Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Волгоградский государственный технический университет»

Кафедра «Программное обеспечение автоматизированных систем»

Семестровая работа

по курсу «Логика и теория алгоритмов»

Реализация программы, эмулирующей работу машины Тьюринга

Выполнил:

Студент группы ПрИн-366

Петренко А.А.

Проверил:

Преподаватель кафедры ПОАС

Пашаев С. В.

Волгоград, 2015

Содержание

[Задание 3](#_Toc438551908)

[1 Теория и понятия 4](#_Toc438551909)

[2 Детали реализации 5](#_Toc438551910)

[Приложение А Интерфейс программы 8](#_Toc438551911)

[Приложение Б Пример файлов, которые можно импортировать в программу 10](#_Toc438551912)

[Приложение В Исходные коды программы 11](#_Toc438551913)

# Задание

Необходимо разработать программу-эмулятор детерминированной машины Тьюринга. Программа должна обеспечивать:

* Наглядное отображение ленты машины, таблицы переходов; показывать текущее состояние и текущий символ ленты.
* Возможность пошагового выполнения.
* Возможность автоматического выполнения и остановки автоматического выполнения.
* Возможность импорта/экспорта ленты из файла/в файл.
* Возможность импорта/экспорта состояния управляющего устройства из файла/в файл.
* Возможность редактирования ленты в программе.
* Возможность редактирования управляющего устройства в программе.
* Корректную обработку ошибок входных данных.

# 1 Теория и понятия

Машина Тьюринга (МТ) — математическая абстракция, представляющая [вычислительную машину](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) общего вида. Была предложена [Аланом Тьюрингом](https://ru.wikipedia.org/wiki/%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3,_%D0%90%D0%BB%D0%B0%D0%BD_%D0%9C%D0%B0%D1%82%D0%B8%D1%81%D0%BE%D0%BD) в [1936](https://ru.wikipedia.org/wiki/1936) году для формализации понятия [алгоритма](https://ru.wikipedia.org/wiki/%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC).

Машина Тьюринга является расширением модели [конечного автомата](https://ru.wikipedia.org/wiki/%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82) и, согласно [тезису Чёрча — Тьюринга](https://ru.wikipedia.org/wiki/%D1%82%D0%B5%D0%B7%D0%B8%D1%81_%D0%A7%D1%91%D1%80%D1%87%D0%B0_%E2%80%94_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0), способна имитировать (при наличии соответствующей программы) любую машину, действие которой заключается в переходе от одного дискретного состояния к другому.

В состав Машины Тьюринга входит бесконечная в обе стороны лента, разделённая на ячейки, и управляющее устройство с конечным числом состояний.

Управляющее устройство может перемещаться влево и вправо по ленте, читать и записывать в ячейки символы некоторого конечного алфавита. Выделяется особый пустой символ, заполняющий все клетки ленты, кроме тех из них (конечного числа), на которых записаны входные данные.

В управляющем устройстве содержится таблица переходов, которая представляет алгоритм, реализуемый данной Машиной Тьюринга. Каждое правило из таблицы предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей клетке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо. Некоторые состояния Машины Тьюринга могут быть помечены как терминальные, и переход в любое из них означает конец работы, остановку алгоритма.

Машина Тьюринга называется детерминированной, если каждой комбинации состояния и ленточного символа в таблице соответствует не более одного правила, и недетерминированной в противном случае.

Конкретная машина Тьюринга задаётся перечислением элементов множества букв алфавита A, множества состояний Q и набором правил, по которым работает машина. Они имеют вид: (если головка находится в состоянии , а в обозреваемой ячейке записана буква , то головка переходит в состояние , в ячейку вместо записывается , головка делает движение , которое имеет три варианта: на ячейку влево (L), на ячейку вправо (R), остаться на месте (N)). Для каждой возможной конфигурации имеется ровно одно правило (для недетерминированной машины Тьюринга может быть большее количество правил). Правил нет только для заключительного состояния, попав в которое машина останавливается. Кроме того, необходимо указать конечное и начальное состояния, начальную конфигурацию на ленте и расположение головки машины.

# 2 Детали реализации

Программа разработана на языке С++ с использованием библиотеки Qt 5.5. Интерфейс программа приведен в приложении А.

Входные и выходные данные, а точнее файлы ленты и управляющего устройства машины, которые можно импортировать/экспортировать являются json файлами и имеют определенную структуру.

Для корректного импорта файла ленты в программу, он должен иметь следующий вид:

{

"tape-view": ["<символ 1>", "<символ 2>", … , "<символ n>"],

"empty-symbol" : "<пустой символ>",

"pointer-index": <число>

}

Для корректного импорта файла управляющего устройства, он должен иметь следующий вид:

{

"alphabet": ["<символ 1>", "<символ 2>", … , "<символ n>"],

"states": ["<состояние 1>", "<состояние 2>", … , "<состояние n>"],

"begin-state": "<состояние i>",

"end-state": " состояние i",

"commands":

{

"<<символ i><состояние j>>":

{

"new-symbol": "<символ>",

"new-state": "<состояние>",

"action": "<действие>"

},

"<<символ j><состояние i>>":

{

"new-symbol": "<символ>",

"new-state": "<состояние>",

"action": "<действие>"

}

}

}

Команды должны быть заданы для каждой пары <символ><состояние>, где символы берутся из алфавита машины, а состояние – из множества состояний. Действие может быть либо “R” – смещение вправо, либо “L” – смещение влево, либо “N” – смещение отсутствует. Если команд будет недостаточно, программа выведет сообщение об этом.

Пример файлов, которые можно импортировать в программу приведен в приложении Б.

Также программа проверяет соответствие ленты и управляющего устройства, причем управляющее устройство имеет приоритет над лентой. Если символы ленты не соответствуют символам алфавита управляющего устройства, машина не сможет начать работу. Для начала работы необходимо исправить символы ленты.

При добавлении нового состояния из программы оно будет автоматически помечено как начальное, конечное и текущее, при условии, что до этого управляющее устройство их не содержало. Также можно принудительно задать новое состояние начальным, конечным и/или текущим.

При добавлении нового символа алфавита из программы, он автоматически будет помечен как пустой символ, при условии, что до этого в ленте он не был задан. Также можно принудительно задать новый символ алфавита пустым символом.

При добавлении новой команды из программы, необходимо соблюдать установленный синтаксис: <символ алфавита>-<состояние>-<действие>. При этом команда будет проверена на корректность, а именно принадлежность символа алфавита текущему алфавиту, состояния – текущему множеству состояний и команды – возможным командам.

При добавлении нового символа ленты он будет проверен на принадлежность алфавиту, при условии, что тот задан. Если алфавит не задан, символ проверен не будет.

После достижения машиной конечного состояния, будет выведено сообщение об этом и машина завершит работу. Для продолжения работы, необходимо перезагрузить машину.

