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

Acknowledgments

We would like to express a gratitude to our thesis supervisor Mgr. Ing. Matúš Jókay, PhD. for his approach and expert advice.

Contents

List of Figures and Tables

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.[1] The application can run a variety of tests, including:

   - **Entropia**
   - **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 seperated by a comma(CSV format).

- **-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 on its website.[2]

```
#=============================================================================#
#          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

3. **Practically random** is a library implemented in the C++ programming language. It is suited for testing random number generators-*RNG*[3]

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.[4]

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 original version 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. [5, 6]

   (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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(j) **The Linear Complexity Test**

The purpose of this test is determine whether the sequence is sufficiently complex to be considered as random. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

(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. [5, 6]

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. [7, 8]

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. [9, 10] 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. [11, 12, 13]

   (b) **Level 2** - this level requires role-based access control, as well as physical security. [11, 12, 13]

   (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. [11, 12, 13]

   (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. [11, 12, 13]

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. [12]

FIPS tests:

- **FIPS monobit test**

  This test analyzes the distribution of ones and zeroes at each bit position. The number of ones and zeroes is likely to be approximately equal, if the sample is randomly generated. [14, 15]

- **FIPS poker test**

  In this test is the bit sequence at each position divides into consecutive, non-overlapping groups of four, and derives a four-bit number from each group. Then there is counting number of occurrences of each of the 16 possible numbers, and performs a chi-square calculation to evaluate this distribution. The distribution of four-bit numbers is likely to be approximately uniform, if was sample randomly generated. [14, 15]

- **FIPS runs tests**

  In this test is the bit sequence at each position divides into runs of consecutive bits which have the same value. Then there is counting the number of runs with a length of 1, 2, 3, 4, 5, and 6 and above. The number of runs with each of these lengths is likely to be within a range determined by the size of the sample set, if the sample is randomly generated. [14, 15]

- **FIPS long runs test**

  Purpose of this test is measured the longest run of bits with the same value at each bit position. the longest run is likely to be within a range determined by the size of the sample set, if the sample is randomly generated. [14, 15]

- **Spectral tests**

