

**SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND
INFORMATION TECHNOLOGY**

**PARANOYA
TEAM PROJECT**

2020

Lóránt Boráros, Filip Budáč, Martin Cehelský, Silvia Holecová

**SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND
INFORMATION TECHNOLOGY**

**PARANOYA
TEAM PROJECT**

Study Programme:	Applied Informatics
Course:	TP – Team project
Lecturer:	Mgr. Ing. Matúš Jókay, PhD.
Teaching assistant:	Mgr. Ing. Matúš Jókay, PhD.

Bratislava 2020 Lóránt Boráros, Filip Budáč, Martin Cehelský, Silvia Holecová

Contents

Introduction	1
1 Analysis	2
1.1 Existing programs	2
1.2 The current status of the application	4
1.3 Methods used in evaluate tests	4
1.4 Used test sets	4
1.5 Solution methods	9
2 UML Diagrams	10
2.1 Use Case Diagram	10
2.2 Sequence Diagrams	11
2.3 Activity Diagrams	19
3 Acceptance tests	30
4 GUI - Graphical user interface	35
4.1 Figma	35
4.2 Structure	35
4.3 Main menu	36
4.4 New Test	37
4.5 Add sequence	38
4.6 Load test	39
4.7 Generate	40
4.8 Run Test	41
4.9 Results	42
5 Implementation	44
5.1 NIST test set	44
5.2 Application	44
5.3 Tests	45
Conclusion	47
Bibliography	48

Appendix	I
A Description of application	II

List of Figures and Tables

Figure 1	Dieharder test suite	3
Figure 2	Use case diagram	10
Figure 3	Sequence diagram - Run	11
Figure 4	Sequence diagram - Continue	11
Figure 5	Sequence diagram - Pause	12
Figure 6	Sequence diagram - Cancel	12
Figure 7	Sequence diagram - Set input/output folder	13
Figure 8	Sequence diagram - Select output option	13
Figure 9	Sequence diagram - Select summary option	14
Figure 10	Sequence diagram - Set parameters for selected tests	14
Figure 11	Sequence diagram - Generator-set destination file	15
Figure 12	Sequence diagram - Generator-set output format	15
Figure 13	Sequence diagram - Generator-set bytecount	16
Figure 14	Sequence diagram - Help	16
Figure 15	Sequence diagram - Show test results	17
Figure 16	Sequence diagram - Save configuration	17
Figure 17	Sequence diagram - Load sequence	18
Figure 18	Activity diagram - Run	19
Figure 19	Activity diagram - Continue	19
Figure 20	Activity diagram - Pause	20
Figure 21	Activity diagram - Stop	21
Figure 22	Activity diagram - Set input/output folder	22
Figure 23	Activity diagram - Select output option	23
Figure 24	Activity diagram - Select summary option	24
Figure 25	Activity diagram - Set parameters for selected tests	25
Figure 26	Activity diagram - Show test results	26
Figure 27	Activity diagram - Save configuration	26
Figure 28	Activity diagram - Load sequence	27
Figure 29	Activity diagram - Generate sequence/Set destination file	27
Figure 30	Activity diagram - Generate sequence/Set output format	28
Figure 31	Activity diagram - Generate sequence/Set bytecount	28
Figure 32	Activity diagram - Help	28

Figure 33	Main menu	36
Figure 34	New Test	37
Figure 35	Add sequence	38
Figure 36	Set test	39
Figure 37	Generate	40
Figure 38	Run Test	41
Figure 39	Results	42
Figure 40	Detailed results	43
Figure A.1	ParanoYa - Main window	II
Figure A.2	ParanoYa - Tests are set	III
Figure A.3	ParanoYa - Additional modification of parameters	III
Figure A.4	ParanoYa - Set the output options	IV
Figure A.5	ParanoYa - pval.html file	V

Introduction

The quality and security of many cryptographic systems is judged based on the generation of unpredictable random or pseudorandom numbers. Generating truly random data is very complicated. In practice various algorithms are used to generate pseudo-random sequences and their aim is to achieve indistinguishability from truly random data. In principle, it is not possible to prove that the data are really random, but exist some sets of tests that can detect non-random behavior. Such test sets are implemented in application ParanoYa.

This application was created on Faculty of Electrical Engineering and Information Technology by students. ParanoYa was developed for statistical testing of pseudo random sequences. In the application are implemented various test sets like NIST, FIBS and Diehard. With this application is also possible to evaluate achieved results for testing sequences that are processed in Microsoft Excel document.

The aims of our team project are create detailed analysis for given problem, create new design for user interface and create prototype of new application with new user interface and chosen test set.

The motivation for processing the topic was improvement of the existing application ParanoYa. With creating of new user interface that would be more intuitive and user-friendly and also with reducing the time required to perform tests, we could increase the user comfort of this application. With actualization of the application for the current development tools, we also could ease the work of developers who would like to extend or change this application in the future.

1 Analysis

This section describes complete analysis of the given problem. There are described existing programs which are similar as paranoYa, current status of the application, methods used in evaluate test and also test sets used in application with detailed description of each of them. There are also included functional requirements for the application.

1.1 Existing programs

Nowadays existing multiple applications which are similar as paranoYa. Analysing of these applications we found that none of them doesn't implement all of these test sets FIPS, NIST and Diehard, as paranoYa. ParanoYa was also designed for future extensions. But these existing applications don't have this possibility. Below are described applications which were analysed:

1. **Ent** is a console application, which is useful to test sequences of pseudo-random number generators for encryption, compression and statistical sampling. The application can run a variety of tests, including:

- **Entropy**
- **Chi-square Test**
- **Arithmetic mean**
- **Monte Carlo Value of Pi**
- **Serial Correlational Coefficient**

Ent offers a number of options regarding input and output formats of the data:

- **-b**
Data input is treated as bit-stream instead of byte-stream.
- **-c** A table of characters is printed to the standard output. The table includes the decimal value of each character paired with the corresponding printable character in ISO 8859-1 Latin-1.
- **-f**
Characters of upper-case letters are changed to lower-case.
- **-t**
Output format is changed to *terse mode* which means that the output values are separated by a comma(CSV format).


```

#####
#               dieharder version 3.29.4beta Copyright 2003 Robert G. Brown               #
#####
Installed dieharder tests:

```

Test Number	Test Name	Test Reliability
-d 0	Diehard Birthdays Test	Good
-d 1	Diehard OPERM5 Test	Suspect
-d 2	Diehard 32x32 Binary Rank Test	Good
-d 3	Diehard 6x8 Binary Rank Test	Good
-d 4	Diehard Bitstream Test	Good
-d 5	Diehard OPSO	Good
-d 6	Diehard OQSO Test	Good
-d 7	Diehard DNA Test	Good
-d 8	Diehard Count the 1s (stream) Test	Good
-d 9	Diehard Count the 1s Test (byte)	Good
-d 10	Diehard Parking Lot Test	Good
-d 11	Diehard Minimum Distance (2d Circle) Test	Good
-d 12	Diehard 3d Sphere (Minimum Distance) Test	Good
-d 13	Diehard Squeeze Test	Good
-d 14	Diehard Sums Test	Do Not Use
-d 15	Diehard Runs Test	Good
-d 16	Diehard Craps Test	Good
-d 17	Marsaglia and Tsang GCD Test	Good
-d 100	STS Monobit Test	Good
-d 101	STS Runs Test	Good
-d 102	STS Serial Test (Generalized)	Good
-d 200	RGB Bit Distribution Test	Good
-d 201	RGB Generalized Minimum Distance Test	Good
-d 202	RGB Permutations Test	Good
-d 203	RGB Lagged Sum Test	Good
-d 204	RGB Kolmogorov-Smirnov Test Test	Good

Figure 1: Dieharder test suite

- **-u**

Prints manual.

2. **Dieharder** is an improved version of *Diehard battery of tests* with a cleaned up source code implemented in C programming language. Thanks to the improvements the tests run considerably faster. Furthermore the new structre enables the incorporation of new sets of tests in the future. It also enables to test generators directly by accepting a infinite stream of numbers.**Dieharder** is an open-source project available for free download in its website.
3. **Practically random** is a library implemented in the C++ programming language. It is suited for testing random number generators-*RNG*
4. **TestU01** is library implemented in ANSI C programming language. The library contains functions for empirical testing of random number generators. The application provides classical statistical test as well as some original ones. Basic plotting of the generated numbers is also available.

1.2 The current status of the application

Application named ParanoYa is used for statistical testing pseudo random sequences. In this application, are implemented various test sets like NIST, FIPS, Diehard. With this application it is also possible to evaluate each testing sequence. Using the application it is also possible to evaluate individual tested sequences based on two methodologies. Output of the application is processed in Microsoft Excel document. With this document we can evaluate the achieved results. Application was created with framework Qt and used test suites are implemented in C. Full description of the application ParanoYa can be found in attachment A.

1.3 Methods used in evaluate tests

The application using two methods for evaluate tests. In output file are saved results of testing sequences. Each of these values is called p-value and represents the probability that another randomly generated sequence is worse in a given test. The tests are evaluated at a confidence level of $\alpha = 0.01$. If some p-value is smaller than α than testing sequence did not pass the test. A generator has not passed a test if less than 96% of the sequences have passed that test. This limit is set for the number of 100 sequences using a confidence interval based on the probability of error of the second kind.

1.4 Used test sets

Testing is a process when is executed one or more test cases based on specified conditions. During this process is compared current and expected behavior. In the application are implemented different sets of tests, for example NIST, FIPS a Diehard.

1. NIST

NIST is statistical package of tests which is used to testing randomness of arbitrarily long binary sequences. This sequences are generated using a random or pseudo-random sequence generator. This package is consists of the following 15 test:

(a) The Frequency (Monobit) Test

The aim of this test is determine whether the ratio of zeros and units in a given sequence corresponds to the expected ratio for a random sequence. The number of units and zeros in the sequence should be approximately equal, which is also examined by the test.[1]

(b) Frequency Test within a Block

This test considers the ratio of zeros and units in M-bit blocks. The aim of the test is to determine whether frequency of M-bit block is approximately $M/2$.

(c) **The Runs Test**

In this test is important the total number of zeros and units in runs in whole sequence, where the run represents a continuous sequence of equal bits. A run of length k means that it consists of k identical bits and is bounded before and after with a bit having the opposite value. The purpose of this test is to determine whether the number of runs of units and zeros of varying length is as expected for random sequences. This test is mainly used to assess whether the variation between such substrings is too slow or too fast.

(d) **Tests for the Longest-Run-of-Ones in a Block**

This test focuses on the longest run units within M-bit blocks. Its purpose is to determine whether the length of the longest run units in the test sequence is consistent with the length of the longest run units expected in random sequences. Irregularity in the expected length of the longest run of units means that there exists an irregularity in the expected length of the longest run of zeroes. Long zero runs are not evaluated separately because of concerns about statistical independence between tests.

(e) **The Binary Matrix Rank Test**

The test is aimed at the discontinuous order of the submatrices in the whole sequence. The purpose of this test is checking the linear dependence in the fixed length of the substrings of the original sequence.

(f) **The Discrete Fourier Transform (Spectral) Test**

The focus of this test are the heights of the peaks in the Fourier transform. The purpose of this test is detect periodic functions (for example, repeating patterns that are close together) in a test sequence that would indicate a deviation from the assumption of randomness.

(g) **The Non-overlapping Template Matching Test**

The random number sequence is divided into independent substrings of length M and the number of occurrences of template B , which represents the m -bit run units in each of the substrings. IfP-value chi-square of statistic is less than the significance level, the test concludes that the test sequence appears random. Otherwise, the test concludes that the retest appears to be random. The throughput is defined by the ratio of the sequences that passed the test.

(h) **The Overlapping Template Matching Test**

This test detects the number of occurrences in pre-specified target strings. The test uses an m -bit window to search for a specific m -bit pattern. If the pattern is not found, the window moves about one bit position. If the searched pattern is found, the window moves only one bit before resuming the search.

(i) **Maurer's "Universal Statistical" Test**

The purpose of this test is determine whether the sequence can be significantly compressed without losing information or not. A too compressed sequence is considered as non-random.

(j) **The Linear Complexity Test**

The purpose of this test is determine whether the sequence is sufficiently complex to be considered as random.

(k) **The Serial Test**

The purpose of this test is determine whether the number of occurrences of overlapping m -bit patterns is approximately the same as would be expected in a random sequence.

(l) **The Approximate Entropy Test**

The test focuses on the frequency of any possible overlap of m -bit patterns in the whole sequence. The purpose of this test is to compare the frequency of the overlapping blocks of two consecutive or adjacent lengths (m and $m + 1$) with the expected result for a random sequence.

(m) **The Cumulative Sums (Cusums) Test**

This test focuses on the maximum deviation (from zero) of the random walk (defined by the cumulative sum of the adjusted $(-1, +1)$ digits in sequence). The aim of the test is determine whether the cumulative sum of the partial sequences occurring in the test sequence is too large or too small relative to the expected behavior of this cumulative sum for the random sequences. This cumulative sum can be considered as a random walk. The random walk deviation should be near zero for a random sequence. For certain types of random sequences, the deviations of this random walk will be greater than zero.

