_load)

    await state.update\_data(input\_file=in\_file)

    await state.update\_data(output\_file=out\_file)

    await state.update\_data(level\_file=lvl\_file)

    # Загрузка фото в буфер для последующей обработки

    await bot.download(

        message.photo[-1],

        destination=image\_for\_load

    )

    await message.answer("I'll try to recognize colors in the photo")

    logger.log\_info(f'Пользователь {message.from\_user.id} отправил фото')

    try:

        # Распознаем цвета и добавляем их в список с последующей сериализации в json

        undef\_colors = found\_colors\_in\_flasks(image\_for\_search=image\_for\_load, id\_client=message.from\_user.id, reload\_image=False)

        await state.update\_data(undefined\_colors=undef\_colors)

    except:

        # Если есть любое прерывание во время распознавания, то просим пользователя загрузить новое фото

        # (генерация прерывания говорит о том, что фото не является скриншотом колб или не соответствует условиям)

        await message.answer(

            'Something went wrong...🤷‍♂️ Please upload another picture',

            reply\_markup=error\_image()

        )

        logger.log\_error('Изображение не подходит для распознавания')

        await state.set\_state(amc.SolveFlasks.start\_solving)

    if undef\_colors:

        # Подготавливаем картинку, в которой подсвечиваем неопределенные области

        create\_image\_for\_replace(json\_name=in\_file, id\_client=message.from\_user.id)

        # Изображение, где подсвечивается первый неопределенный цвет

        with open(lvl\_file, 'rb') as open\_image:

            await message.answer\_photo(

                BufferedInputFile(

                    open\_image.read(),

                    filename='solve\_flasks'

                ),

                caption="Please check if I recognized everything correctly? If I misrecognized some colors or you noticed some other error, then feel free to let me know about the problem\nTo do this, click the button below the message (send a photo with which the error occurred and describe the problem)🙂",

                reply\_markup=feedback()

            )

        await message.answer(

            'Please select from the options provided the color that should be in place of the green circle',

            reply\_markup=colors(undef\_colors)

        )

        logger.log\_info(f'Изображение для пользователя {message.from\_user.id} успешно создано и готово для редактирования')

        await state.set\_state(amc.SolveFlasks.set\_color)

    else:

        # Формируем итоговый json

        create\_image\_for\_replace(json\_name=in\_file, id\_client=message.from\_user.id)

        # Итоговое изображение

        with open(lvl\_file, 'rb') as open\_image:

            await message.answer\_photo(

                BufferedInputFile(

                    open\_image.read(),

                    filename='solve\_flasks'

                ),

                caption="I'll look for a solution from this position. Wait, this may take a while"

            )

        logger.log\_info(f'Пользователь {message.from\_user.id} заполнил все пустоты')

        try:

            # Вызываем функцию перебора переливаний

            is\_solved = transfusion\_manage(task=in\_file, result=out\_file)

        except TelegramBadRequest:

            logger.log\_error('Превышено время ожидания ответа на начало поиска решения')

        # В случае, если файл пустой или не был создан сообщаем, что решение не найдено, иначе выводим решение

        if not is\_solved:

            await message.answer(

                f"😖😖😖Unfortunately, I was unable to find a solution for this arrangement.\nIf you want to change the order of undefined colors, click '🔄️🖼️Reload image'.\nIf you know all the colors, but the solution still hasn't been found, then I can add another empty flask, to do this, click '➕🧪Add an empty flask'\nOr you can upload a new image, to do this, click '📩🖼️Upload new image'",

                reply\_markup=no\_result()

            )

            await state.set\_state(amc.SolveFlasks.set\_color)

        else:

            with open(out\_file, "r") as result:

                await message.answer(

                    f'Yay!🥳🥳🥳I found a solution for you!!!🥳🥳🥳\n{result.read()}\nLet me know if you want a solution for another screenshot🙂',

                    reply\_markup=upload\_new()

                )

            await state.set\_state(amc.SolveFlasks.start\_solving)

        # Удаление временных файлов

        if os.path.isfile(out\_file):

            os.remove(out\_file)

        os.remove(in\_file)

        os.remove(lvl\_file)

@rtr.message(amc.SolveFlasks.send\_photo)

async def sending\_photo\_incorrectly(message: Message):

    '''Функция для отслеживания любых действий кроме отправки фото'''

    logger.log\_info(f'Пользователь {message.from\_user.id} отправил что-то кроме фото')

    msg = await message.answer('Send a photo please')

    await asyncio.sleep(10)

    await message.delete()

await msg.delete()

Листинг 5. Программный код для хэндлера последующей обработки изображения.

from aiogram import Router, F  # Подключение библиотек

from aiogram.types import Message, CallbackQuery, BufferedInputFile

from aiogram.fsm.context import FSMContext

from aiogram.exceptions import TelegramBadRequest

from transfusion\_of\_liquids import transfusion\_manage

from found\_colors import found\_colors\_in\_flasks, replace\_in\_json, create\_image\_for\_replace, add\_empty\_flask

import config

import classes.all\_my\_classes as amc

from keyboards.all\_my\_keyboards import error\_image, colors, feedback, upload\_new, no\_result

import asyncio

import os

rtr = Router()

logger = amc.ConfigLogger(\_\_name\_\_)

@rtr.callback\_query(

    amc.SolveFlasks.set\_color,

    F.data.in\_(

        [

            "LIGHT BLUE", "ORANGE", "YELLOW", "RED", "LIGHT GREEN", "BLUE", "BURGUNDY",

            "GREEN", "PINK", "CRIMSON", "CREAM", "PURPLE", "GRAY","LILAC",

            'reload\_image', 'add\_an\_empty\_flask', 'upload\_new\_image'

        ]

    )

)

async def fill\_undef\_values(callback: CallbackQuery, state: FSMContext):

    '''Функция дозаполнения неопределенных цветов вручную'''

    if callback.data == 'upload\_new\_image':

        await callback.message.answer('Upload a new screenshot as an image, please')

        await callback.answer()

        await state.set\_state(amc.SolveFlasks.send\_photo)

        return

    # Получаем данные с путями к папкам

    paths = await state.get\_data()

    image\_for\_load = paths['original\_image']

    in\_file, out\_file = paths['input\_file'], paths['output\_file']

    lvl\_file = paths['level\_file']

    if callback.data == 'reload\_image' or callback.data == 'add\_an\_empty\_flask':

        logger.log\_info(f'Изображение от пользователя {callback.from\_user.id} отправлено на перезагрузку с/без добавления пустой колбы')

        try:

            # Распознаем цвета и добавляем их в список с последующей сериализации в json

            undef\_colors = found\_colors\_in\_flasks(image\_for\_search=image\_for\_load, id\_client=callback.from\_user.id, reload\_image=True)

            await state.update\_data(undefined\_colors=undef\_colors)

        except:

            # Если есть любое прерывание во время распознавания, то просим пользователя загрузить новое фото

            # (генерация прерывания говорит о том, что фото не является скриншотом колб или не соответствует условиям)

            await callback.message.answer(

                'Something went wrong...🤷‍♂️ Please upload another picture',

                reply\_markup=error\_image()

            )

            logger.log\_error('Изображение не подходит для распознавания')

            await callback.answer()

            await state.set\_state(amc.SolveFlasks.start\_solving)

    else:

        undef\_colors = paths['undefined\_colors']

    if callback.data != 'reload\_image' and callback.data != 'add\_an\_empty\_flask':

        # Удаление цвета нажатой кнопки из словаря и замена неопределенного цвета цветом кнопки

        if undef\_colors:

            for variation in config.color\_variations:

                if callback.data == variation:

                    if variation == 'LIGHT GREEN':

                        undef\_colors['LIGHTGREEN'] -= 1

                        if undef\_colors['LIGHTGREEN'] == 0:

                            undef\_colors.pop('LIGHTGREEN')

                        replace\_in\_json(json\_name=in\_file, color\_name='LIGHTGREEN')

                        break

                    elif variation == 'LIGHT BLUE':

                        undef\_colors['LIGHTBLUE'] -= 1

                        if undef\_colors['LIGHTBLUE'] == 0:

                            undef\_colors.pop('LIGHTBLUE')

                        replace\_in\_json(json\_name=in\_file, color\_name='LIGHTBLUE')

                        break

                    else:

                        undef\_colors[variation] -= 1

                        if undef\_colors[variation] == 0:

                            undef\_colors.pop(variation)

                        replace\_in\_json(json\_name=in\_file, color\_name=variation)

                        break

    if undef\_colors:

        if callback.data == 'add\_an\_empty\_flask':

            # Добавляем пустую колбу

            add\_empty\_flask(json\_name=in\_file)

            logger.log\_info(f'В изображение пользователя {callback.from\_user.id} была добавлена пустая колба')

        # Подготавливаем картинку, в которой подсвечиваем неопределенные области

        create\_image\_for\_replace(json\_name=in\_file, id\_client=callback.from\_user.id)

        if callback.data == 'reload\_image' or callback.data == 'add\_an\_empty\_flask':

            # Изображение, где подсвечивается первый неопределенный цвет

            with open(lvl\_file, 'rb') as open\_image:

                await callback.message.answer\_photo(

                    BufferedInputFile(

                        open\_image.read(),

                        filename='solve\_flasks'

                    ),

                    caption="Please check if I recognized everything correctly? If I misrecognized some colors or you noticed some other error, then feel free to let me know about the problem\nTo do this, click the button below the message (send a photo with which the error occurred and describe the problem)🙂",

                    reply\_markup=feedback()

                )

            await callback.message.answer(

                'Please select from the options provided the color that should be in place of the green circle',

                reply\_markup=colors(undef\_colors)

            )

            await callback.answer()

            logger.log\_info(f'Изображение для пользователя {callback.from\_user.id} перезагружено для дальнейшего редактирования')

        else:

            # Изображение, где подсвечивается первый неопределенный цвет

            await callback.message.delete()

            with open(lvl\_file, 'rb') as open\_image:

                await callback.message.answer\_photo(

                    BufferedInputFile(

                        open\_image.read(),

                        filename='solve\_flasks'

                    ),

                    caption="Please select from the options provided the color that should be in place of the green circle",

                    reply\_markup=colors(undef\_colors)

                )

            await callback.answer()

            logger.log\_info(f'Изображение для пользователя {callback.from\_user.id} дополнено и отправлено для дальнейшего редактирования')

    else:

        # Формируем итоговый json

        create\_image\_for\_replace(json\_name=in\_file, id\_client=callback.from\_user.id)

        # Итоговое изображение

        await callback.message.delete()

        with open(lvl\_file, 'rb') as open\_image:

            await callback.message.answer\_photo(

                BufferedInputFile(

                    open\_image.read(),

                    filename='solve\_flasks'

                ),

                caption="I'll look for a solution from this position. Wait, this may take a while"

            )

        await callback.answer()

        logger.log\_info(f'Пользователь {callback.from\_user.id} заполнил все пустоты')

        try:

            # Вызываем функцию перебора переливаний

            is\_solved = transfusion\_manage(task=in\_file, result=out\_file)

        except TelegramBadRequest:

            logger.log\_error('Превышено время ожидания ответа на начало поиска решения')

        # В случае, если файл пустой или не был создан сообщаем, что решение не найдено, иначе выводим решение

        if not is\_solved:

            await callback.message.answer(

                f'😖😖😖Unfortunately, I was unable to find a solution for this arrangement.\nIf you want to change the order of undefined colors, click "🔄️🖼️Reload image".\nIf you know all the colors, but the solution still hasn’t been found, then I can add another empty flask, to do this, click “➕🧪Add an empty flask”\nOr you can upload a new image, to do this, click "📩🖼️Upload new image"',

                reply\_markup=no\_result()

            )

            await state.set\_state(amc.SolveFlasks.set\_color)

        else:

            with open(out\_file, "r") as result:

                await callback.message.answer(

                    f'Yay!🥳🥳🥳I found a solution for you!!!🥳🥳🥳\n{result.read()}\nLet me know if you want a solution for another screenshot :)',

                    reply\_markup=upload\_new()

                )

            await state.set\_state(amc.SolveFlasks.start\_solving)

        # Удаление временных файлов

        if os.path.isfile(out\_file):

            os.remove(out\_file)

        os.remove(in\_file)

        os.remove(lvl\_file)

@rtr.message(amc.SolveFlasks.set\_color)

async def filling\_incorrectly(message: Message):

    '''Функция для отслеживания любых действий кроме заполнения цветом'''

    logger.log\_info(f'Пользователь {message.from\_user.id} проигнорировал кнопки')

    msg = await message.answer('Please select a color from above or choose what to do with the image')

    await asyncio.sleep(10)

    await message.delete()

await msg.delete()

Листинг 6. Содержимое файла с глобальными переменными.

API\_TOKEN = 'TOKEN'  # Токен для работы с API

# Вариации цветов

color\_variations = [

    'LIGHT BLUE', 'ORANGE', 'YELLOW', 'RED', 'LIGHT GREEN', 'BLUE', 'BURGUNDY',

    'GREEN', 'PINK', 'CRIMSON', 'CREAM', 'PURPLE', 'GRAY', 'LILAC'

]

Листинг 7. Программный код клавиатур.

from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup

import classes.all\_my\_classes as amc

logger = amc.ConfigLogger(\_\_name\_\_)

def create\_undef\_buttons(color\_buttons\_list):

    '''Функция для расстановки кнопок с цветами'''

    color\_buttons, button\_line = [], []

    # "Красивая" расстановка кнопок

    for i in range(len(color\_buttons\_list)):

        if i % 3 == 0 and i == len(color\_buttons\_list) - 1:

            color\_buttons.append(button\_line)

            button\_line = []

            button\_line.append(color\_buttons\_list[i])

            color\_buttons.append(button\_line)

        elif i % 3 == 0 and i != 0:

            color\_buttons.append(button\_line)

            button\_line = []

            button\_line.append(color\_buttons\_list[i])

        elif i == len(color\_buttons\_list) - 1:

            button\_line.append(color\_buttons\_list[i])

            color\_buttons.append(button\_line)

        else:

            button\_line.append(color\_buttons\_list[i])

    logger.log\_info('Расстановка кнопок в правильном порядке')

    return color\_buttons

def start\_keyboard():

    start\_button = [

        [

            InlineKeyboardButton(text='🚀Start solving', callback\_data='start\_solving')

        ]

    ]

    kb = InlineKeyboardMarkup(inline\_keyboard=start\_button)

    logger.log\_info('Вывод приветствия')

    return kb

def error\_image():

    reload\_img = [

        [

            InlineKeyboardButton(text='📩🖼️Upload new image', callback\_data='upload\_new\_image')

        ]

    ]

    kb = InlineKeyboardMarkup(inline\_keyboard=reload\_img)

    return kb

def colors(undef\_colors):

    color\_buttons\_list = []

    # Создание списка кнопок с цветмаи, которыми можно будет заменить неопределенные значения

    for color in undef\_colors.keys():

        for \_ in range(undef\_colors[color]):

            if color == 'LIGHTGREEN':

                color\_buttons\_list.append(InlineKeyboardButton(text='LIGHT GREEN', callback\_data='LIGHT GREEN'))

            elif color == 'LIGHTBLUE':

                color\_buttons\_list.append(InlineKeyboardButton(text='LIGHT BLUE', callback\_data='LIGHT BLUE'))

            else:

                color\_buttons\_list.append(InlineKeyboardButton(text=color, callback\_data=color))

    color\_buttons = create\_undef\_buttons(color\_buttons\_list)

    kb = InlineKeyboardMarkup(inline\_keyboard=color\_buttons)

    logger.log\_info('Составлен список неопределенных кнопок')

    return kb

def feedback():

    feedback\_button = [

        [

            InlineKeyboardButton(text='Feedback to me🙃', url=f"tg://user?id={984089348}")

        ]

    ]

    kb = InlineKeyboardMarkup(inline\_keyboard=feedback\_button)

    return kb

def upload\_new():

    upload\_new\_button = [

        [

            InlineKeyboardButton(text='📩🖼️Upload new image', callback\_data='upload\_new\_image')

        ]

    ]

    kb = InlineKeyboardMarkup(inline\_keyboard=upload\_new\_button)

    logger.log\_info('Результат успешно найден')

    return kb

def no\_result():

    no\_result\_button = [

        [

            InlineKeyboardButton(text='🔄️🖼️Reload image', callback\_data='reload\_image'),

            InlineKeyboardButton(text='➕🧪Add an empty flask', callback\_data='add\_an\_empty\_flask')

        ],

        [

            InlineKeyboardButton(text='📩🖼️Upload new image', callback\_data='upload\_new\_image')

        ]

    ]

    kb = InlineKeyboardMarkup(inline\_keyboard=no\_result\_button)

    logger.log\_info('Результат не найден')

    return kb

Листинг 8. Программный код инициализации классов.

from aiogram.fsm.state import StatesGroup, State

import logging

from logging.handlers import RotatingFileHandler

import os

class ConfigLogger:

    '''Класс логгирования'''

    def \_\_init\_\_(self, filename) -> None:

        # Создание папки для логов

        if not os.path.isdir('./logs'):

            os.mkdir('./logs')

        self.logger = logging.getLogger(filename)

        self.logger.setLevel(logging.INFO)

        handler = logging.FileHandler(f'./logs/{filename}.log', 'a')

        handler = RotatingFileHandler(f'./logs/{filename}.log', maxBytes=1e6, backupCount=6)

        formatter = logging.Formatter('%(name)s\t%(asctime)s\t%(levelname)s\t%(message)s')

        handler.setFormatter(formatter)

        self.logger.addHandler(handler)

    def log\_message(self, level, message):

        if level == 'info':

            self.logger.info(message)

        elif level == 'warning':

            self.logger.warning(message, exc\_info=True)

        elif level == 'error':

            self.logger.error(message, exc\_info=True)

    def log\_info(self, message):

        self.log\_message('info', message)

    def log\_warning(self, message):

        self.log\_message('warning', message)

    def log\_error(self, message):

        self.log\_message('error', message)

class SolveFlasks(StatesGroup):

    '''Класс для машины состояний'''

    start\_solving = State()

    send\_photo = State()

set\_color = State()

Листинг 9. Программный код для модуля распознавания цветов в колбах.

'''Этот модуль отвечает за поиск и распознавание цветов в каждой колбе'''

import cv2

import numpy as np

import json

import os

class BreakAction(Exception):

    pass

variations = [

    ('LIGHTBLUE', (np.array((30, 50, 210), np.uint8), np.array((106, 255, 255), np.uint8)), (224, 161, 103)),

    ('ORANGE', (np.array((0, 165, 203), np.uint8), np.array((19, 255, 255), np.uint8)), (68, 144, 226)),

    ('YELLOW', (np.array((22, 46, 192), np.uint8), np.array((34, 255, 255), np.uint8)), (64, 185, 227)),

    ('RED', (np.array((0, 148, 114), np.uint8), np.array((7, 255, 255), np.uint8)), (30, 43, 173)),

    ('LIGHTGREEN', (np.array((41, 0, 160), np.uint8), np.array((65, 255, 255), np.uint8)), (70, 187, 108)),

    ('BLUE', (np.array((103, 181, 135), np.uint8), np.array((120, 255, 255), np.uint8)), (207, 90, 39)),

    ('BURGUNDY', (np.array((158, 135, 84), np.uint8), np.array((255, 255, 127), np.uint8)), (53, 32, 95)),

    ('GREEN', (np.array((86, 121, 86), np.uint8), np.array((96, 255, 255), np.uint8)), (100, 97, 46)),

    ('PINK', (np.array((140, 0, 197), np.uint8), np.array((154, 255, 255), np.uint8)), (219, 153, 212)),

    ('CRIMSON', (np.array((140, 88, 183), np.uint8), np.array((195, 255, 255), np.uint8)), (128, 109, 218)),

    ('CREAM', (np.array((0, 0, 241), np.uint8), np.array((20, 255, 255), np.uint8)), (194, 218, 248)),

    ('PURPLE', (np.array((131, 157, 186), np.uint8), np.array((255, 255, 255), np.uint8)), (201, 64, 132)),

    ('GRAY', (np.array((0, 0, 94), np.uint8), np.array((255, 29, 116), np.uint8)), (109, 107, 106)),

    ('LILAC', (np.array((117, 155, 136), np.uint8), np.array((125, 255, 255), np.uint8)), (187, 62, 71))

]

def preprocessing\_image(image):

    '''Функция предобработки изображения'''

    # Чтение обрезанного изображения в ч/б формате

    gray\_noise = cv2.imread(image, 0)