  In this test is analysing the bit sequence at each position and identifying evidence of non-randomness in some samples that pass the other statistical tests. The test treats each series of consecutive numbers as coordinates in a multi-dimensional space. Then there is plot a point in this space at each location determined by these co-ordinates. The distribution of points within this space is likely to be approximately uniform, the appearance of clusters within the space indicates that the data is likely to be non-random, if the sample is randomly generated. [14, 15]

- **Correlation test**

  This test describes that it is necessary to test for any statistically significant relationships between the values at different bit positions within the tokens. A value at a given bit position is equally likely to be accompanied by a one or a zero at any other bit position, if the sample is randomly generated. This test computes the probability of the relationships at each position observed with bits at other positions arising if the tokens are random. The test adjusts the significance level of the bit whose significance level is lower based on all of the other bit-level tests, to prevent arbitrary results, when a degree of correlation is observed between two bits. [14, 15]

- **Compression test**

  This test attempts to compress the bit sequence at each position using standard ZLIB compression. After this compression, there is indicate the proportional reduction in the size of the bit sequence. The data is less likely to be randomly generated, what indicates a higher degree of compression. [14, 15]

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. [16, 17, 18]

   (b) **Overlapping permutations**

   Purpose of this test is analyzing sequences of five consecutive random numbers. [16, 17, 18]

   (c) **Ranks of matrices**

In this test are selecting a number of bits from a number of random numbers to form a matrix above [0,1]. Later is determining order of matrix. [17, 19, 18]

(d) **Monkey test**

This test is also known as bitstream test. Name of this test is from an endless "monkey theorem". Test is processing sequences of a certain number oa bits as "words". During testing are counting the overlapping words in the stream. [17, 18]

(e) **Count the 1s**

This test performs 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". [16, 17, 18]

(f) **Parking lot test**

In this test are randomly place unit circles in a 100x100 square. This test is repeated if the circle overlaps an existing one. After 12,000 tries, the number of successfully "packed" circles, we should get a curve that should be similar to those provided by a perfect random number generator. [16, 17]

(g) **Minimum distance test**

In this test are also randomly place 8000 points in a 10,000 x 10,000 square. After placed points is finding the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean. [16, 17, 18]

(h) **Random spheres test**

In a cube of edge 1000 are randomly chosen 4000 points. Then it is centering a sphere on each point that has radius with minimum distance to another point. Sphere's volume which is the smallest should be exponentially distributed with certain mean. [17, 18]

(i) **The squeeze test**

In this test is multiplying 231 by float random integers on [0,1). This multiplying is repeated until is reach 1. It should be repeated 100,000 times. After these repetitions, the number of floats needed to reach 1 should follow a certain distribution. [16, 17, 18]