(n) **The Random Excursions Test**

The test is focused on the number of cycles that have exactly K occurrences in the cumulative sum of random steps. The cumulative sum can be found

if the subtotals $(0, 1)$ of the sequence are adjusted to $(-1, +1)$. The random deviation of the random steps consists of a sequence of n steps of unit length. The purpose of the test is determine whether the number of occurrences of the state with random-step exceeds what is expected of the random sequence.

(o) **The Random Excursions Variant Test**

This test examines how many times is occurred specific status in a cumulative sum of random steps. The goal is detect deviations from the expected number of occurrences of different states in random steps.

These tests deal with the different types of randomness that might arise in sequence. Some of the tests could be broken down into different subtests. The order in which the tests are run is arbitrary, but it is recommended that the Frequency test be run first, because if this test fails, the probability of failing further tests is very high.

2. **FIPS** nist sp-822,fips 140-2 Test Federal Information Processing is the US government security standard used to validate cryptographic modules. FIPS provides different types of security based on a defined level of security. There are four such levels:

- (a) **Level 1** - the lowest security level that does not require specific physical security mechanisms but requires the use of at least one approved security algorithm or function
- (b) **Level 2** - this level requires role-based access control, as well as physical security
- (c) **Level 3** - in this level is provided identity-based authentication and physical security. It should include an attack detection mechanism. If the system were hacked, the system should be able to delete critical security parameters
- (d) **Level 4** - it is the highest level of security. In addition to the above-mentioned requirements for the system, the requirements of physical security are tighten, it is especially advantageous for working in a physically unprotected environment

FIPS validation involves intensive testing to identify specific deficiencies and weaknesses. For the system to meet FIPS validation, it needs to include cryptographic algorithms and hash functions. The three best known examples are AES, Triple DES, and HMAC SHA-1.

3. Diehard

Diehard tests are statistical tests used to evaluate the quality of the random number generator. The Diehard test battery consists of various, independent statistical tests. The results of these assays are referred to as p-values. Diehard's tests include:

(a) **The Birthday spacings test**

This test first selects m birthdays in a year with n days, then it is a list of birthday gaps between birthdays. Finally, the Poisson asymptotically distribution of j value is assessed. The j value is the number of values that are in the list of spaces. If it is multiple times in the list, then j is asymptotically Poisson divided with diameter $m^3/(4n)$. n must be large enough to compare the results with the Poisson distribution.

(b) **Overlapping permutations**

This test follows a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states for $5!$ possible arrangement of five numbers.

(c) **Ranks of matrices**

This test is performed by selecting a number of bits from a number of random numbers to form a matrix above $[0,1]$ and then is determining the matrix order. The number of rows should follow a certain distribution.

(d) **Monkey test**

Also called as bitstream test. This test has its name from an endless "monkey theorem". It is best achieved by processing sequences of a certain number of bits as "words" and counting the overlapping words in the steam. The number of "words" that do not appear should follow the known distribution.

(e) **Count the 1s**

The test is done through counting the 1 bits in each of either successive or chosen bytes and converting the counts to "letters", and counting the occurrences of five-letter "words".

(f) **Parking lot test**

Randomly place unit circles in a 100×100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "packed" circles should follow a certain normal distribution.

(g) **Minimum distance test**

Randomly place 8000 points in a 10,000 x 10,000 square and then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean.

(h) **Random spheres test**

Randomly choose 4000 points in a cube of edge 1000. Center a sphere on each point, whose radius is the minimum distance to another point. The smallest sphere's volume should be exponentially distributed with certain mean.

(i) **The squeeze test**

Multiply 231 by float random integers on $[0,1)$ until you reach 1. Repeat this 100,000 times. The number of floats needed to reach 1 should follow a certain distribution.

(j) **Overlapping sums test**

Generate a long sequence of random floats on $[0,1)$. Add sequence of 100 consecutive floats. The sums should be normally distributed with characteristic mean and sigma.

(k) **Runs test**

Generate a long sequence of random floats on $[0,1)$. Count ascending and descending runs. The counts should follow a certain distribution.

(l) **The craps test**

Play 200,000 games of craps, counting the wins and the number of throws per game. Each count should follow a certain distribution.

1.5 Solution methods

The core of the application, test sets, are represented as C libraries. User interface is created using Qt framework and application output is currently presented within an *.xls* file, readable in spreadsheet editors. The solution design is divided into several steps:

1. **Project actualisation compatible with current design environments** We decided to use Python for developing user interface. Using Cython library we created an python-c interface. A shared object file ".so" was created from the C libraries. The shared object enables us to dynamically connect library with different programs.

2 UML Diagrams

This sections belongs UML diagrams which describe entire functionality of developing application.

2.1 Use Case Diagram

Each Use case describes a sequence of actions that provide something of measurable value to an Actor and is drawn as a horizontal ellipse. In our diagram are described actions, which are offered to the Actor operating with an app. Actor in our case is capable of several actions, to name a few, *File options*, *Selects tests*, *Tasks*, *Tests evaluation* etc. Each action has its respective Action and Sequence diagram, describing action more detaily in pages below.

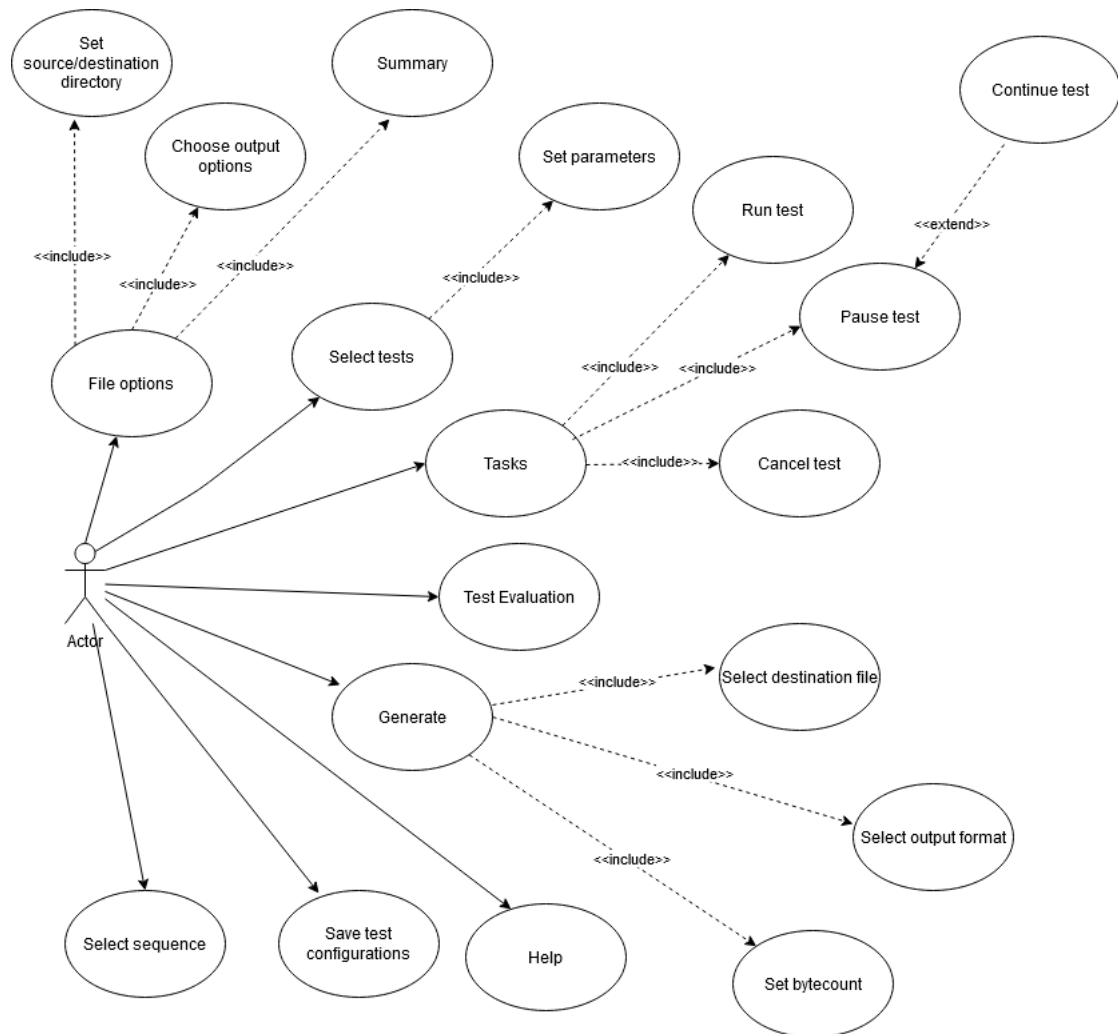


Figure 2: Use case diagram

2.2 Sequence Diagrams

In this subsections are belonged sequence diagrams. These diagrams displayed processes which are performed sequentially. With these diagrams we can examine behavior of the system.

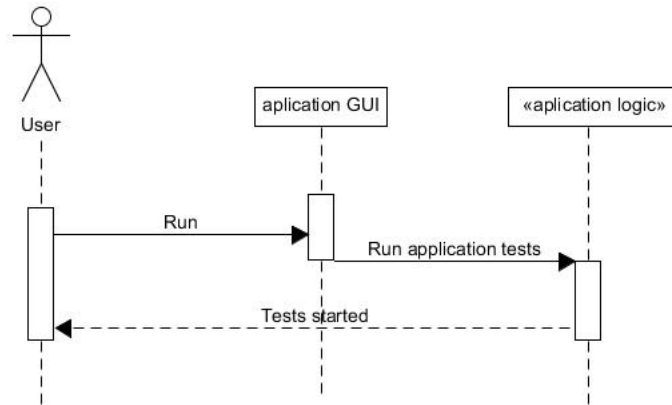


Figure 3: Sequence diagram - Run

The user interacts with the app's graphical interface. In the *Tasks* tab in the application navigation bar, selects *Run*. Pseudo-random sequence testing starts. Start-up is preceded by loading a sequence, selecting a methodology.

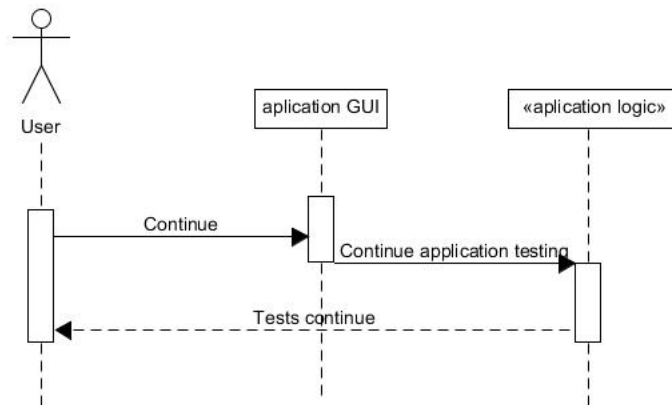


Figure 4: Sequence diagram - Continue

The user interacts with the app's graphical interface. In the *Tasks* tab in the application navigation bar, selects *Continue*. Pseudo-random sequence testing continues. Actions needed before that *Run* and *Pause* the testing.

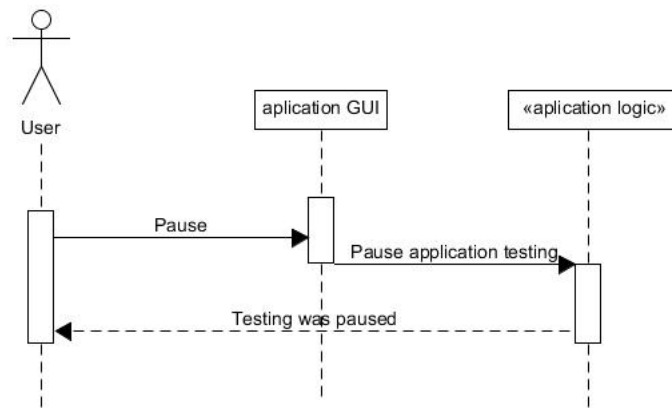


Figure 5: Sequence diagram - Pause

The user interacts with the app's graphical interface. In the *Tasks* tab in the application navigation bar, selects *Pause*. The pseudo-random sequence testing is discontinued. The interrupt is preceded by *Run* testing.

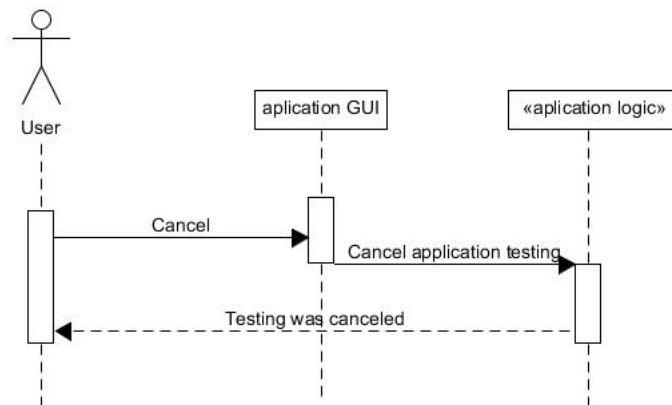


Figure 6: Sequence diagram - Cancel

The user interacts with the app's graphical interface. In the *Tasks* tab in the application navigation bar, selects *Cancel*. The pseudo-random sequence testing stops. Stopping is preceded by *Run* the testing.