    # Размытие фона для ч/б изображения

    blurred = cv2.GaussianBlur(

        gray\_noise,

        (5, 5),

        0

    )

    # Пороговая обработка изображения

    thresholder = cv2.threshold(

        blurred,

        68,

        255,

        cv2.THRESH\_BINARY

    )[1]

    return thresholder

def found\_rect(contour, my\_list, coeff\_width, coeff\_height, is\_flask=False):

    '''Функция распознавания прямоугольника'''

    rect = cv2.minAreaRect(contour)

    box = np.intp(cv2.boxPoints(rect))

    if is\_flask ==True:

        if rect[1][0] >= coeff\_height and rect[1][1] >= coeff\_height:

            my\_list.append(rect)

    else:

        if (rect[1][0] >= rect[1][1] and rect[1][1] >= coeff\_width and rect[1][0] >= coeff\_height) or \

            (rect[1][0] < rect[1][1] and rect[1][0] >= coeff\_width and rect[1][1] >= coeff\_height):

            # Добавляем прямоугольники с колбами в список

            my\_list.append(rect)

    return my\_list, box

def crop\_rects(contours, cropped\_image, id\_client):

    '''Функция для выделения каждой отдельной колбы или цвета в ней для распознавания цветов'''

    flasks\_info = []

    for cnt in contours:

        filename = f'./{id\_client}/tmp/flask\_{cnt}.jpg'

        # Взаимодействие с колбой

        if cnt[2] > 45:

            height\_flask = [round(cnt[0][1] - cnt[1][0] / 2), round(cnt[0][1] + cnt[1][0] / 2)]

            width\_flask = [round(cnt[0][0] - cnt[1][1] / 2), round(cnt[0][0] + cnt[1][1] / 2)]

        else:

            height\_flask = [round(cnt[0][1] - cnt[1][1] / 2), round(cnt[0][1] + cnt[1][1] / 2)]

            width\_flask = [round(cnt[0][0] - cnt[1][0] / 2), round(cnt[0][0] + cnt[1][0] / 2)]

        flask = cropped\_image[height\_flask[0]:height\_flask[1], width\_flask[0]:width\_flask[1]]

        cv2.imwrite(filename, flask)

        flasks\_info.append((filename, (width\_flask[1] - width\_flask[0], height\_flask[1] - height\_flask[0])))

    return flasks\_info

def create\_color\_list(image):

    '''Функция для создания списка колб с цветами вместо числовых значений'''

    # Более агрессивный подход для удаления ненужных шумов с изображения с использованием эрозии

    morph\_kernel = np.ones((3, 3))

    erode\_image = cv2.erode(cv2.imread(image), kernel=morph\_kernel, iterations=3)

    cv2.imwrite(image, erode\_image)

    color\_pixels = cv2.imread(image)

    height, width, \_ = color\_pixels.shape

    hsv\_colors = cv2.cvtColor(color\_pixels, cv2.COLOR\_BGR2HSV)

    colors\_info, count\_colors = [], []

    # Подбор коэффициентов

    coeff\_width = round(width / 1.5)

    coeff\_height = round(height / 6.8)

    for variation in variations:

        # Проверяем пороговое значение для каждой вариации цвета на картинке и находим контуры

        thresholder = cv2.inRange(hsv\_colors, variation[1][0], variation[1][1])

        contours\_color, \_ = cv2.findContours(

            thresholder,

            cv2.RETR\_TREE,

            cv2.CHAIN\_APPROX\_SIMPLE

        )

        if contours\_color:

            # В случае если контур был найден, определяем координаты и размеры прямоугольника с цветом

            color = []

            for cnt in contours\_color:

                color\_coords, \_ = found\_rect(cnt, color, coeff\_width, coeff\_height, is\_flask=True)

            for cnt in color\_coords:

                # Исключаем наложение прямоугольников друг на друга

                add\_flag = True

                if len(colors\_info) > 0:

                    for add\_color in colors\_info:

                        if abs(cnt[0][1] - add\_color[1][1]) < 38:

                            add\_flag = False

                            break

                if add\_flag == True:

                    # Добавляем в список информацию о цвете и его местоположении

                    color\_name = variation[0]

                    colors\_info.append([color\_name, cnt[0], cnt[1]])

                    count\_colors.append([variation[0]])

    # Поддержка 2 и более цветов друг за другом

    if len(colors\_info) > 1:

        min\_color\_rect = min(

            colors\_info,

            key=lambda

            item:

            item[2][0]

        )[2][0]

        idx\_line = 0

        while idx\_line < len(colors\_info):

            if colors\_info[idx\_line][2][0] > 2.75 \* min\_color\_rect:

                for \_ in range(2):

                    colors\_info.insert(idx\_line, colors\_info[idx\_line])

                idx\_line += 2

            elif colors\_info[idx\_line][2][0] > 1.75 \* min\_color\_rect:

                colors\_info.insert(idx\_line, colors\_info[idx\_line])

                idx\_line += 2

            else:

                idx\_line += 1

    # Не учитываем пустые списки (пустые колбы будут добавляться отдельно)

    if len(colors\_info) < 4 and len(colors\_info) > 0:

        # Добавляем неопределившиеся значения список цветов в колбе

        for \_ in range(4 - len(colors\_info)):

            colors\_info.append(['UNDEFINED', (0, height)])

    return colors\_info

def sorted\_flasks(flasks\_list):

    '''Пользовательская функция для сортировки колб в нужном порядке'''

    min\_coord = min(

        flasks\_list,

        key=lambda

        item:

        item[0][1]

    )[0][1]

    max\_flask\_height = max(

        flasks\_list,

        key=lambda

        item:

        item[1][0]

    )[1][0]

    layer\_height = max\_flask\_height \* 1.55

    sorted\_flask\_list, layer\_1, layer\_2, layer\_3 = [], [], [], []

    for coord\_flask in flasks\_list:

        number\_layer = round((coord\_flask[0][1] - min\_coord) / layer\_height)

        if number\_layer == 0:

            layer\_1.append(coord\_flask)

        elif number\_layer == 1:

            layer\_2.append(coord\_flask)

        elif number\_layer == 2:

            layer\_3.append(coord\_flask)

    if len(layer\_1) != 0:

        layer\_1 = sorted(layer\_1)

    if len(layer\_2) != 0:

        layer\_2 = sorted(layer\_2)

    if len(layer\_3) != 0:

        layer\_3 = sorted(layer\_3)

    for element in layer\_1:

        sorted\_flask\_list.append(element)

    for element in layer\_2:

        sorted\_flask\_list.append(element)

    for element in layer\_3:

        sorted\_flask\_list.append(element)

    return sorted\_flask\_list

def replace\_undefined(flasks\_list):

    '''Функция для составления списка неопределенных значений недостающими цветами'''

    # Подготовление списка с цвтеами и их количеством, которые нужно добавить

    flasks\_list = np.asarray(flasks\_list)

    colors, counts = np.unique(flasks\_list, return\_counts=True)

    colors\_dict = dict(zip(colors, counts))

    added\_colors = dict()

    for key in colors\_dict.keys():

        if colors\_dict[key] < 4:

            if key != 'UNDEFINED':

                added\_colors[key] = 4 - colors\_dict[key]

    return added\_colors

def found\_colors\_in\_flasks(image\_for\_search, id\_client, reload\_image):

    '''Основная функция для распознавания цветов на картинке и добавления их в массив'''

    # Чтение изображения в цветном формате

    original\_image = cv2.imread(image\_for\_search)

    # Получение параметров размера изображения и вывод параметров обрезки

    height, width, \_ = original\_image.shape

    if reload\_image == False:

        cropped\_height = [round(height \* 0.125), round(height \* 0.875)]

    else:

        cropped\_height = [0, height]

    # Обрзка изображения под определенные границы (чтобы были видны только колбы)

    cropped\_image = original\_image[cropped\_height[0]:cropped\_height[1], 0:width]

    cv2.imwrite(image\_for\_search, cropped\_image)

    # Предобработка начального изображения после кропа

    # Определение контуров элементов и их отрисовка на цветном изображении

    contours\_of\_flasks, \_ = cv2.findContours(

        preprocessing\_image(image\_for\_search),

        cv2.RETR\_TREE,

        cv2.CHAIN\_APPROX\_SIMPLE

    )

    # Задаем эмпирически полученные коэффициенты отношения высоты и ширины экрана к высоте и ширине колбы

    coeff\_width\_flask = round(width / 11)

    coeff\_height\_flask = round((cropped\_height[1] - cropped\_height[0]) / 5.2)

    flasks = [] # Список прямоугольников-колб

    # Проходим по всем контурам и подсвечиваем прямоугольники целых колб

    for contour in contours\_of\_flasks:

        # Определение границ прямоугольников и добавление цвета прямоугольника в список

        flasks, \_ = found\_rect(contour, flasks, coeff\_width\_flask, coeff\_height\_flask)

    flasks = sorted\_flasks(flasks)

    images\_of\_flasks = crop\_rects(flasks, cropped\_image, id\_client)

    flasks\_list = []    # Список цветов в колбах

    for images\_contour in images\_of\_flasks:

        # Находим контуры цветов внутри каждой колбы

        internal\_colors = []

        colors\_list = create\_color\_list(images\_contour[0])

        if colors\_list:

            for colors\_contours in colors\_list:

                internal\_colors.append((colors\_contours[0], colors\_contours[1]))

            internal\_colors = sorted(

                internal\_colors,

                key=lambda

                item:

                item[1][1],

                reverse=True

            )

            colors = []

            for color in internal\_colors:

                colors.append(color[0])

            flasks\_list.append(colors)

    # Вручную добавляем 2 пустые колбы

    for \_ in range(2):

        flasks\_list.append(['EMPTY', 'EMPTY', 'EMPTY', 'EMPTY'])

    # Удаление всех временных файлов для экономии места

    for flask\_info in images\_of\_flasks:

        os.remove(flask\_info[0])

    create\_json(flasks\_list, id\_client)

    return replace\_undefined(flasks\_list)

def create\_json(flasks\_list, id\_client):

    '''Создание и заполнение json файла с распознанными цветами'''

    with open(f"./{id\_client}/tmp/start\_level\_{id\_client}.json", "w") as this\_level:

        json.dump({"bottles": flasks\_list}, this\_level, indent=2)

def replace\_in\_json(json\_name, color\_name):