(j) **Overlapping sums test**

In this test is generating a long sequence of random floats on [0,1). After generating is added sequence of 100 consecutive floats. The sums should be normally distributed with characteristic mean and sigma. [16, 17, 18]

(k) **Runs test**

In this test is also generating a long sequence of random floats on [0,1). Then is counting ascending and descending runs. These counts should follow a certain distribution. [16, 17, 18]

(l) **The craps test**

This test is described with game craps.This game is playing 200,000 times. During games are counting the wins and the number of throws per game. Each of this counts should follow a certain distribution. [16, 17, 18]

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.

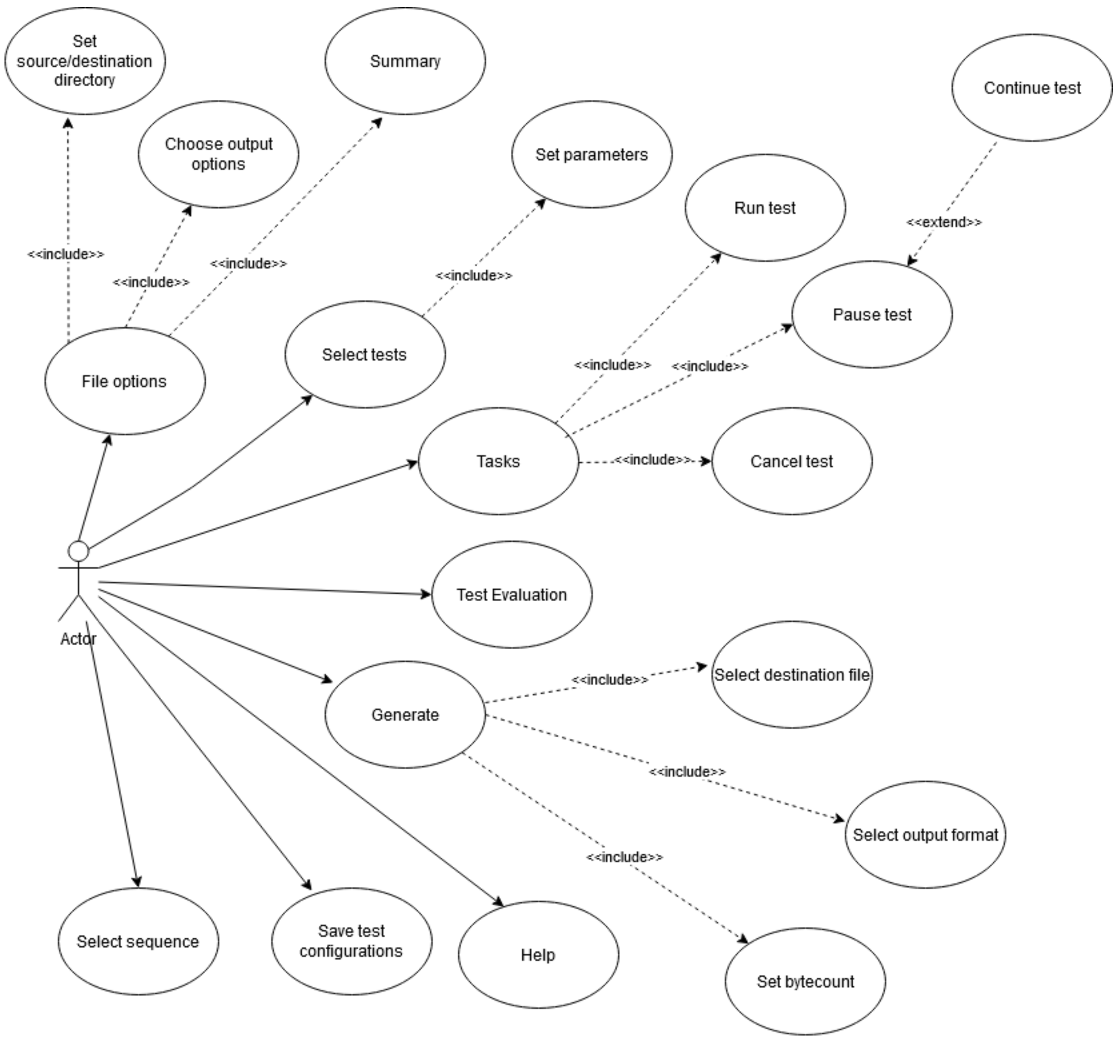

Figure 2: Use case diagram

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.

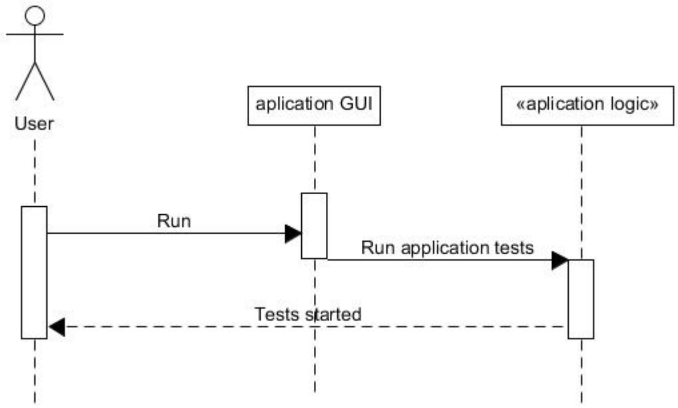

Figure 3: Sequence diagram - Run

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.

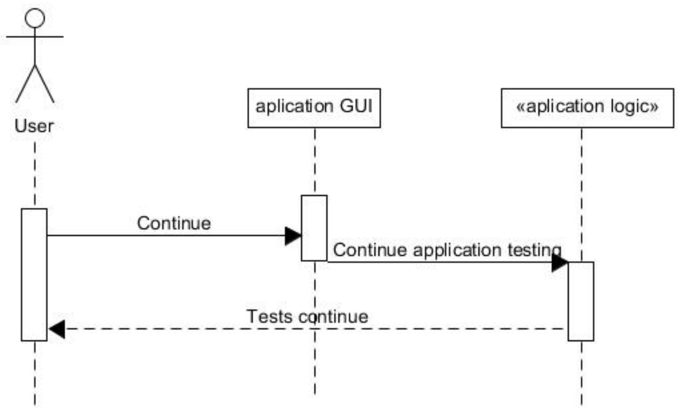

Figure 4: Sequence diagram - Continue

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.

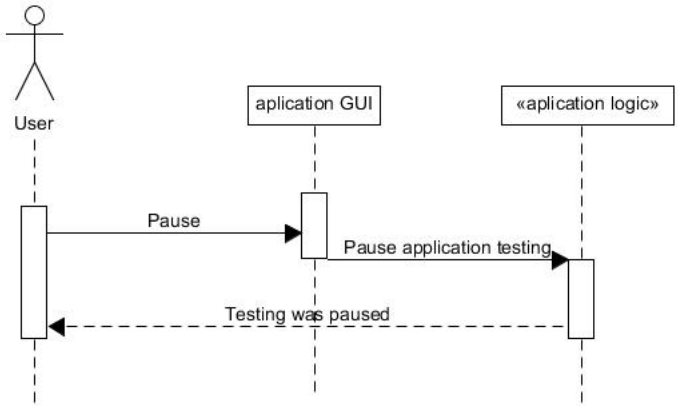

Figure 5: Sequence diagram - Pause

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.

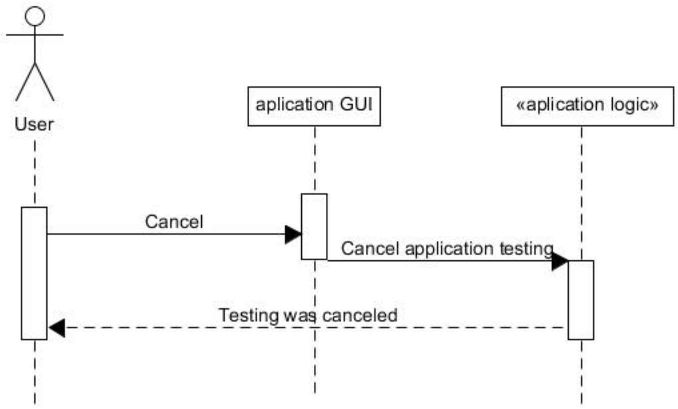

Figure 6: Sequence diagram - Cancel

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.

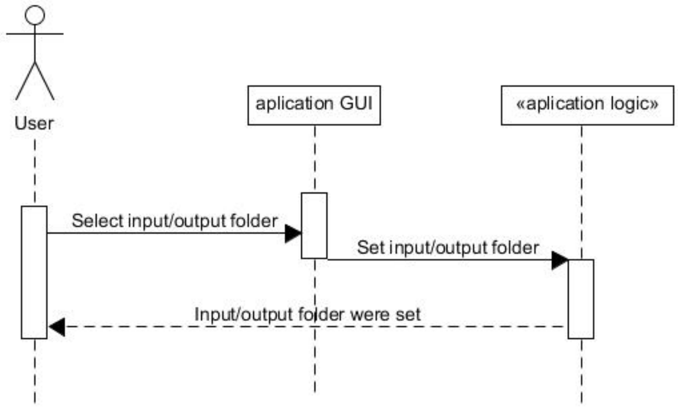

Figure 7: Sequence diagram - Set input/output folder

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.

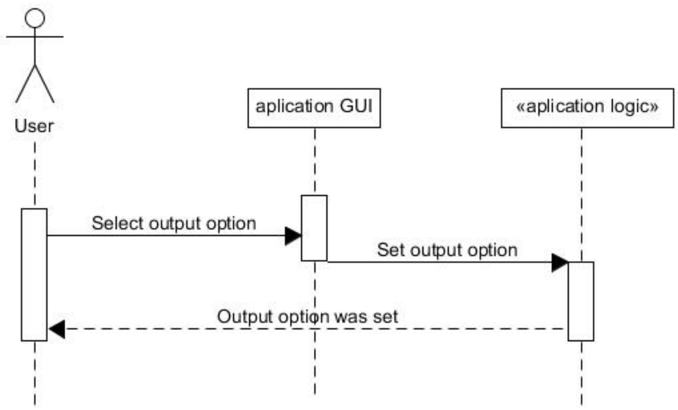

Figure 8: Sequence diagram - Select output option

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.

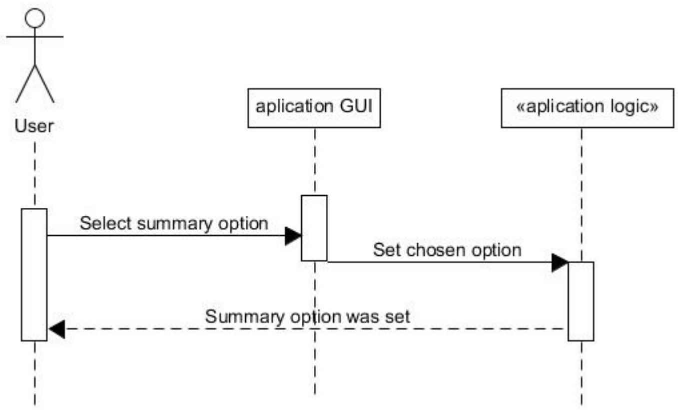

Figure 9: Sequence diagram - Select summary option

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..

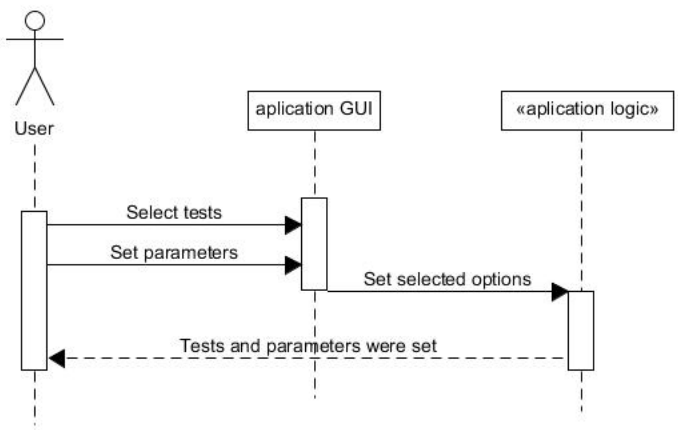

Figure 10: Sequence diagram - Set parameters for selected tests

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.

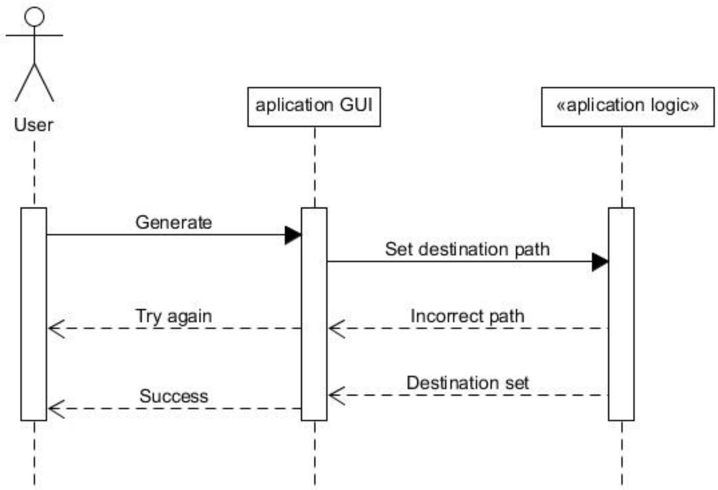

Figure 11: Sequence diagram - Generator-set destination file

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.

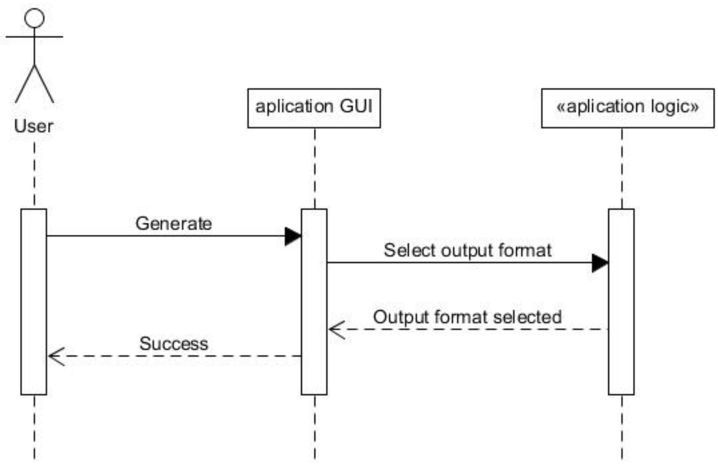

Figure 12: Sequence diagram - Generator-set output format

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.

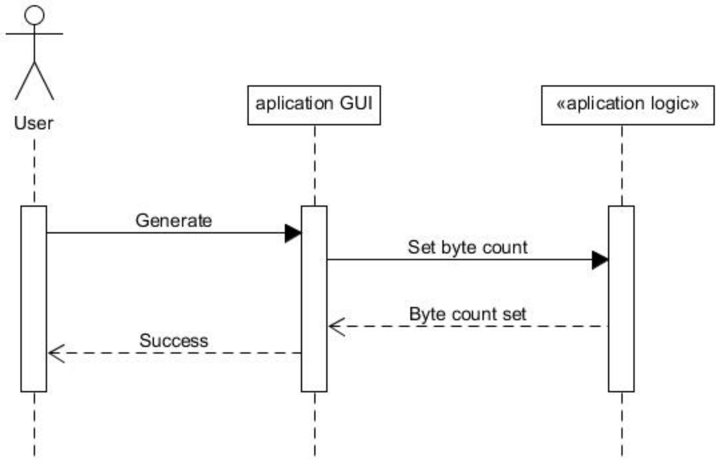

Figure 13: Sequence diagram - Generator-set bytecount

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.

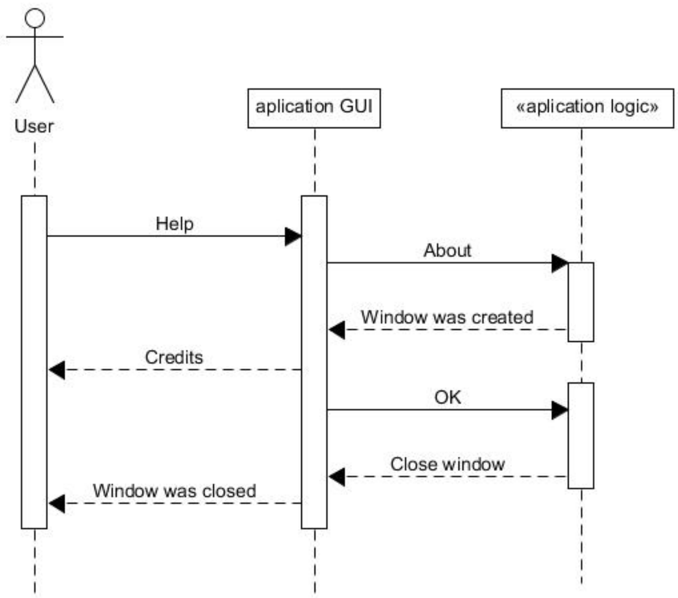

Figure 14: Sequence diagram - Help

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.

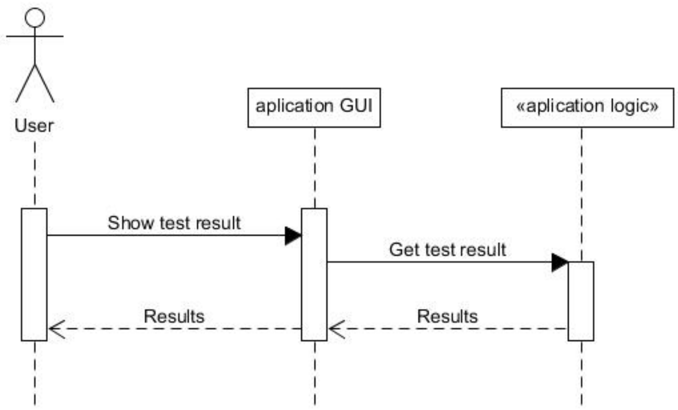

Figure 15: Sequence diagram - Show test results

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.

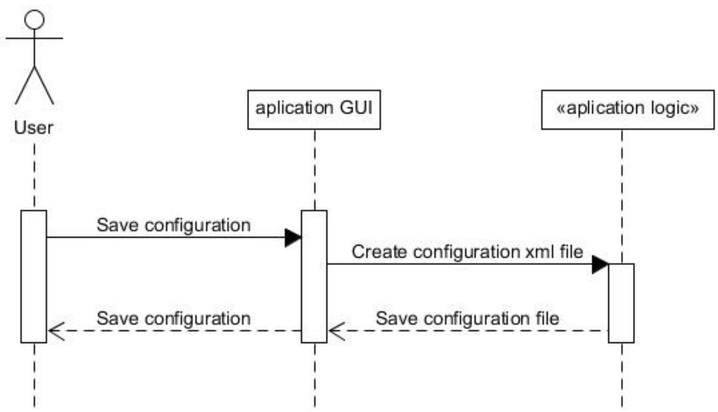

Figure 16: Sequence diagram - Save configuration

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.

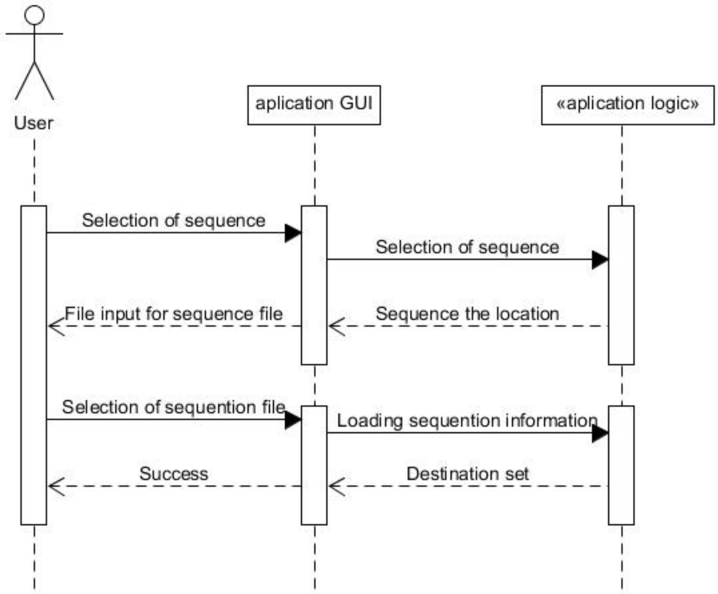

Figure 17: Sequence diagram - Load sequence

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.

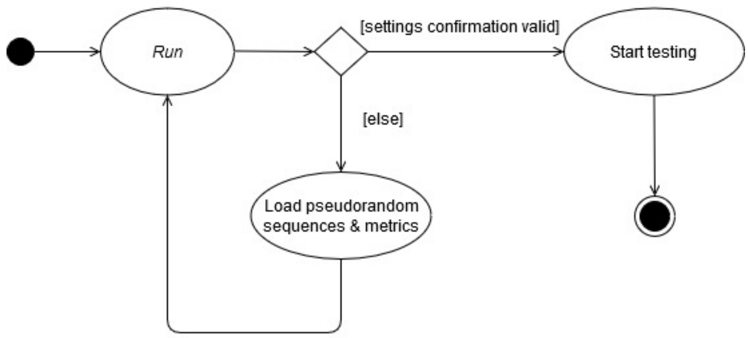

Figure 18: Activity diagram - Run

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

