Веб интерпретатор одномерной и двумерной машин Тьюринга

КУРАТОР: МАКСИМ ДОБРОХВАЛОВ

СТУДЕНТ-МАГИСТР: КИРИЛЛ ВИНОГРАДОВ

СТУДЕНТЫ-БАКАЛАВРЫ: АНТОН КИБАРДИН

ДМИТРИЙ ЧЕШУИН

КИРИЛЛ КРЫЖАНОВСКИЙ

Задача и функциональность

Задача: создать веб-приложение, которое будет интерпретатором одно- и двумерной машины Тьюринга, а также движком для решения простейших задач на ней.

Функциональность:

- создание, редактирование, отображение, импорт и экспорт программ для Машин,
- визуальная симуляция работы Машин,
- выполнение программы по шагам,
- примитивные инструменты отладки,
- общие интерфейсы для создания задач,
- задачи-примеры из https://github.com/OSLL/adfmp20-turing и курса Информатика,
- сохранение результатов решения задач в БД,
- поддержка пользователей,
- поддержка протокола LTI*.

Задачи 2 итерации

- Реализовать логику машины Тьюринга на сервере Flask
- Создать API БД на основе спроектированной диаграммы БД
- Создать прототип визуализации поля
- Создать страницу с формой авторизации
- Спроектировать стандарты взаимодействия между северами Flask и Vue
- Создать таблицу правил

Инструкция по развертыванию и запуску проекта

Клиент:

- Загрузить проект https://github.com/moevm/mse_turing_tasks/tree/master/client
- В папке проекта установить зависимости с помощью команды **npm install**
- Для запуска выполнить команду **npm run serve**

Сервер:

- Установить интерпретатор Python3
- Загрузить проект https://github.com/moevm/mse_turing_tasks/tree/master/server
- Установить зависимости с помощью команды: **pip install –r requirements.txt**
- Создать новый исполняемый файл *.py и добавить в него строки:

```
from app import app
app.run()
```

- Запустить созданный ранее файл

БД:

- Установить зависимости командой **pip3 install -r requirements.txt**
- Выполнить скрипт **startMongo.sh**

Результаты работы: Клиент

1. Создана страница авторизации
2. Создан основной интерфейс веб-приложения
3. Реализована возможность ввода таблицы состояний машины
4. Реализована возможность задавать ленту для одномерной машины, размер плоскости и стартовую позицию для двумерной машины
5. Реализован вывод результата полного прогона машины в консоль

Результаты работы: Сервер

1. Создана универсальная многомерная машина состояний
   - Полный прогон
   - Пошаговый прогон
2. Реализованы классы многомерного поля и многомерной точки
3. Разработаны стандарты для взаимодействия сервера и клиента
4. Частично реализовано api соответствующее стандартам п.3
   - Имеется возможность полностью запустить машину любой мерности с необходимым набором передвижений, состояний, своим полем соответствующей мерности.

