Russian Fiction Text Document Analysis

1st Polina Chainikova
University of Oulu,
Oulu, Finland
pchainik21@student.oulu.fi

2nd Daria Efimova
University of Oulu,
Oulu, Finland
defimova21@student.oulu.fi

***Abstract*—Today's topical task is natural language processing of text data. It is devoted to teaching machines to understand human languages and extract meaning from text, which finds its application in translators, chatbots, text classification, sentimental analysis, extraction of named entities, facts and relations. One of the tasks of text processing is the task of abstracting texts of fiction. This is the task we decided to implement in this project. As students from Russia, we decided to create a system that extracts this information from Russian fiction. Namely, extracting the names and essences of the characters. To process the texts we used the nltk library, as well as the Russian library Natasha. We also compared the performance of the Russian pymystem3 and pymorphy2 analyzers. Next, we extracted the entities of the characters, such as name, description of appearance, actions of the characters and their direct speech. A simple interface using PySimpleGUI was also built to make the system more user-friendly. We have completed all the tasks but this system is not perfect and needs to be refined, such as creating new rules for algorithms to improve their completeness and accuracy.**

***Index Terms*—NLTK, python, lemmatization, morphological analysis, syntactic analysis**

I. Introduction

When researching literary works of authors and studying their works of fiction, it is often necessary to solve the problem of abstracting, which can simplify the work with large texts and help in characterizing the characters of the works. Also, an automatic search for character descriptions may be necessary for the work of question-and-answer systems in the field of fiction. For example in our Petrozavodsk State University this system could help philologists in processing different kinds of texts. Philologists work with huge literary works all the time and, very often, they need to extract certain information from a text for writing essays, term papers, etc. That is why the goal of this project is to create a system that abstracts Russian literary works. The task of abstracting here refers to finding the characteristics of the characters, namely their names, actions, descriptions of their appearance, as well as highlighting direct speech. In addition to philological departments at universities, this can also be very useful in schools for teachers of literature, who also work with literary works and make tests on them for their students. To solve the problem of automatic processing of written texts in natural language using computers helps computer morphology, which analyzes and synthesizes words by software means.

This project is different in its individuality as we have not found anything similar to it. For word processing we will use well-known library nltk and also Russian library Natasha. Testing the system in practice was carried out on the basis of literary works of the writer Ivan Sergeyevich Shmelev. We also used 30 sentences taken from the website of the national corpus of the Russian language to compare the performance of two analyzers pymorphy2 and pymystem3. The comparison of the two analyzers was carried out in order to choose the best one for further work.

So, the aim of the work is to create algorithms for automatic identification of actors and their characteristics in literary texts, where the characteristics means the description of the appearance, actions and speech of a character. Objectives:

1) Extraction of named entities from the text, that is, the names of all the characters of a literary work.

2) Making a character dossier, which includes a description of appearance, actions and speech, by developing and implementing algorithms to find the relevant entities in the text.

II. Methodology

In order to solve the tasks set, it was necessary, first, to choose tools in order to subsequently create a basic program with their help. The processing of a fiction text goes through several stages, which include (Fig. 1):

1) tokenization
2) morphological analysis
3) lemmatization
4) syntactic analysis

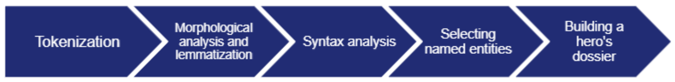

Tokenization
Morphological analysis and lemmatization
Syntax analysis
Selecting named entities
Building a hero's dossier

Fig. 1. The process of processing a literary work.

A. Text extraction

Initially, it was necessary to select the necessary tools for extracting text from the Web site. The library beautiful soup and request were used for this subtask. BeautifulSoup is a Python library for parsing HTML and XML documents. It is often used for scraping web pages. BeautifulSoup allows you to transform a complex HTML document into a complex tree of different Python objects. These can be tags, navigation or comments. Requests is a Python module you can use to send all kinds of HTTP requests. It's an easy-to-use library with many features, from passing parameters in URLs to sending custom headers and SSL validation.

B. Tokenization

Tokenization is the process of breaking text into text units - sentences and words. Tokenization can be done using various Python language tools. In our programs, we used the tokenize module from the NLTK library to break text into sentences, to perform morphological and syntactic analysis.

C. Morphological analysis and lemmatization

Lemmatization is the reduction of a word to its original (initial) form [9]. One auxiliary task was to write two programs to extract lemmas in words, using two libraries for morphological analysis pymorphy2 and pymystem3 to compare them.

- pymystem3. Pymystem3 is a shell for MyStem developed by Yandex.
- pymorphy2 [3]. pymorphy2 uses OpenCorpora dictionary for Russian language, which contains word grammars.

These analyzers are able to build hypotheses for unfamiliar words, lemmatize, put the word in the right form and return grammatical information about the word (number, gender, case, part of speech, etc.) and lemmas, as well as build hypotheses for unfamiliar words.

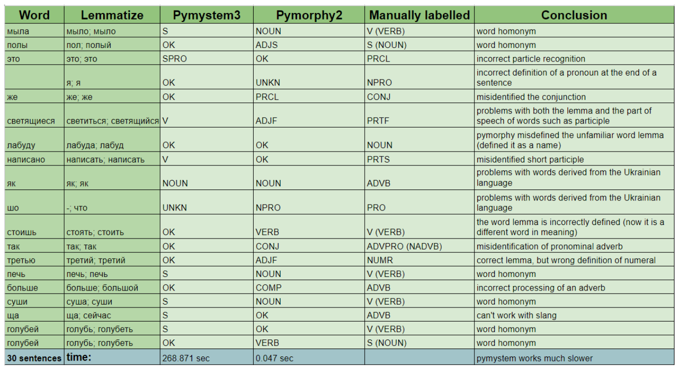
These libraries were chosen because they are quite common and their functions satisfy the objectives. It was necessary to compare the performance of the two aforementioned libraries and choose one of them, which would prove to be better in application as a tool for solving the main task. To analyze the performance of the two programs, 30 sentences of varying complexity were taken from the national corpus of the Russian language, so that we could compare the results obtained when using different libraries for morphological analysis. The complexity of the sentence depended on the number of words in the sentence, the number of commas and related conjunctions (a feature of Russian grammar), the presence of modern words (slang), and the presence of words such as homonyms. Homonyms are words that are the same in sound or spelling, but have different meanings.