```
Function StartTests(event):
    if settings_valid then
        start_testing()
    else
        load_sequence()
        run()
    end
```

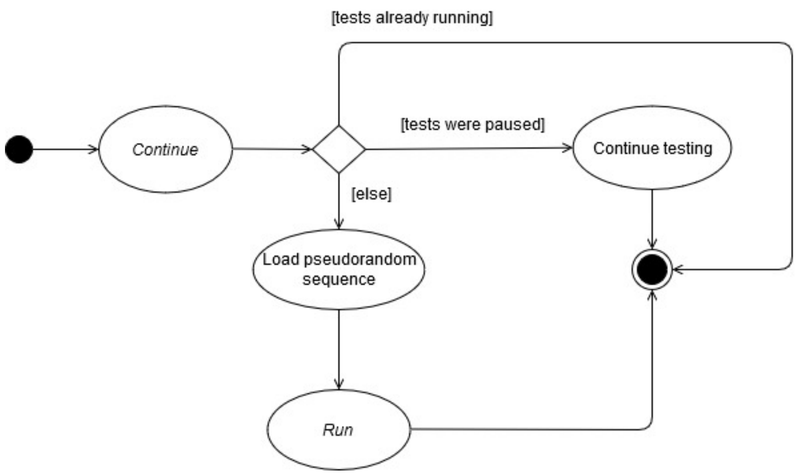

Figure 19: Activity diagram - Continue

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