Результаты работы: БД

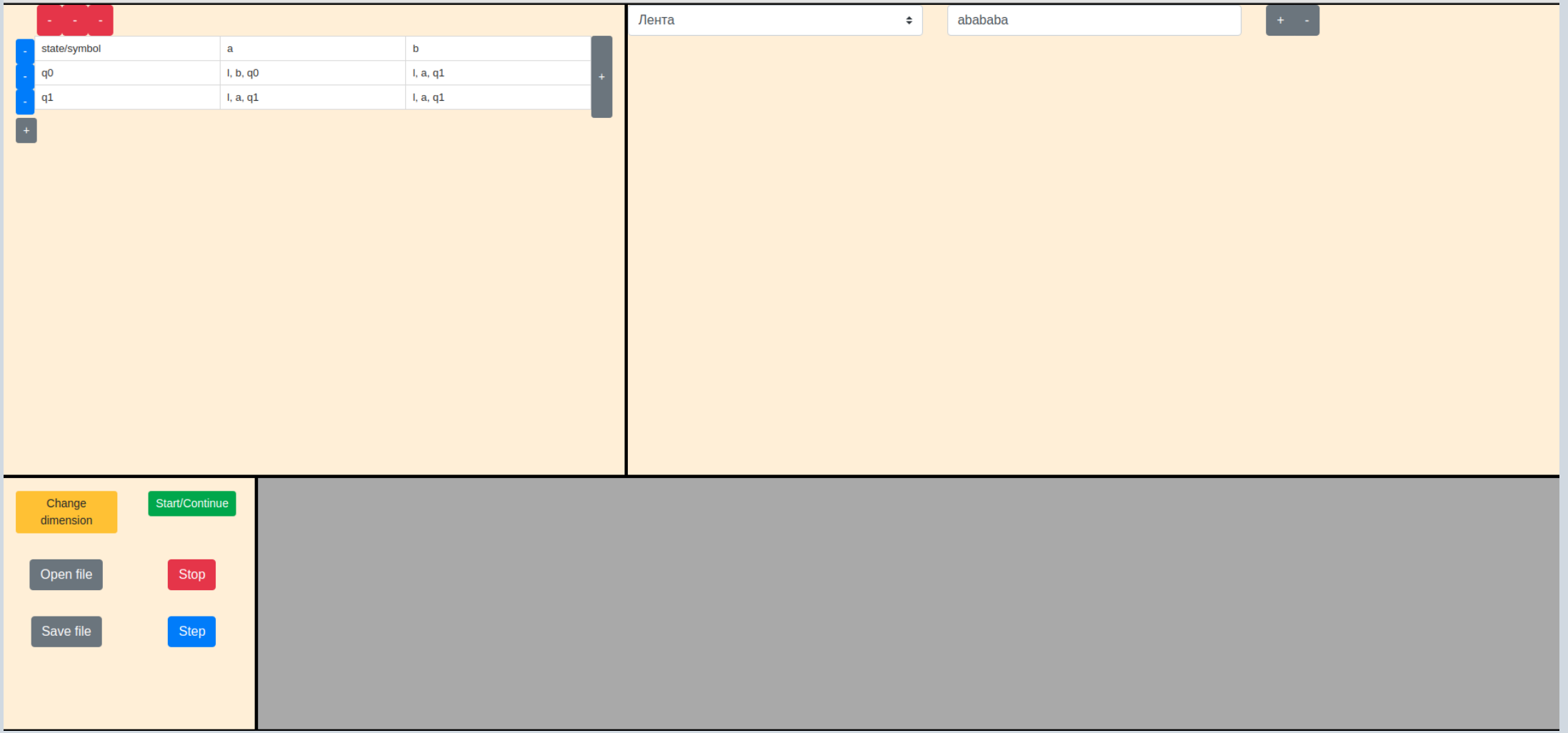
В ходе данной итерации были разработаны следующие классы:

- DataBase - для связи Python'a с mondoDB
  - Методы:insert_user(user), insert_program(program), find_user(id_), find_program(id_), users(), programs(), remove()
- User - класс пользователя, который начал работать с приложением
- Program - программа: поле, начальная позиция и набор правил, по который будет работать машина Тьюринга
- Session - сессия, для того, чтобы продолжить выполнения программы с сохраненного места
- State - состояние, элемент из списка состояний
- Way - путь: по символу на поле решает, какое действие ему соответствует
- Action - действие, то, что нужно выполнить при соответствующих ему состоянию и символу
- Position - позиция, координата (x, y) для возможности указать точку на плоскости