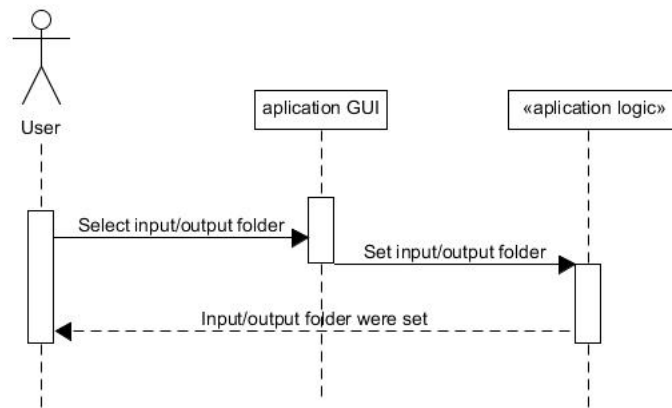


Figure 7: Sequence diagram - Set input/output folder

The user interacts with the app's graphical interface. In the *File* tab in the application navigation bar, selects *Batch process....* Next window is shown. This window belongs Source directory, Destination directory, Output options and Summary. After clicking on the button *Set...*, the user selects Source directory in the option Source directory and then he clicks button *OK*. This directory is also set as Destination directory by default. If user would like to change destination directory, he sets it in a similar way like Source directory.

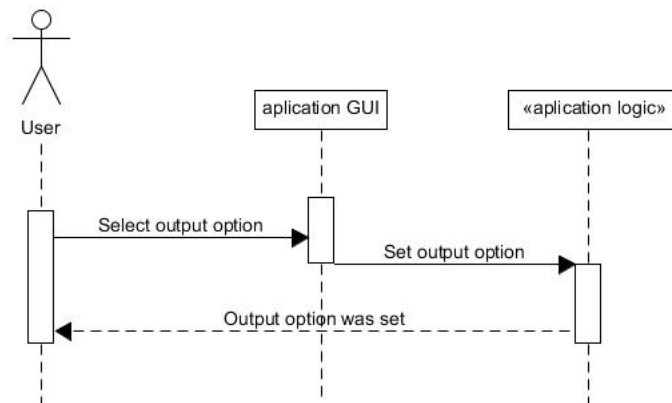


Figure 8: Sequence diagram - Select output option

The user interacts with the app's graphical interface. In the *File* tab in the application navigation bar, selects *Batch process....* Next window is shown. This window belongs Source directory, Destination directory, Output options and Summary. In the part Output options, the user selects one of the following options: XML, HTML, XML + HTML.

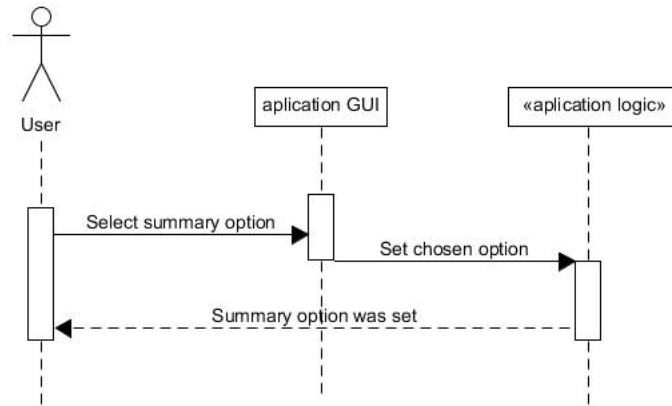


Figure 9: Sequence diagram - Select summary option

The user interacts with the app's graphical interface. In the *File* tab in the application navigation bar, selects *Batch process....* Next window is shown. This window belongs Source directory, Destination directory, Output options and Summary. In the part Summary, the user selects none, one or both of the following options: Generate summary HTML file, Prase P-values from all sequences..

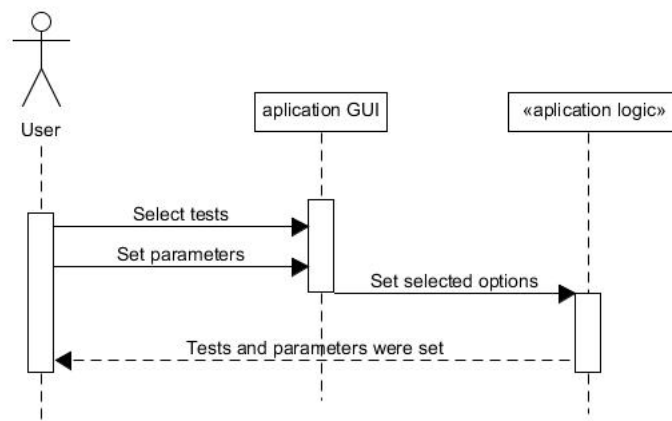


Figure 10: Sequence diagram - Set parameters for selected tests

The user interacts with the app's graphical interface. In the main menu selects test, which

would like to run. Clicks button *Add new* and test is inserted. User can set parameters for inserted test by inscribing values, for example. N - length of input string or M - length in bits of each block.

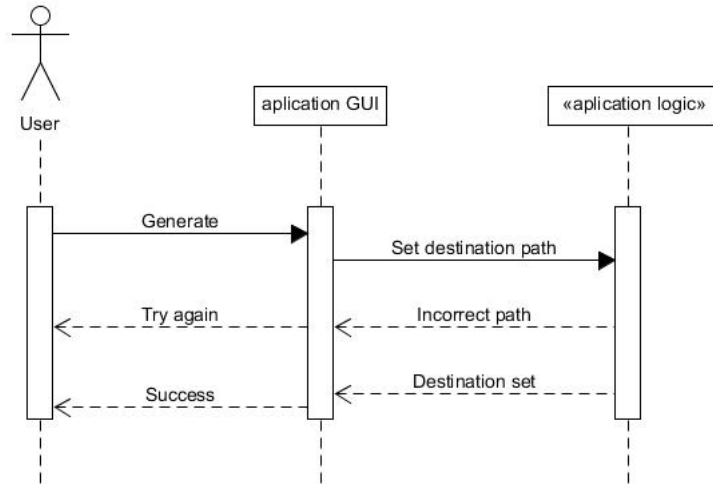


Figure 11: Sequence diagram - Generator-set destination file

The user interacts with the application GUI. In the main menu clicks Generate. A new window opens with three tasks. The User selects the path to a *Destination file*, where the generated sequence is going to be saved. If the User enters an incorrect path, he will be notified until a valid path is given.

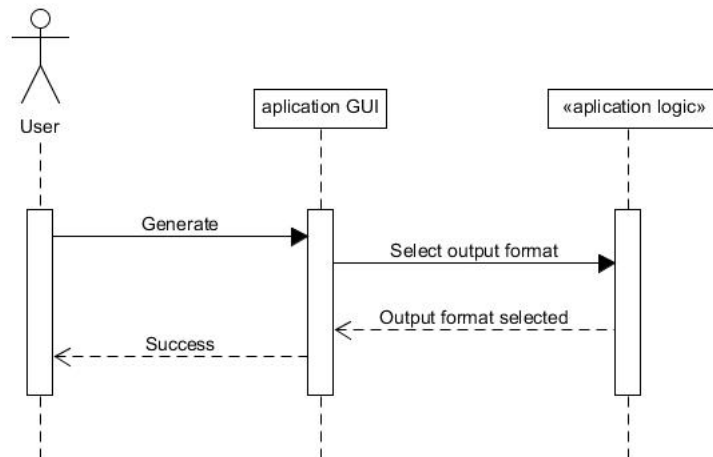


Figure 12: Sequence diagram - Generator-set output format

The user interacts with the application GUI. In the main menu clicks Generate. A new

window opens with three tasks. The User has to choose an *output format* from a predefined list of available formats.

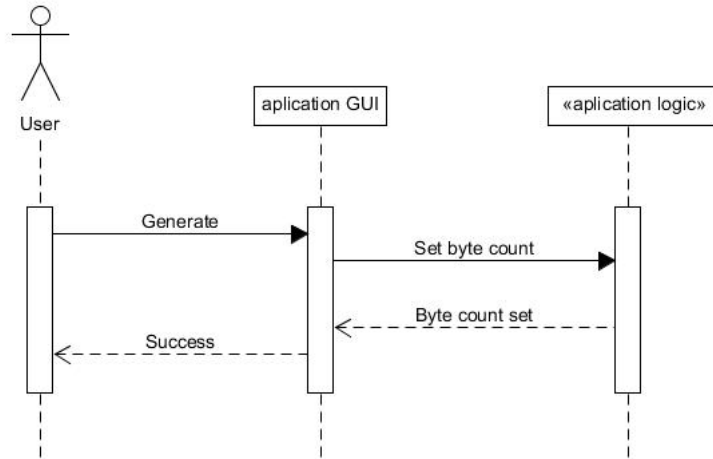


Figure 13: Sequence diagram - Generator-set bytecount

The user interacts with the application GUI. In the main menu clicks Generate. A new window opens with three tasks. The User has to provide a *byteCount*, which will be a number written to a text field.

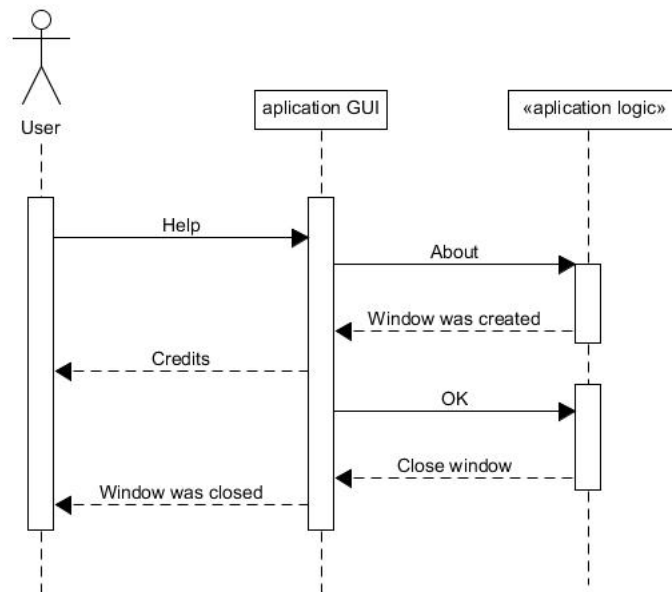


Figure 14: Sequence diagram - Help

The user interacts with the application GUI. In the main menu clicks Help. a submenu

appears with one element named *About...* Clicking the *About...* button will open the Credits window. The Credit window can be closed with the ok button.

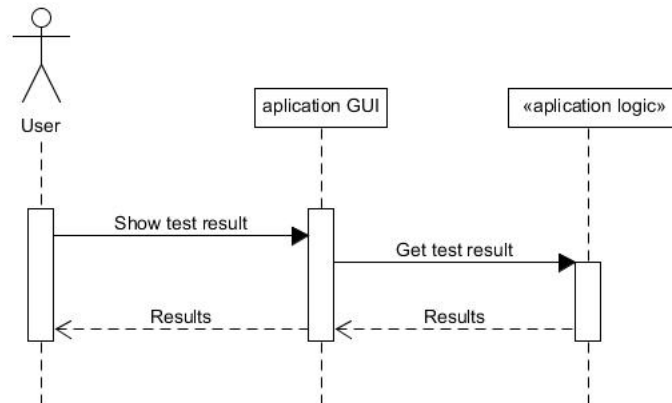


Figure 15: Sequence diagram - Show test results

After testing has ended, user has an option to show test results. When selected, it retrieves results from application logics and displays it to user via application GUI.

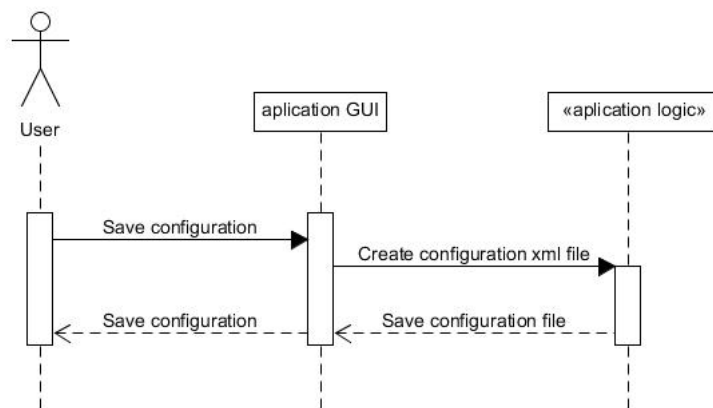


Figure 16: Sequence diagram - Save configuration

The user has an option to save current configuration in an XML file. The configuration is exported by application logics to an XML file which is sent back to user.