    '''Фуекция для замены неопределенных цветов в json на выбранные пользователем'''

    with open(json\_name, "r") as this\_level:

        file = json.load(this\_level)

    try:

        for \_, flasks in file.items():

            for colors in range(len(flasks)):

                for color in range(len(flasks[colors])):

                    if flasks[colors][color] == 'UNDEFINED':

                        file[\_][colors][color] = color\_name

                        raise BreakAction

    except BreakAction:

        pass

    with open(json\_name, "w") as this\_level:

        json.dump(file, this\_level, indent=2)

def create\_image\_for\_replace(json\_name, id\_client):

    '''Функция для отрисовки изображения с подсветкой того цвета, который нужно заполнить'''

    with open(json\_name, "r") as this\_level:

        file = json.load(this\_level)

    # Создание и сохранение пустого черного изображения

    filename = f'./{id\_client}/tmp/level\_for\_{id\_client}.jpg'

    height, width = 1800, 1400

    template = np.zeros((height, width, 3), np.uint8)

    cv2.imwrite(filename, template)

    # Установка количества линий с колбами и размеров колбы

    for \_, flasks in file.items():

        count\_flasks = len(flasks)

    height\_flask, width\_flask = 400, 100

    count\_lines = np.trunc(count\_flasks / 6) + 1

    flasks\_centers = []

    # Заполнение массива с центрами колб

    try:

        for y in range(int(height / (count\_lines + 1)), height, int(height / (count\_lines + 1))):

            for x in range(int(width / 7), width, int(width / 7)):

                flasks\_centers.append([x, y])

                if len(flasks\_centers) == count\_flasks:

                    raise BreakAction

    except BreakAction:

        pass

    cnt\_undef = 0

    # Отрисовка всех колб с цветами и пустыми полями внутри них

    for colors in range(len(flasks)):

        x1, y1 = flasks\_centers[colors][0] - width\_flask / 2, flasks\_centers[colors][1] - height\_flask / 2

        x2, y2 = flasks\_centers[colors][0] + width\_flask / 2, flasks\_centers[colors][1] + height\_flask / 2

        flask\_rect = cv2.rectangle(template, (int(x1), int(y1)), (int(x2), int(y2)), (176, 176, 90), 6)

        cv2.imwrite(filename, flask\_rect)

        for color in range(len(flasks[colors])):

            circle\_x, circle\_y = flasks\_centers[colors][0], y2 - (y2 - y1) \* (2 \* color + 1) / 8

            for variation in variations:

                if flasks[colors][color] == 'UNDEFINED':

                    cnt\_undef += 1

                    if cnt\_undef == 1:

                        color\_circle = cv2.circle(template, (int(circle\_x), int(circle\_y)), 47, (0, 255, 0), 6)

                    else:

                        color\_circle = cv2.circle(template, (int(circle\_x), int(circle\_y)), 47, (255, 255, 255), 6)

                    break

                elif flasks[colors][color] == variation[0]:

                    color\_circle = cv2.circle(template, (int(circle\_x), int(circle\_y)), 47, variation[2], -1)

                    break

            cv2.imwrite(filename, color\_circle)

def add\_empty\_flask(json\_name):

    '''Функция для добавления пустой колбы в конец json'''

    with open(json\_name, "r") as this\_level:

        file = json.load(this\_level)

    file['bottles'].append(['EMPTY', 'EMPTY', 'EMPTY', 'EMPTY'])

    with open(json\_name, "w") as this\_level:

        json.dump(file, this\_level, indent=2)

Листинг 10. Программный код для решения уровня.

import json

def get\_level\_from\_json(level):

    '''Функция для считывания данных из json и преобразования их в список'''

    with open(level, 'r') as this\_level:

        file = json.load(this\_level)

    return list(file['bottles'])

def send\_result\_to\_txt(result, steps):

    '''Функция для записи результата решения в текстовый файл'''

    with open(result, 'w') as result\_level:

        for step in steps:

            result\_level.write(f'{step}\n')

def check\_solving(position):

    '''Функция проверки получения решения'''

    full\_color = 0

    for flask in position:

        # После очередного перемещения проходим по списку колб

        if flask[0] == flask[1] == flask[2] == flask[3]:

            full\_color += 1

        else:

            break

    if full\_color == len(position):

        # Количество одноцветных колб совпадает с количеством колб

        return True

    return False

def possible\_moves(position):

    '''Функция для определения всех возможных перемещений для конкретной ситуации'''

    moves = []  # Список перемещений

    count\_flasks = len(position)    # Количество колб

    count\_colors = len(position[0]) # Максимальное количество цветов в колбе (во всех колбах одинаковое количество элементов)

    # Перебираем все колбы из которых можно перелить

    for idx\_solve\_flask in range(count\_flasks):

        # Перебираем все цвета, начиная с верхнего (последний в списке)

        mono\_color\_height = 0

        solve\_upper\_color = (

            (idx\_solve\_flask, 0),

            ('EMPTY', mono\_color\_height)

        )  # Пустая колба

        for idx\_color in range(count\_colors - 1, -1, -1):

            # Получаем необходимую информацию о самом верхнем цвете

            if position[idx\_solve\_flask][idx\_color] != 'EMPTY':

                mono\_color\_height += 1

                solve\_upper\_color = (

                    (idx\_solve\_flask, idx\_color),

                    (position[idx\_solve\_flask][idx\_color], mono\_color\_height)

                )

                break

        # Из пустой колбы ничего перелить нельзя

        if solve\_upper\_color[1][0] == 'EMPTY':

            continue

        # Проверка того, что следующие блоки такого же цвета (переливаться будет сразу весь цвет и это влияет на решение)

        for idx\_color in range(solve\_upper\_color[0][1] - 1, -1, -1):

            if position[idx\_solve\_flask][idx\_color] == solve\_upper\_color[1][0]:

                mono\_color\_height += 1

            else:

                break

        # Нельзя переливать полностью заполненную колбу

        if mono\_color\_height == count\_colors:

            continue

        solve\_upper\_color = (

            (solve\_upper\_color[0][0], solve\_upper\_color[0][1]),

            (solve\_upper\_color[1][0], mono\_color\_height)

        )

        # Перебираем все колбы в которые можно перелить

        for idx\_target\_flask in range(count\_flasks):

            # Переливать колбу саму в себя нельзя

            if idx\_solve\_flask != idx\_target\_flask:

                # Перебираем все цвета, начиная с верхнего (последний в списке)

                count\_empty\_slots = 0

                target\_upper\_color = (

                    (idx\_target\_flask, 0),

                    ('EMPTY', count\_empty\_slots)

                )

                for idx\_color in range(count\_colors - 1, -1, -1):

                    if position[idx\_target\_flask][idx\_color] != 'EMPTY':

                        target\_upper\_color = (

                            (idx\_target\_flask, idx\_color),

                            (position[idx\_target\_flask][idx\_color], count\_empty\_slots)

                        )

                        break

                    else:

                        count\_empty\_slots += 1

                if count\_empty\_slots > 0:

                    target\_upper\_color = (

                        (target\_upper\_color[0][0], target\_upper\_color[0][1]),

                        (target\_upper\_color[1][0], count\_empty\_slots)

                    )

                # В полную колбу ничего перелить нельзя

                if target\_upper\_color[0][1] == count\_colors - 1:

                    continue

                # Исключение переливания одного цвета в пустую колбу (бесполезное действие)

                if not (target\_upper\_color[1][0] == 'EMPTY' and

                        len(set(position[idx\_solve\_flask])) == 2 and

                        'EMPTY' in position[idx\_solve\_flask]):

                    # Переливание возможно только если верхние цвета совпадают или если переливаем в пустую колбу и места в целевой колбе достаточно

                    if solve\_upper\_color[1][0] == target\_upper\_color[1][0] and target\_upper\_color[1][1] >= solve\_upper\_color[1][1]:

                        target\_upper\_color = (

                            (target\_upper\_color[0][0], target\_upper\_color[0][1] + 1),

                            ('EMPTY', target\_upper\_color[1][1])

                        )

                        moves.append((solve\_upper\_color, target\_upper\_color))

                    elif target\_upper\_color[1][0] == 'EMPTY':

                        moves.append((solve\_upper\_color, target\_upper\_color))

    return moves

def apply\_move(position, move):

    '''Функция для применения перемещения к текущему положению для получения нового'''

    # Получение данных о колбах, которые задействуются

    solve\_flask, target\_flask = move

    # Замена цвета в решающей колбе на пустое и заполнение места в целевой колбе

    for cnt in range(solve\_flask[1][1]):

        position[target\_flask[0][0]][target\_flask[0][1] + cnt] = solve\_flask[1][0]

        position[solve\_flask[0][0]][solve\_flask[0][1] - cnt] = 'EMPTY'

    step = f'{solve\_flask[0][0] + 1} -> {target\_flask[0][0] + 1}'

    return position, step

def go\_back\_move(position, move):

    '''Функция отмены перемещения'''

    # Получение данных о колбах, которые задействуются

    solve\_flask, target\_flask = move

    # Замена цвета в целевой колбе на пустое и заполнение места в решающей колбе

    for cnt in range(solve\_flask[1][1]):

        position[solve\_flask[0][0]][solve\_flask[0][1] - cnt] = solve\_flask[1][0]

        position[target\_flask[0][0]][target\_flask[0][1] + cnt] = target\_flask[1][0]

    return position

def transfusion\_of\_liquids(position):

    '''Функция перемещения цвета в текущей позиции и записи последовательности шагов'''

    visited\_states = set()

    stack = [(position, [])]

    while stack:

        now\_position, steps = stack[0]

        now\_position = tuple(tuple(flask) for flask in now\_position)

        # Проверяем пришли ли мы в начальную позицию снова? если да, то решения не было найдено

        if len(stack) == 1:

            if now\_position in visited\_states:

                return False, None

        # Если начального состояния нет в списке посещенных, то добавляем его (начало поиска)

        if not visited\_states:

            visited\_states.add(now\_position)

        now\_position = list(list(flask) for flask in now\_position)

        steps = list(steps)

        # Проверяем решена ли задача

        if check\_solving(now\_position):

            return True, steps

        if not possible\_moves(now\_position):

            # Поскольку мы сначала заполняем стек вершинами, то для корректных позиций нужен откат действия

            go\_back\_move(now\_position, move)

            stack.pop(0)

        else:

            # Обходим массив возможных перемещений, рекурсивно погружаясь в глубину

            moves = possible\_moves(now\_position)

            for move in moves:

                # Применяем действие

                now\_position, step = apply\_move(now\_position, move)

                now\_position\_tuple = tuple(tuple(flask) for flask in now\_position)

                # Если текущая позиция уже была посещена ранее, то возвращаем движение назад

                if now\_position\_tuple in visited\_states:

                    if move == moves[len(moves) - 1]:

                        stack.pop(0)

                    go\_back\_move(now\_position, move)

                    continue

                # Добавляем текущую позицию в список посещенных

                visited\_states.add(now\_position\_tuple)

                steps.append(step)

                steps = tuple(steps)

                stack.insert(0, (now\_position\_tuple, steps))

                break

def transfusion\_manage(task, result):

    '''Основная функция модуля, регулирующая процесс переливания'''

    # Вызываем функцию для считывания файлов из json

    start\_position = get\_level\_from\_json(task)

    # Возвращаем флаг решения и список шагов, если решение есть

    is\_solved, steps\_list = transfusion\_of\_liquids(start\_position)

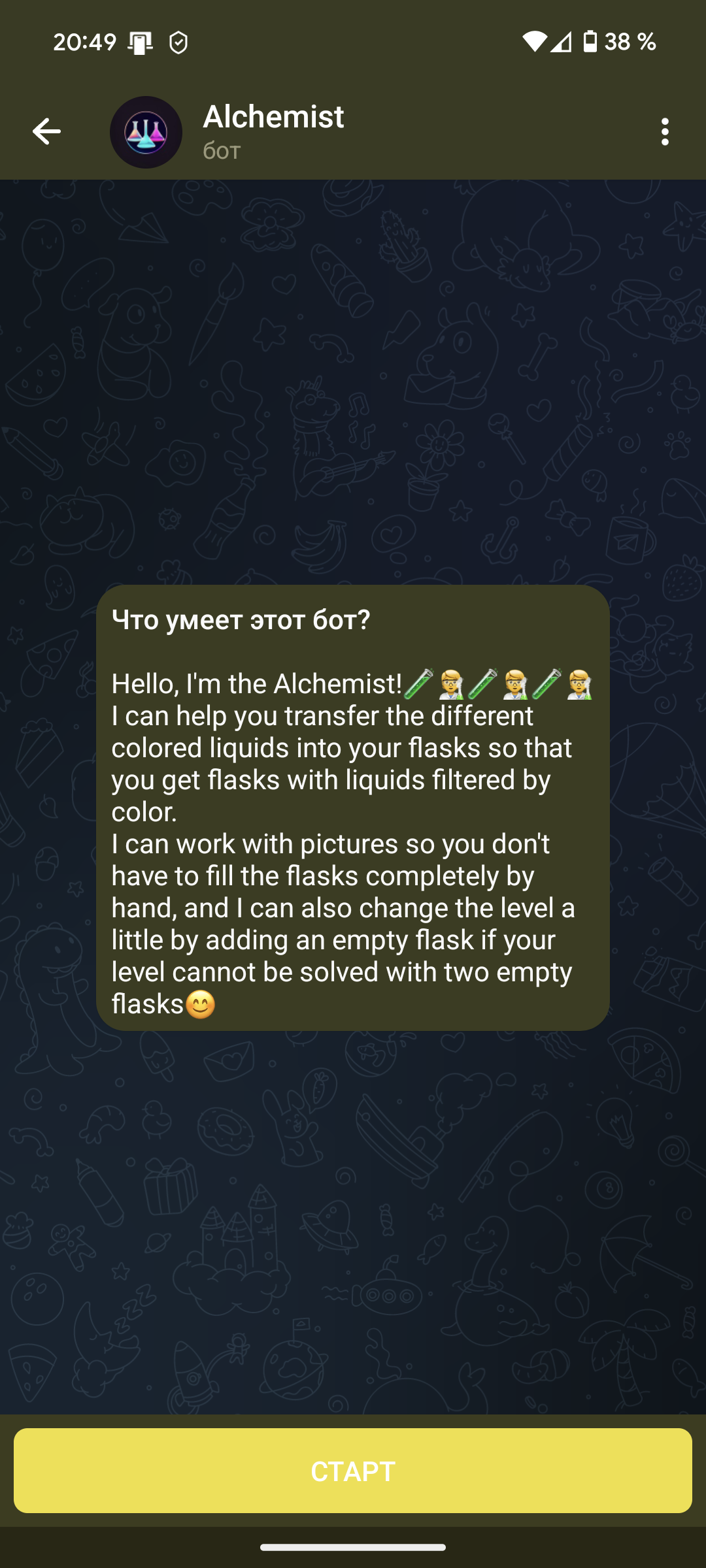
    if is\_solved:

        send\_result\_to\_txt(result, steps\_list)

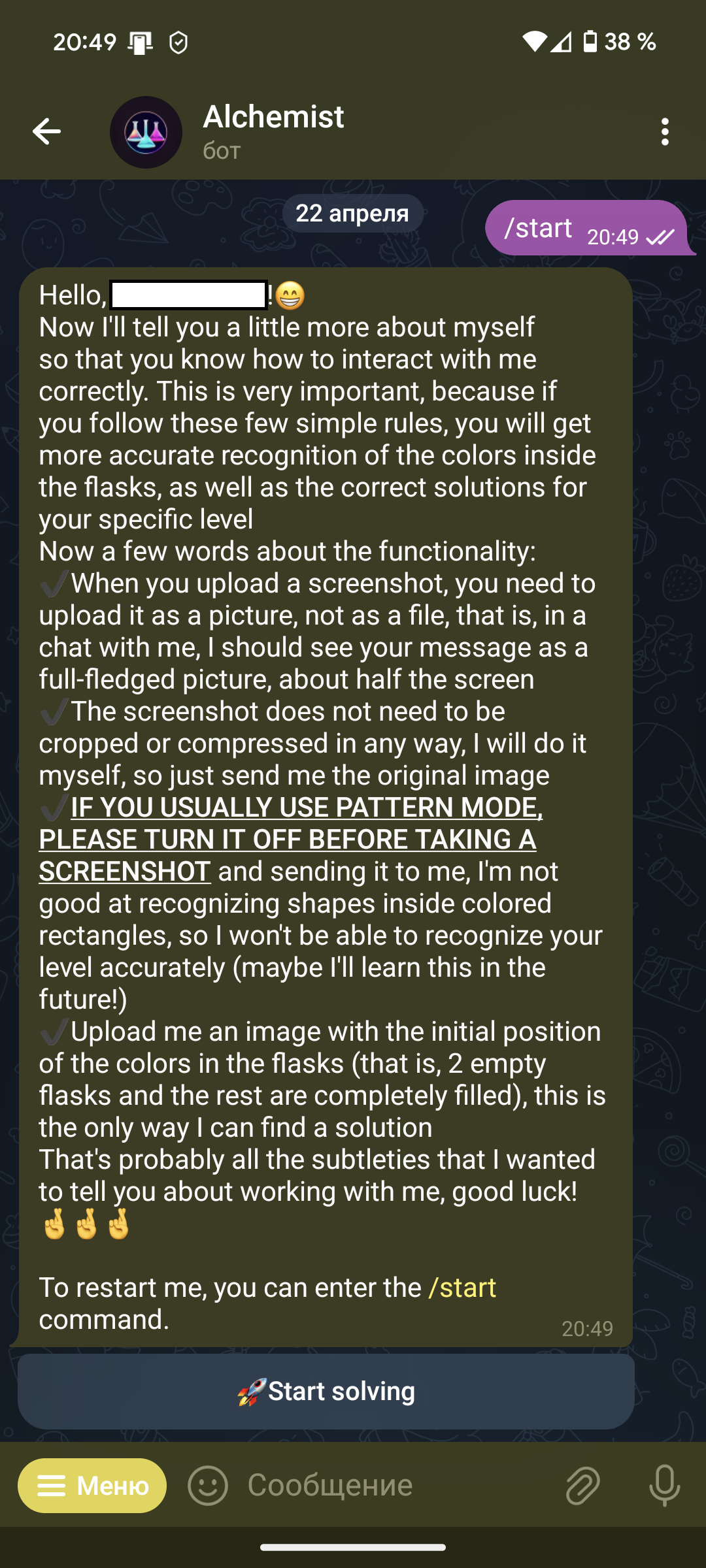
        return True

    return False

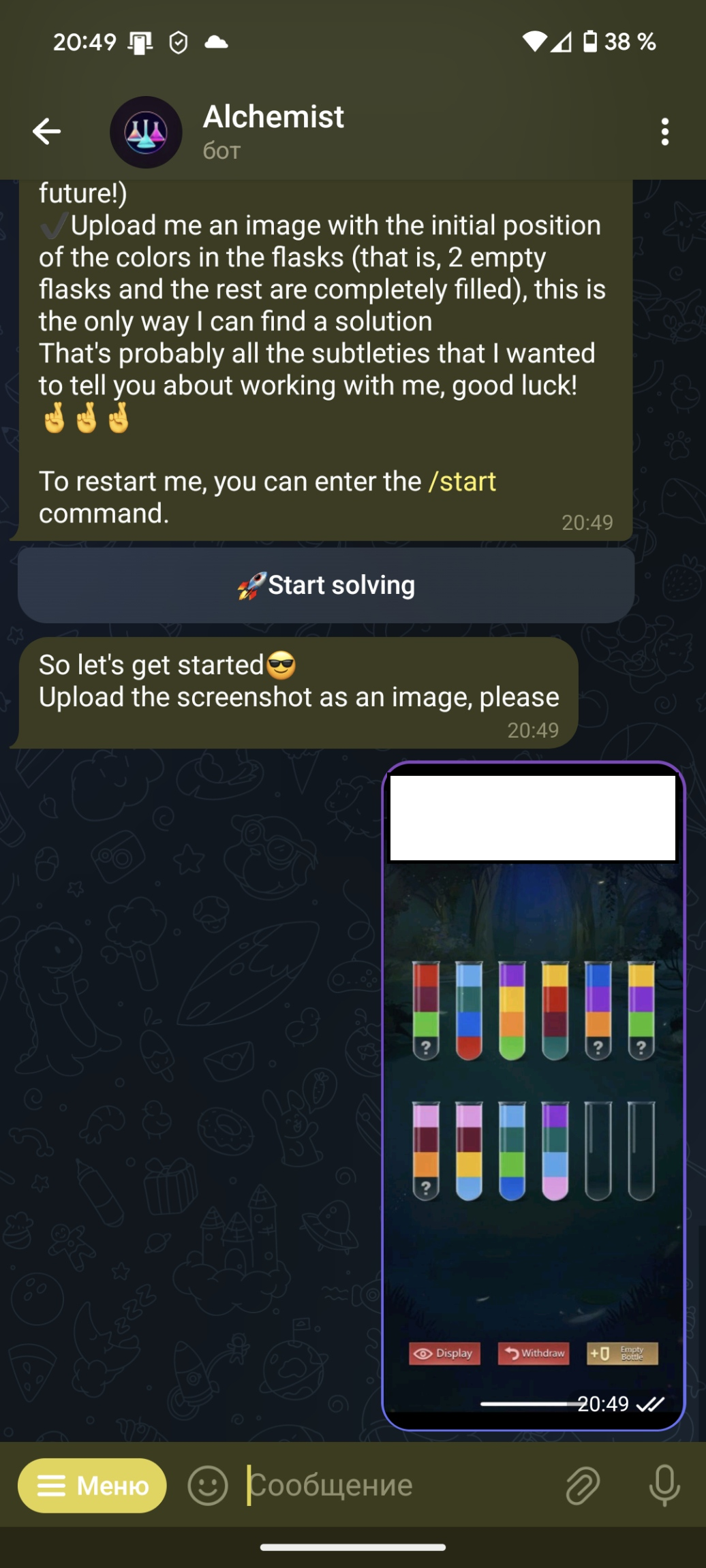
Приложение B (скриншоты пунктов меню)