Each sentence has been hand-tagged. The code of these programs can also be found on github:

- https://github.com/mu-stardash/Russian-Fiction-Text-Document-Analysis/blob/main/mystem.py
- https://github.com/mu-stardash/Russian-Fiction-Text-Document-Analysis/blob/main/pymorphy_2.py

As a result, we got that Pymystem3 library is more accurate and correct in determining the morphological analysis of words and their lemmata, but is inferior in speed of execution, and is not able to handle slang. However, both Pymystem3 and Pymorphy2 perform poorly on words such as homonyms.

It was also found that Pymystem3 cannot detect grammars of numbers (written in letters).



Word	Lemmatize	Pymystem3	Pymorphy2	Manually labelled	Conclusion
мыла	мыло; мыло	S	NOUN	V (VERB)	word homonym
полы	пол; полый	OK	ADJS	S (NOUN)	word homonym
это	это; это	SPRO	OK	PRCL	incorrect particle recognition
	я; я	OK	UNKN	NPRO	incorrect definition of a pronoun at the end of a sentence
же	же; же	OK	PRCL	CONJ	misidentified the conjunction
светящиеся	светиться; светящийся	V	ADJF	PRTF	problems with both the lemma and the part of speech of words such as participle
лабуду	лабуда; лабуд	OK	OK	NOUN	pymorphy misdefined the unfamiliar word lemma (defined it as a name)
написано	написать; написать	V	OK	PRTS	misidentified short participle
як	як; як	NOUN	NOUN	ADVB	problems with words derived from the Ukrainian language
шо	-; что	UNKN	NPRO	PRO	problems with words derived from the Ukrainian language
стоишь	стоять; стоить	OK	VERB	V (VERB)	the word lemma is incorrectly defined (now it is a different word in meaning)
так	так; так	OK	CONJ	ADVPRO (NADVB)	misidentification of pronominal adverb
третью	третий; третий	OK	ADJF	NUMR	correct lemma, but wrong definition of numeral
печь	печь; печь	S	NOUN	V (VERB)	word homonym
больше	больше; большой	OK	COMP	ADVB	incorrect processing of an adverb
суши	суша; суши	S	NOUN	V (VERB)	word homonym
ща	ща; сейчас	S	OK	ADVB	can't work with slang
голубей	голубь; голубеть	S	OK	V (VERB)	word homonym
голубей	голубь; голубеть	OK	VERB	S (NOUN)	word homonym
30 sentences	**time:**	268.871 sec	0.047 sec		pymystem works much slower

Fig. 2. Results of comparing mistakes of two analyzers

D. Syntax analysis

Syntactic analysis is used to build a tree of word dependencies in a sentence. For each dependency, an appropriate linguistic characteristic is defined, which describes the type of relationship between words.

Two modules, Navec and Slovnet, from the Natasha library for the Python language, which is designed to extract structured information from texts, can be used for parsing [11, 12, 13].

- Module Navec is a collection of vector word representations for the Russian language [14].
- Module Slovnet is a set of models for extracting named entities, parsing morphology and syntax [15].

E. Selecting named entities

There is a standard task of extracting named entities from text, which consists of extracting names, dates, addresses, amounts of money, geographical objects, etc. from text. It is often necessary to work with texts and their analysis.

It was the task of extracting named entities from the text, and specifically the names of all characters in a literary work, that needed to be solved.

To solve this problem for the Russian language there is the library Natasha, which also performs tokenization, segmentation of sentences, word embedding, tagging morphology, lemmatization, normalization of phrases, syntactic analysis, tagging NER (named entities), extraction of facts [11].

Also previously selected was a library for morphological analysis of the text - pymorphy2, in which character names can be found using the standard tools of the library (grammatical characteristics of words - grammas [16]) and intuitive analytical rules for extracting names from the text.

It was with the help of these two libraries that the first of the tasks set in the work was solved - the extraction from the text of named entities, that is, the names of all the characters of a literary work.

To compare the correctness of the two programs, two texts by Ivan Sergeyevich Shmelev were taken: ”Fever” [18] and ”How I went to see Tolstoy” [17].

1) Proprietary rules and pymorphy2: The program receives text from a website, which needs to be processed: remove punctuation marks, unnecessary words (so-called stop words), as well as perform tokenization. To remove punctuation marks the library string is used, and to remove stop words the library nltk, which contains its own list of stop words and for the Russian language [19, 20].

In order to find character names in the processed text, rules were written: the name begins with a capital letter, the word is a noun, animate. These rules can be determined by parsing each word morphologically and highlighting grammars. Each parse has a tag (p.tag). A tag is a set of grammemes that characterize a given word. For example, the tag ’VERB, perf, intr plur, past, indc’ means that the word is a verb (VERB) of perf, intr, plur, past, indc [3]. The pymorphy2 uses the OpenCorpora dictionary for the Russian language. This dictionary contains grammars for us of certain rules, including ’Name’, ’Surn’, ’Patr’ for first name, patronymic and surname respectively. For a noun it is ’NOUN’, for an animate name it is ’anim’ [3].

Also pymorphy2 returns all possible variants, but in practice we usually need only one variant, the correct one. Every expression has a score parameter. Score is a score P(tag—word), an estimate of the probability that the given parse is correct [1]. For named entities $score \geq 0.30$, so we also include this condition in the name lookup rules.

The program checks for combinations of words: consecutive first and last names, first and last name, etc. This is needed to select such a name only as for one character, not several different ones. The resulting names are written to the set() (to exclude repeated names) and output to the output.txt file. You can see code of this program with this link: https://github.com/mu-stardash/Russian-Fiction-Text-Document-Analysis/blob/main/names_rules.py

2) With Natasha library: Let's look at a program that searches for names in text with the help of the Natasha library (https://github.com/mustardash/Russian-Fiction-Text-Document-Analysis/blob/main/names_Natasha.py). The algorithm of the program includes the following steps:

- Importing the necessary modules from the Natasha library and applying their methods for further use.
- Obtaining text from a website. Doc combines annotators, initially only the text field is defined for it.
- It does sentence segmentation, tokenization, morphology tagging, parsing, NER (named entities) tagging. Spans has a normalize method that uses syntax annotation with syntax_parser to change phrases, phrase normalization happens.
- To highlight names (first name and last name of one person, etc.), a check is used to see if the word (or phrase) is a named entity PER (i.e. person - character). [11, 13, 12]. The pymystem3 library is additionally used to bring words into their initial form. As shown in the previous result (see Table X0), pymystem3 is slower but more correct in lemmatizing than pymorphy2

The resulting names are written to a list (checking for names already added) and output to the output.txt file.

3) Considering results: Consider the results of the two programs in the form of comparative tables (Table V, it is located at the end of the report) first on the story "How I went to Tolstoy", and then on the story "Fever".