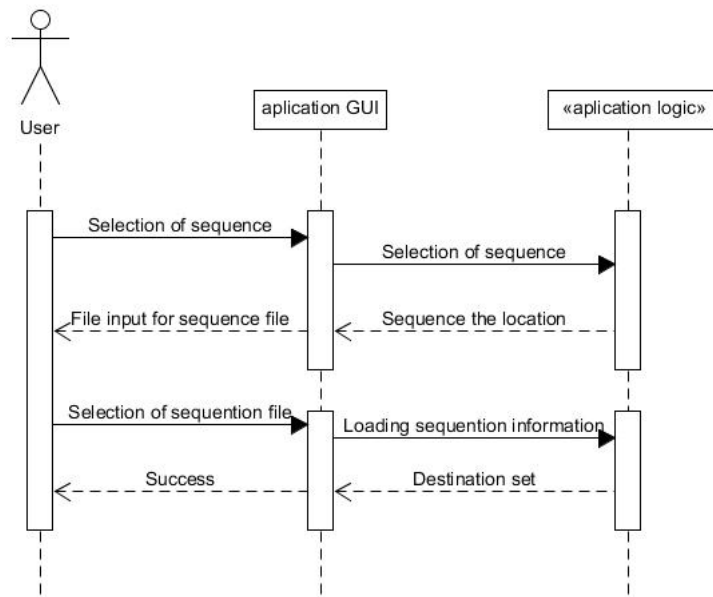


Figure 17: Sequence diagram - Load sequence

By selecting File -> Load Sequence, user is able to load a sequence into the program.

2.3 Activity Diagrams

This subsection contains all activity diagrams with pseudo codes corresponding to functionality of the application ParanoYa.

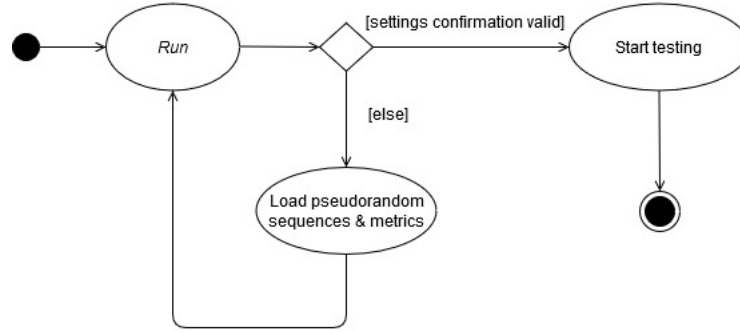


Figure 18: Activity diagram - Run

Algorithm 1: Start testing. Function triggered after user click event.

```

1 Function StartTests(event):
2   if settings_valid then
3     | start_testing()
4   else
5     | load_sequence()
6     | run()
7   end
  
```

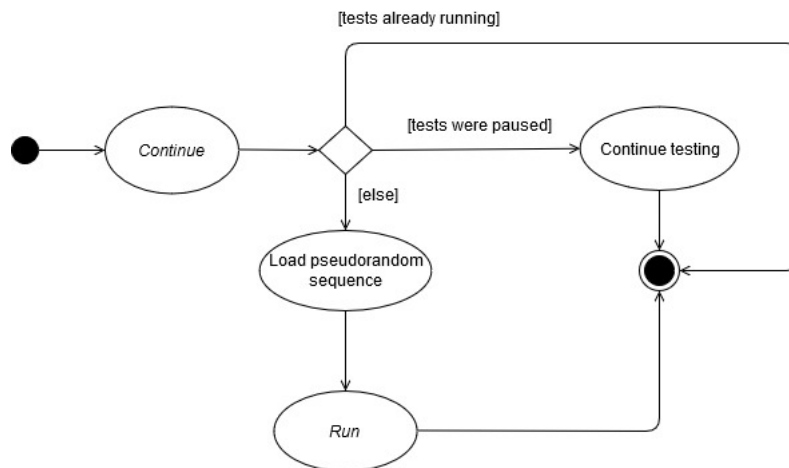


Figure 19: Activity diagram - Continue

Algorithm 2: Continue testing. Function triggered after user click event.

```
1 Function ContinueTests(event):  
2   if tests_running then  
3     return  
4   end  
5   if tests_paused then  
6     continue_testing()  
7   else  
8     load_sequence()  
9     run()  
10  end
```

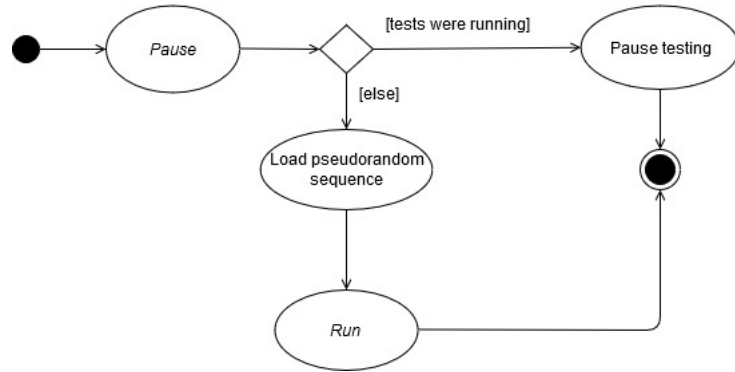


Figure 20: Activity diagram - Pause

Algorithm 3: Pause testing. Function triggered after user click event.

```
1 Function PauseTests(event):  
2   if tests_running then  
3     pause_testing()  
4   else  
5     load_sequence()  
6     run()  
7   end
```

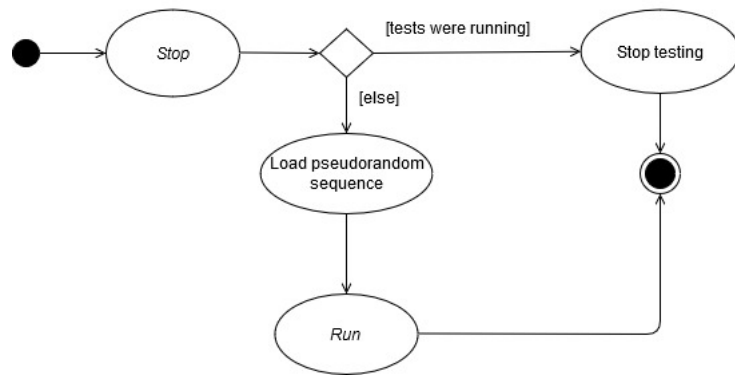


Figure 21: Activity diagram - Stop

Algorithm 4: Stop testing. Function triggered after user click event.

```

1 Function StopTests(event):
2   if tests_paused then
3     return
4   end
5   if tests_running then
6     stop_testing()
7   else
8     load_sequence()
9     run()
10  end

```

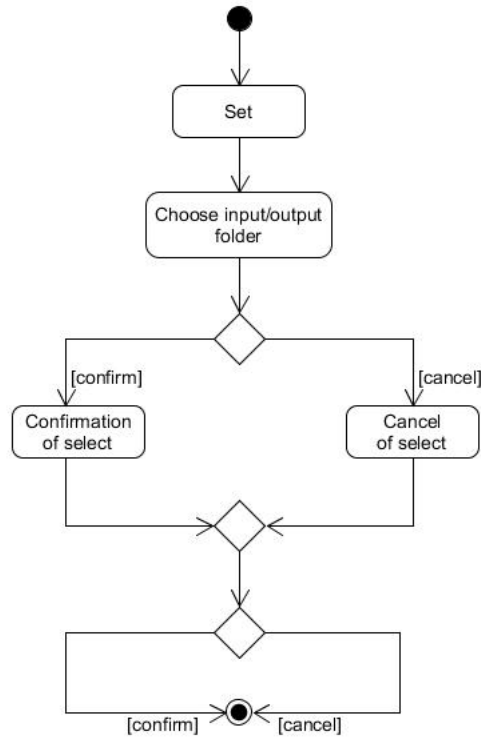


Figure 22: Activity diagram - Set input/output folder

Algorithm 5: Set input/output folder. Function triggered after user click event.

```

1 Function SetInputOutputFolder(event):
2   chosenFolder <- choose_folder();
3   if ok then
4     | sourceDirectory <- chosenFolder;
5   end
6   if cancel then
7     end

```

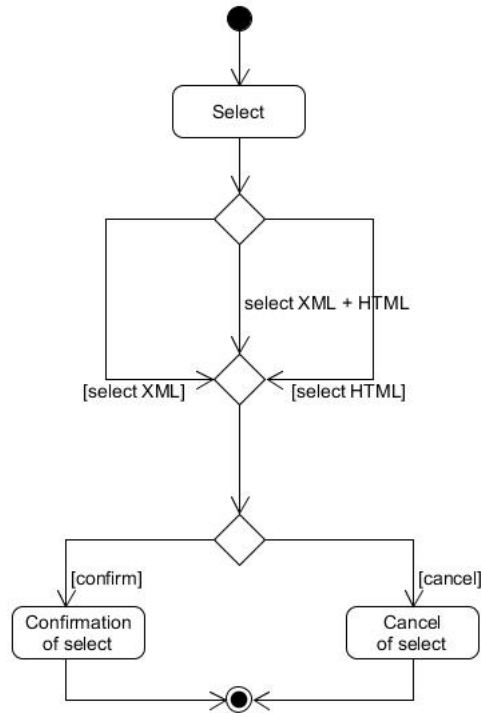


Figure 23: Activity diagram - Select output option

Algorithm 6: Select output option. Function triggered after user click event.

```

1 Function SelectOutputOption(event):
2   if XMLselected then
3     |   outputOption < - XML;
4   end
5   if HTMLselected then
6     |   outputOption < - HTML;
7   else
8     |   outputOption < - XMLHTML;
9   end

```

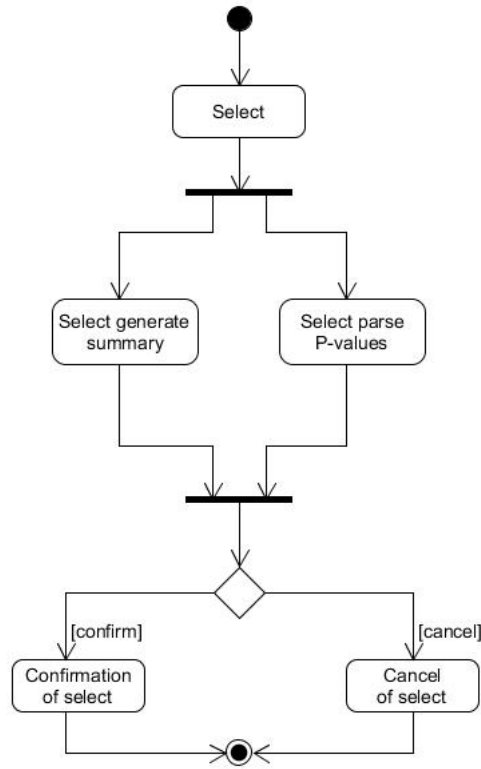


Figure 24: Activity diagram - Select summary option

Algorithm 7: Select summary option. Function triggered after user click event.

```

1 Function SelectSummary(event):
2   if isClickedGenerate then
3     | generateSum < - true;
4   else
5     | generateSum < - false;
6   end
7   if isClickedParse then
8     | parsePvalues < - true;
9   else
10    | parsePvalues < - false;
11  end

```

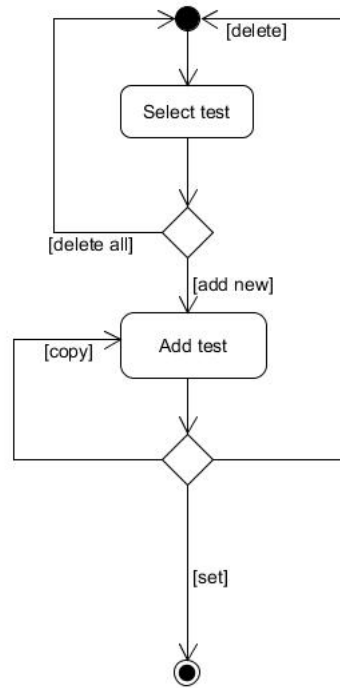


Figure 25: Activity diagram - Set parameters for selected tests

Algorithm 8: Set parameters for selected tests. Function triggered after user click event.

```

1 Function SetParameters(event):
2   select_test();
3   if add then
4     add_test();
5     if copy then
6       | copy_test();
7     end
8     if delete then
9       | delete_test();
10    end
11  end
12  if deleteAll then
13    | delete_all_tests();
14  end

```

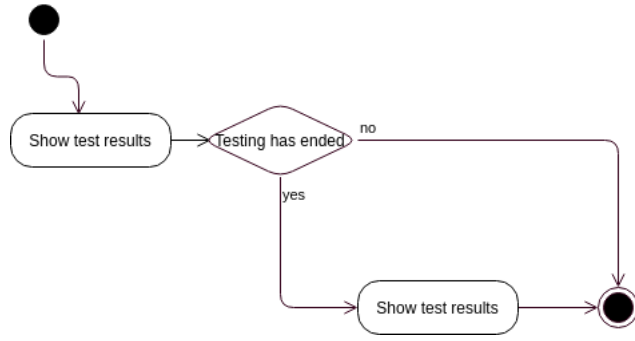


Figure 26: Activity diagram - Show test results

Algorithm 9: Show test results, after testing has ended, triggered after user click

```

1 Function ShowTestResults(event):
2   if testing_has_ended then
3     | show_test_results()
4   end