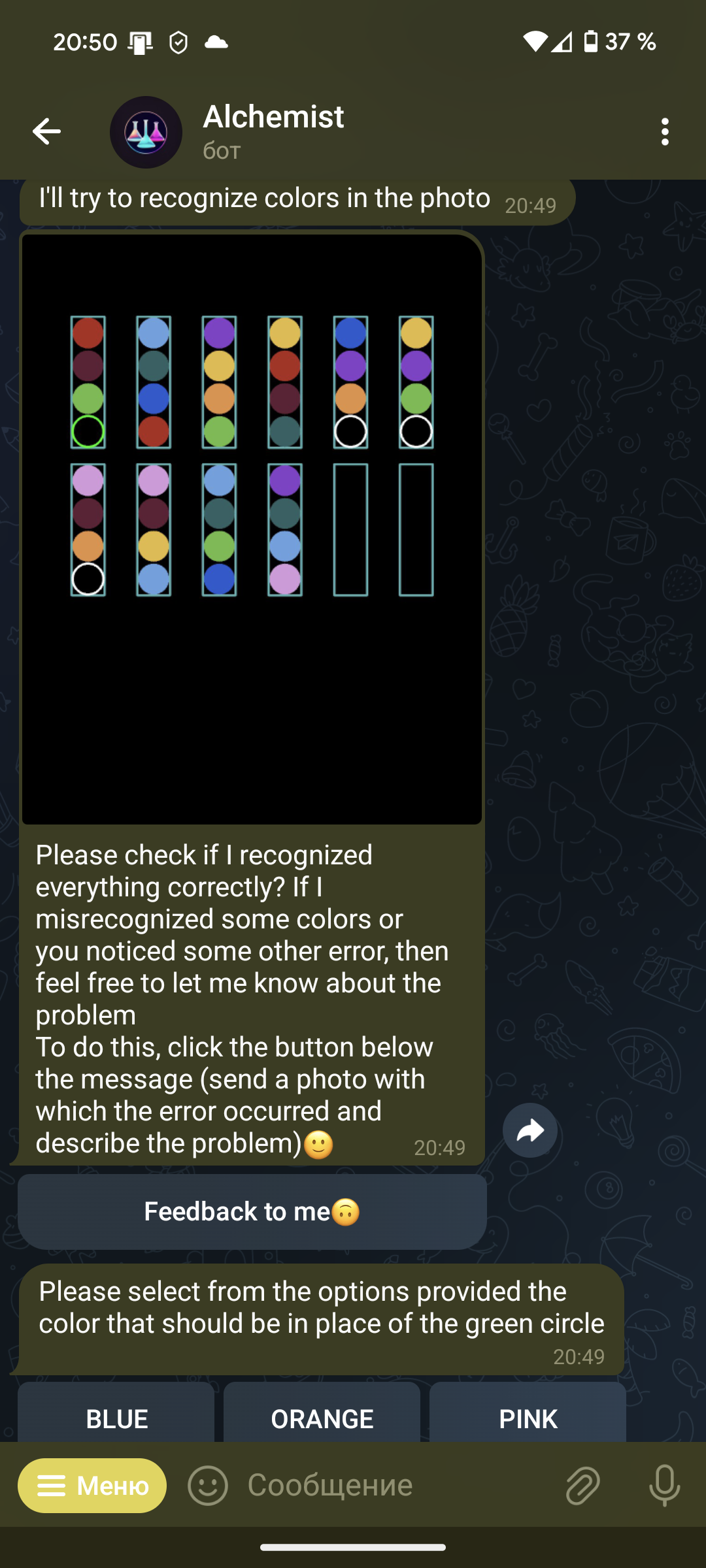
*Рис. 1. Приветственное меню бота.*



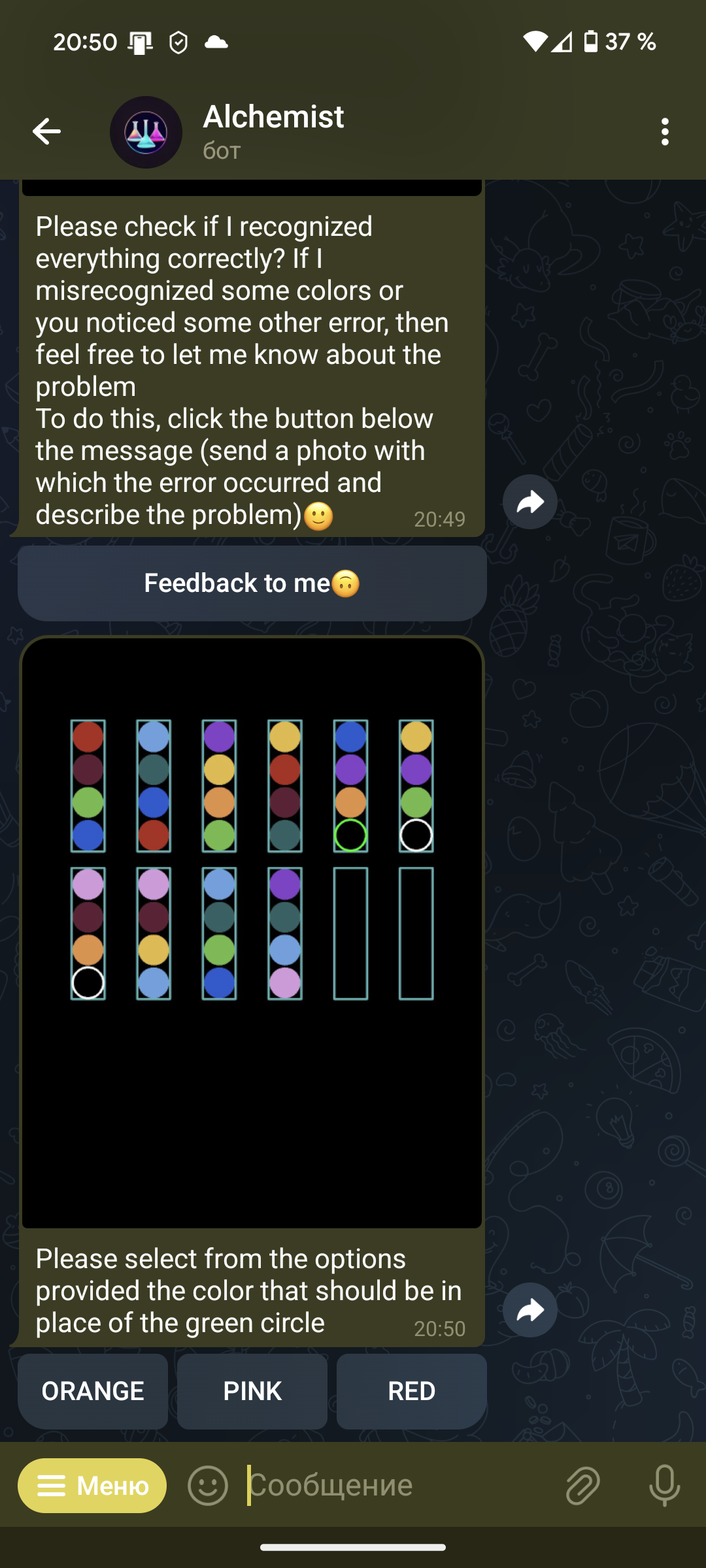
*Рис. 2. Стартовое сообщение.*



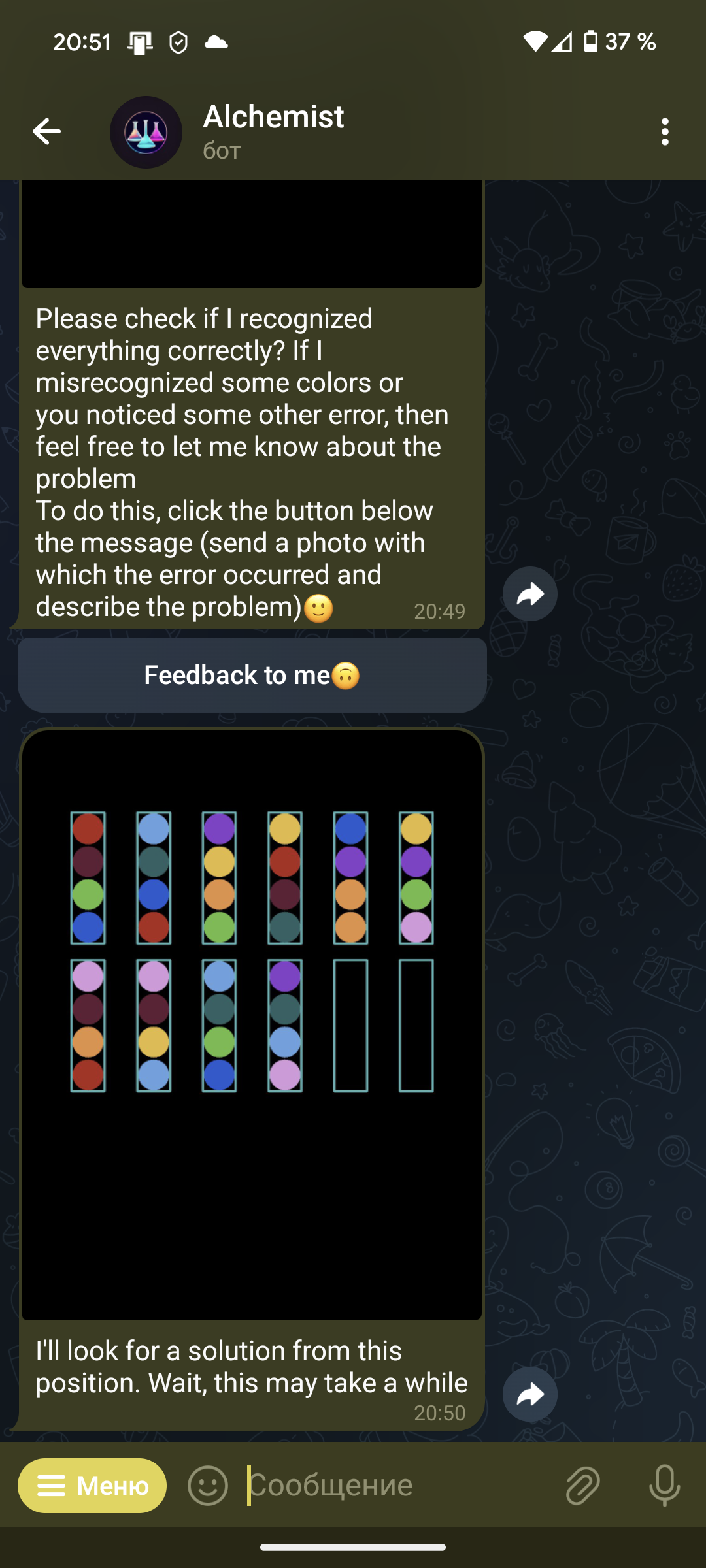
*Рис. 3. Загрузка изображения.*