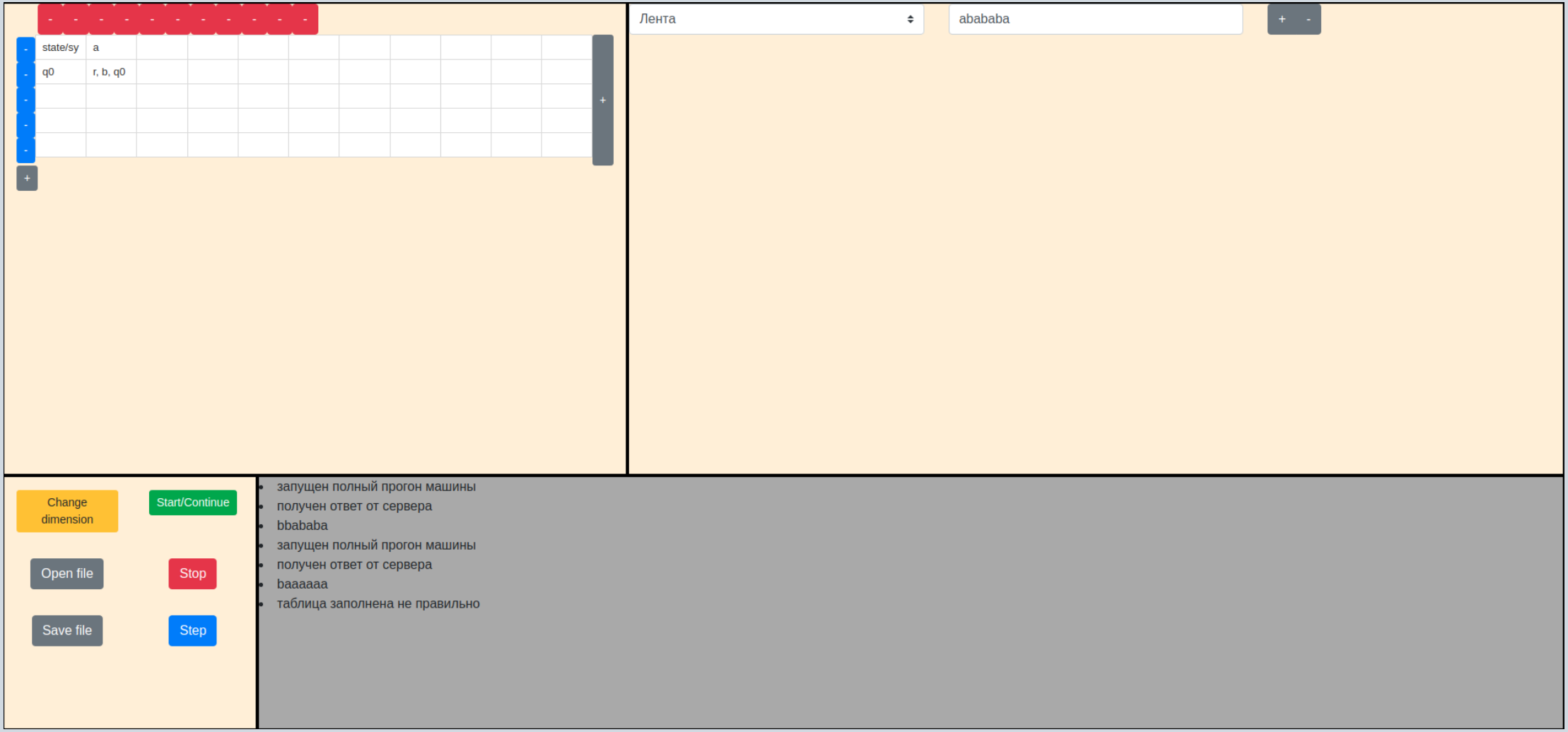
Для классов User, Program были разработаны методы

- __init__([_user|__program]=None, _json=None) -- для создания соотвествующих экземляров классов из [_user/_program] или соответвующим им JSON объектов
- to_json() -- метод, переводящий данный экземпляр класса в соответвуюищй ему объект