```

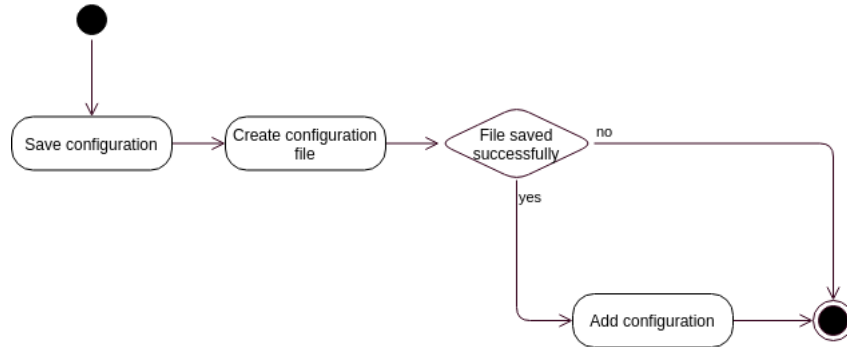


Figure 27: Activity diagram - Save configuration

Algorithm 10: Save configuration, triggered by selecting the option

```

1 Function SaveConfiguration(event):
2   file <- create_configuration_file()
3   if file then
4     | add_configuration()
5     | save_configuration()
6   end

```

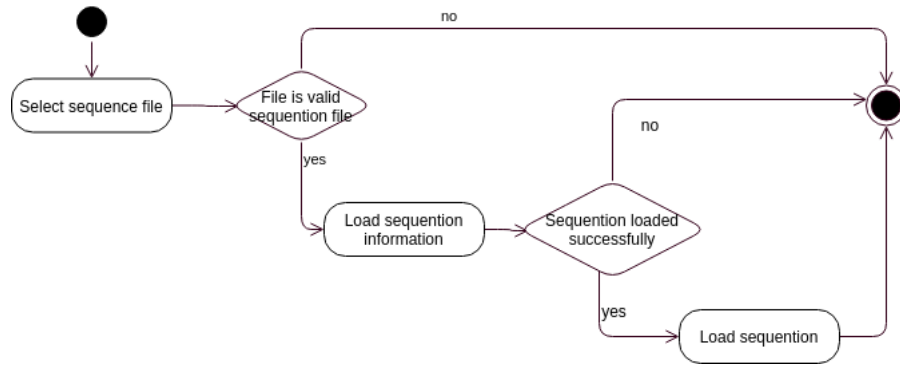


Figure 28: Activity diagram - Load sequence

Algorithm 11: Load sequence, triggered by selecting the option

```

1 Function LoadSequence(file):
2   if is_valid_sequention_file(file) then
3     sequention <- import_sequention(file)
4     if sequention then
5       load_sequention(sequention)
6     end
7   end

```

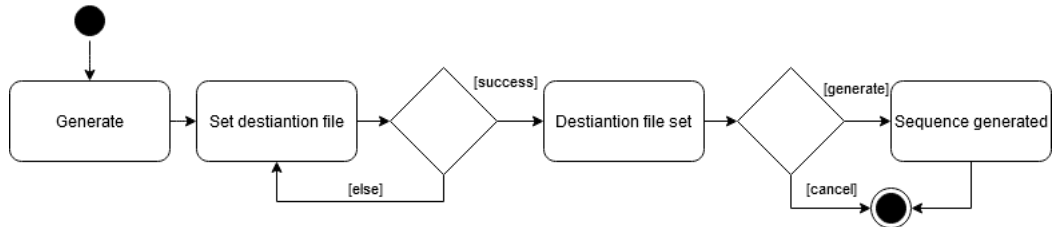


Figure 29: Activity diagram - Generate sequence/Set destination file

Algorithm 12: Generate sequence into file.

```

1 Function Generate(event):
2   destinationFile_select();
3   if success then
4     destinationFile_set() ;
5   else
6     destinationFile_select() ;
7   end
8   generate();

```

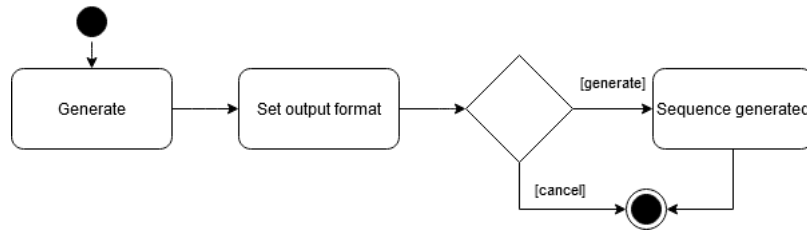


Figure 30: Activity diagram - Generate sequence/Set output format

Algorithm 13: Generate sequence into file.

```

1 Function Generate(event):
2   |   OutputFormat_set();
3   |   generate();

```

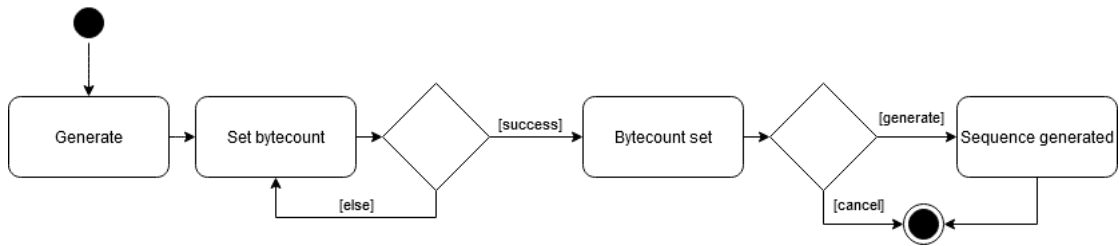


Figure 31: Activity diagram - Generate sequence/Set bytecount

Algorithm 14: Generate sequence into file.

```

1 Function Generate(event):
2   |   byteCount_select();
3   |   if success then
4   |   |   byteCount_set() ;
5   |   else
6   |   |   byteCount_select() ;
7   |   end
8   |   generate();

```



Figure 32: Activity diagram - Help

Algorithm 15: Open help for inforamtion

1 **Function** `Help(event)`:

2 `help()`;

3 `About_click()`;

4 `Credits.show()`;

3 Acceptance tests

Section acceptance tests includes tables which describes these tests. Individually tables contain ID of acceptance test, name of test, interface for test, input, output, steps with action and expected reaction.

ID	1	Name	Show test results
Interface	Client / application GUI / application logics		
Input	Successfully ended testing		
Output	Test results are displayed to user in application GUI		
Step	Action	Expected reaction	
1	Testing ended	Application GUI shows an option to display test results	
2	Users selects to show test results	Test results are displayed to user	

ID	2	Name	Save configuration
Interface	Client / application GUI / application logics		
Input	-		
Output	Configuration is saved in a XML file		
Step	Action	Expected reaction	
1	User makes a change in a configuration	Application saves change for configuration	
2	Users selects to save configuration	Configuration is saved in a XML file	

ID	3	Name	Load sequence
Interface	Client / application GUI / application logics		
Input	-		
Output	Sequence is loaded into application		
Step	Action	Expected reaction	
1	User selects to load sequence	A file input is displayed to user	
2	Users selects valid configuration file	A sequence is loaded into application from the chosen file	

ID	4	Name	Run
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Tests started		
Step	Action	Expected reaction	
1	User enters tab Settings	Tab window is opened	
2	User selects option Run	Tests start running	

ID	5	Name	Continue
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Tests continue		
Step	Action	Expected reaction	
1	User enters tab Settings	Tab window is opened	
2	User selects option Continue	Stopped tests will run	

ID	6	Name	Pause
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Tests were paused		
Step	Action	Expected reaction	
1	User enters tab Settings	Tab window is opened	
2	User selects option Pause	Running tests will be paused	

ID	7	Name	Cancel
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Tests stopped		
Step	Action	Expected reaction	
1	User enters tab Settings	Tab window is opened	
2	User selects option Cancel	Running tests will stop	

ID	8	Name	Cancel
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Set input/output folder		
Step	Action	Expected reaction	
1	User chooses input/output folder	Input/output folder is chosen	
2	User selects option Cancel	Chosen folders are canceled	
3	User selects option OK	Chosen folders are set	
4	User selects option Cancel	Selected options are canceled	
5	User selects option OK	Selected options are successfully set	

ID	9	Name	Cancel
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Selected output option		
Step	Action	Expected reaction	
1	User selects one output option	Output option is selected	
2	User selects option Cancel	Selected options are canceled	
3	User selects option OK	Selected options are successfully set	

ID	10	Name	Cancel
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Selected summary option		
Step	Action	Expected reaction	
1	User selects one summary option	Summary option is selected	
2	User selects option Cancel	Selected options are canceled	
3	User selects option OK	Selected options are successfully set	

ID	11	Name	Cancel
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Set parameters for selected tests		
Step	Action	Expected reaction	
1	User selects test	Selected test is shown	
2	User selects option Add new	Advanced options are shown	
3	User set parameters for chosen test	Parameters are set	
4	User selects option Copy	Test is copied with set parameters	
5	User selects option Delete	Current test is deleted	
6	User selects option Delete All	All tests are deleted	

ID	12	Name	Gen-Set file path
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Selected summary option		
Step	Action	Expected reaction	
1	User provides path to destination file	Path saved	
2	User selects option OK	Path to destination file successfully set	

ID	13	Name	Gen-Set output
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Set output format		
Step	Action	Expected reaction	
1	User selects output format option	Output format selected	
2	User selects option OK	Selected output format is successfully set	

ID	14	Name	Gen-Set byte-count
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Bytecount set		
Step	Action	Expected reaction	
1	User provides bytecount	Bytecount saved	
2	User selects option OK	Selected bytecount is successfully set	

ID	15	Name	Help
Interface	Client / application GUI / application logic		
Input	Click event		
Output	Shows credits window		
Step	Action	Expected reaction	
1	User enters help submenu	Help submenu appears	
2	User selects option About...	Credits window opens	
3	User selects option OK	Credits window closes	

4 GUI - Graphical user interface

This sections contains new design for user interface of application ParanoYa. There is described tool in which was created new user interface. There are also displayed individual windows with their functionality.

4.1 Figma

Design concepts were created in a design tool named Figma which is cloud-based and freely accessible. The environment is accessible from any operating system which supports browsers, which means it accessible from any computer with access to the Internet. It also supports offline work with a desktop application. Figma offers a wide variety of design tools and also supports prototyping. The finished concepts can be exported to various file formats which includes generated CSS code.

4.2 Structure

From the main menu the user can access every feature of the application. Accessing a feature is realized by clicking the corresponding button which opens up a dialog window. To access another feature or the main menu the currently opened dialog window has to be closed. Only one dialog window can be opened at a time.

The user can create test sets by clicking the **New test** [Figure 34] button which will open a dialog window. Previously saved test sets can be loaded easily by clicking on the **Set test** button which will open a dialog window [Figure 36]. By clicking the **Add sequence** button the user can add sequences for testing. This event opens up another dialog window [Figure 35]. Sequences can also be generated by the application. To open the generate dialog window [Figure 37] the user has to click the **Generate** button. After the necessary parameters are set the user can start testing by clicking the **Run test** button which will open a dialog window [Figure 38].

After all the test have been performed the results of the test are shown in the Results window [Figure 39] which opens automatically. The user can see the details of the results for every sequence by simply clicking on the corresponding cell in the results table. This action will open a new dialog window [Figure 40].

4.3 Main menu

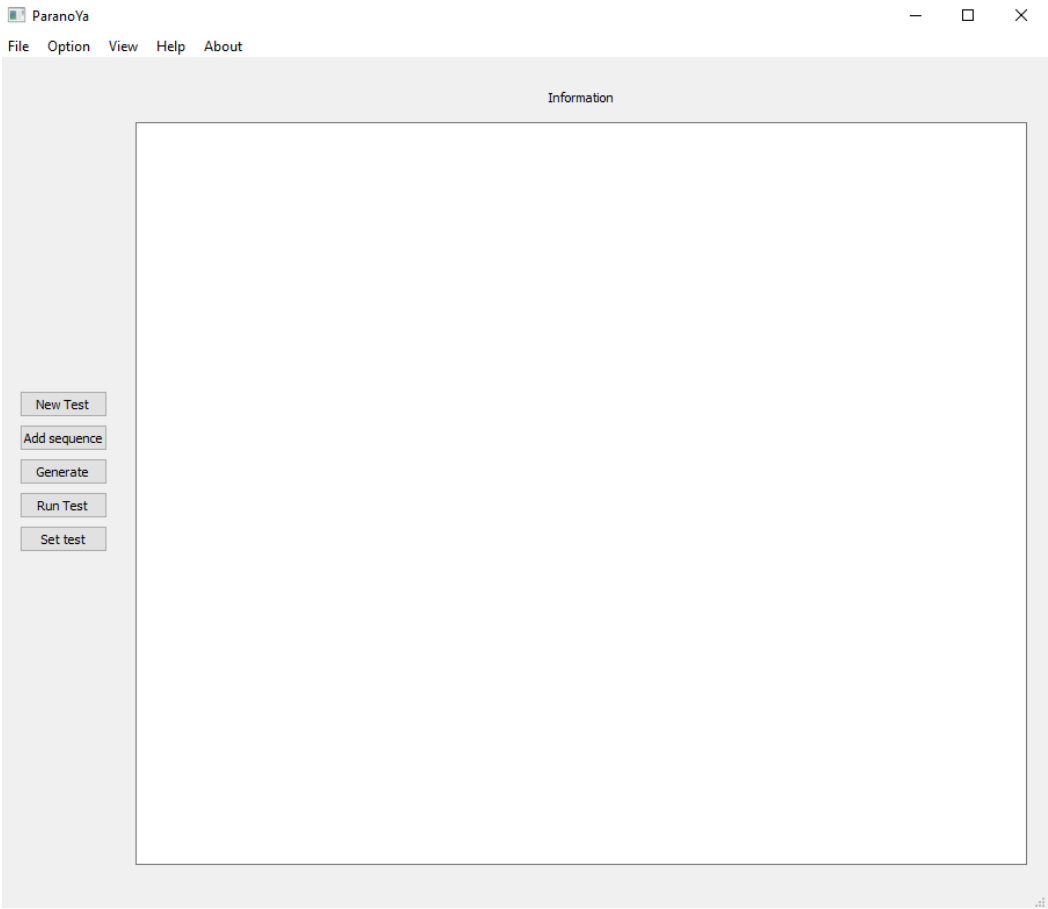


Figure 33: Main menu

Purpose:	Every function of the application is available from this window: <ul style="list-style-type: none">• New test• Add sequence• Generate sequence• Run test• Set test
Navigation and User Interaction:	The user can choose an action by clicking the buttons. The corresponding window will appear immediately after that.