**

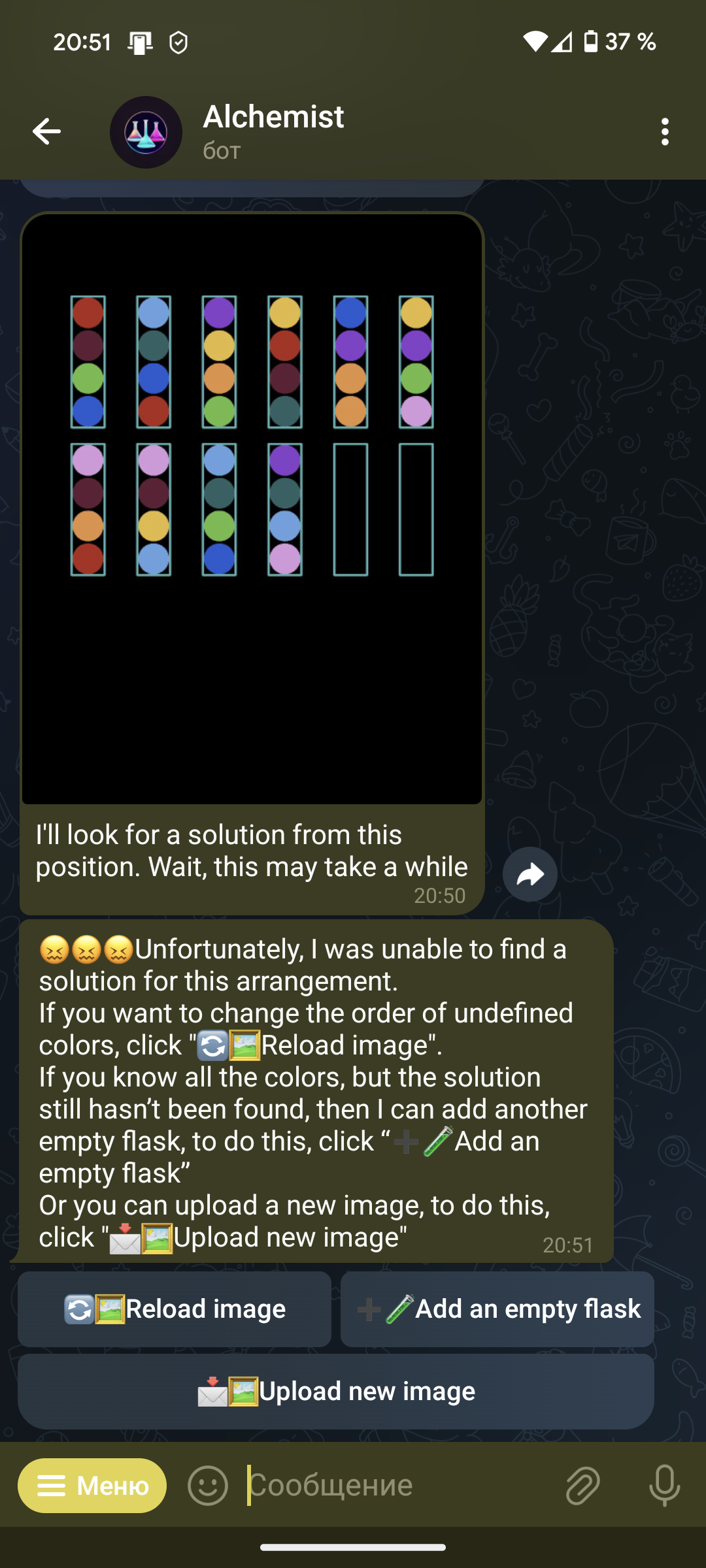
*Рис. 4. Результат распознавания цветов или перезагрузки фото.*

**

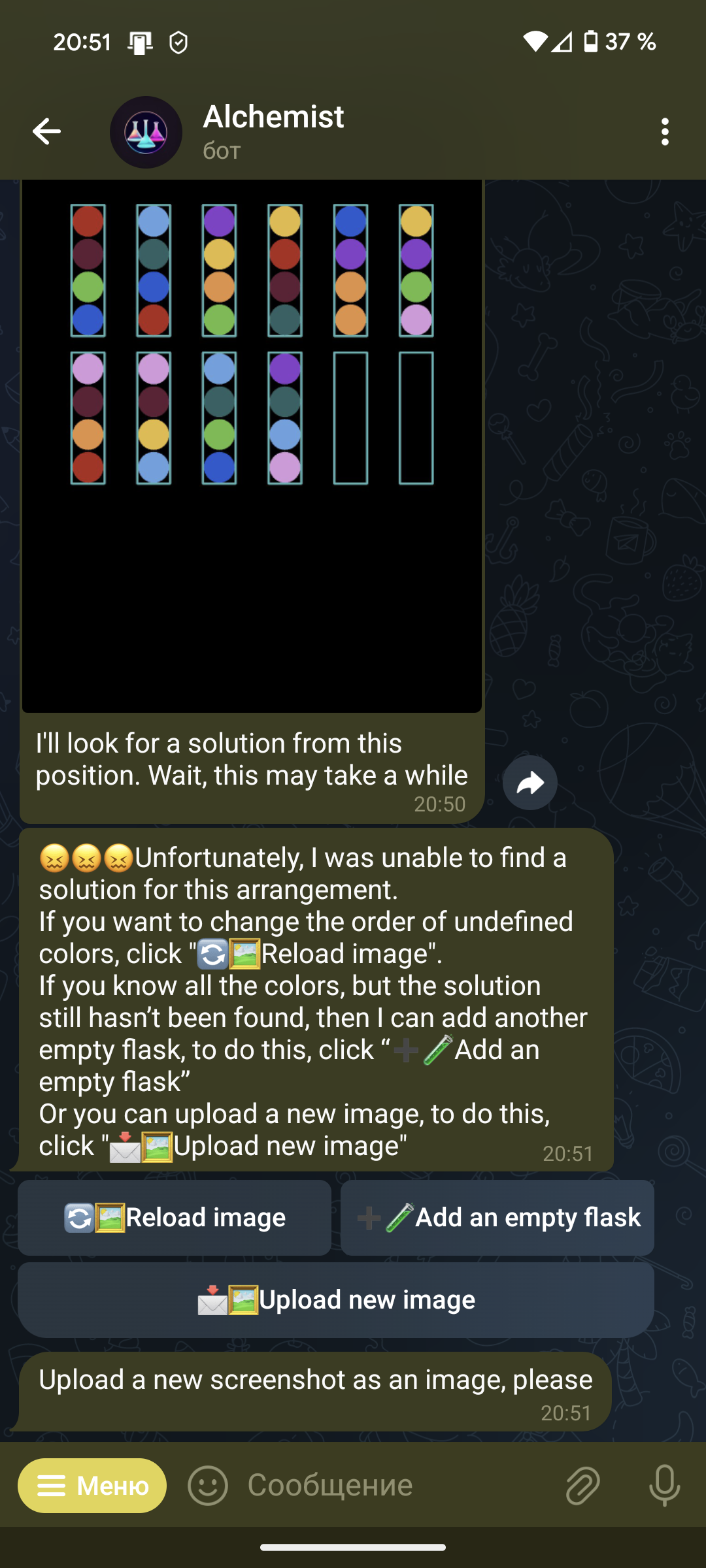
*Рис. 5. Промежуточное заполнение неопределенных цветов.*