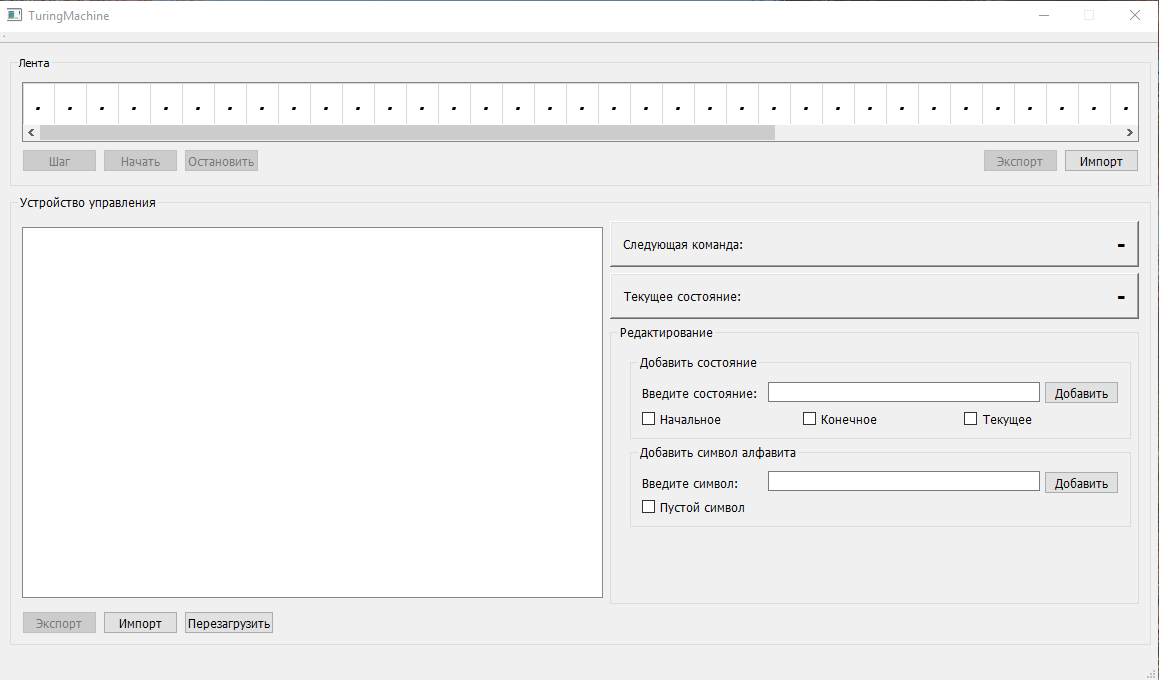
Внутри программы машина представлена тремя классами:

1. Machine – логическая модель машины, которая управляет лентой и управляющим устройством, а также состояниями машины. Именно с ней работает представление – класс главного окна TuringMachine.
2. Tape – класс ленты машины, агрегирован в класс Machine, отвечает за ленту машины.
3. Controller – класс управляющего устройства, агрегирован в класс Machine, отвечает за управляющее устройство машины.

Исходные коды программы приведены в приложении В.

# Приложение А

# Интерфейс программы



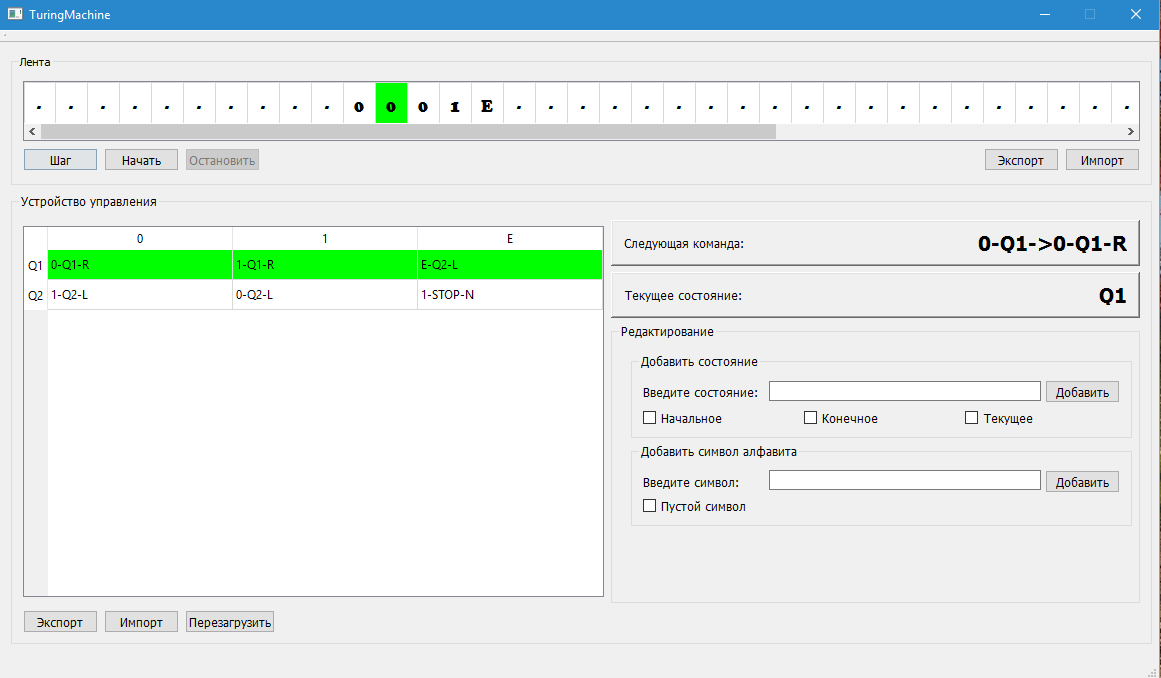
Рисунок 1. Главное окно программы при запуске.

Рисунок 2. Главное окно программы, когда машина готова к работе.

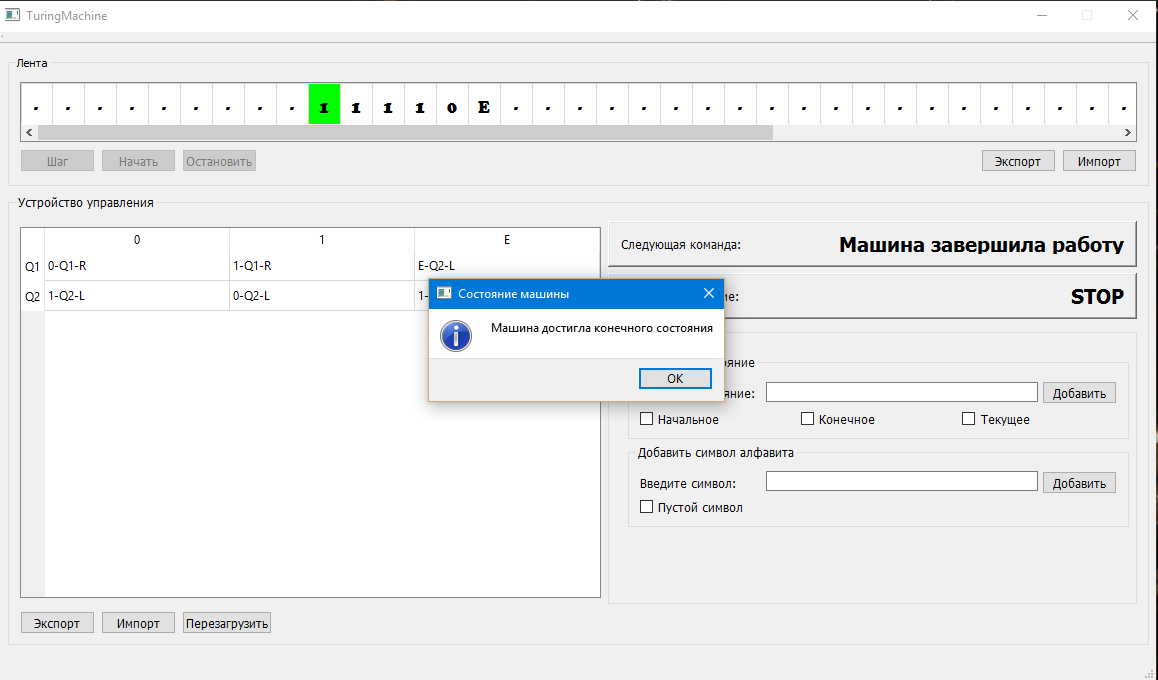


Рисунок 3. Главное окно программы, когда машина завершила работу.