4.4 New Test

The screenshot shows a window titled "Add Test" with a standard OS window control bar (minimize, maximize, close) and a help icon. The main area contains a dropdown menu currently set to "Test1". Below the dropdown are four buttons: "Add", "Add All", "Remove", and "Remove All". The "Remove All" button is highlighted with a blue border. To the right of these buttons is a list box containing "Test1" and "Test2". Further right are three input fields labeled "Param1", "Param2", and "Param3". "Param1" contains the value "0.01", "Param2" contains "0.02", and "Param3" contains "8848". Below these input fields is a "Set" button. At the bottom of the dialog are two buttons: "Save" and "Load".

Figure 34: New Test

Purpose:	The user is able to create a new set of tests, save that set or load from an existing, previously saved set.
Navigation and User Interaction:	<p>By clicking the button Add the user is able to add a new test to the set. The new test will be chosen from a drop-down list containing all available test.</p> <p>Test parameters can be set on the right handside of the window and by clicking the Set button the changes will be saved.</p> <p>A selected test can be removed by clicking the Remove button.</p> <p>By clicking the Add All button the user is able to easily add all available tests this set.</p> <p>By clicking the Remove All button the user can remove all tests from the set.</p> <p>By clicking the button Save the user is able to save the current set of tests.</p> <p>By clicking the button Load the user is able load a previously save set of tests.</p>

4.5 Add sequence

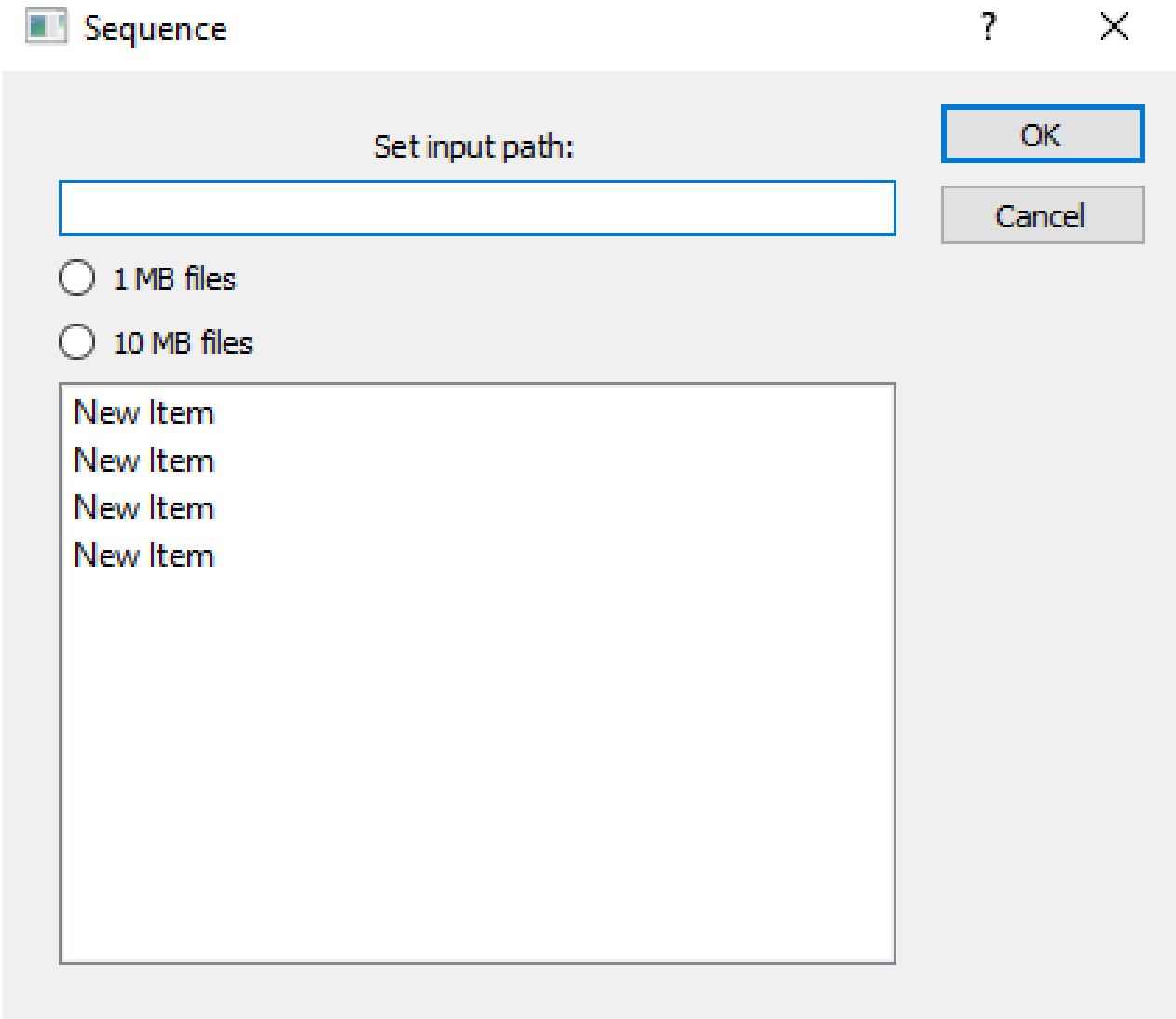


Figure 35: Add sequence

Purpose:	The user can add sequences for testing.
Navigation and User Interaction:	The user can specify the source folder which contains the sequences to be tested. They have to specify the size of the source files. Currently ParanoYa support fixed sizes of 1MB and 10MB. After clicking the button Ok the files in the specified folder will be imported. By clicking the Cancel button, the window is closed.

4.6 Load test

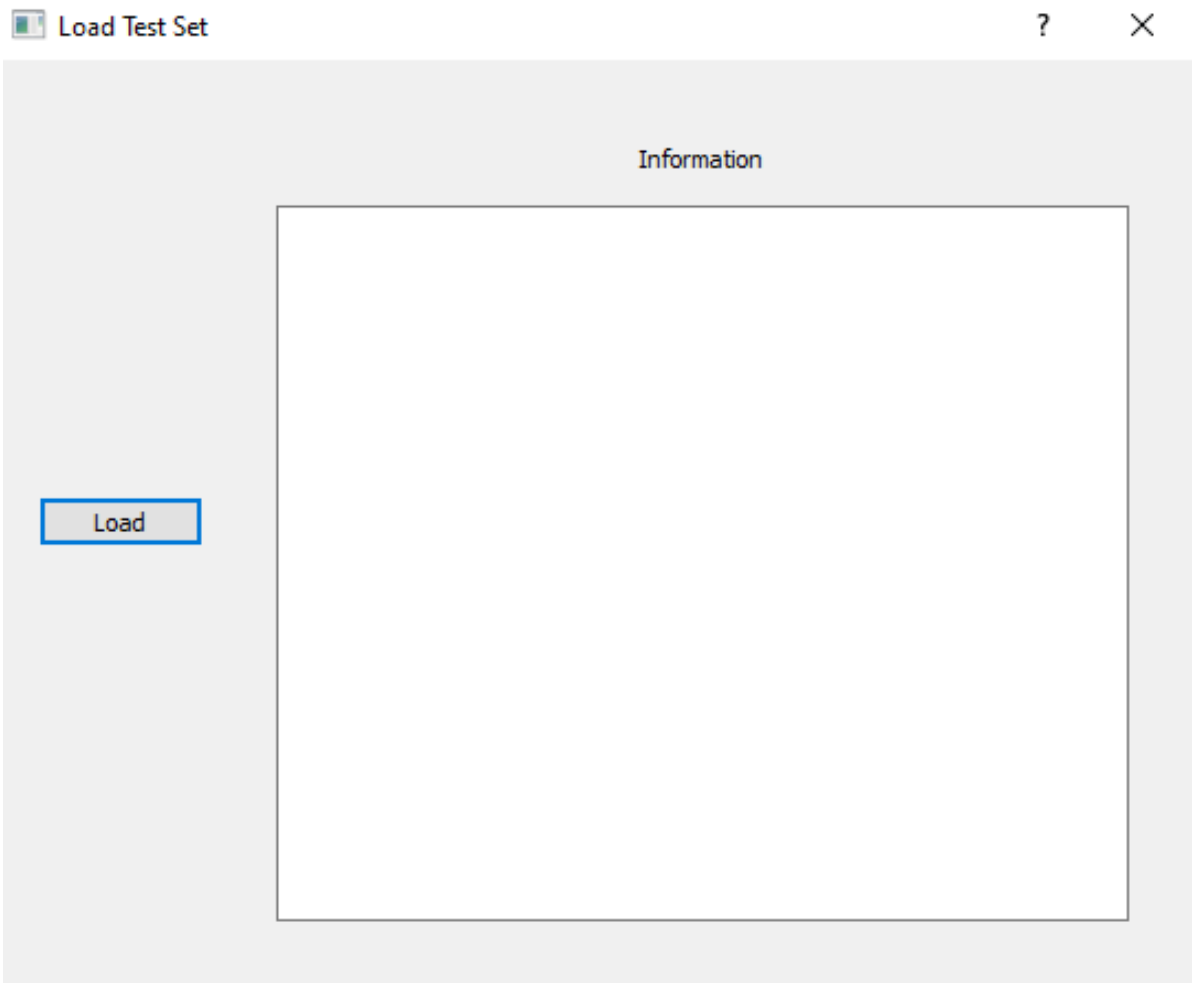


Figure 36: Set test

Purpose:	The user can choose from previously created test sets.
Navigation and User Interaction:	The user can choose from a list of predefined test set and load a selected set by clicking the Load button.

4.7 Generate

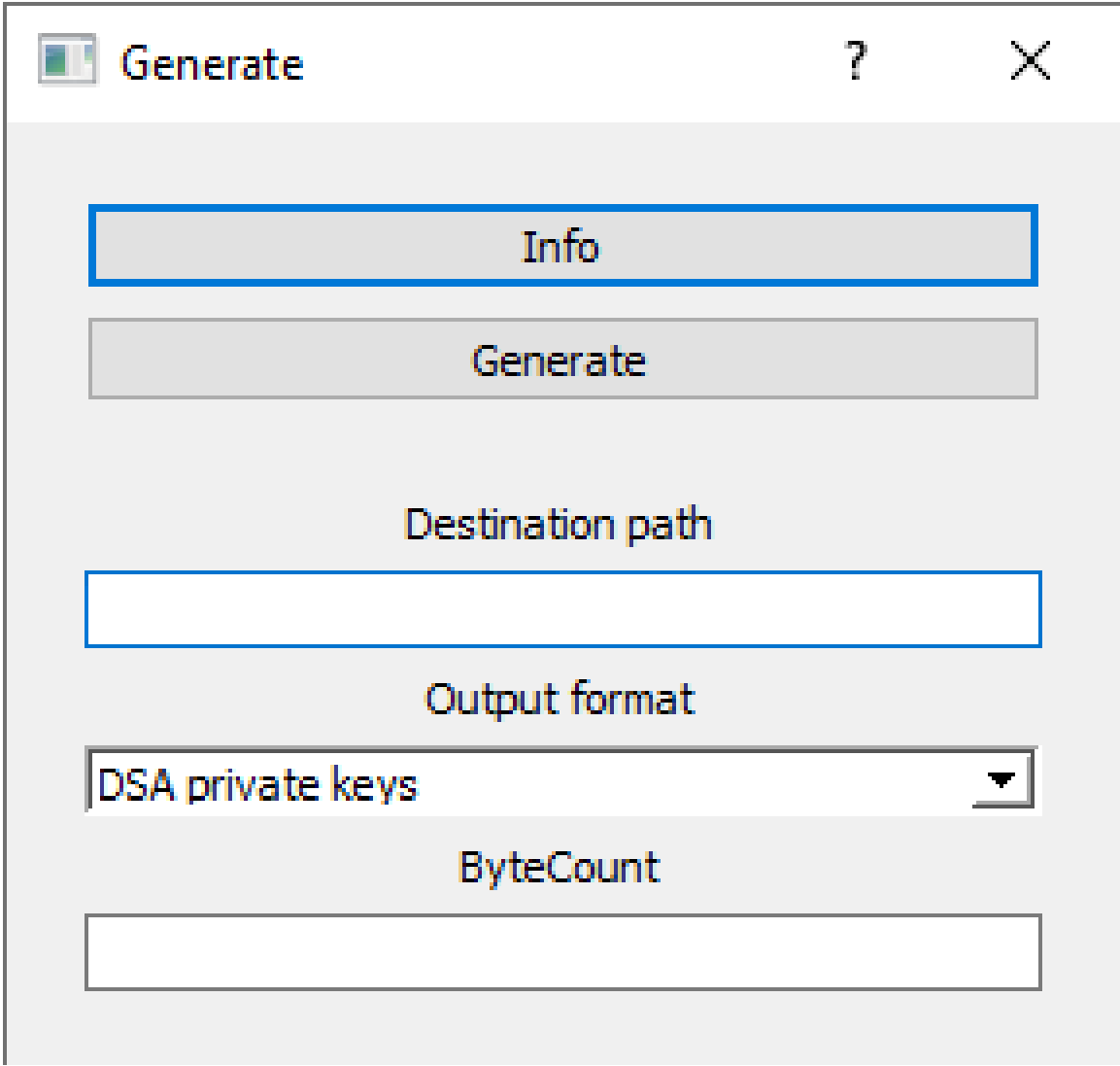


Figure 37: Generate

Purpose:	The user is able to generate sequences for testing
Navigation and User Interaction:	Some basic information is available for the parameters and their optimal value or range of value. After the parameters are set the user can initiate the generation process by clicking the Generate button.

