Пояснительная записка

направление Devops

Ермин Михаил Александрович

2025

Оглавление

[Реализация 2](#_Toc192627663)

[Вычисления 2](#_Toc192627664)

[REST API 4](#_Toc192627665)

[Использование 6](#_Toc192627666)

[Docker 6](#_Toc192627667)

[Обычное использование 7](#_Toc192627668)

# Реализация

## Вычисления

**Описание работы программы PathFinder**

Программа предназначена для поиска оптимальной точки сбора персонажей в лабиринте с различными типами местности. Каждый тип местности имеет свою стоимость перемещения. Программа загружает лабиринт из файла, находит оптимальную точку сбора и строит кратчайшие пути для каждого героя с учётом сложности местности.

**Вербальная модель**

1. Загрузка лабиринта из текстового файла, который содержит размеры лабиринта и его структуру.
2. Определение начальных позиций героев, отмеченных цифрами от '1' до '9'.
3. Поиск оптимальной точки сбора:
   * Перебор всех доступных клеток.
   * Для каждой клетки вычисление общей стоимости пути от всех героев.
   * Выбор точки с минимальной суммарной стоимостью.
4. Построение маршрута для каждого героя до найденной точки с использованием алгоритма Дейкстры (поиск кратчайшего пути на графе с весами).
5. Вывод информации о путях и визуализация лабиринта с отображением маршрутов и точки сбора.

**Математическая модель**

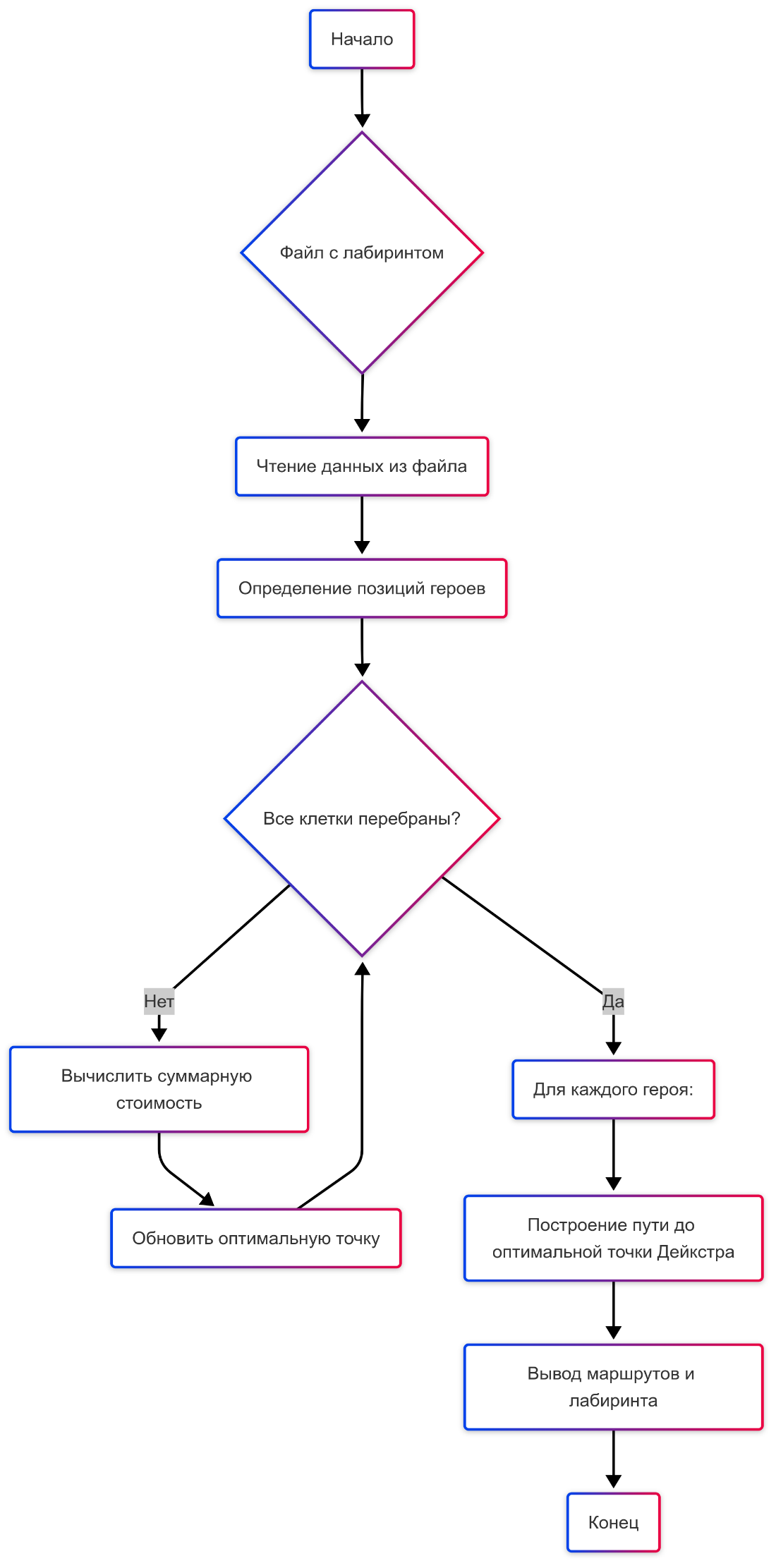
Пусть:

* M – матрица лабиринта размером h \* w, где M[i][j] – тип местности.
* H={h1,h2,…,hn} – множество позиций героев, где hi=(xi,yi).
* d(h, p) – длина кратчайшего пути от героя h до точки p с учётом стоимости клеток.

Целевая функция:

Где p – точка сбора, которая минимизирует суммарные затраты на перемещение.

**Блок-схема**



**Комментарии к модулям и объектам**

1. **Point**: структура, представляющая координаты (x,y)(x, y) на карте.
2. **terrainCost**: карта с типами местности и соответствующими им весами.
3. **readMaze**: функция чтения лабиринта из файла. Определяет размеры, загружает карту и находит героев.
4. **isValid**: проверяет, доступна ли точка для перемещения.
5. **findOptimalMeetingPoint**: перебирает все клетки и находит точку с минимальной стоимостью до всех героев.
6. **findPath**: поиск кратчайшего пути с использованием алгоритма Дейкстры и очереди с приоритетами (PriorityQueue).
7. **PriorityQueue**: реализация очереди с приоритетами на основе контейнера heap.
8. **printMazeWithPaths**: визуализация лабиринта с маршрутами и точкой сбора.

## REST API

API-сервер принимает запросы с данными о лабиринте, сохраняет их в файл, запускает внешний процесс для вычисления оптимальной точки сбора персонажей и возвращает найденные координаты.

**1. Запуск сервера**

Программа запускает HTTP-сервер на порту 8080 и обрабатывает запросы по маршруту /process. Сервер принимает только POST-запросы.

**2. Обработка запроса**

Когда сервер получает POST-запрос, он выполняет следующие шаги:

1. Проверяет, что метод запроса — POST. Если метод отличается, возвращает ошибку 405 Method Not Allowed.
2. Декодирует JSON-данные из тела запроса и сохраняет их в структуре MazeRequest. Если JSON некорректный, возвращает ошибку 400 Bad Request.

**3. Сохранение лабиринта**

Данные лабиринта сохраняются в текстовый файл по пути ../maps/maze-api.txt. Если возникает ошибка при записи в файл, сервер возвращает код 500 Internal Server Error и сообщение с описанием ошибки.

**4. Запуск внешней программы**

После успешного сохранения файла сервер запускает внешний процесс:

* Выполняется команда go run ../game/main.go -m ../maps/maze-api.txt.
* Эта команда запускает основную программу, которая анализирует лабиринт и вычисляет оптимальную точку сбора персонажей.

Если программа завершается с ошибкой, сервер возвращает статус 500 Internal Server Error и передаёт в ответе текст ошибки и вывод программы.

**5. Извлечение оптимальной точки**

После выполнения внешней программы сервер анализирует её вывод. Для поиска координат точки сбора используется регулярное выражение:

Оптимальная точка сбора: {X:(\d+) Y:(\d+)}

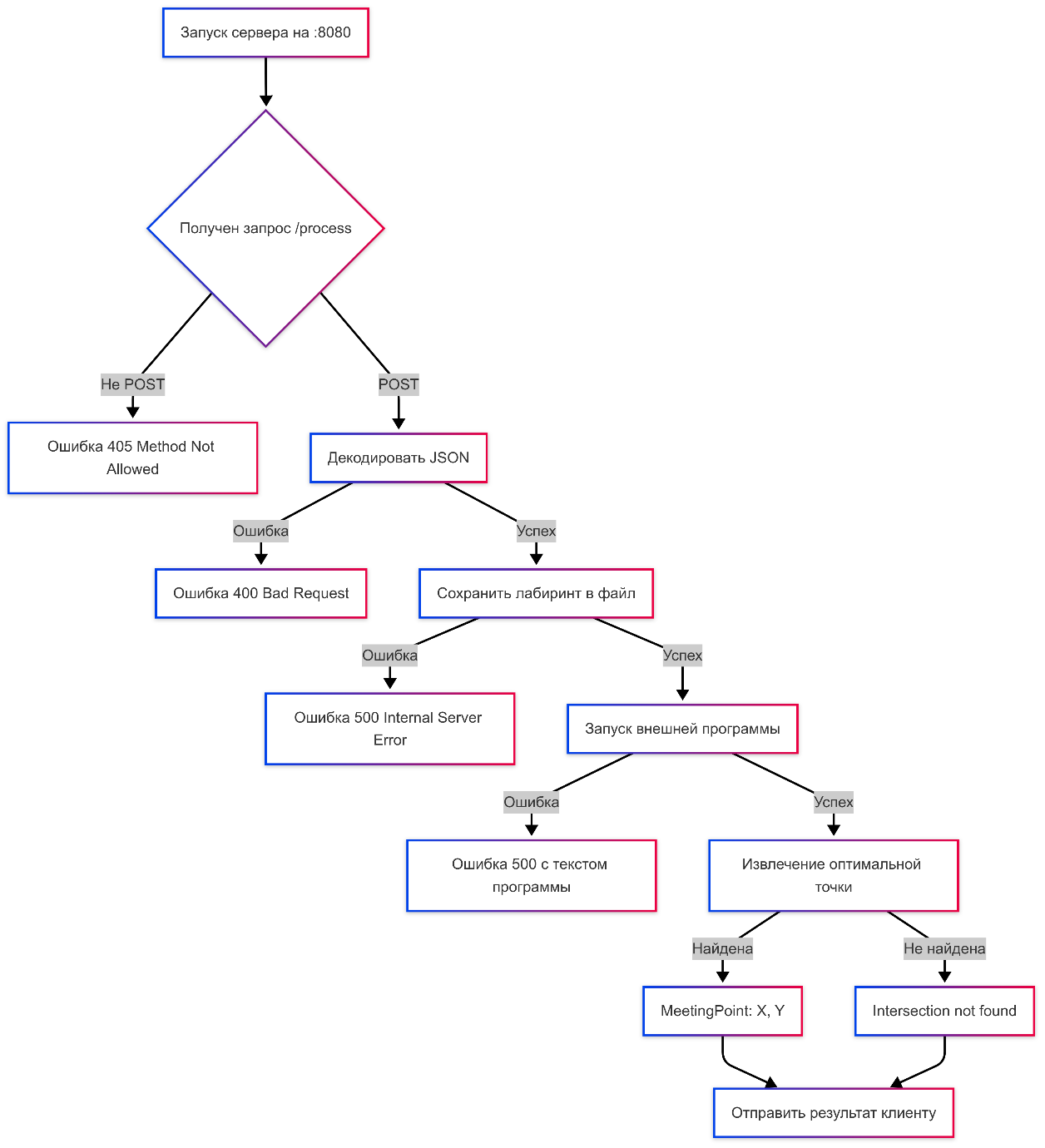
Если точка найдена, она форматируется в виде строки: MeetingPoint: X:<значение>, Y:<значение>.

Если точка сбора не найдена, сервер возвращает сообщение Intersection not found.

**6. Ответ клиенту**

Сервер отправляет клиенту текстовый ответ с найденной точкой сбора или сообщением об отсутствии результата. Формат ответа — text/plain с кодировкой UTF-8.

**Блок схема**

****

# Использование

Для использования необходимо клонировать репозиторий  
git clone <https://github.com/magnoliAHAH/tbolimpiada-2025.git>

Переходим в директорию проекта  
cd tbolimpiada-2025

## Docker

Создаём докер образ:

sudo docker build -t pathfinder .

Запускаем контейнер:

sudo docker run -p 8080:8080 --name pathfinder pathfinder

Теперь rest api сервер запущен на 8080 порту, для обращения можно использовать curl, инструменты windows или специализированные приложения для работы с api.

В директории tbolimpiada-2025/api располагаются файлы curl-command.sh и windows-request.txt  
Чтобы использовать в linux необходимо изменить IP в файле curl-command.sh (при желании и карту), затем сделать файл исполняемым:

Chmod +x ./curl-command.sh

Далее запустить его:

sudo ./tbolimpiada-2025/api/curl-command.sh



Пример вывода команды curl

Для windows необходимо в файле windows-request.txt изменить IP (карту при желании), скопировать команду и вставить в терминал.