# Приложение Б

# Пример файлов, которые можно импортировать в программу

Файл ленты машины:

{"tape-view": ["0", "0", "0", "1", "E"],

"empty-symbol" : "E",

"pointer-index": 1}

Файл управляющего устройства машины:

{"alphabet": ["0", "1", "E"],

"states": ["Q1", "Q2", "STOP"],

"begin-state": "Q1",

"end-state": "STOP",

"commands":

{

"1Q1":

{

"new-symbol": "1",

"new-state": "Q1",

"action": "R"},

"0Q1":

{

"new-symbol": "0",

"new-state": "Q1",

"action": "R"},

"EQ1":

{

"new-symbol": "E",

"new-state": "Q2",

"action": "L"},

"1Q2":

{

"new-symbol": "0",

"new-state": "Q2",

"action": "L"},

"0Q2":

{

"new-symbol": "1",

"new-state": "Q2",

"action": "L"},

"EQ2":

{

"new-symbol": "1",

"new-state": "STOP",

"action": "N"}}

}

# Приложение В

# Исходные коды программы

Файл controller.h

#ifndef CONTROLLER\_H

#define CONTROLLER\_H

#include <QObject>

#include <QMap>

struct Command;

class Controller : public QObject

{

Q\_OBJECT

public:

Controller();

~Controller();

bool setCurrentState(const QString& state);

void setStates(const QStringList& states);

void setAlphabet(const QStringList& alphabet);

void setBeginEndStates(const QString& beginState,

const QString& endState) throw(QString&);

void addComand(const QString& key, const Command& cmd) throw(QString&);

QStringList getAlphabet() const;

QStringList getStates(bool includeStopState) const;

QString getCurrentState() const;

void reset();

private:

friend class Machine;

bool isStateValid(const QString& state) const;

bool isAlphabetSymbolValid(const QString& symbol) const;

QString endState;

QString beginState;

QString currentState;

QStringList states;

QStringList alphabet;

QMap<QString, Command> commands;

};

struct Command

{

QString newSymbol;

QString newState;

QString action;

Command()

{

this->newSymbol = "";

this->newState = "";

this->action = "";

}

Command(const QString& newSymbol, const QString& newState, const QString& action)

{

this->newSymbol = newSymbol;

this->newState = newState;

this->action = action;

}

Command& operator=(const Command& other)

{

this->newSymbol = other.newSymbol;

this->newState = other.newState;

this->action = other.action;

return \*this;

}

};

#endif // CONTROLLER\_H

Файл controller.cpp

#include "controller.h"

Controller::Controller()

{

}

Controller::~Controller()

{

}

bool Controller::setCurrentState(const QString& state)

{

if (isStateValid(state))

{

this->currentState = state;

return true;

}

return false;

}

QString Controller::getCurrentState() const

{

return currentState;

}

void Controller::setStates(const QStringList& states)

{

this->states = states;

}

void Controller::setAlphabet(const QStringList& alphabet)

{

this->alphabet = alphabet;

}

void Controller::setBeginEndStates(const QString& beginState, const QString& endState) throw(QString&)

{

QString beginSt, endSt;

if (isStateValid(beginState))

{

this->beginState = beginState;

this->currentState = beginState;

}

else

{

QString errorString = "Начальное состояние задано не верно.\nОтсутствует в множестве состояний";

throw errorString;

}

if (isStateValid(endState))

{

this->endState = endState;

}

else

{

QString errorString = "Конечное состояние задано не верно.\nОтсутствует в множестве состояний";

throw errorString;

}

}

void Controller::addComand(const QString& key, const Command& cmd) throw(QString&)

{

if (!isAlphabetSymbolValid(cmd.newSymbol))

{

QString errorString = "Символ задан не верно.\nОтсутствует в алфавите.";

throw errorString;

}

if (!isStateValid(cmd.newState))

{

QString errorString = "Cостояние задано не верно.\nОтсутствует в множестве состояний";

throw errorString;

}

if (cmd.action != "R" && cmd.action != "L" && cmd.action != "N")

{

QString errorString = "Действие задано не верно.\nВозможные действия:\"R\",\"L\",\"N\".";

throw errorString;

}

commands.insert (key,cmd);

}

QStringList Controller::getStates(bool includeStopState) const

{

QStringList tmpStates = states;

if (!includeStopState)

{

int index = tmpStates.indexOf(endState);

tmpStates.removeAt(index);

}

return tmpStates;

}

void Controller::reset()

{

alphabet.clear();

states.clear();

beginState.clear();

endState.clear();

currentState.clear();

commands.clear();

}

QStringList Controller::getAlphabet() const

{

return alphabet;

}

bool Controller::isStateValid(const QString& state) const

{

return states.contains(state, Qt::CaseSensitive);

}

bool Controller::isAlphabetSymbolValid(const QString& symbol) const

{

return alphabet.contains(symbol, Qt::CaseSensitive);

}

Файл tape.h

#ifndef TAPE\_H

#define TAPE\_H

#include <QObject>

#define TAPE\_BLANK "-"

class Tape : public QObject

{

Q\_OBJECT

public:

Tape();

~Tape();

void setData(const QStringList &tapeView,

unsigned int tapePointer,

const QString& emptySymbol);

void reset();

QStringList getTapeView() const;

unsigned int getTapePointer() const;

QString getCurrentSymbol() const;

QString getEmptySymbol() const;

void setCurrentSymbol(const QString& symbol);

void setSymbolAt(const QString& symbol, unsigned int index);

void setCurrentTapePointer(int index) throw(QString&);

void appendToTape(const QString &symbol);

void prependToTape(const QString &symbol);

void incrementPointer();

void decrementPointer();

private:

friend class Machine;

QStringList tapeView;

unsigned int tapePointer;

QString emptySymbol;

};

#endif // TAPE\_H

Файл tape.cpp

#include "tape.h"

Tape::Tape()

{

}

Tape::~Tape()

{

}

QStringList Tape::getTapeView() const

{

return tapeView;

}

unsigned int Tape::getTapePointer() const

{

return tapePointer;

}

QString Tape::getCurrentSymbol() const

{

return tapeView.at(tapePointer);

}

QString Tape::getEmptySymbol() const

{

return emptySymbol;

}

void Tape::setCurrentSymbol(const QString& symbol)

{

tapeView[tapePointer] = symbol;

}

void Tape::setSymbolAt(const QString& symbol, unsigned int index)

{

tapeView[index] = symbol;

}

void Tape::setCurrentTapePointer(int index) throw(QString&)

{

int size = tapeView.size();

/\* Текущая лента в крайней левой позиции \*/

if (index == size)

{

tapeView.push\_back (emptySymbol);

tapePointer = index;

}

else if (index == -1)

{

tapeView.push\_front(emptySymbol);

tapePointer = 0;

}

else if (index > size)

{

QString errorString = "Некорректное смещение головки ленты";

throw errorString;

}

else

{

tapePointer = index;

if (tapeView[tapePointer] == TAPE\_BLANK)

{

tapeView[tapePointer] = emptySymbol;

}

}

}

void Tape::appendToTape(const QString &symbol)

{

tapeView.append(symbol);

}

void Tape::prependToTape(const QString &symbol)

{

tapeView.prepend(symbol);

}

void Tape::incrementPointer()

{

if (tapePointer < (tapeView.size() - 1) )

{

++tapePointer;

}

}

void Tape::decrementPointer()

{

if (tapePointer > 0)

{

--tapePointer;

}

}

void Tape::setData(const QStringList &tapeView, unsigned int tapePointer, const QString& emptySymbol)

{

this->tapeView = tapeView;

this->tapePointer = tapePointer;

this->emptySymbol = emptySymbol;

}

void Tape::reset()

{

tapeView.clear();

tapePointer = 0;

emptySymbol.clear();

}

Файл machine.h

#ifndef MACHINE\_H

#define MACHINE\_H

#include <QObject>

#include "controller.h"

#include "tape.h"

#define TAPE\_BLANK "-"

#define DEFAULT\_EMPTY\_SYMBOL ""

class Machine : public QObject

{

Q\_OBJECT

signals:

void machineError(QString&);

void machineFinished();

void machineStopped();

void machineReady(bool);

void tapeChanged();

void tapeSymbolChanged(unsigned int index, QString newSymbol);

void tapePointerChanged(unsigned int oldPos, unsigned int newPos);

public:

Machine();

~Machine();

/\* Сет-методы \*/

void setTape(const QStringList& tapeView,

unsigned int tapePointer,

const QString& emptySymbol = DEFAULT\_EMPTY\_SYMBOL);

void setStates(const QStringList& states);

void setAlphabet(const QStringList& alphabet);

void setBeginEndStates(const QString& beginState,

const QString& endState) throw(QString&);

void addComand(const QString& key, const Command& cmd) throw(QString&);

void addState(const QString& state);

void addSymbol(const QString& symbol);

void appendToTape(const QString& symbol);

void prependToTape(const QString& symbol);

void incrementTapePointer();

void decrementTapePointer();

void setTapeSymbol(const QString& symbol, unsigned int index);

void setCurrentState(const QString& state);

void setBeginState(const QString& state);

void setEndState(const QString& state);

void setEmptySymbol(const QString& symbol);