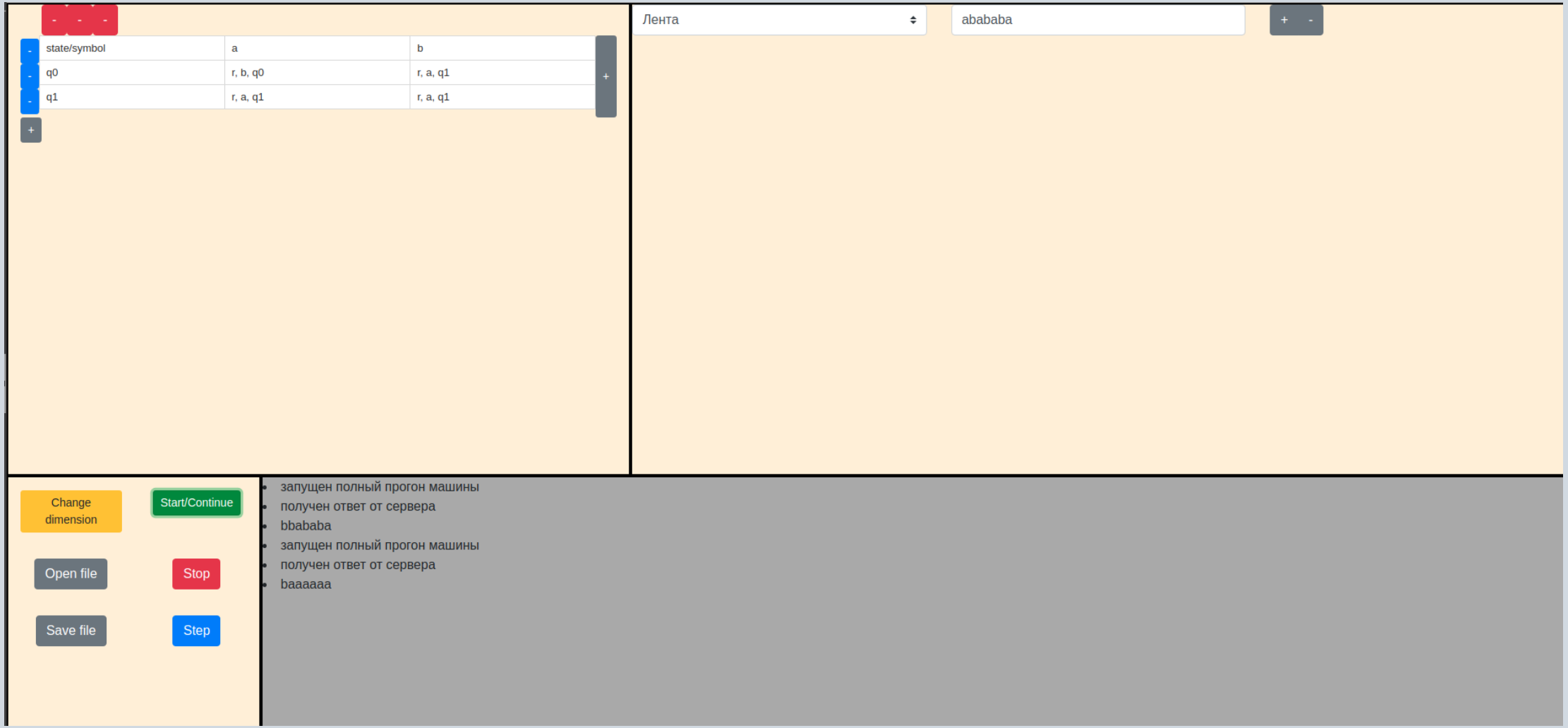
Корректный ввод данных в таблицу правил



Некорректный ввод данных в таблицу правил

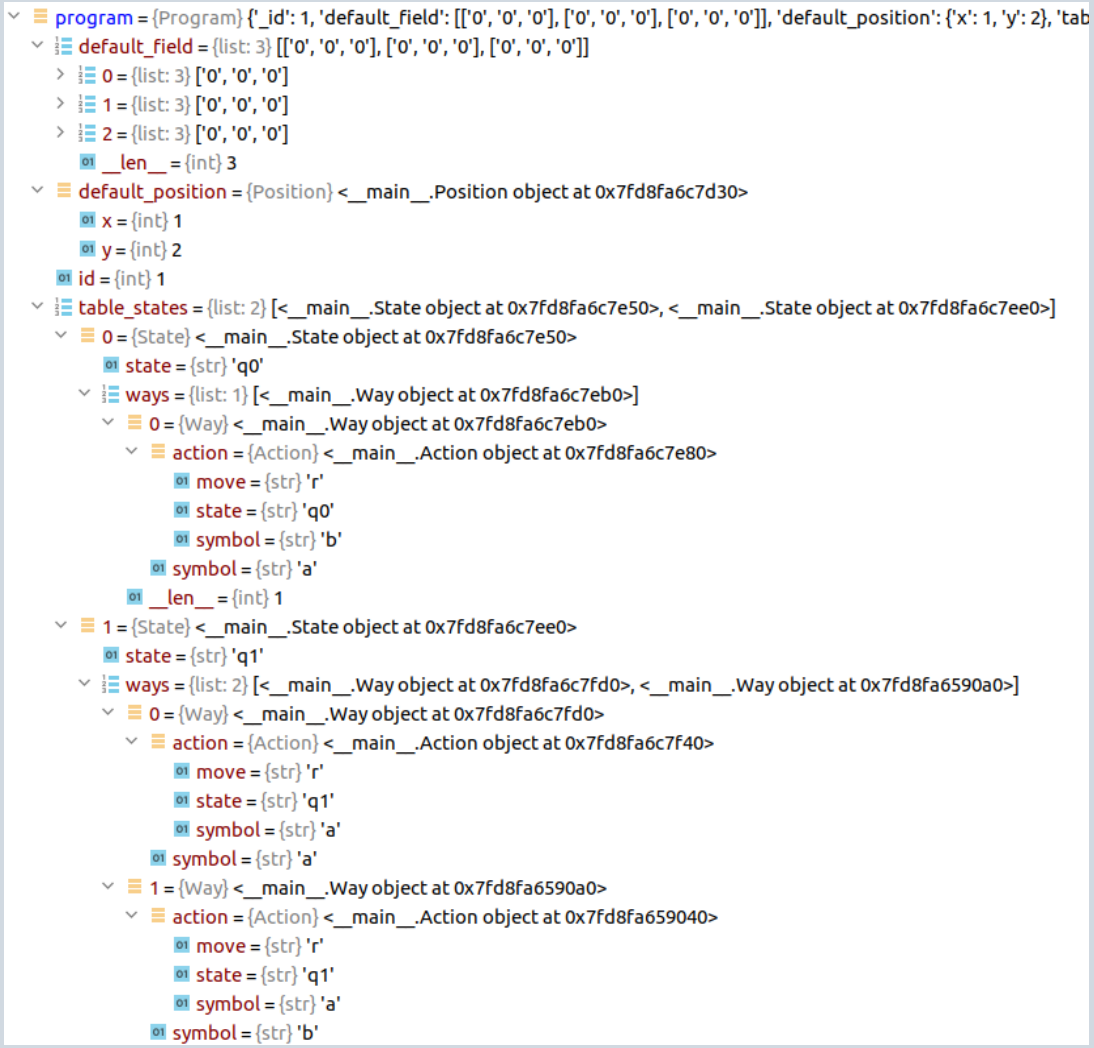


Исполнение программы



Преобразование объекта Program в json формат

```
{
  "_id": 1,
  "default_field": [
    ["0", "0", "0"],
    ["0", "0", "0"],
    ["0", "0", "0"]
  ],
  "default_position": {
    "x": 1,
    "y": 2
  },
  "table_states": [
    {
      "state": "q0",
      "ways": [
        {
          "symbol": "a",
          "action": {
            "symbol": "b",
            "state": "q0",
            "move": "r"
          }
        }
      ]
    },
    {
      "state": "q1",
      "ways": [
        {
          "symbol": "a",
          "action": {
            "symbol": "a",
            "state": "q1",
            "move": "r"
          }
        },
        {
          "symbol": "b",
          "action": {
            "symbol": "a",
            "state": "q1",
            "move": "r"
          }
        }
      ]
    }
  ]
}
```



```
program = {Program} {'_id': 1, 'default_field': [['0', '0', '0'], ['0', '0', '0'], ['0', '0', '0']], 'default_position': {'x': 1, 'y': 2}, 'tab
  default_field = {list: 3} [['0', '0', '0'], ['0', '0', '0'], ['0', '0', '0']]
    0 = {list: 3} ['0', '0', '0']
    1 = {list: 3} ['0', '0', '0']