```
Function ContinueTests(event):
    if tests_running then
        return
    end
    if tests_paused then
        continue_testing()
    else
        load_sequence()
        run()
    end
```

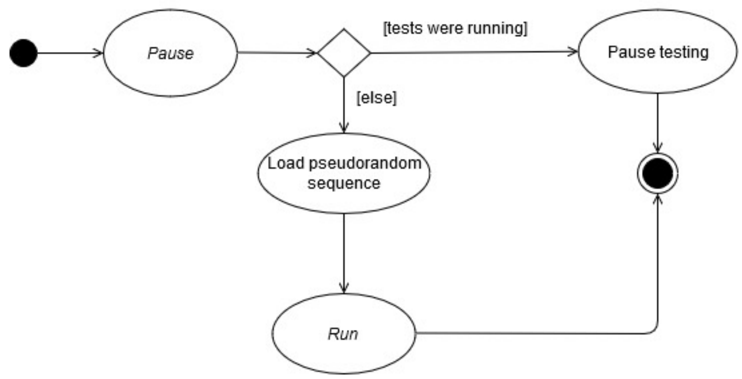

Figure 20: Activity diagram - Pause

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

```
Function PauseTests(event):
    if tests_running then
        pause_testing()
    else
        load_sequence()
        run()
    end
```

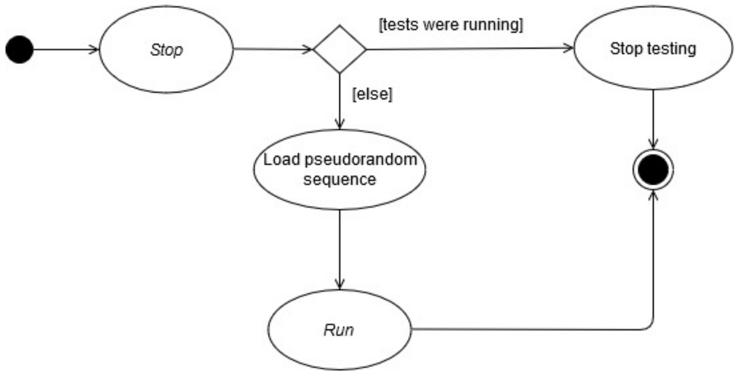

Figure 21: Activity diagram - Stop

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