/\* Гет-методы \*/

QStringList getTape() const;

QString getEmptySymbol() const;

QStringList getAlphabet() const;

unsigned int getTapePointer() const;

QString getCurrentTapeSymbol() const;

QString getCommand(const QString& key) const;

QStringList getStates(bool includeStopState) const;

QString getBeginState() const;

QString getEndState() const;

QString getCurrentState() const;

QStringList exportTape() const;

QMap<QString, Command> getCommands() const;

/\* Методы проверки и управления состоянием машины \*/

bool isReady() const;

bool isTapeLoaded() const;

bool isControllerLoaded() const;

bool isFinished() const;

bool isTapeSymbolValid(const QString& symbol) const;

bool isStateValid(const QString& state) const;

bool isStateUnique(const QString& state) const;

bool isSymbolUnique(const QString& symbol) const;

void setControllerLoaded();

void setTapeLoaded();

/\* Перезагрузка машины \*/

void reset();

void resetController();

void resetTape();

/\* Работа машины \*/

void oneStep(bool emitSignals);

void startWork();

void stopWork();

private:

/\* Проверить, соответствует ли лента УУ \*/

void verifyTape();

unsigned int getNewTapePointer(const QString& action, unsigned int prevPointer);

Tape\* tape;

Controller\* controller;

bool tapeLoaded;

bool controllerLoaded;

enum class MachineState

{

IN\_WORK,

PAUSED,

FINISHED,

}machineState;

};

#endif // MACHINE\_H

Файл machine.cpp

#include "machine.h"

Machine::Machine()

{

tape = new Tape;

controller = new Controller;

tapeLoaded = false;

controllerLoaded = false;

machineState = MachineState::PAUSED;

}

Machine::~Machine()

{

delete tape;

delete controller;

}

void Machine::setTape(const QStringList& tapeView, unsigned int tapePointer, const QString& emptySymbol)

{

tape->setData(tapeView, tapePointer,emptySymbol);

}

void Machine::setCurrentState(const QString& state)

{

controller->setCurrentState(state);

}

void Machine::setBeginState(const QString& state)

{

controller->beginState = state;

}

void Machine::setEndState(const QString& state)

{

controller->endState = state;

}

void Machine::setEmptySymbol(const QString& symbol)

{

tape->emptySymbol = symbol;

}

void Machine::setAlphabet(const QStringList& alphabet)

{

controller->setAlphabet(alphabet);

}

void Machine::setBeginEndStates(const QString& beginState, const QString& endState) throw(QString&)

{

try

{

controller->setBeginEndStates(beginState, endState);

}

catch (QString& errorString)

{

emit machineError(errorString);

}

}

void Machine::addComand(const QString& key, const Command& cmd) throw(QString&)

{

try

{

controller->addComand(key, cmd);

}

catch (QString& errorString)

{

emit machineError(errorString);

}

}

void Machine::addState(const QString& state)

{

controller->states.append(state);

if (!controller->alphabet.isEmpty())

{

setControllerLoaded();

}

}

void Machine::addSymbol(const QString& symbol)

{

controller->alphabet.append(symbol);

if (!controller->states.isEmpty())

{

setControllerLoaded();

}

}

void Machine::appendToTape(const QString& symbol)

{

tape->appendToTape(symbol);

}

void Machine::prependToTape(const QString& symbol)

{

tape->prependToTape(symbol);

}

void Machine::incrementTapePointer()

{

tape->incrementPointer();

}

void Machine::decrementTapePointer()

{

tape->decrementPointer();

}

void Machine::setTapeSymbol(const QString& symbol, unsigned int index)

{

tape->setSymbolAt(symbol, index);

}

QStringList Machine::getStates(bool includeStopState) const

{

return controller->getStates(includeStopState);

}

QString Machine::getBeginState() const

{

return controller->beginState;

}

QString Machine::getEndState() const

{

return controller->endState;

}

QStringList Machine::getTape() const

{

return tape->getTapeView();

}

QString Machine::getEmptySymbol() const

{

return tape->getEmptySymbol();

}

unsigned int Machine::getTapePointer() const

{

return tape->getTapePointer();

}

QString Machine::getCommand(const QString& key) const

{

QString command = "";

if (controller->commands.contains(key))

{

Command cmd = controller->commands.value(key);

command = cmd.newSymbol + "-" + cmd.newState + "-" + cmd.action;

}

return command;

}

QString Machine::getCurrentState() const

{

return controller->getCurrentState();

}

QStringList Machine::exportTape() const

{

QStringList toExport = tape->getTapeView();

toExport.removeAll(TAPE\_BLANK);

return toExport;

}

QMap<QString, Command> Machine::getCommands() const

{

return controller->commands;

}

QString Machine::getCurrentTapeSymbol() const

{

return tape->getCurrentSymbol();

}

bool Machine::isReady() const

{

return tapeLoaded && controllerLoaded;

}

bool Machine::isTapeLoaded() const

{

return tapeLoaded;

}

bool Machine::isControllerLoaded() const

{

return controllerLoaded;

}

bool Machine::isFinished() const

{

return machineState == MachineState::FINISHED;

}

void Machine::setTapeLoaded()

{

tapeLoaded = true;

if (controllerLoaded)

{

verifyTape();

}

}

void Machine::setControllerLoaded()

{

controllerLoaded = true;

if (tapeLoaded)

{

verifyTape();

}

}

bool Machine::isTapeSymbolValid(const QString& symbol) const

{

if (controllerLoaded)

{

return controller->isAlphabetSymbolValid(symbol);

}

return true;

}

bool Machine::isStateValid(const QString& state) const

{

if (controllerLoaded)

{

return controller->isStateValid(state);

}

return true;

}

bool Machine::isStateUnique(const QString& state) const

{

return !(controller->states.contains(state, Qt::CaseSensitive));

}

bool Machine::isSymbolUnique(const QString& symbol) const

{

return !(controller->alphabet.contains(symbol, Qt::CaseSensitive));

}

void Machine::reset()

{

tape->reset();

controller->reset();

tapeLoaded = false;

controllerLoaded = false;

}

void Machine::resetController()

{

controller->reset();

controllerLoaded = false;

}

void Machine::resetTape()

{

tape->reset();

tapeLoaded = false;

}

void Machine::oneStep(bool emitSignals)

{

/\* Получаем команду по символу ленты и текущему состоянию \*/

QString currentState = getCurrentState();

QString currentSymbol = getCurrentTapeSymbol();

QString command = getCommand(currentSymbol + currentState);

if (command.isEmpty())

{

machineState = MachineState::FINISHED;

QString errorString = "Не задана команда для символа \"" +

currentSymbol + "\" и состояния \"" +

currentState + "\".";

emit machineError(errorString);

return;

}

/\* Разбираем команду на данные \*/

QStringList splittedCommand = command.split('-');

QString newSymbol = splittedCommand.at(0);

QString newState = splittedCommand.at(1);

QString action = splittedCommand.at(2);

unsigned int oldTapePointer = tape->getTapePointer();

/\* Устанавливаем новый символ в старую позицию ленты \*/

tape->setSymbolAt(newSymbol,oldTapePointer);

if (emitSignals)

{

emit tapeSymbolChanged(oldTapePointer, newSymbol);

}

/\* Перемещаем головку ленты на новую позицию \*/

int newTapePointer = getNewTapePointer(action, oldTapePointer);

try

{

tape->setCurrentTapePointer(newTapePointer);

newTapePointer = tape->getTapePointer();

}

catch (QString& errorString)

{

machineState = MachineState::FINISHED;

emit machineError(errorString);

return;

}

if (action != "N" && emitSignals)

{

emit tapePointerChanged(oldTapePointer, newTapePointer);

}

/\* Устанавливаем новое состояние УУ \*/

setCurrentState(newState);

/\* Проверка конечного состояния \*/

if (controller->currentState == controller->endState)