1) Having independently counted the number of names in the story "How I went to Tolstoy" (a total of 25 Russian + 3 foreign names) and analyzing Table V (a), we can say: The program, written with the help of self-made rules and using the capabilities of the library pymorphy2, found 12 names in the story, only 7 are correct. The quality of the work can be evaluated using criteria such as completeness and accuracy. Since the pymorphy2 dictionary is not designed for foreign first and last names, we will not include them in the number of names in the story.
   Completeness = Correctly found / number of names in the text $= 7/25 = 0.28$;
   Accuracy = True found / number of found = $8/12 \approx 0.58$.
   The program using the library Natasha with ready-made rules, found in a story 30 names, correct of which 23. Let's calculate the quality of the work:
   completeness $= 23/25 = 0.92$;
   accuracy $= 23/30 \approx 0.77$
   It is worth noting that the program was also able to find foreign names: Bockl, Smiles, Spencer.
2) Now let's compare the results of our work on the second story "Fever. Having independently counted the number of names in the second story (a total of 16) and analyzing Table V (b), we can say
   a) The first program found 9 names in the story, 4 of which are correct. Let's calculate the quality of the result:
      Completeness $= 4/16 = 0.25$;
      accuracy $= 4/9 \approx 0.44$
   b) The second program found 32 names in the story, 15 of which were correct. Quality of this result:
      Completeness $= 15/16 \approx 0.94$;
      accuracy $= 15/32 \approx 0.47$

Thus, the following results were obtained for the two programs, calculated as the arithmetic mean of the previously obtained results of completeness and accuracy: For the program with self-made rules:

- Completeness $\approx (0.25 + 0.28)/2 \approx 0.27$;
- Accuracy $\approx (0.44 + 0.58)/2 \approx 0.51$;

For the program using ready-made rules:

- Completeness $\approx (0.93 + 0.92)/2 \approx 0.93$;
- Accuracy $\approx (0.47 + 0.77)/2 \approx 0.62$;

Analyzing the received statistics of quality of work of two programs it is possible to tell that the program which uses ready rules of library Natasha, practically in full finds all names, but makes some inaccuracies, as gives out superfluous words which are simply nouns or verbs. It is possible to correct this situation by making up and writing new rules in the yargy parser yourself. Also an advantage of Natasha is to find some foreign names and surnames.

The program, which works only on the self-made rules, looking for words in a very small amount. Having tested the program on other texts it was found out that it's possible to correct this error with the best text processing from unwanted characters, but the use of regular expressions (re library) gave worse results than using the string library.

As a result - Natasha works without additional training at a sufficient level to use it in practice. At the same time there is an opportunity to refine the rules.

The result of checking the work on several texts is shown in the Table I.

Quality Metrics	1 method	2 method
Completeness	57%	95%
Accuracy	72%	72%

TABLE I
RESULTS OF CHECKING THE WORKING OF PROGRAMS FOR FINDING NAMES ON SIX TEXTS (1 METHOD - WITH OUR OWN RULES, 2 METHOD - USING NATASHA LIBRARY).

The FreqDist class from the nltk.probability library was used to count and display the frequency of names in the text. As you can see from these results, the program using the Natasha library (Fig. 4) searches for names in a larger volume than method with proprietary rules and pymorphy2 (Fig. 3). It is also possible to find the main characters by the most frequent names.

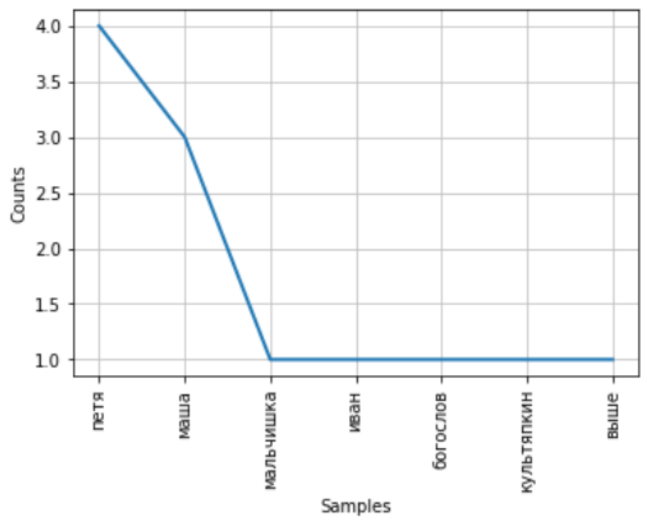


Fig. 3. (a) Three planes in dynamic texture (b) LBP histogram from each plane (c) Combined feature histogram.

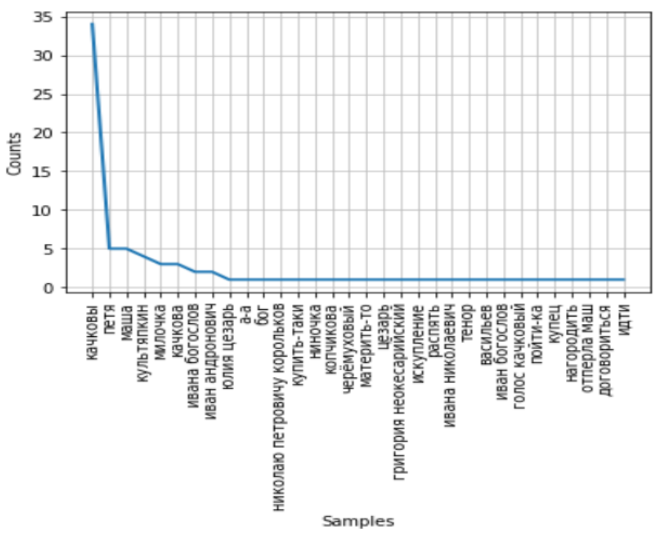


Fig. 4. (a) Three planes in dynamic texture (b) LBP histogram from each plane (c) Combined feature histogram.

F. Creating a dossier of characters

When analyzing literary works, it is necessary to know not only the names of the acting characters, but also their descriptions, appearances, characteristics, actions and speech. Therefore, let us move on to the next task set - compiling the dossier of characters, by developing and implementing algorithms for searching in the text for relevant entities.

G. Searching for direct speech of characters

We got acquainted with the rules of punctuation when composing direct speech in the text, namely: direct speech goes before, between and around the words of the author; in dialogues, direct speech is often accompanied without the author's words, we decided to use regular expressions (Fig. 5).

A regular expression is a string that defines a pattern for searching for substrings in a text. Many different lines can correspond to one pattern [9].