```
Function StopTests(event):
    if tests_paused then
        return
    end
    if tests_running then
        stop_testing()
    else
        load_sequence()
        run()
    end
```

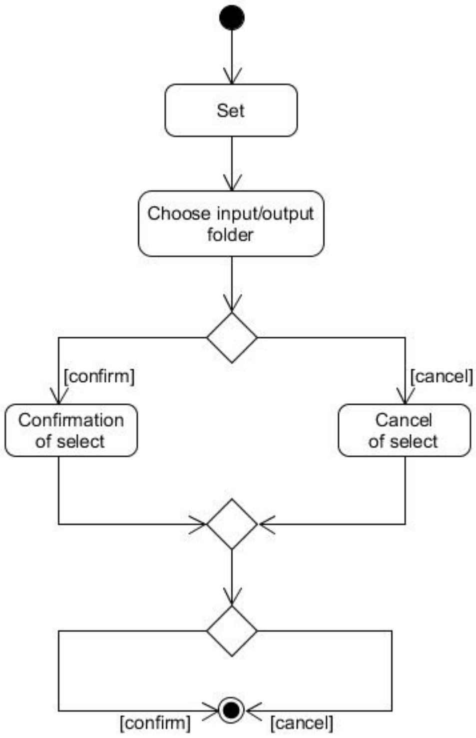

Figure 22: Activity diagram - Set input/output folder

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

Function `SetInputOutputFolder`(*event*)**:**

  $chosenFolder$ <- choose_folder();

  if ok **then**

    $sourceDirectory$ < - $chosenFolder$;

  end

  if $cancel$ **then**

  end

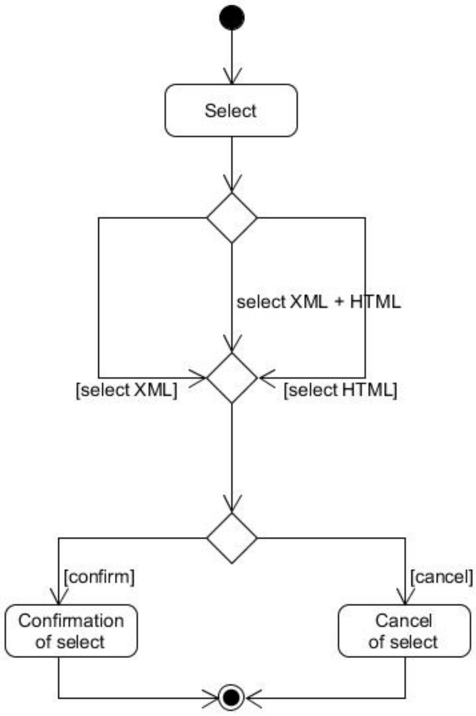

Figure 23: Activity diagram - Select output option

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

Function `SelectOutputOption`($event$):
    if $XMLselected$ **then**
        $outputOption <- XML;$
    end
    if $HTMLselected$ **then**
        $outputOption <- HTML;$
    else
        $outputOption <- XMLHTML;$
    end

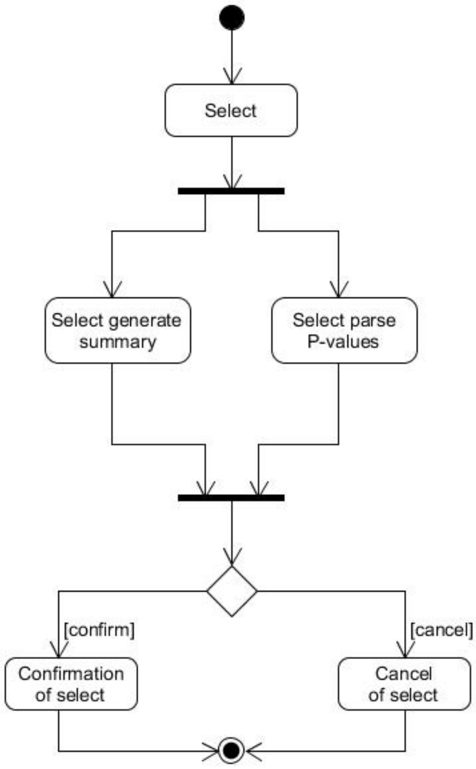

Figure 24: Activity diagram - Select summary option

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

```
Function SelectSummary(event):
    if isClickedGeneratet then
        generateSum < - true;
    else
        generateSum < - false;
    end
    if isClickedParse then
        parsePvalues < - true;
    else
        parsePvalues < - false;
    end
```

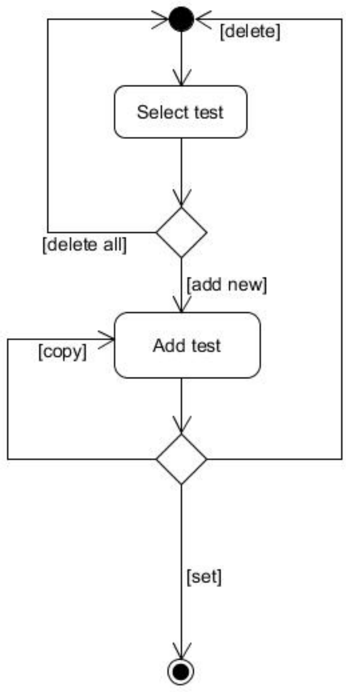

Figure 25: Activity diagram - Set parameters for selected tests

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

```
Function SetParameters(event):
    select_test();
    if add then
        add_test();
        if copy then
            copy_test();
        end
        if delete then
            delete_test();
        end
    end
    if deleteAll then
        delete_all_tests();
    end
```

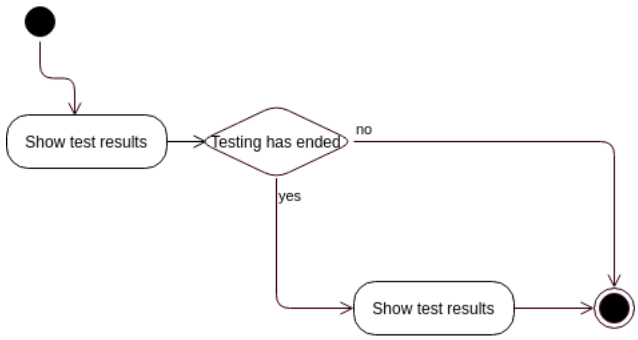

Figure 26: Activity diagram - Show test results

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

```
Function ShowTestResults(event):
    if testing_has_ended then
        show_test_results()
    end
```

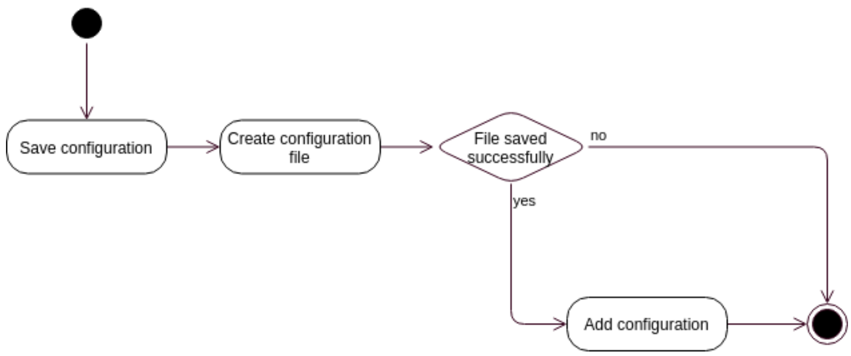

Figure 27: Activity diagram - Save configuration

Algorithm 10: Save configuration, triggered by selecting the option