{

machineState = MachineState::FINISHED;

emit machineFinished();

}

}

void Machine::startWork()

{

while (machineState != MachineState::FINISHED)

{

oneStep(false);

}

emit tapeChanged();

}

void Machine::stopWork()

{

machineState = MachineState::FINISHED;

emit machineStopped();

}

void Machine::verifyTape()

{

bool isOk = true;

QStringList tapeView = tape->getTapeView();

QString emptySymbol = tape->getEmptySymbol();

for (auto i : tapeView)

{

if (i != TAPE\_BLANK && !controller->isAlphabetSymbolValid(i))

{

isOk = false;

break;

}

}

if (!controller->isAlphabetSymbolValid(emptySymbol))

{

isOk = false;

}

if (!isOk)

{

QString message = "Лента машины не соответствует алфавиту управляющего устройства.";

tapeLoaded = false;

emit machineError(message);

}

else

{

emit machineReady(true);

}

}

unsigned int Machine::getNewTapePointer(const QString& action, unsigned int prevPointer)

{

if (action == "R")

{

return (prevPointer + 1);

}

else if (action == "L")

{

return (prevPointer - 1);

}

else if (action == "N")

{

return prevPointer;

}

else

{

QString errorString = "Найдена неизвестная команда \"" + action + "\".";

emit machineError(errorString);

}

}

QStringList Machine::getAlphabet() const

{

return controller->getAlphabet();

}

void Machine::setStates(const QStringList& states)

{

controller->setStates(states);

}

Файл turingmachine.h

#ifndef TURINGMACHINE\_H

#define TURINGMACHINE\_H

#include <QtWidgets/QMainWindow>

#include <QFileDialog>

#include <QList>

#include <QDebug>

#include <QIODevice>

#include <QJsonDocument>

#include <QJsonArray>

#include <QJsonObject>

#include <QMessageBox>

#include <QDataStream>

#include <QtConcurrent/QtConcurrentRun>

#include <tuple>

#include "ui\_turingmachine.h"

#include "machine.h"

#define DEFAULT\_CONTROLLER\_FILE "./controller.json"

#define DEFAULT\_TAPE\_FILE "./tape.json"

#define DEFAULT\_TAPE\_COLUMNS 50

#define TAPE\_OFFSET 10

#define TAPE\_BLANK "-"

class TuringMachine : public QMainWindow

{

Q\_OBJECT

public:

TuringMachine(QWidget \*parent = 0);

~TuringMachine();

private slots:

void resetMachine();

void addState();

void addSymbol();

void importTape();

void exportTape();

void importController();

void exportController();

void machineStep();

void machineBeginWork();

void machineStopWork();

void machineErrorReceived(QString &errorString);

void machineFinished();

void machineStopped();

void machineTapeSymbolChanged(unsigned int index, QString newSymbol);

void machineTapePointerChanged(unsigned int oldTapePointer, unsigned int newTapePointer);

void expandTape(int currentRow,

int currentColumn,

int previousRow,

int previousColumn);

void showLoadedTape();

void showLoadedController();

void showCurrentState();

void showNextCommand();

void repaintTape();

void setControlButtonsEnabled(bool isEnabled);

void rememberCurrentTapeSymbol(QTableWidgetItem\* item);

void rememberCurrentCommand(QTableWidgetItem\* item);

void modifyCurrentTapeSymbol(QTableWidgetItem\* item);

void modifyCurrentCommand(QTableWidgetItem\* item);

private:

QByteArray prepareDataToExportTape() const;

QByteArray prepareDataToExportController() const;

void addRow(QTableWidget\* table, const QString& row);

void addColumn(QTableWidget\* table, const QString& column);

void initializeTape();

void paintRow(int rowNumber, const QColor& color);

void setConnections() const;

void parseControllerFile(const QString& data) throw(QString&);

void parseCommands(const QJsonObject commands) throw(QString&);

std::*tuple*<QStringList, unsigned int, QString> parseTapeFile(const QString& data) const throw(QString&);

Ui::TuringMachineClass ui;

Machine\* machine;

int tapeOffet;

QString currentTapeSymbol;

QString currentCommand;

};

#endif // TURINGMACHINE\_H

Файл turingmachine.cpp

#include "turingmachine.h"

TuringMachine::TuringMachine(QWidget \*parent)

: QMainWindow(parent)

{

ui.setupUi(this);

setControlButtonsEnabled(false);

ui.stop->setEnabled(false);

tapeOffet = TAPE\_OFFSET;

currentTapeSymbol = "";

machine = new Machine;

ui.tape->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

ui.controller->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

initializeTape();

setConnections();

}

TuringMachine::~TuringMachine()

{

delete machine;

}

void TuringMachine::resetMachine()

{

ui.tape->blockSignals(true);

ui.controller->blockSignals(true);

machine->reset();

setControlButtonsEnabled(false);

ui.stop->setEnabled(false);

ui.exportTape->setEnabled(false);

ui.exportController->setEnabled(false);

ui.newState->clear();

ui.newSymbol->clear();

ui.beginState->setChecked(false);

ui.endState->setChecked(false);

ui.currState->setChecked(false);

ui.emptySymbol->setChecked(false);

ui.nextCommand->setText(TAPE\_BLANK);

ui.currentState->setText(TAPE\_BLANK);

ui.tape->clear();

ui.tape->setColumnCount(0);

ui.tape->setRowCount(0);

ui.controller->clear();

ui.controller->setColumnCount(0);

ui.controller->setRowCount(0);

initializeTape();

ui.tape->blockSignals(false);

ui.controller->blockSignals(false);

}

void TuringMachine::addState()

{

ui.controller->blockSignals(true);

QString newState = ui.newState->text();

if (!newState.isEmpty())

{

bool isBeginState = ui.beginState->isChecked();

bool isEndState = ui.endState->isChecked();

bool isCurrState = ui.currState->isChecked();

if (machine->isStateUnique(newState))

{

machine->addState(newState);

if (isBeginState || machine->getBeginState().isEmpty())

{

machine->setBeginState(newState);

}

if (isEndState || machine->getEndState().isEmpty())

{

machine->setEndState(newState);

}

if (isCurrState || machine->getCurrentState().isEmpty())

{

machine->setCurrentState(newState);

}

addRow(ui.controller, newState);

ui.exportController->setEnabled(true);

}

else

{

QMessageBox::warning(this, "Ошибка", "Такое состояние уже существует в машине.");

ui.controller->blockSignals(false);

return;

}

}

else

{

QMessageBox::warning(this, "Ошибка", "Название нового состояния не введено.");

ui.controller->blockSignals(false);

return;

}

ui.controller->blockSignals(false);

}

void TuringMachine::addSymbol()

{

ui.controller->blockSignals(true);

QString newSymbol = ui.newSymbol->text();

if (!newSymbol.isEmpty())

{

bool isEmptySymbol = ui.emptySymbol->isChecked();

if (machine->isSymbolUnique(newSymbol))

{

machine->addSymbol(newSymbol);

if (isEmptySymbol || machine->getEmptySymbol().isEmpty())

{

machine->setEmptySymbol(newSymbol);

}

addColumn(ui.controller, newSymbol);

showCurrentState();

ui.exportController->setEnabled(true);

}

else

{

QMessageBox::warning(this, "Ошибка", "Такой символ уже существует в алфавите машины.");

ui.controller->blockSignals(false);

return;

}

}

else

{

QMessageBox::warning(this, "Ошибка", "Символ не введен.");

ui.controller->blockSignals(false);

return;

}

ui.controller->blockSignals(false);

}

void TuringMachine::importTape()

{

ui.tape->blockSignals(true);

QString fileName = QFileDialog::getOpenFileName( this,

"Выберите файл ленты",

"./",

"JSON файлы(\*.json)");

if (!fileName.isEmpty())

{

QFile file(fileName);

if (file.open(QIODevice::ReadOnly | QIODevice::Text))

{

QDataStream stream(&file);

QString data = file.readAll();

try

{

auto tuple = parseTapeFile(data);

machine->setTape( std::*get*<0>(tuple),

std::*get*<1>(tuple),

std::*get*<2>(tuple));