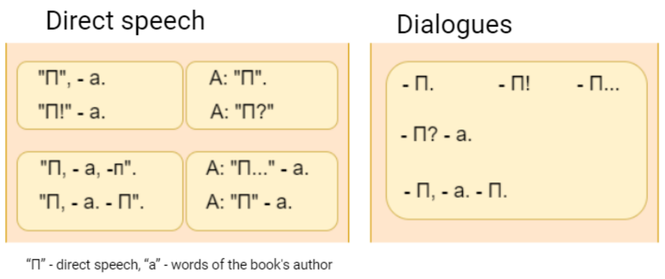


Fig. 5. Direct speech and dialogues

An additional action was to check the use of capital letters in the sentences found, which made it possible to exclude the author's words. Thus, we got a list of all direct speech, which provides an excellent basis for further completing the assignment.

A dictionary is used to match each character with all of his or her statements. It contains sentences that contain the names of the characters. Each found sentence with its position in the text is examined and, if this sentence contains a direct speech from the list found by a regular expression, this sentence is added to the list. Since literary works, for the most part, use dialogs, we check whether there are more sentences that contain direct speech. Dialogues are searched using the position of the sentence in the text. Every second sentence after the current one is checked, and if it is not in the list with direct speech, the search is terminated because the dialogue has ended. Thus, all names and sentences are checked, filling the dictionary with the found sentences corresponding to the names of the characters. You can see code of this program with this link:https://github.com/mustardash/Russian-Fiction-Text-Document-Analysis/blob/main/direct_speech.py

H. Searching for the appearance of characters

To test the algorithm in practice, the story of Ivan Sergeyevich Shmelev: ”Miron and Dasha” [21] was taken, since it contains many descriptions of the characters' appearance, which is convenient for illustrating the results of solving the problem.

An algorithm was developed that uses the syntactic analyzer [15] to find the word combinations describing the appearance, characterizing the sentence members and the relations between them [22].

Syntactic analysis is performed for all sentences containing character names. During the parsing process, a linguistic characteristic is determined for each word in the sentence. Each of them has a complex interpretation, so which characteristics should be used to find the word combinations describing the appearance, was determined not on the basis of their definitions, but in the process of preliminary analysis during the parsing of several sentences from literary works by the parser, tracking which characteristics are most often found in the desired word combinations.

The following linguistic characteristics turned out to be suitable: nmod(the nominal modifier is a noun or phrase that functions as a non-main argument or adjunct), acl: relcl(the relation is used for relative clauses), conj (a conjunct is a relation between two elements connected by a conjunctive conjunction such as and, or, comma or other punctuation mark), acl(denotes finite and non-final sentences that modify a nominal), appos(a nominal immediately following the first noun that serves to define or modify that noun) [22]. An example of syntax analysis is shown in the Figure 6.

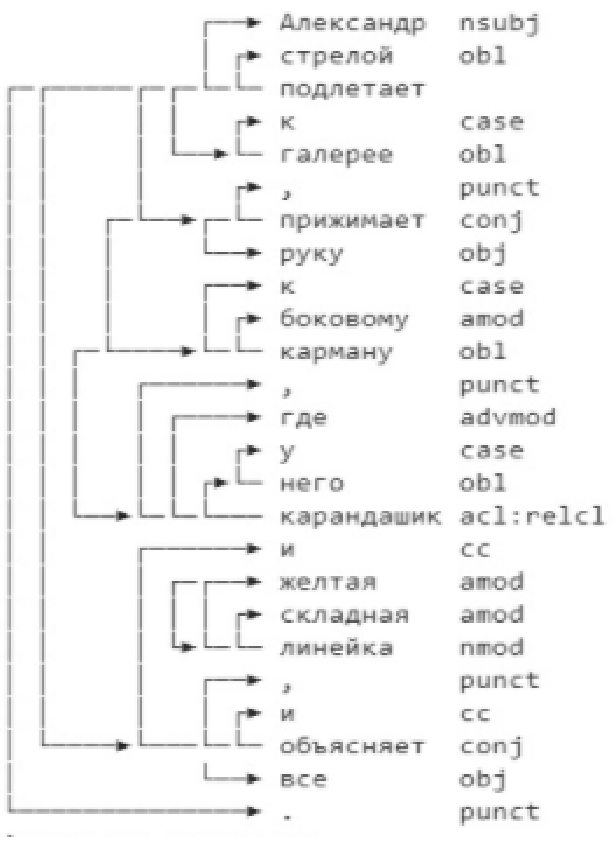


Fig. 6. An example of syntax analysis.

The algorithm is as follows.

Going through all the words in the sentences, their linguistic characteristics are checked. If the right type (acl, acl:relcl, appos, conj, nmod) is found, the word is selected and then the words dependent on it, which are not punctuation marks (punct), unions (cc) and some classes of pronouns (det - possessive, indicative, determinative pronouns) are checked [17]. These dependencies are also revealed as a result of the analysis.

The selected words are combined into word combinations and added to the set. Word combinations, which have only one word, which is not an adjective, or word combinations, which have a verb, are discarded, because with this structure they cannot be a description of appearance.

The algorithm results in a general list of phrases and a set of appearance descriptions corresponding to a particular character. The implementation of this algorithm is presented in the program You can see code of this program with this link: https://github.com/mu-stardash/Russian-Fiction-Text-Document-Analysis/blob/main/appearance.py

I. Searching for character actions

To find the actions of a character in the works, we decided to use syntactic analysis of a sentence. The answer is the subject and the predicate of the sentence. The algorithm was tested on the works of I. S. Shmelev ”Nanny from Moscow” and ”Decay”.

Consider the work of the algorithm. For the extraction of the named entities we use the library Natasha, which is better than pymorphy2 on the basis of the rules. After running parsing, a tree of word dependencies in a sentence is built. For each dependency, an appropriate linguistic characteristic is determined which describes the type of relationship between the words.

Based on the parsing, the actions of the characters are found. From the main word (as a rule, it is a predicate) the connections of words in the sentence are constructed and if the connection points to a word that is a name from the collection, then these are the subjects. To make sure that the predicate is not a single word, the links to other words are checked; if the linguistic characteristic of the word is (conj) or (xcomp), it is the predicate (An example of syntax analysis is shown in the Figure 6). In this way, each word in the name collection will be checked. The screen will display the name of the character and the actions he committed. You can see code of this program with this link: https://github.com/mu-stardash/Russian-Fiction-Text-Document-Analysis/blob/main/temp.py

III. Development of a graphical interface

To create a system for automatically compiling dossiers of literary characters, which uses our already created algorithms, we decided to make a GUI using the new library PySimpleGUI. This GUI framework takes all these popular and well-established GUI frameworks (Tkinter, Qt, Remi, WxPython) and wraps them in a single library that is easy to learn and then build applications. This was very useful for our team, since we had no experience in building a GUI. When writing code using PySimpleGUI, its dimensionality would be much smaller and easier to understand.