Изображение выглядит как текст, снимок экрана, Шрифт

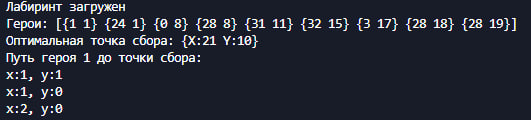
Контент, сгенерированный ИИ, может содержать ошибки.

Пример вывода для windows

## Обычное использование

Можно собрать директорию game в исполняемый файл с помощью go build, если он установлен на устройстве или же использовать

go run ./main.go -m [путь до карты]



Пример вывода go run ./main.go -m [путь до карты] (информация о карте)

Изображение выглядит как шаблон, снимок экрана, Красочность, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Пример вывода go run ./main.go -m [путь до карты] (карта с отмеченными путями)

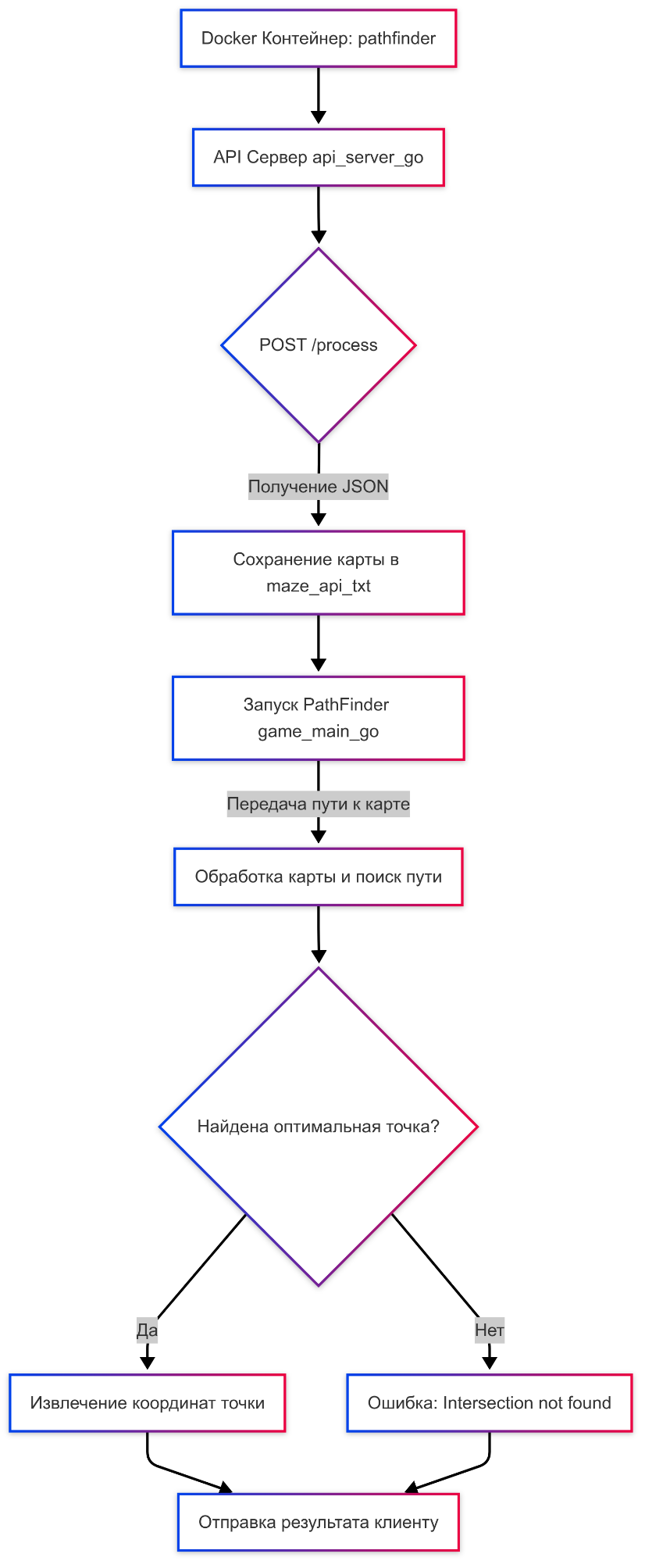


Схема взаимодействия внутри docker