}

catch (QString& errorString)

{

QMessageBox::critical(this, "Ошибка разбора файла \"" + fileName + "\"", errorString);

file.close();

ui.tape->blockSignals(false);

return;

}

file.close();

machine->setTapeLoaded();

ui.exportTape->setEnabled(true);

showLoadedTape();

}

}

ui.tape->blockSignals(false);

}

void TuringMachine::exportTape()

{

QString fileName = QFileDialog::getSaveFileName( this,

"Сохранить файл ленты",

"./",

"JSON файлы(\*.json)");

if (!fileName.isEmpty())

{

QFile file(fileName);

if (file.open(QIODevice::WriteOnly | QIODevice::Text))

{

QByteArray toExport = prepareDataToExportTape();

file.write(toExport);

file.close();

}

}

}

void TuringMachine::importController()

{

ui.controller->blockSignals(true);

QString fileName = QFileDialog::getOpenFileName( this,

"Выберите файл ленты",

"./",

"JSON файлы(\*.json)");

if (!fileName.isEmpty())

{

QFile file(fileName);

if (file.open(QIODevice::ReadOnly | QIODevice::Text))

{

QDataStream stream(&file);

QString data = file.readAll();

try

{

parseControllerFile(data);

}

catch (QString& errorString)

{

QMessageBox::critical(this, "Ошибка разбора файла \"" + fileName + "\"", errorString);

file.close();

machine->resetController();

ui.controller->blockSignals(false);

return;

}

file.close();

machine->setControllerLoaded();

ui.exportController->setEnabled(true);

showLoadedController();

}

}

ui.controller->blockSignals(false);

}

void TuringMachine::exportController()

{

QString fileName = QFileDialog::getSaveFileName( this,

"Сохранить файл управляющего устройства",

"./",

"JSON файлы(\*.json)");

if (!fileName.isEmpty())

{

QFile file(fileName);

if (file.open(QIODevice::WriteOnly | QIODevice::Text))

{

QByteArray toExport = prepareDataToExportController();

file.write(toExport);

file.close();

}

}

}

void TuringMachine::machineStep()

{

ui.tape->blockSignals(true);

if (machine->isReady())

{

machine->oneStep(true);

showCurrentState();

showNextCommand();

}

else

{

QMessageBox::warning(this, "Машина", "Машина еще не готова к работе.\nПожалуйста, убедитесь, что лента и управляющее устройство заданы.");

}

ui.tape->blockSignals(false);

}

void TuringMachine::machineBeginWork()

{

ui.tape->blockSignals(true);

if (machine->isReady())

{

ui.stop->setEnabled(true);

QtConcurrent::run(this->machine,&Machine::startWork);

}

else

{

QMessageBox::warning(this, "Машина", "Машина еще не готова к работе.\nПожалуйста, убедитесь, что лента и управляющее устройство заданы.");

}

ui.tape->blockSignals(false);

}

void TuringMachine::machineStopWork()

{

ui.tape->blockSignals(true);

machine->stopWork();

ui.tape->blockSignals(false);

}

void TuringMachine::machineErrorReceived(QString &errorString)

{

QMessageBox::critical(this, "Ошибка машины", errorString);

}

void TuringMachine::machineFinished()

{

ui.nextCommand->setText("Машина завершила работу");

setControlButtonsEnabled(false);

ui.stop->setEnabled(false);

showCurrentState();

QMessageBox::information(this, "Состояние машины", "Машина достигла конечного состояния");

ui.tape->blockSignals(false);

}

void TuringMachine::machineStopped()

{

ui.stop->setEnabled(false);

showCurrentState();

showNextCommand();

ui.tape->blockSignals(false);

}

void TuringMachine::machineTapeSymbolChanged(unsigned int index, QString newSymbol)

{

ui.tape->blockSignals(true);

QTableWidgetItem\* item = ui.tape->item(0, index);

if (item != nullptr)

{

ui.tape->item(0, index)->setText(newSymbol);

}

else

{

qDebug() << "Метод:machineTapeSymbolChanged()\nНе удалось перезаписать текущий символ ленты.";

}

ui.tape->blockSignals(false);

}

void TuringMachine::machineTapePointerChanged(unsigned int oldTapePointer, unsigned int newTapePointer)

{

ui.tape->blockSignals(true);

QTableWidgetItem\* item = nullptr;

int tapeSize = ui.tape->columnCount();

/\* Удаляем старый указатель \*/

ui.tape->item(0, oldTapePointer)->setBackgroundColor(Qt::white);

if ((oldTapePointer == 0 && newTapePointer == 0) ||

(newTapePointer == tapeSize))

/\* В ленту добавился крайнией символ слева или справа \*/

{

ui.tape->insertColumn(newTapePointer);

item = new QTableWidgetItem(machine->getEmptySymbol());

item->setTextAlignment(Qt::AlignCenter);

ui.tape->setItem(0, newTapePointer, item);

item->setBackgroundColor(Qt::green);

}

else

{

item = ui.tape->item(0, newTapePointer);

item->setBackgroundColor(Qt::green);

item->setTextAlignment(Qt::AlignCenter);

}

/\* Ставим пустой символ машины, если в текущей ячейке ленты пусто \*/

if (item->text() == TAPE\_BLANK)

{

item->setText(machine->getEmptySymbol());

}

ui.tape->blockSignals(false);

}

void TuringMachine::expandTape(int currentRow, int currentColumn, int previousRow, int previousColumn)

{

ui.tape->blockSignals(true);

int columns = ui.tape->columnCount();

if (currentColumn == (columns - 1))

/\* Добавляем ячейку справа \*/

{

QTableWidgetItem\* item = new QTableWidgetItem(TAPE\_BLANK);

item->setTextAlignment(Qt::AlignCenter);

ui.tape->setColumnCount(columns + 1);

ui.tape->setItem(0, columns, item);

if (machine->isTapeLoaded())

{

machine->appendToTape(TAPE\_BLANK);

}

}

else if (currentColumn == 0)

/\* Добавляем ячейку слева \*/

{

QTableWidgetItem\* item = new QTableWidgetItem(TAPE\_BLANK);

item->setTextAlignment(Qt::AlignCenter);

ui.tape->insertColumn(0);

ui.tape->setColumnCount(columns + 1);

ui.tape->setItem(0, 0, item);

++tapeOffet;

if (machine->isTapeLoaded())

{

machine->prependToTape(TAPE\_BLANK);

if (machine->getTapePointer() == 0)

{

machine->incrementTapePointer();

}

}

}

ui.tape->blockSignals(false);

}

void TuringMachine::showLoadedTape()

{

ui.tape->blockSignals(true);

QStringList tape = machine->getTape ();

int size = tape.size();

ui.tape->setColumnCount(size);

if (machine->isTapeLoaded() && size != 0)

{

for (int i = 0; i < size; ++i)

{

QTableWidgetItem\* item = ui.tape->item(0, i);

if (item != nullptr)

{

item->setText(tape[i]);

}

else

{

item = new QTableWidgetItem(tape[i]);

ui.tape->setItem(0, i, item);

}

}

ui.tape->item(0, machine->getTapePointer())->setBackgroundColor(Qt::green);

if (machine->isControllerLoaded())

{

showNextCommand();

setControlButtonsEnabled(true);

}

}

ui.tape->blockSignals(false);

}

void TuringMachine::showLoadedController()

{

ui.controller->blockSignals(true);

QStringList alphabet = machine->getAlphabet();

QStringList states = machine->getStates(false);

int columns = alphabet.size();

int rows = states.size();

if (machine->isControllerLoaded() && columns != 0 && rows != 0)

{

ui.controller->setColumnCount(columns);

ui.controller->setRowCount(rows);

ui.controller->setHorizontalHeaderLabels(alphabet);

ui.controller->setVerticalHeaderLabels(states);

for (int i = 0; i < rows; ++i)

{

QString currentState = ui.controller->verticalHeaderItem(i)->text();

for (int j = 0; j < columns; ++j)

{

QString currentLetter = ui.controller->horizontalHeaderItem(j)->text();

QString key = currentLetter + currentState;

QString command = machine->getCommand(key);

if (!command.isEmpty())