PySimpleGUI has built-in widgets that we need (for example: buttons, input field and list), and is very easy to customize the appearance of the interface, where all the elements can be arranged by lines (building a so-called layout). This framework has a huge selection of pre-defined themes for the window interface, so we also decided to give the user the option to change the interface theme to something he likes best. This is very popular these days, and we also set a default dark theme that is comfortable for most users. A detailed description of the interface elements is presented in Figure 9 in the last section.

You can see code of this program with this link: https://github.com/mu-stardash/Russian-Fiction-Text-Document-Analysis/blob/main/interface_NLP.py

IV. Results and Discussion

Now let’s consider the completeness and accuracy of our algorithms.

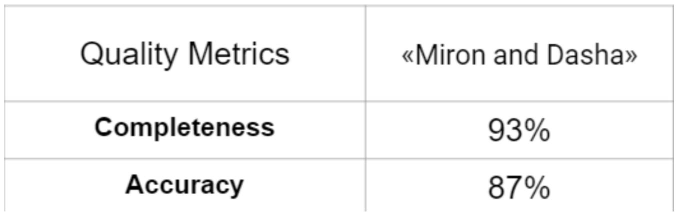
1) After testing the work of the written algorithm for searching direct speech, the following results were obtained: the algorithm finds only 35% of all statements, half of which are wrong (Table II). The main mistakes are not always correct finding of speech in dialogues; incorrect finding of direct speech if there are two or more names in front of it in the sentence.

Quality Metrics	«Fever»	«Misha»
Completeness	33%	37%
Accuracy	49%	55%

TABLE II
Results of the direct speech search algorithm.

2) The completeness of finding character descriptions using a syntax analysis is 93%, and the

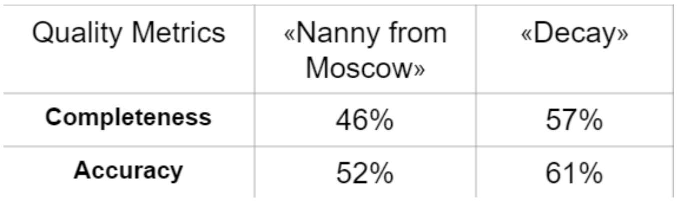
accuracy is 87% on one text that contains a large number of such elements (Table III). It can be concluded that it is very effective to find character descriptions using the syntax analysis, but the main errors are associated with inaccurate parsing of Natasha.



Quality Metrics	«Miron and Dasha»
Completeness	93%
Accuracy	87%

TABLE III
RESULTS OF THE APPEARANCE SEARCH ALGORITHM.

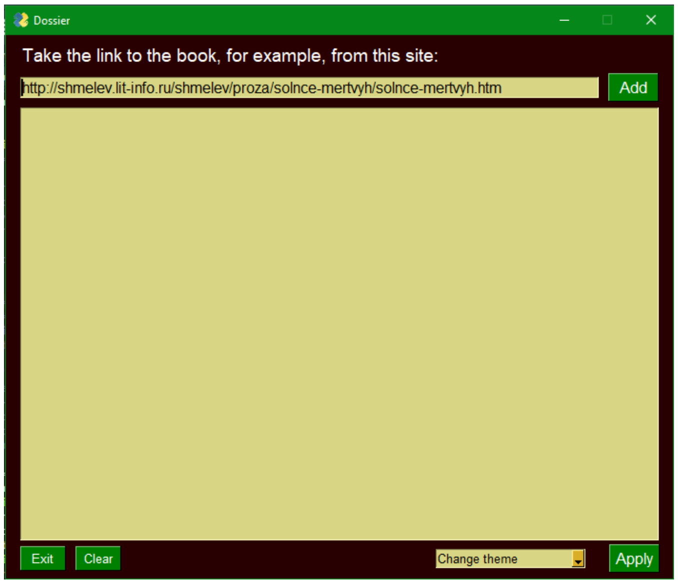
3) The results of the operation of the algorithm for searching for the actions of the characters using a syntax analysis were as follows (Table IV): 51% of the actions were found, half of which turned out to be erroneous. The main bugs are problems with Natasha's imprecise parsing, which leads to a decrease in the search quality.



Quality Metrics	«Nanny from Moscow»	«Decay»
Completeness	46%	57%
Accuracy	52%	61%

TABLE IV
THE RESULTS OF THE ALGORITHM SEARCHING THE ACTIONS OF THE CHARACTERS.

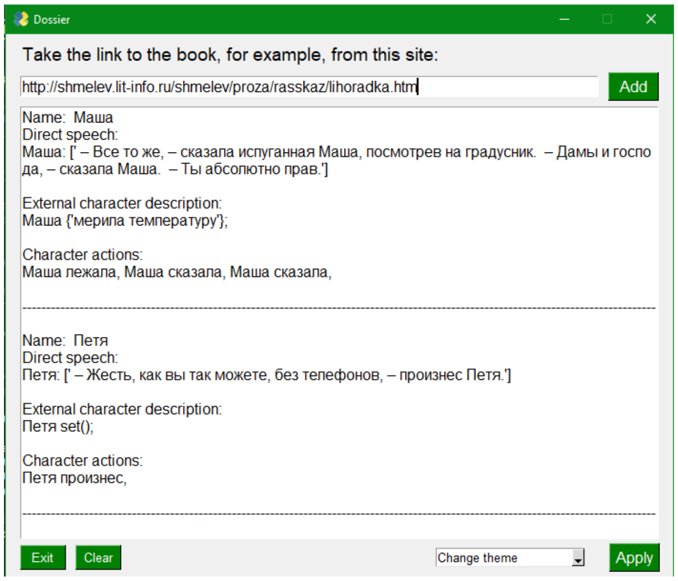
4) By combining all the algorithms together and using a graphical interface, we got a system for automatically compiling a dossier of characters in Russian literary works. The interface contains an input field for a link to the desired work from the website (the inserted link is also checked by the program for correctness), adding which the user after a while will receive in the output field all the characters of this work with their characteristics. You can see the view of this system in Figure 7 and Figure 8.



Take the link to the book, for example, from this site:

http://shmelev.lit-info.ru/shmelev/proza/solnce-mertvyh/solnce-mertvyh.htm

Fig. 7. Automatic dossier system home screen.