```
Function SaveConfiguration(event):
    file <- create_configuration_file()
    if file then
        add_configuration()
        save_configuration()
    end
```

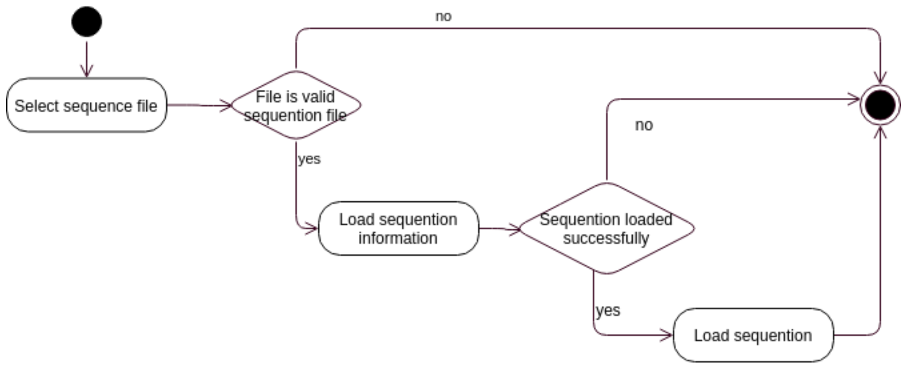

Figure 28: Activity diagram - Load sequence

Algorithm 11: Load sequence, triggered by selecting the option

```
Function LoadSequence(file):
    if is_valid_sequention_file(file) then
        sequention <- import_sequention(file)
        if sequention then
            load_sequention(sequention)
        end
    end
```

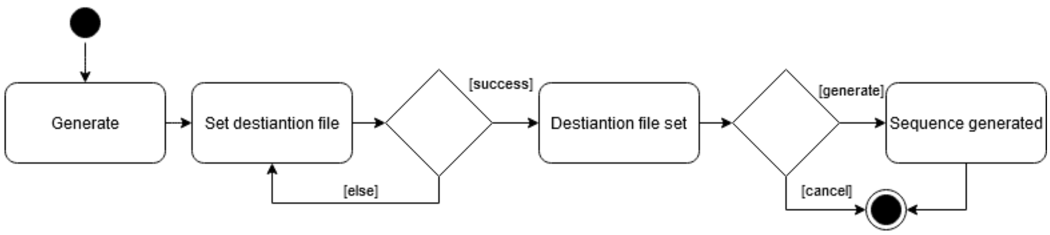

Figure 29: Activity diagram - Generate sequence/Set destination file

Algorithm 12: Generate sequence into file.

```
Function Generate(event):
    destinationFile_select();
    if success then
        destinationFile_set() ;
    else
        destinationFile_select() ;
    end
    generate();
```

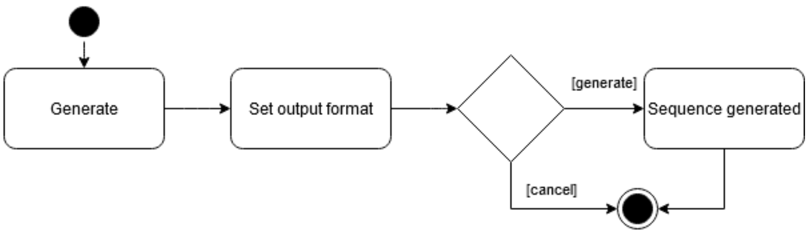

Figure 30: Activity diagram - Generate sequence/Set output format

Algorithm 13: Generate sequence into file.

```
Function Generate(event):
    OutputFormat_set();
    generate();
```

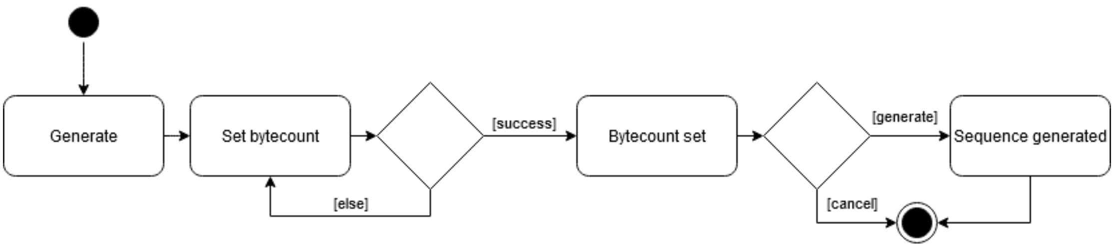

Figure 31: Activity diagram - Generate sequence/Set bytecount

Algorithm 14: Generate sequence into file.

```
Function Generate(event):
    byteCount_select();
    if success then
        byteCount_set() ;
    else
        byteCount_select() ;
    end
    generate();
```

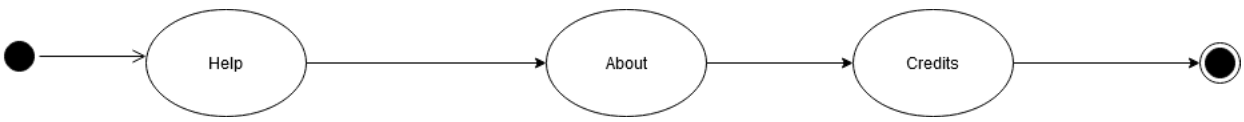

Figure 32: Activity diagram - Help

Algorithm 15: Open help for inforamtion