{

QTableWidgetItem\* item = new QTableWidgetItem(command);

ui.controller->setItem(i, j,item);

}

}

showCurrentState();

ui.controller->blockSignals(true);

if (machine->isTapeLoaded())

{

showNextCommand();

setControlButtonsEnabled(true);

}

}

}

ui.controller->blockSignals(false);

}

void TuringMachine::showCurrentState()

{

ui.controller->blockSignals(true);

int rows = ui.controller->rowCount();

QString currentState = machine->getCurrentState();

for (int i = 0; i < rows; ++i)

{

paintRow(i, Qt::white);

QString state = ui.controller->verticalHeaderItem(i)->text ();

if (state == currentState)

{

paintRow(i, Qt::green);

}

}

ui.currentState->setText(currentState);

ui.controller->blockSignals(false);

}

void TuringMachine::showNextCommand()

{

if (!machine->isFinished ())

{

QString currentState = machine->getCurrentState();

QString currentSymbol = machine->getCurrentTapeSymbol();

QString key = currentSymbol + currentState;

QString cmd = machine->getCommand(key);

QString text;

if (cmd.isEmpty())

{

text = "Команда не задана";

}

else

{

text = currentSymbol + "-" + currentState + "->" + cmd;

}

ui.nextCommand->setText(text);

}

}

void TuringMachine::repaintTape()

{

ui.tape->blockSignals(true);

ui.tape->clear();

QStringList tapeView = machine->getTape();

int tapeSize = tapeView.size();

/\* Перерисовываем ленту \*/

ui.tape->setColumnCount(tapeSize);

for (int i = 0; i < tapeSize; ++i)

{

QTableWidgetItem\* item = new QTableWidgetItem(tapeView[i]);

item->setTextAlignment(Qt::AlignCenter);

ui.tape->setItem(0, i, item);

}

/\* Устанавливаем указатель \*/

unsigned int tapePointer = machine->getTapePointer();

ui.tape->item(0,tapePointer)->setBackgroundColor(Qt::green);

ui.tape->blockSignals(false);

}

void TuringMachine::setControlButtonsEnabled(bool isEnabled)

{

ui.step->setEnabled(isEnabled);

ui.*begin*->setEnabled(isEnabled);

}

void TuringMachine::rememberCurrentTapeSymbol(QTableWidgetItem\* item)

{

currentTapeSymbol = item->text();

qDebug() << "Cell symbol - " + currentTapeSymbol;

}

void TuringMachine::rememberCurrentCommand(QTableWidgetItem\* item)

{

currentCommand = item->text();

qDebug() << "Command - " + currentCommand;

}

void TuringMachine::modifyCurrentTapeSymbol(QTableWidgetItem\* item)

{

ui.tape->blockSignals(true);

int itemIndex = item->column();

QString itemText = item->text();

if (!machine->isTapeLoaded())

/\* Если ленты еще нет \*/

{

if (machine->isTapeSymbolValid(itemText))

{

/\* Создаем ленту \*/

QVector<QString> tmp;

tmp.*fill*(TAPE\_BLANK, ui.tape->columnCount());

QStringList list(QList<QString>::fromVector(tmp));

list[itemIndex] = itemText;

machine->setTape(list, itemIndex);

ui.tape->item(0, machine->getTapePointer())->setBackgroundColor(Qt::green);

machine->setEmptySymbol(itemText);

machine->setTapeLoaded();

ui.exportTape->setEnabled(true);

}

}

else

/\* Лента загружена \*/

{

if (machine->isTapeSymbolValid(itemText))

{

machine->setTapeSymbol(itemText, itemIndex);

}

else

{

item->setText(currentTapeSymbol);

QMessageBox::warning(this,

"Ошибка ввода",

"Невозможно установить \"" +

itemText +

"\" в ленту, так как этот символ не найден в алфавите машины.");

}

}

ui.tape->blockSignals(false);

}

void TuringMachine::modifyCurrentCommand(QTableWidgetItem\* item)

{

ui.controller->blockSignals(true);

QString newCommand = item->text();

QString state = ui.controller->verticalHeaderItem(item->row())->text ();

QString symbol = ui.controller->horizontalHeaderItem(item->column())->text();

if (newCommand.*count*('-') != 2)

{

QMessageBox::warning(this, "Ошибка", "Неверный синтаксис команды.\nКоманда должна задаваться:\n<символ алфавита>-<состояния>-<действие>");

item->setText(currentCommand);

ui.controller->blockSignals(false);

return;

}

else

{

QStringList list = newCommand.split('-');

if (!machine->isTapeSymbolValid(list[0]))

{

QMessageBox::warning(this, "Ошибка", "Символ \"" + list[0] + "\" отсутствует в алфавите.");

item->setText(currentCommand);

ui.controller->blockSignals(false);

return;

}

if (!machine->isStateValid(list[1]))

{

QMessageBox::warning(this, "Ошибка", "Состояние \"" + list[1] + "\" отсутствует в множестве состояний.");

item->setText(currentCommand);

ui.controller->blockSignals(false);

return;

}

if (list[2] != "R" && list[2] != "L" && list[2] != "N")

{

QMessageBox::warning(this, "Ошибка", "Неизвестное действие \"" + list[2] + "\".\nВозможные действия:\"R\",\"L\",\"N\".");

item->setText(currentCommand);

ui.controller->blockSignals(false);

return;

}

/\* Создаем команду \*/

Command cmd(list[0], list[1], list[2]);

QString key = symbol + state;

machine->addComand(key, cmd);

}

ui.controller->blockSignals(false);

}

QByteArray TuringMachine::prepareDataToExportTape() const

{

QJsonObject main;

QJsonArray tapeView = QJsonArray::fromStringList(machine->exportTape());

QJsonValue emptySymbol(machine->getEmptySymbol());

QJsonValue pointerIndex((int)machine->getTapePointer() - TAPE\_OFFSET);

main.insert("tape-view", tapeView);

main.insert("empty-symbol", emptySymbol);

main.insert("pointer-index", pointerIndex);

return QJsonDocument(main).toJson(QJsonDocument::Indented);

}

QByteArray TuringMachine::prepareDataToExportController() const

{

QJsonObject main;

QJsonArray alphabet = QJsonArray::fromStringList(machine->getAlphabet());

QJsonArray states = QJsonArray::fromStringList(machine->getStates(true));

QJsonValue beginState(machine->getBeginState());

QJsonValue endState(machine->getEndState());

QJsonObject commands;

QMap<QString, Command> controllerCommands = machine->getCommands();

QStringList keys = controllerCommands.keys();

/\* Экспортируем команды \*/

for (auto i : keys)

{

QJsonObject tmp;

Command cmd = controllerCommands.value(i);

QJsonValue newSymbol(cmd.newSymbol);

QJsonValue newState(cmd.newState);

QJsonValue action(cmd.action);

tmp.insert("new-symbol", newSymbol);

tmp.insert("new-state", newState);

tmp.insert("action", action);

commands.insert(i, tmp);

}

main.insert("alphabet", alphabet);

main.insert("states", states);

main.insert("begin-state", beginState);

main.insert("end-state", endState);

main.insert("commands", commands);

return QJsonDocument(main).toJson(QJsonDocument::Indented);

}

void TuringMachine::addRow(QTableWidget\* table, const QString& row)

{

int rows = table->rowCount();

table->insertRow(rows);

QTableWidgetItem\* header = new QTableWidgetItem(row);

table->setVerticalHeaderItem(rows, header);

table->setRowCount(rows + 1);

int columns = table->columnCount();

for (int i = 0; i < columns; ++i)

{

QTableWidgetItem\* item = table->item(rows, i);

if (item == nullptr)

{

item = new QTableWidgetItem();

table->setItem(rows, i, item);

}

}

}

void TuringMachine::addColumn(QTableWidget\* table, const QString& column)

{

int columns = table->columnCount();

table->insertColumn(columns);

QTableWidgetItem\* header = new QTableWidgetItem(column);

table->setHorizontalHeaderItem(columns, header);

table->setColumnCount(columns + 1);

int rows = table->rowCount();

for (int i = 0; i < rows; ++i)