Take the link to the book, for example, from this site:

http://shmelev.lit-info.ru/shmelev/proza/rasskaz/lihoradka.htm

```
Name:  Маша
Direct speech:
Маша: [' – Все то же, – сказала испуганная Маша, посмотрев на градусник.  – Дамы и госпо
да, – сказала Маша.  – Ты абсолютно прав.']

External character description:
Маша {'мерила температуру'};

Character actions:
Маша лежала, Маша сказала, Маша сказала,

-----------------------------------------------------------------------------------------

Name:  Петя
Direct speech:
Петя: [' – Жесть, как вы так можете, без телефонов, – произнес Петя.']

External character description:
Петя set();

Character actions:
Петя произнес,

-----------------------------------------------------------------------------------------
```

Fig. 8. An example of the system where the user changed the theme.

Summing up, we can say that algorithms have been developed to automatically identify the characters, describe their appearance, actions and direct speech. The algorithms are not yet perfect and need to be improved. Also, the processing time of text by three algorithms takes quite a long time, since in our algorithms we used the Pymystem3 library for lemmatization, which. it has been found to work much slower, but it gives better results (for example, names are searched better). Also, the work of the program is slowed down by the use of the Natasha library, which takes a long time to load the modules it needs.

V. Overall Discussion

Let's sum up the results of the completed project. So, to date, the following results have been obtained:

- We compared two Russian analyzers (Pymystem3 and Pymorphy2) for 30 sentences of varying complexity. The sentences were taken from the Russian National Corpus, and their complexity depends on the peculiarities of Russian grammar and on the use of words such as homonyms. After comparing the analyzers, conclusions were drawn on their further use in the main algorithms for putting words in their initial form, as well as for isolating morphemes (by conducting morphological analysis).
- Tested the use of various Python language libraries for text processing. Namely: removing stop words from the Russian text (the most common words that do not carry a meaningful load for finding names in the text), tokenization of the text into sentences and into words, and removing punctuation marks. All of these functions were accomplished by using the nltk library. The string, re libraries were also tried to remove punctuation marks from the text. To extract the text of a work of fiction from a website, libraries BeautifulSoup, request were disassembled and used.
- Explored in more detail the capabilities of the Natasha Library to compose a syntactic analysis of sentences.
- Algorithms for automatic search for characters, direct speech, including in dialogues, appearance and actions have been developed. The algorithms are not perfect and need some work.
  - To find all characters, two programs were developed: one using the rules themselves invented and using the previously considered pymorphy2 analyzer, the second using the Natasha library, the rules for extracting names in which are already written, and are available through NamesExtractor(). Natasha showed the best results, so it was decided to use this method for further tasks.
  - To search for direct speech, an algorithm was created, where the main attribute is the search for all direct speeches using regular expressions, as well as a compiled function of correlating direct speech to the desired character, using the position of the sentence in the text (the position in the text helped to implement the search for dialogues).
  - To search for appearance and actions, the studied parsing from the Natasha library was used, where the necessary linguistic characteristics were highlighted (they describe the relationship between two words in a sentence, as shown in Figure 6).

As mentioned at the beginning of the report, we did not find similar systems on the Internet, like our system, so we have nothing to compare the work done. It can also be concluded that our system is unique and relevant in the modern world. This will be very useful for people writing articles and studying the literary works of Russian writers. These can be researchers, teachers of literature at school, teachers and students at the Faculty of Philology. Having improved the quality of the algorithms for the automatic compilation of character dossiers and also accelerated the time of their work, we would like to offer this system for the Faculty of Philology at our home university in Russia. Having learned our idea, the teachers were interested in the implementation of this system.

VI. Conclusion

In conclusion, we can conclude that we have completed all the points of the project. As a result, a system was developed for the automatic compilation of dossiers of characters in Russian works. The system includes four algorithms:

- Search for named entities. After comparing the two algorithms for finding names in the text, the best one was chosen. This algorithm finds almost complete names, but sometimes it recognizes extra words as names. Intuitively, a Russian user can easily recognize these names as erroneous. This algorithm was used in the further development of the main tasks.
- Search for direct speech.
  The algorithm showed not very good results, but they can be corrected by improving the direct speech search algorithm. Now the algorithm finds only dialogues (therefore, monologues

were not taken into account in calculating the results) and only finds the first sentence from the direct speech of the hero (due to the applied regular expressions).

- Search for appearance and characteristics.
  The algorithm uses a parser from the Natasha library and has shown excellent results on text containing a large amount of descriptions of appearance. The result is not 100% due to not always accurate parsing thanks to Natasha.
- Search for action.
  As in the previous step, the algorithm uses a parser. The results turned out worse, since very often Natasha defines verbs incorrectly and incorrectly builds connections in a sentence. There is also a problem with the Name - Verb connection, since sometimes the verb is incorrectly defined to the main word.

A. Future work

In the future, it is planned to add sentences to the search in which a specific character is mentioned not through a name, but through a personal pronoun or a noun personifying it, improve algorithms to increase their completeness and accuracy, and also expand the possibilities for compiling a more extensive dossier of characters.

Also, for the created algorithms in the future, it is planned to add a search for monologues (for direct speech), expand the dictionary of nouns and improve the rules for finding phrases - descriptions of appearance and actions using a parser.

B. Delineation of group member responsibilities

The work was done jointly and collaboratively, but we distributed some work to complete the tasks more quickly:

- Daria Efimova: manual analysis of 30 sentences on the pymystem3 analyzer and analyzed the program's work on 30 sentences; developed a program to search for names using ready-made rules from Natasha's library; developed an interface.
- Polina Chaynikova: manual analysis of 30 sentences on the pymorphy2 analyzer and analyzed the work of the program on 30 sentences; developed a program to search for names using the rules we made up; studied the work of parsing from Natasha's library.

The rest of the tasks were done together: writing the three main algorithms for the system (direct speech search, action search, and appearance search).

C. The complexity of the tasks performed

For us, the project was not easy, because in a very short time we needed to get acquainted with a large number of new libraries (pymorphy2, pymystem3, Natasha) and implement rather complex algorithms. Also, none of our team worked with a graphical interface, so we also had to familiarize ourselves with one of the simple PySimpleGUI libraries.

The work was also painstaking, since we had to evaluate the results of the programs, independently calculating the necessary data in the texts (names, actions, appearance, etc.), and also independently mark the 30 sentences used to compare the analyzers.