4.8 Run Test

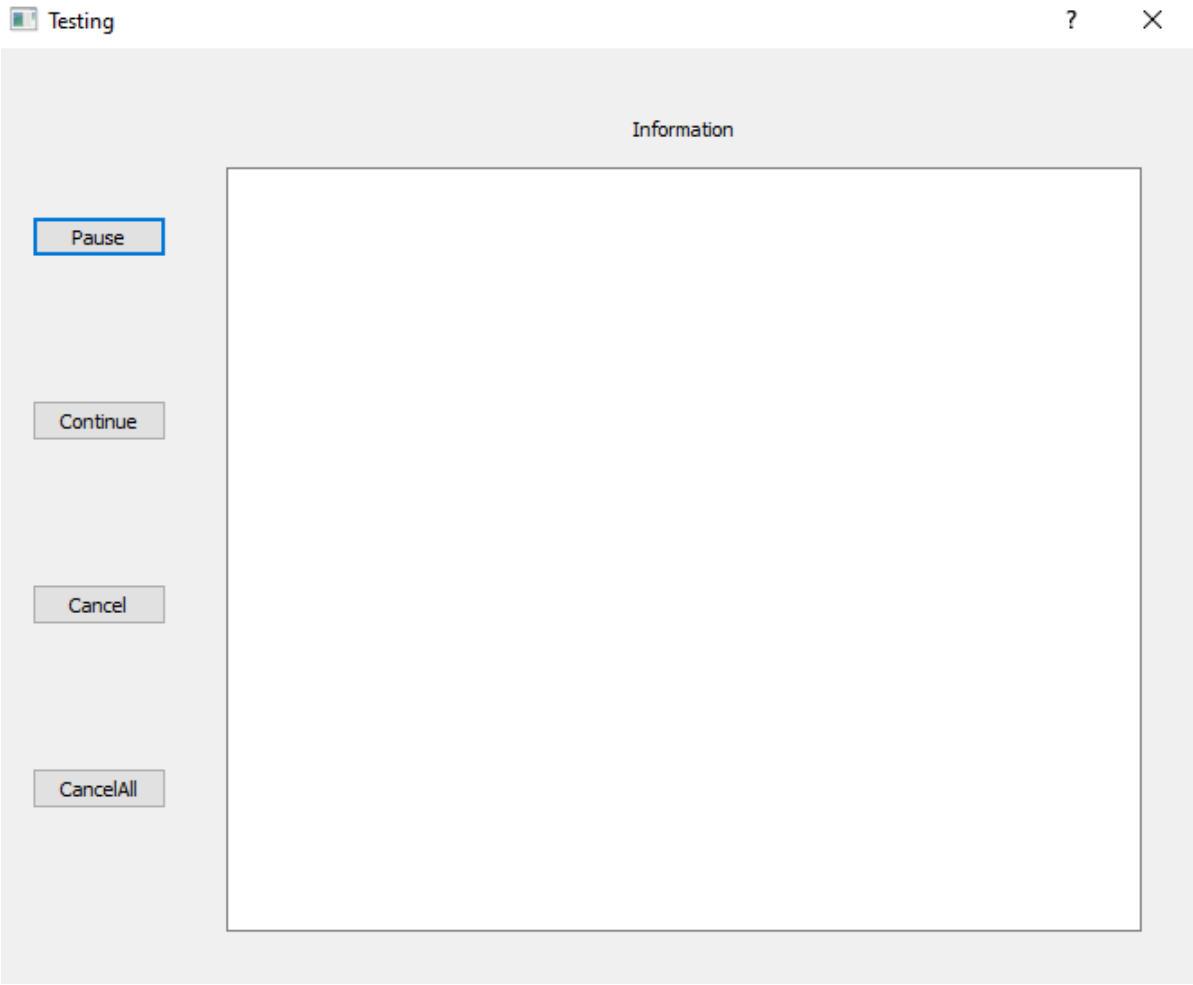


Figure 38: Run Test

Purpose:	The user can pause, continue or cancel ongoing tests.
Navigation and User Interaction:	The user can pause an ongoing test by clicking the Pause button. The user can continue an already paused test by clicking the Continue button. The user can cancel the ongoing test by clicking the Cancel button.

4.9 Results

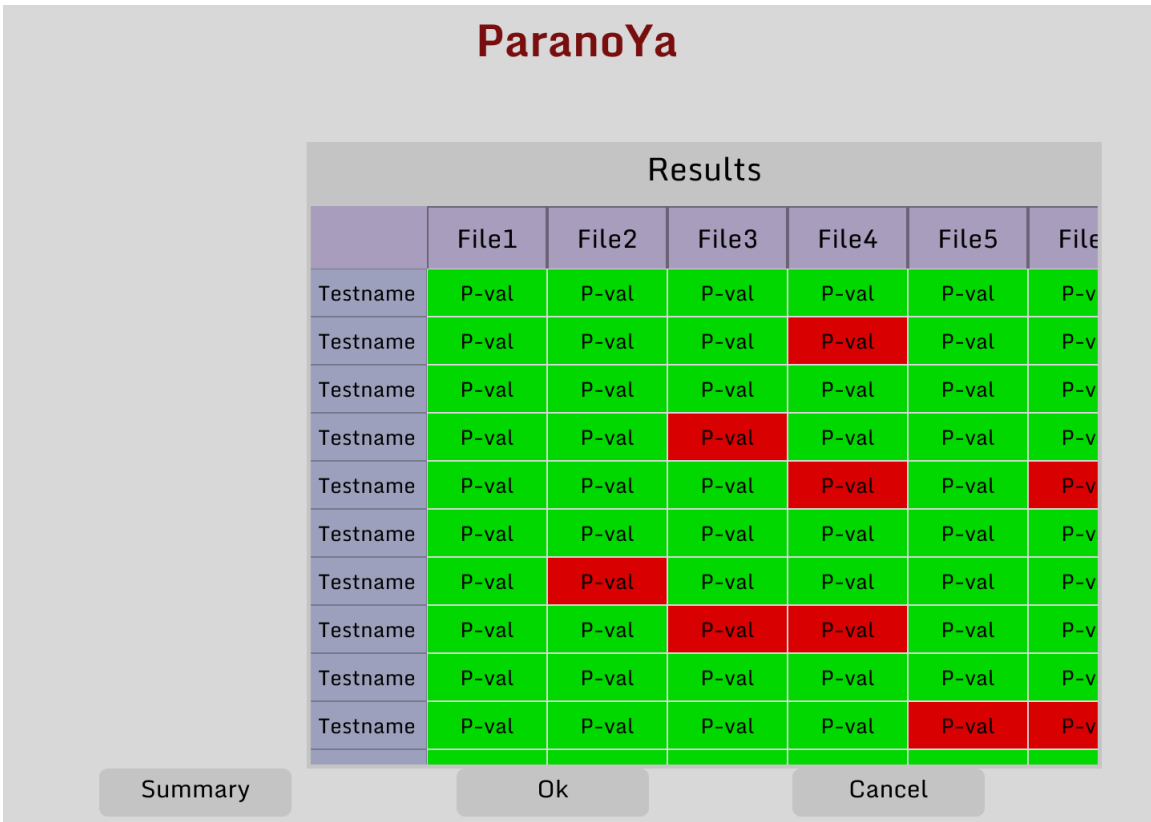


Figure 39: Results

Purpose:	The user is able to evaluate the results of the testing.
Navigation and User Interaction:	The result are summarized in a table. The rows represent the list of performed tests, the columns represent the tested sequences. The cells hold information about the success of the tests. The user can click on every cell to check why the chosen sequence failed or passed the test. By clicking the Summary button the user can generate a file containing the summmary of the test.

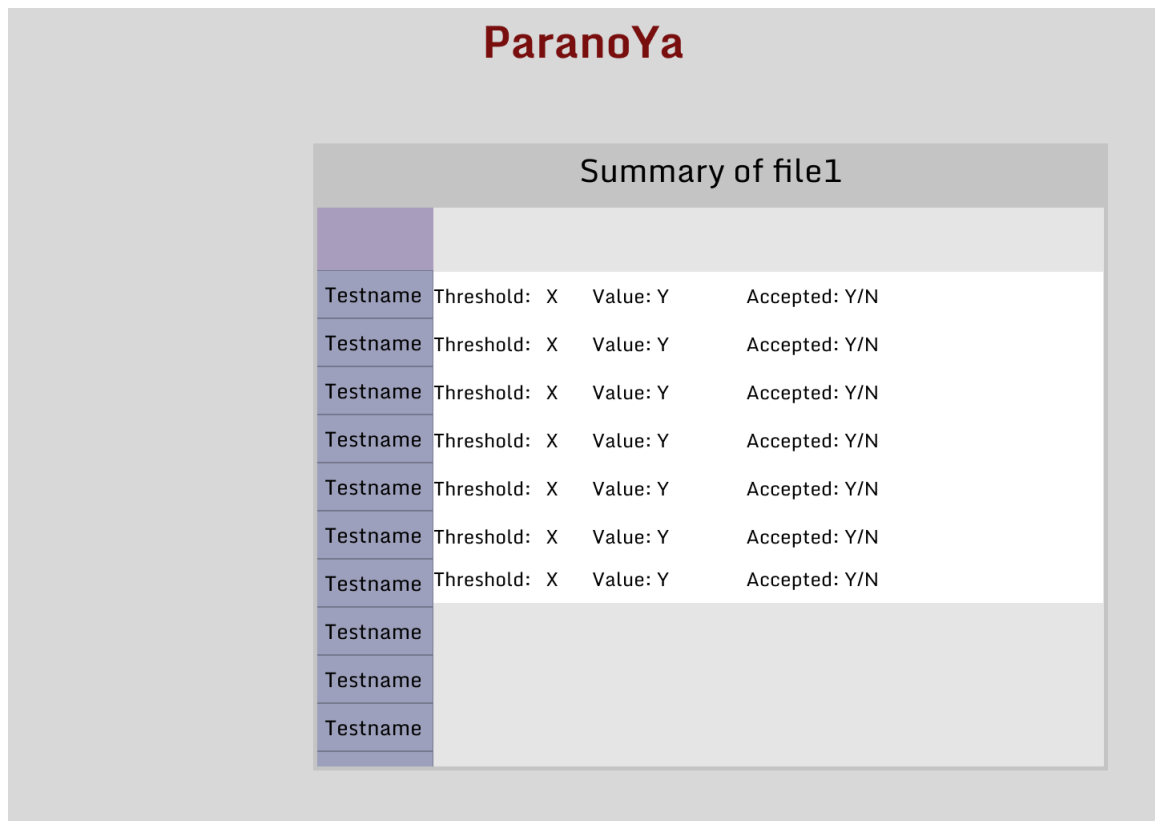


Figure 40: Detailed results

Purpose:	The user is able to evaluate the results of performed tests by every sequence.
Navigation and User Interaction:	This table shows information about every test performed on the selected sequence. The information includes the predefined threshold to pass the test, the actual value of the performed test and final verdict of the test.

5 Implementation

In this section are described tools, libraries which we used for development and detailed process of implementation.

5.1 NIST test set

We used NIST test set library developed by GINARTeam in Python in the github repository

<https://github.com/GINARTeam/NIST-statistical-test>

This test set contains 15 NIST tests. It reads an input byte sequence file, runs the tests and outputs the result in csv files.

We implemented this test set in the application. We extended the library by implementing another functionality. User is now able to select multiple files to be tested. User can also choose tests to be tested.

5.2 Application

The application folder is used to manage testing and connect application logic and GUI.

```
class MainWindow:
    __instance = None

    @staticmethod
    def getInstance():
        if MainWindow.__instance is None:
            MainWindow()
        return MainWindow.__instance

    def __init__(self):
        if MainWindow.__instance is not None:
            raise Exception("This class is a singleton!")
        else:
            MainWindow.__instance = self
```

The main window of application is represented by a singleton MainWindow. This is the place that all of user interactions are being handled. MainWindow stores information about its UI and it also handles changing UI in case user switches to another window.

When opened the main window, user has several possible actions. By selecting New Test, user is served a new window and user is able to create a custom test set and choose which tests he wants to run out of all of the available tests.