{

QTableWidgetItem\* item = table->item(i, columns);

if (item == nullptr)

{

item = new QTableWidgetItem();

table->setItem(i, columns, item);

}

}

}

void TuringMachine::initializeTape()

{

ui.tape->setRowCount(1);

ui.tape->setColumnCount(DEFAULT\_TAPE\_COLUMNS);

for (int i = 0; i < DEFAULT\_TAPE\_COLUMNS; ++i)

{

QTableWidgetItem\* item = new QTableWidgetItem(TAPE\_BLANK);

item->setTextAlignment(Qt::AlignCenter);

ui.tape->setItem(0, i, item);

}

}

void TuringMachine::paintRow(int rowNumber, const QColor& color)

{

int columns = ui.controller->columnCount();

for (int j = 0; j < columns; ++j)

{

QTableWidgetItem\* tmp = ui.controller->item(rowNumber, j);

if (tmp != nullptr)

{

tmp->setBackgroundColor(color);

}

}

}

void TuringMachine::setConnections() const

{

connect(ui.resetButton, SIGNAL(clicked(bool)), SLOT(resetMachine()));

connect(ui.step, SIGNAL(clicked(bool)), SLOT(machineStep ()));

connect(ui.stop, SIGNAL(clicked(bool)), SLOT(machineStopWork()));

connect(ui.exportTape, SIGNAL(clicked(bool)), SLOT(exportTape()));

connect(ui.importTape, SIGNAL(clicked(bool)), SLOT(importTape()));

connect(ui.*begin*, SIGNAL(clicked(bool)), SLOT(machineBeginWork()));

connect(ui.exportController, SIGNAL(clicked(bool)), SLOT(exportController()));

connect(ui.importController, SIGNAL(clicked(bool)), SLOT(importController()));

connect(ui.tape, SIGNAL(currentCellChanged(int, int,int,int)), this, SLOT(expandTape(int, int,int,int)));

connect(ui.tape, SIGNAL(itemPressed(QTableWidgetItem\*)), this, SLOT(rememberCurrentTapeSymbol(QTableWidgetItem\*)));

connect(ui.tape, SIGNAL(itemChanged(QTableWidgetItem\*)), this, SLOT(modifyCurrentTapeSymbol(QTableWidgetItem\*)));

connect(machine, SIGNAL(machineError(QString&)), this, SLOT(machineErrorReceived(QString&)));

connect(machine, SIGNAL(machineFinished()), this, SLOT(machineFinished()));

connect(machine, SIGNAL(tapeSymbolChanged(unsigned int, QString)), this, SLOT(machineTapeSymbolChanged(unsigned int,QString)));

connect(machine, SIGNAL(tapePointerChanged(unsigned int, unsigned int)), this, SLOT(machineTapePointerChanged(unsigned int, unsigned int)));

connect(machine, SIGNAL(tapeChanged()), this, SLOT(repaintTape()));

connect(machine, SIGNAL(machineStopped()), this, SLOT(machineStopped()));

connect(machine, SIGNAL(machineReady(bool)), this, SLOT(setControlButtonsEnabled(bool)));

connect(ui.addNewState, SIGNAL(clicked(bool)), SLOT(addState()));

connect(ui.addNewSymbol, SIGNAL(clicked(bool)), SLOT(addSymbol()));

connect(ui.controller, SIGNAL(itemPressed(QTableWidgetItem\*)), this, SLOT(rememberCurrentCommand(QTableWidgetItem\*)));

connect(ui.controller, SIGNAL(itemChanged(QTableWidgetItem\*)), this, SLOT(modifyCurrentCommand(QTableWidgetItem\*)));

}

void TuringMachine::parseControllerFile(const QString& data) throw(QString&)

{

QJsonDocument doc = QJsonDocument::fromJson(data.toUtf8());

QJsonObject obj = doc.object();

/\* Считываем и устанавливаем алфавит \*/

if (obj.contains("alphabet"))

{

QJsonArray arr = obj["alphabet"].toArray();

QStringList alphabet;

for (auto i : arr)

{

QString tmp = i.toString();

alphabet << tmp;

}

machine->setAlphabet(alphabet);

}

else

{

QString errorString = "Не найден алфавит по тегу \"alphabet\".";

throw errorString;

}

/\* Считываем и устанавливаем состояния \*/

if (obj.contains("states"))

{

QJsonArray arr = obj["states"].toArray();

QStringList states;

for (auto i : arr)

{

QString tmp = i.toString();

states << tmp;

}

machine->setStates(states);

}

else

{

QString errorString = "Не найдены состояния по тегу \"states\".";

throw errorString;

}

/\* Считываем и устанавливаем начальные и конечные состояния \*/

QString beginState;

QString endState;

if (obj.contains("begin-state"))

{

beginState = obj["begin-state"].toString();

}

else

{

QString errorString = "Не найдено начальное состояние по тегу \"begin-state\".";

throw errorString;

}

if (obj.contains("end-state"))

{

endState = obj["end-state"].toString();

}

else

{

QString errorString = "Не найдено конечное состояние по тегу \"end-state\".";

throw errorString;

}

machine->setBeginEndStates(beginState, endState);

/\* Считываем команды \*/

if (obj.contains("commands"))

{

QJsonObject commands = obj["commands"].toObject();

parseCommands(commands);

}

else

{

QString errorString = "Не найдены команды по тегу \"commands\".";

throw errorString;

}

}

void TuringMachine::parseCommands(const QJsonObject commands) throw(QString&)

{

QStringList states = machine->getStates(false);

QStringList alphabet = machine->getAlphabet();

QStringList commandsList;

/\* Формируем список команд по алфавиту и множеству состояний \*/

for (int i = 0; i < states.size(); ++i)

{

for (int j = 0; j < alphabet.size(); ++j)

{

QString command = states[i];

command.prepend(alphabet[j]);

commandsList << command;

}

}

/\* Считываем и запоминаем команды \*/

for (auto i : commandsList)

{

if (commands.contains(i))

{

QJsonObject tmp = commands[i].toObject();

Command cmd(tmp["new-symbol"].toString(),

tmp["new-state"].toString(),

tmp["action"].toString());

try

{

machine->addComand(i, cmd);

}

catch (QString& errorString)

{

errorString.prepend("Команда \"" + i + "\":\n");

throw errorString;

}

}

else

{

QString errorString = "Недостаточно команд. Не найдена команда \"" + i + "\"";

throw errorString;

}

}

}

std::tuple<QStringList, unsigned int,QString> TuringMachine::parseTapeFile(const QString& data) const throw(QString&)

{

QString string;

unsigned int number;

QVector<QString> tmp;

tmp.*fill*(TAPE\_BLANK, ui.tape->columnCount());

QStringList list(QList<QString>::fromVector (tmp));

QJsonDocument doc = QJsonDocument::fromJson(data.toUtf8());

QJsonObject obj = doc.object();

if (obj.contains("tape-view"))

{

QJsonArray arr = obj["tape-view"].toArray();

int size = arr.size();

for (int i = 0; i < size;++i)

{

QString tmp = arr[i].toString();

int insertIndex = TAPE\_OFFSET + i;

if (list.size() < insertIndex)

{

list.insert((TAPE\_OFFSET + i),tmp);

}

else

{

list[insertIndex] = tmp;

}

}

}

else

{

QString errorString = "Не найдены данные о состоянии ленты по тегу \"tape-view\".";

throw errorString;

}

if (obj.contains("pointer-index"))

{

number = obj["pointer-index"].toInt() + TAPE\_OFFSET;

if (number < 0 || number >= list.size())

{

QString errorString = "Указатель ленты по тегу \"pointer-index\" задан не верно.";

throw errorString;

}

}

else

{

QString errorString = "Не найдены данные об указателе ленты по тегу \"pointer-index\".";

throw errorString;

}

if (obj.contains("empty-symbol"))

{

string = obj["empty-symbol"].toString();

}

else

{

QString errorString = "Не найдены данные о пустом символе ленты по тегу \"empty-symbol\".";

throw errorString;

}

return std::*make\_tuple*(list, number,string);

}