    2 = {list: 3} ['0', '0', '0']
    __len__ = {int} 3
  default_position = {Position} <__main__.Position object at 0x7fd8fa6c7d30>
    x = {int} 1
    y = {int} 2
  id = {int} 1
  table_states = {list: 2} [<__main__.State object at 0x7fd8fa6c7e50>, <__main__.State object at 0x7fd8fa6c7ee0>]
    0 = {State} <__main__.State object at 0x7fd8fa6c7e50>
      state = {str} 'q0'
      ways = {list: 1} [<__main__.Way object at 0x7fd8fa6c7eb0>]
        0 = {Way} <__main__.Way object at 0x7fd8fa6c7eb0>
          action = {Action} <__main__.Action object at 0x7fd8fa6c7e80>
            move = {str} 'r'
            state = {str} 'q0'
            symbol = {str} 'b'
          symbol = {str} 'a'
        __len__ = {int} 1
    1 = {State} <__main__.State object at 0x7fd8fa6c7ee0>
      state = {str} 'q1'
      ways = {list: 2} [<__main__.Way object at 0x7fd8fa6c7fd0>, <__main__.Way object at 0x7fd8fa6590a0>]
        0 = {Way} <__main__.Way object at 0x7fd8fa6c7fd0>
          action = {Action} <__main__.Action object at 0x7fd8fa6c7f40>
            move = {str} 'r'
            state = {str} 'q1'
            symbol = {str} 'a'
          symbol = {str} 'a'
        1 = {Way} <__main__.Way object at 0x7fd8fa6590a0>
          action = {Action} <__main__.Action object at 0x7fd8fa659040>
            move = {str} 'r'