D. Acquired skills

In the process of implementing the project, we have acquired a lot of new skills:

- Having familiarized ourselves with the capabilities of the NLTK library, we learned how to preprocess input data. In our case, the input data were links to websites, the text from which was extracted thanks to the learned libraries BeautifulSoup and Request.
- We learned how to make morphological analysis of words in a sentence and lemmatize them using popular Russian analyzers.
- We learned how to make a syntactic analysis of sentences, which can be used when searching for actions, characteristics, descriptions of characters in texts.
- We learned how to extract named entities from text.
- We learned how to create a simple graphical interface in Python.

VII. References employed

[1] User's Manual. Morphological analyzer pymorphy2 [Electronic resource]. Mikhail Korobov Revision, copyright 2013-2020. URL: https://pymorphy2.readthedocs.io/en/stable/user/guide.html

[2] Python shell for morphological and social analyzer of Russian language Yandex MyStem 3.1 [Electronic resource]. Denis

Sukhonin, Alexander Panchenko. URL: https://pypi.org/project/pymystem3/

[3] Decoding of grammars MyStem [Electronic resource]. 2020 Yandex. URL: https://yandex.ru/dev/mystem/doc/grammemes-values.html/

[4] National Corpus of the Russian language [Electronic resource]. © 2003-2020. URL: 1. https://ruscorpora.ru/new/search-main.html

[5] Shmelev Ivan Sergeyevich [Electronic resource]. © 2000-2021 NIV. URL: Ivan Shmelev (lit-info.ru)

[6] Project Natasha. A set of quality open source tools for natural Russian language processing (NLP) [Electronic resource]. URL: https://habr.com/ru/post/516098/

[7] How to clean up text for machine learning with Python [Electronic resource]. URL: https://www.machinelearningmastery.ru/clean-text-machine-learning-python/

[8] Text preprocessing in NLP [Electronic resource]. URL: https://python-school.ru/nlp-text-preprocessing/

[9] I. S. Nikolaev, O. V. Mitrenina, T. M. Lando. Applied and computer linguistics. URSS Publisher, 2016. 316 c.

[10] Regular expressions in Python from simple to complex [Electronic resource]. URL: Regular expressions in Python from easy to complex. Details, examples, pictures, exercises / Habr (habr.com)

[11] Natasha : docs.ipynb [Electronic resource] - URL : https://nbviewer.jupyter.org/github/natasha/natasha/blob/master/docs.ipynb (Accessed 6.10.2021).

[12] Github : natasha [Electronic resource] - URL : https://github.com/natasha (Accessed 6.10.2021).

[13] NLPub : Natasha [Electronic resource] - URL : https://nlpub.ru/Natasha(accessed 6.10.2021).

[14] Github : natasha/ navec [Electronic resource] - URL : https://github.com/natasha/navec (Accessed 8.10.2021).

[15] Github : natasha/slovnet [Electronic resource] - URL : https://github.com/natasha/slovnet (Accessed 9.10.2021)

[16] OpenCorpora : Grammes [Electronic resource] - URL: http://opencorpora.org/dict.php?act=gram (Accessed 9.10.2020).

[17] Ivan Shmelev [Electronic resource]. Date of accession: 10.10.2021. URL: http://tolstoy-lit.ru/tolstoy/vospominaniya/shmelev-kak-ya-hodil-k-tolstomu.htm

[18] Lev Tolstoy [Electronic resource]. Date of accession: 10.10.2021. URL: http://tolstoy-lit.ru/tolstoy/vospominaniya/shmelev-kak-ya-hodil-k-tolstomu.htm

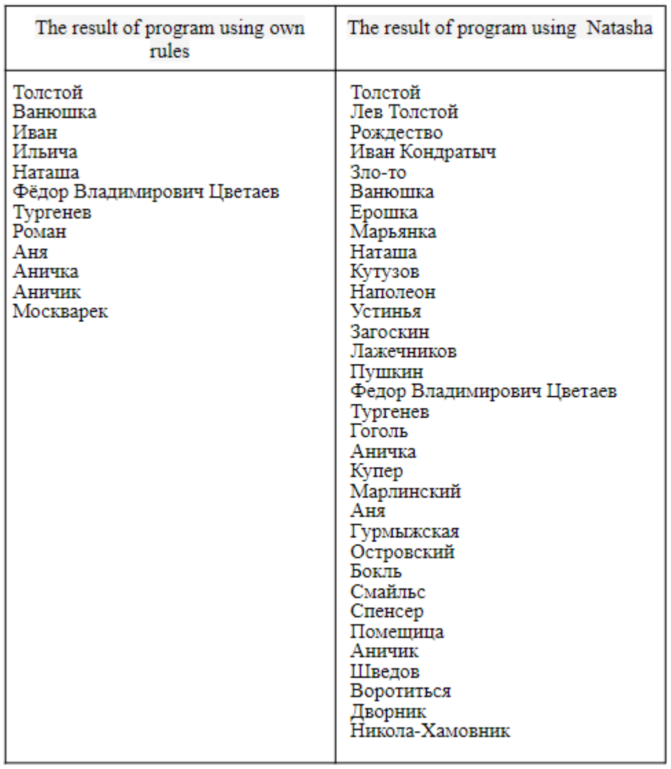
[19] How to clean text for machine learning with Python [Electronic resource]. Date of access: 01.12.2020. URL: https://www.machinelearningmastery.ru/clean-text-machine-learning-python/

[20] Text preprocessing in NLP [Electronic resource]. Date of reference: 01.12.2020. URL: https://python-school.ru/nlp-text-preprocessing/

[21] Library of Russian and Soviet Classics : Ivan Sergeyevich Shmelev : Miron and Dasha [Electronic resource] - URL : https://ruslit.traumlibrary.net/book/shmelev-ss05-08/shmelev-ss05- 08.html (accessed 13.10.2021).

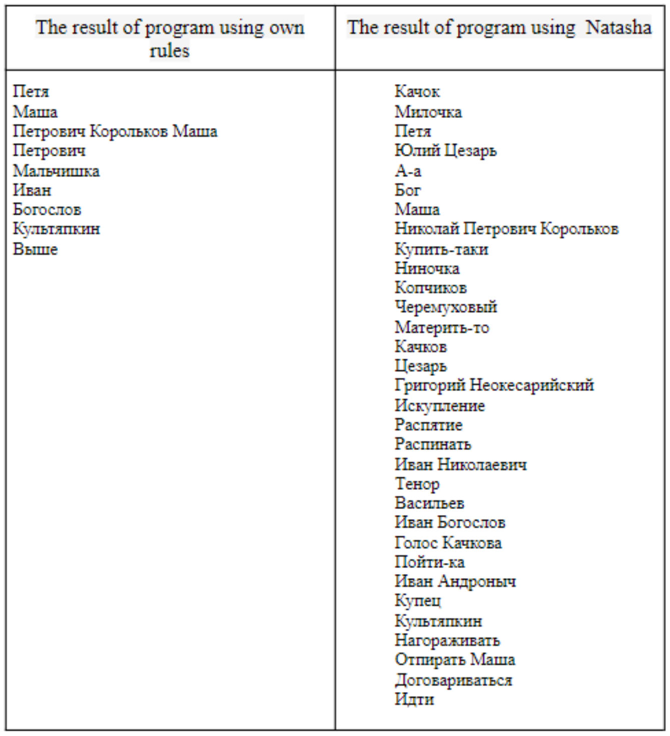
[22] Universal Dependencies : Universal Dependency Relations [Electronic resource] - URL : https://universaldependencies.org/u/dep/index.html (accessed 13.10.2021)