**

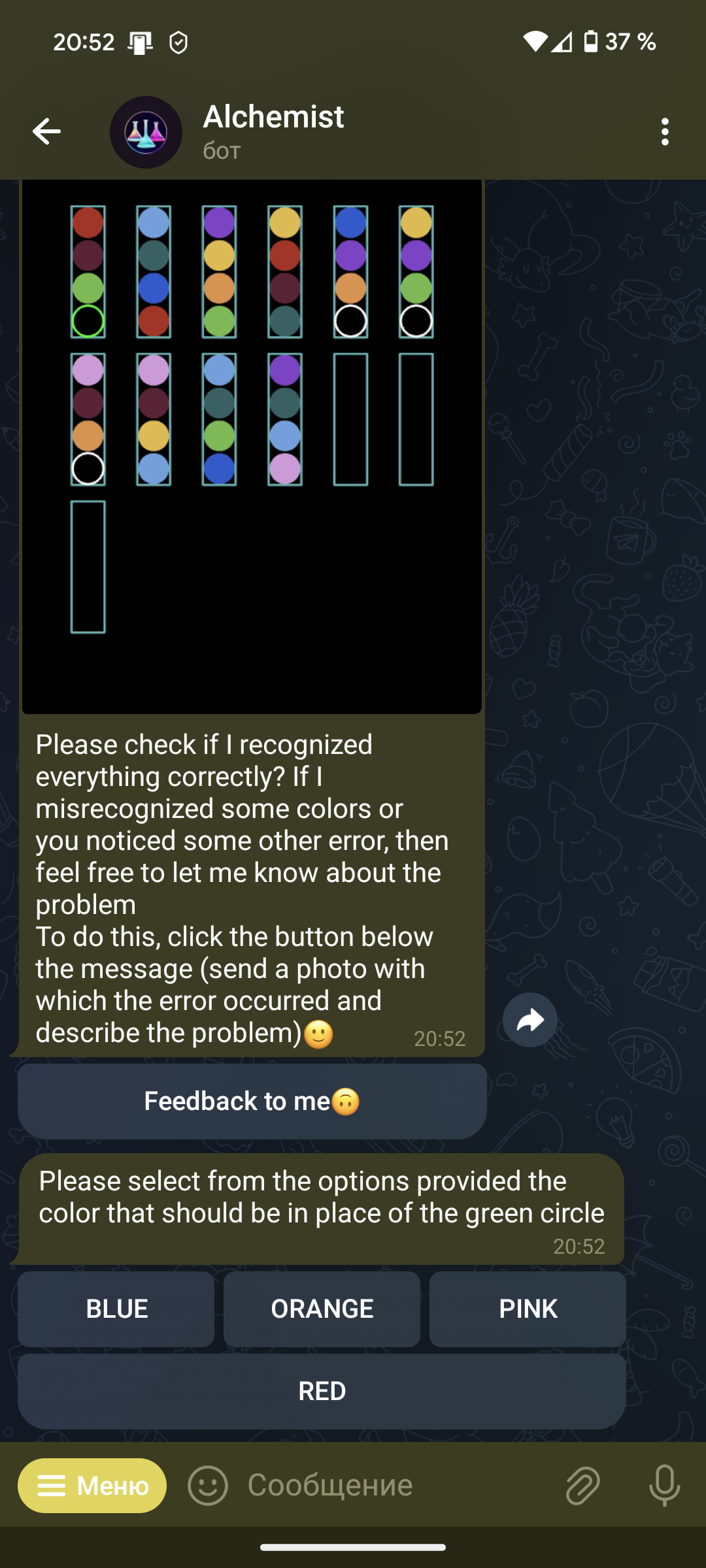
*Рис. 6. Конечное ответное изображение перед решением.*

**

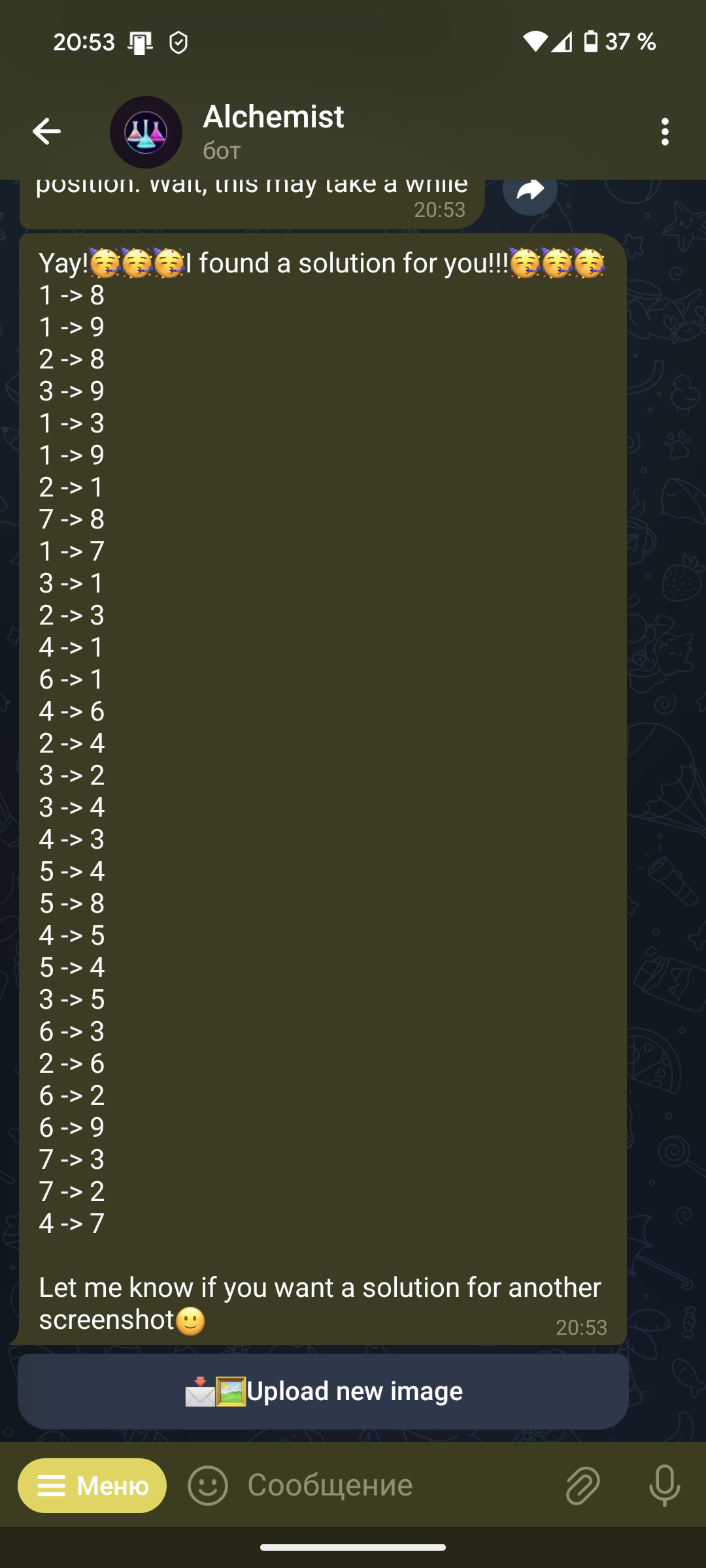
*Рис. 7. Результат неудачного поиска решения.*

**

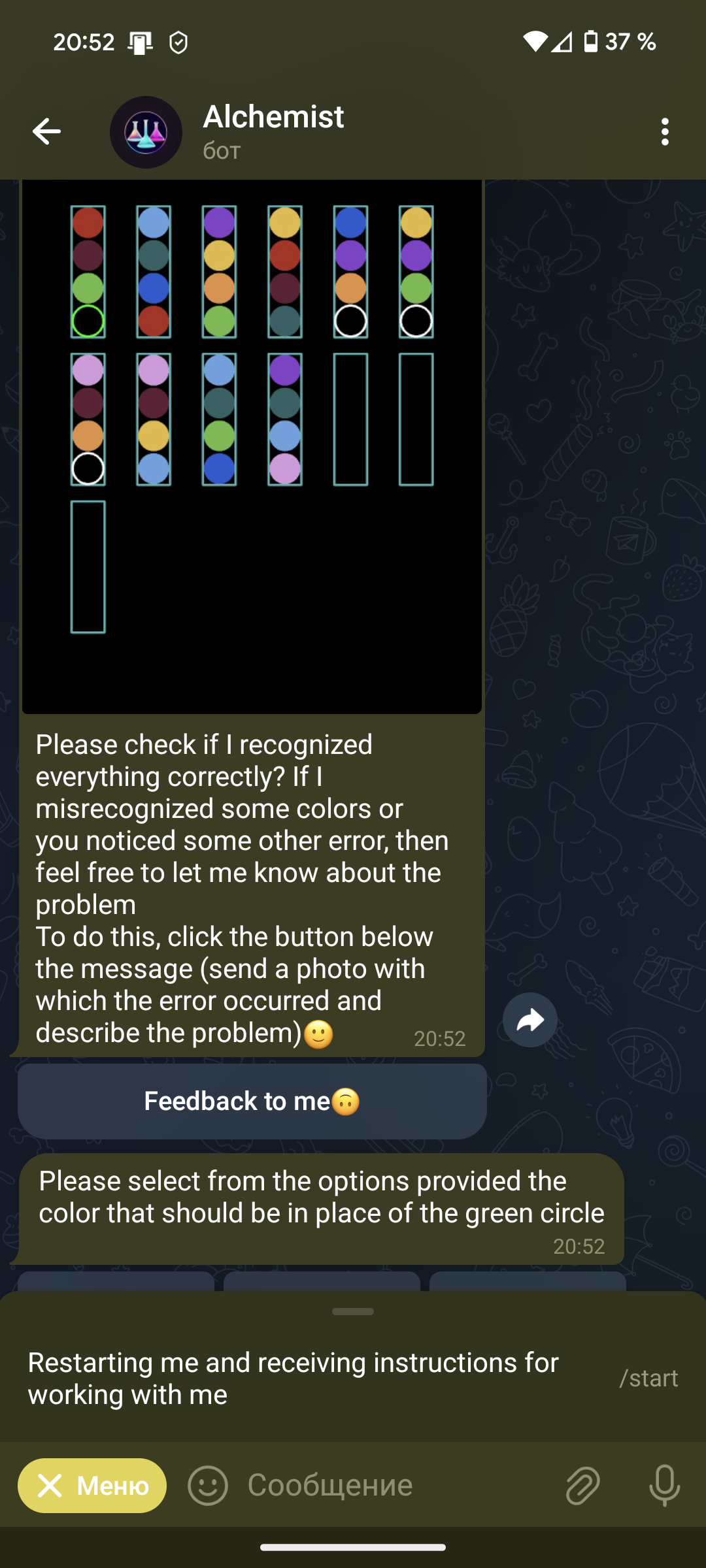
*Рис. 8. Нажатие кнопки загрузки нового изображения.*

**

*Рис. 9. Выбор опции добавления пустой колбы.*

**

*Рис. 10. Удачная попытка поиска решения.*

**

*Рис. 11. Выпадающее меню.*

*Рис. 11. Выпадающее меню.*