Button Add sequence opens a new window which handles loading byte sequence file which is then being tested. Button Generate is used to generate random byte sequence file in case user does not own the specific byte sequence file but wants to run the tests anyway. It pops up a new window displaying the options of generated byte sequence file.

Button Run Tests opens a testing window and starts testing immediately. User is able to cancel testing or to Pause / Continue testing. After testing is done, user is displayed results.

Button Set Test is used to load a specific test set. User can choose from a list of test sets by clicking Load button on the window popped up.

5.3 Tests

The NIST test set contains a total of 15 tests:

1. Monobit test
2. Frequency within block test
3. Runs test
4. Longest run ones in a block test
5. Binary matrix test
6. DFT test
7. Non overlapping template matching test
8. Overlapping template matching test
9. Maurer's universal test
10. Linear complexzity test
11. Serial test
12. Approximal entropy test
13. Cumulative sums test

14. Random excursion test

15. Random excursion variant test

These tests are testing the null hypothesis of a byte sequence, which says that the sequence is random. Each applied test accepts or rejects the hypothesis based on the statistic value. The determined critical value is used to clarify whether the results should be accepted or rejected. The tests results are being output into csv files. The files have specific form of output for each different test. However all of the tests compare the P-value with critical value and then decides if the sequence is accepted or rejected.

P-value represents the probability that the chosen test statistic will assume values that are equal to or worse than the observed test statistic value when considering the null hypothesis.

The tests have the normal and the chi-squared as reference distributions. The standard normal distribution is used to compare the value of the test statistic obtained from the random number generator with the expected value of the statistic under the assumption of randomness. The Chi-squared distribution is used to compare the goodness-of-fit of the observed frequencies of a sample measure to the corresponding expected frequencies of the hypothesized distribution

Conclusion

In the introduction of our work was set the aims that should be achieved by elaborating the given topic. These aims: creation of detailed analysis for given problem, creation of new design for user interface and creation of prototype of new application with new user interface and chosen test set, we managed to fulfill.

We achieved these aims in three main phases. In the first phase we elaborated detailed analysis for the given problem. We described current state of the application, implemented test sets which are used in the application and we analysed existing programs similar as ParanoYa. We also analysed methods used for evaluate testing sequences.

In the second phase, we described entire functionality of the application with UML diagrams. We also created acceptance tests for every application use case. In this phase, we also created new design for user interface.

In the last phase, we implement new graphical user interface and prototype of new application with statistical test suite NIST. This prototype was created in programming language Python.

Original version of the application ParanoYa was actualized for the current development tools and environment. It could help developers who would like to extend or change this application in future. The application has new user interface that is more intuitive and user-friendly. We also reduced testing time for chosen test set. With these updates, we increased comfort for the user of this application.

Possible extensions for further development:

- Extension of methodologies by tests that have not yet been implemented in the application.
- Recasting the core of NIST tests according to Marek Sys's library.
- Extension of the application for evaluation of outputs (according to the Excel document model).
- Description of the methodology of testing for new implemented modifications in the application .

Bibliography

1. BRATKOVÁ, Eva (comp.). *Metody citování literatury a strukturování bibliografických záznamů podle mezinárodních norem ISO 690 a ISO 690-2: metodický materiál pro autory vysokoškolských kvalifikačních prací* [online]. Verze 2.0, aktualiz. a rozšíř. Praha: Odborná komise pro otázky elektronického zpřístupňování vysokoškolských kvalifikačních prací, Asociace knihoven vysokých škol ČR, 2008 [visited on 2011-02-02]. Available from: <http://www.evskp.cz/SD/4c.pdf>.

Appendix

A	Description of application	II
---	--------------------------------------	----

A Description of application

On the picture below we can see main window of application ParanoYa which will show after start the application. On the left side, there are all available tests and on the right side we can add new tests and set their parameters.

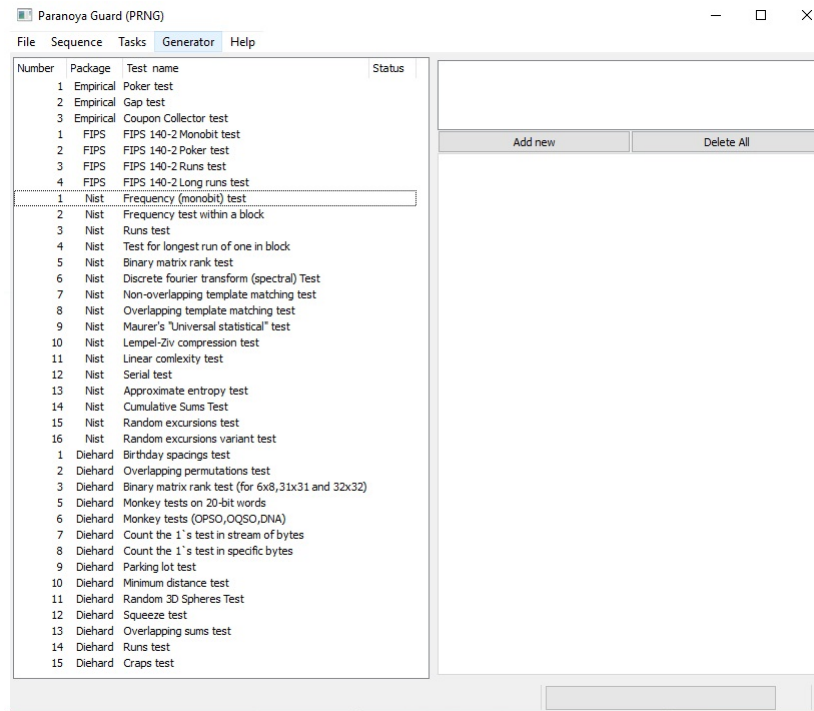


Figure A.1: ParanoYa - Main window

When we would like to test some sequences we can insert our methodical template. In the File tab in the application navigation bar. There is necessary selects Open and choose template that we want to use. Needed tests are set based on which template was chosen. On the picture below we can see that some tests are set after inserting the template.

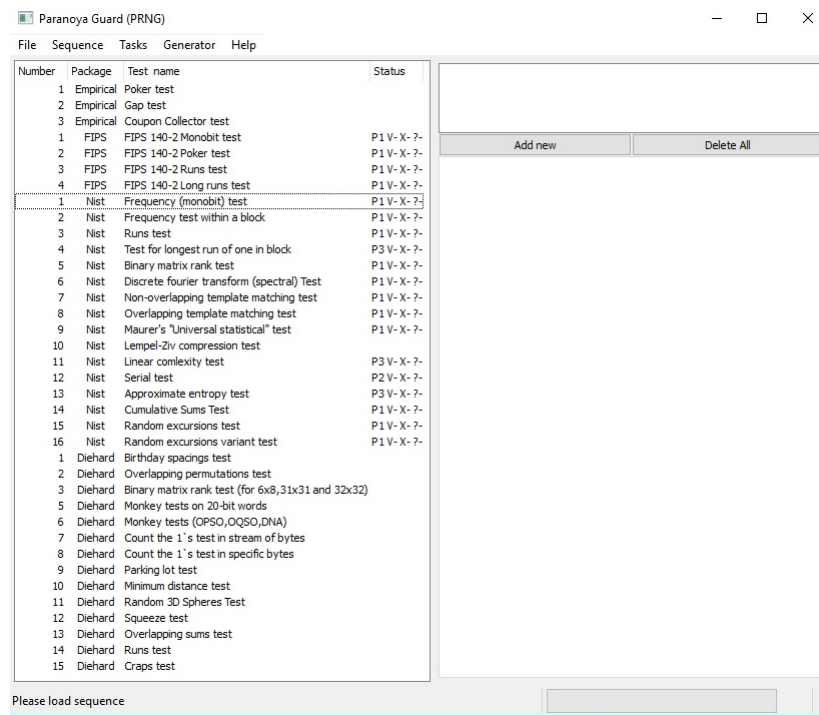


Figure A.2: ParanoYa - Tests are set

Some tests also contains various parameters that would be additionally modified on the right side of application what is displayed on the following picture.

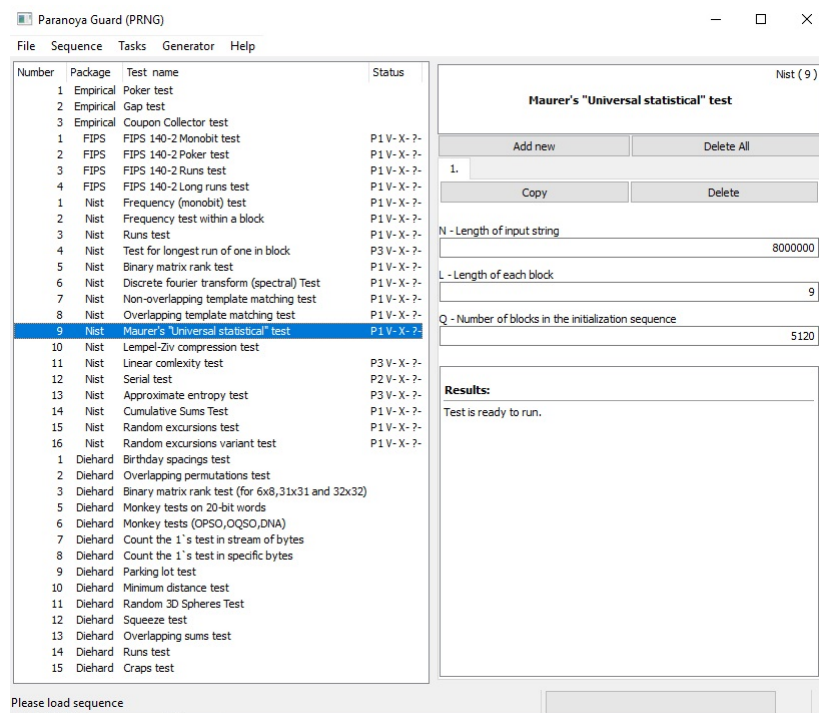


Figure A.3: ParanoYa - Additional modification of parameters

When all tests and parameters are set, there is also necessary to set source directory where are saved files with sequences, destination directory where ParanoYa will generate output files, output options and summary. These options we can see on the picture below. We can set these things after click the Batch process... in the File tab in the application navigation bar. After click the button Ok testing will start.

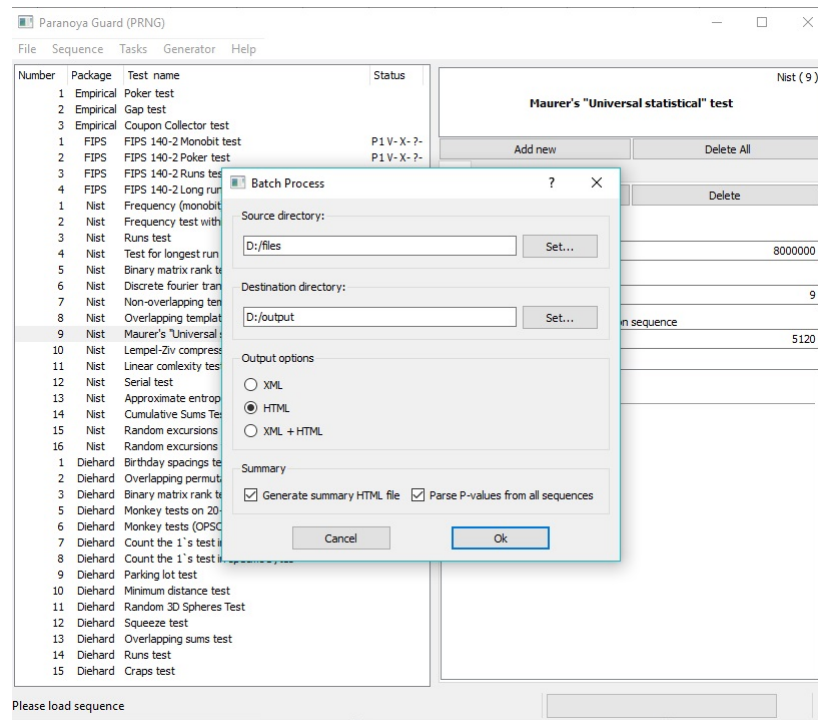


Figure A.4: ParanoYa - Set the output options

ParanoYa generates files for every testing sequence and also two files: pval.html and summary.html into destination directory. We can open the file pval.html in Microsoft Excel and evaluate results of testing sequences. File pval.html opened in Microsoft Excel is displayed on the next picture.

	A	B	C
1		D:/paranoYa/metodicke	D:/metodicke
2	Frequency (monobit) test	sablony/metodicka10MB.xml	sablony/metodicka1MB.xml
3	Frequency test within a block	0.000000	0.000000
4	Runs test	1.000000	1.000000
5	Test for longest run of one in block	Irr	Irr
6		0.000000	0.000000
7		0.000000	0.000000
8	Binary matrix rank test	0.000000	0.000000
9	Discrete fourier transform (spectral)		
10	Test	0.000000	0.000000
11	Non-overlapping template matching	0.000000	0.000000
12	test	0.000000	0.000000
13		0.167619	0.873030
14		0.000000	0.000000
15		0.000000	0.000000
16		0.000000	0.005786
17		0.026754	0.000000
			0.019009

Figure A.5: ParanoYa - pval.html file