VIII. Appendix



The result of program using own rules	The result of program using Natasha
Толстой	Толстой
Ванюшка	Лев Толстой
Иван	Рождество
Ильича	Иван Кондратыч
Наташа	Зло-то
Фёдор Владимирович Цветаев	Ванюшка
Тургенев	Ерошка
Роман	Марьянка
Аня	Наташа
Аничка	Кутузов
Аничик	Наполеон
Москварек	Устинья
	Загоскин
	Лажечников
	Пушкин
	Федор Владимирович Цветаев
	Тургенев
	Гоголь
	Аничка
	Купер
	Марлинский
	Аня
	Гурмыжская
	Островский
	Бокль
	Смайльс
	Спенсер
	Помещица
	Аничик
	Шведов
	Воротиться
	Дворник
	Никола-Хамовник

a) on the work ”How I went to Tolstoy”

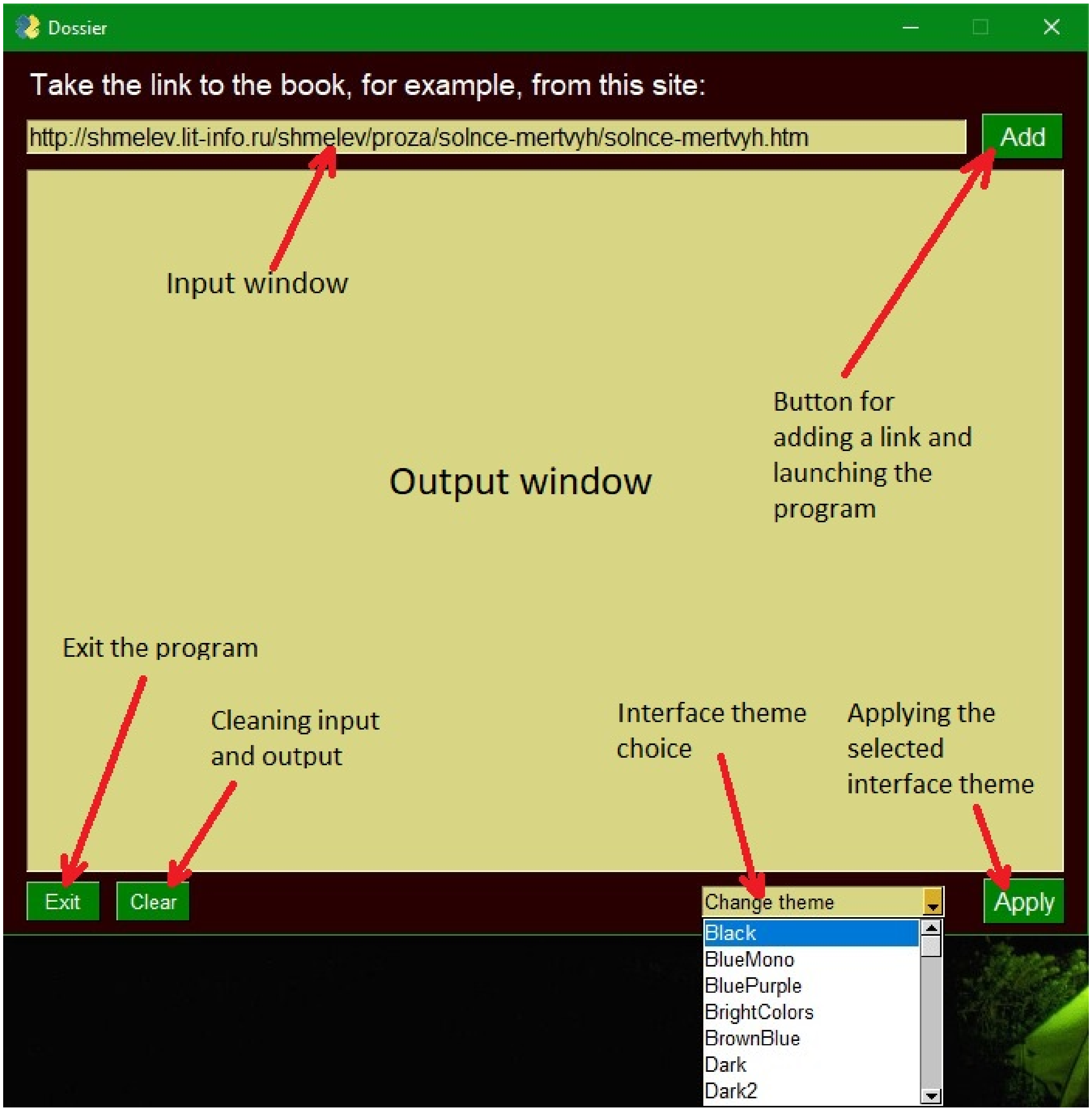


The result of program using own rules	The result of program using Natasha
Петя	Качок
Маша	Милочка
Петрович Корольков Маша	Петя
Петрович	Юлий Цезарь
Мальчишка	А-а
Иван	Бог
Богослов	Маша
Культяпкин	Николай Петрович Корольков
Выше	Купить-таки
	Ниночка
	Копчиков
	Черемуховый
	Материть-то
	Качков
	Цезарь
	Григорий Неокесарийский
	Искупление
	Распятие
	Распинать
	Иван Николаевич
	Тенор
	Васильев
	Иван Богослов
	Голос Качкова
	Пойти-ка
	Иван Андроныч
	Купец
	Культяпкин
	Нагораживать
	Отпирать Маша
	Договариваться
	Идти

b) on the work ”Fever”

TABLE V

Comparison of the results of the programs on the search for names.


Take the link to the book, for example, from this site:

http://shmelev.lit-info.ru/shmelev/proza/solnce-mertvyh/solnce-mertvyh.htm

Fig. 9. Detailed description of interface elements.