            state = {str} 'q1'
            symbol = {str} 'a'
          symbol = {str} 'b'
```

Результаты развёртывания

1. Сервер развёрнут на бесплатном хостинге flask веб-приложений pythonanywhere.com
   - Имеется возможность взаимодействия с api по адресу https://wintari.pythonanywhere.com
2. Клиент развёрнут на сервисе google Firebase и доступен по адресу https://turing-app-2020-etu.web.app/main
3. Получена полноценная распределённая 2-х уровневая архитектура приложения, позволяющая изменять клиент и сервер независимо друг от друга.

Планы на следующую итерацию

- Добавить в машину обработку внутренних ошибок: бесконечный цикл, вызов несуществующего состояния, вызов несуществующего перемещения.
- Добавить возможность выполнения программы по шагам с клиента
- Добавить отрисовку поля для 1 и 2 мерных машин.
- Добавить страницу регистрации
- Добавить сессии пользователей
- Осуществить переход на MongoDB Atlas
- Добавить сохранение и загрузку машины из файла
- Добавить обработку полей таблицы состояний и начальной координаты
- Улучшить дизайн UI: корректировка размера и позиции элемнетов, корректировка цветовой схемы и т.д.
- Улучшить логгирование: добавление дополнительной информации (время, адрес сервера и т.д.), представление структуры и т.д.
- Разработать юнит-тесты