```
Function Help(event):
    help();
    About_click();
    Credits.show();
```

3 Acceptance tests

This section includes tables which describes acceptance 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	Ouptut 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

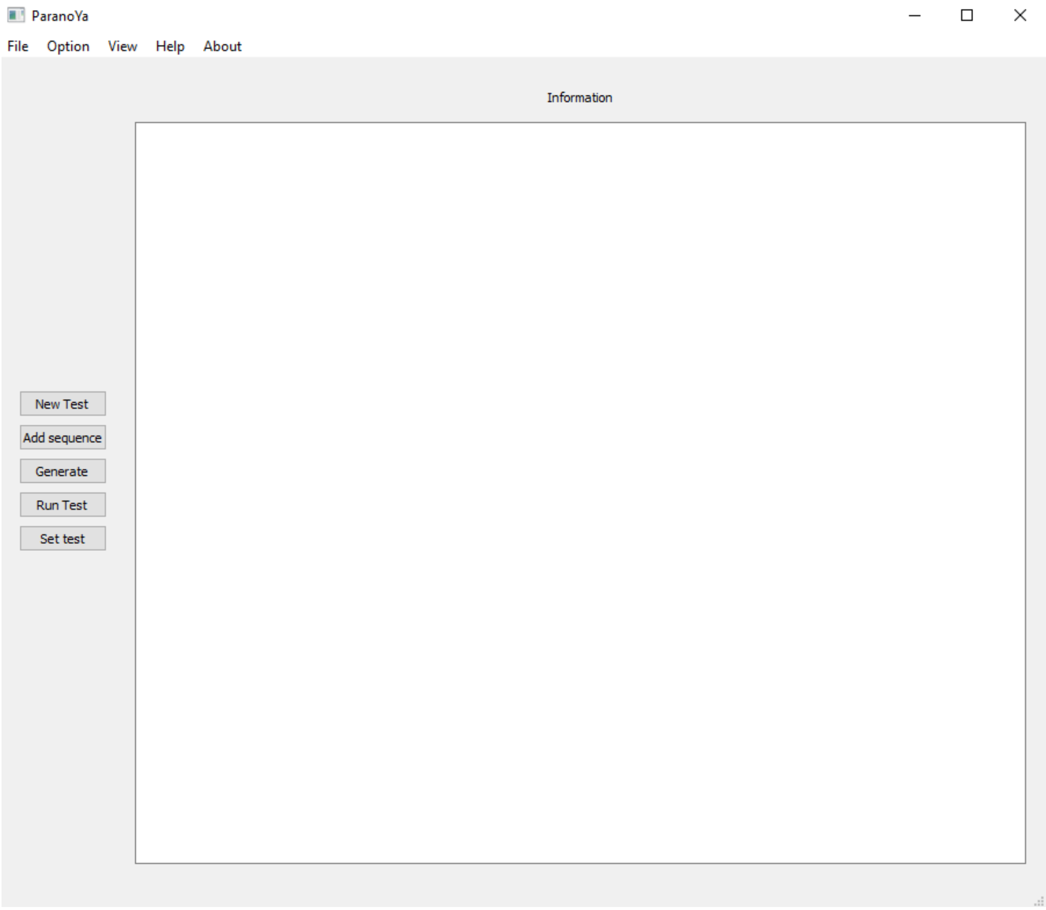

Figure 33: Main menu

Purpose:	Every function of the application is available from this window: • **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

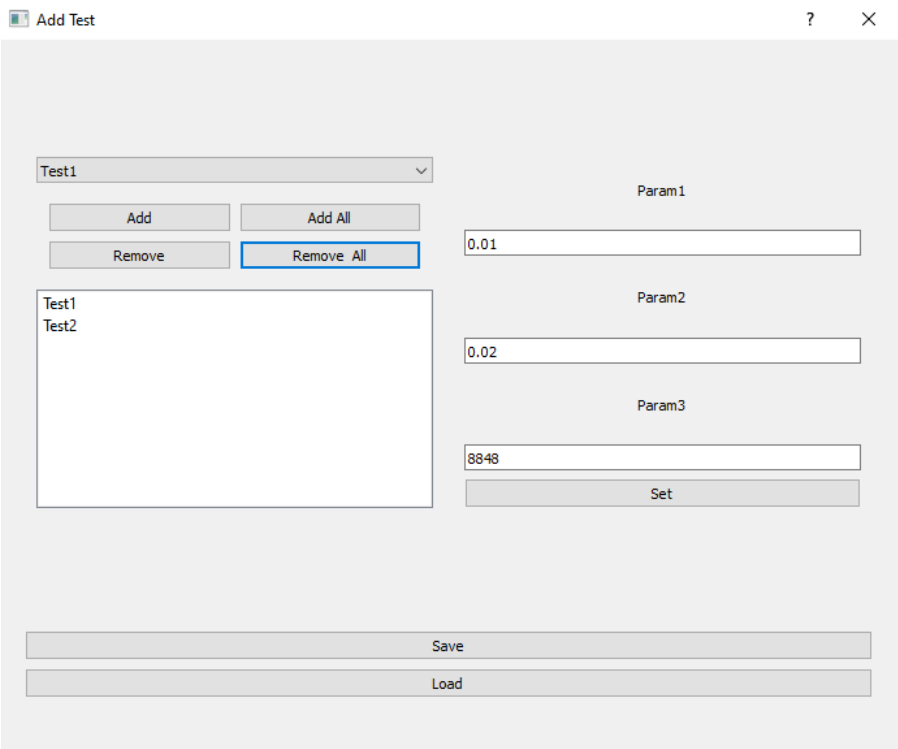

Figure 34: New Test

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:	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. Test parameters can be set on the right handside of the window and by clicking the **Set** button the changes will be saved. A selected test can be removed by clicking the **Remove** button. By clicking the **Add All** button the user is able to easily add all available tests this set. By clicking the **Remove All** button the user can remove all tests from the set. By clicking the button **Save** the user is able to save the current set of tests. By clicking the button **Load** the user is able load a previously save set of tests.

4.5 Add sequence

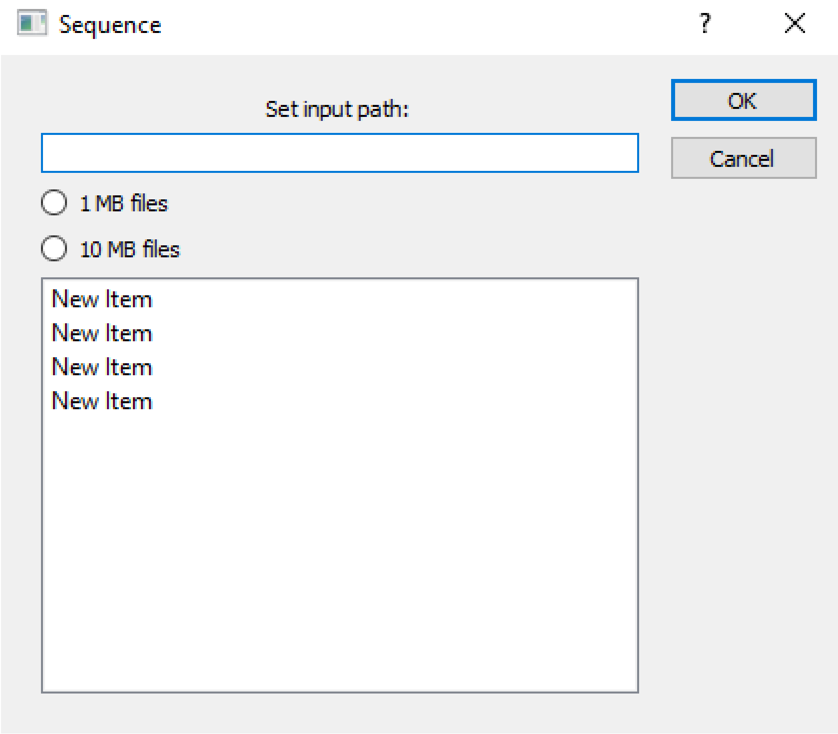

Figure 35: Add sequence

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

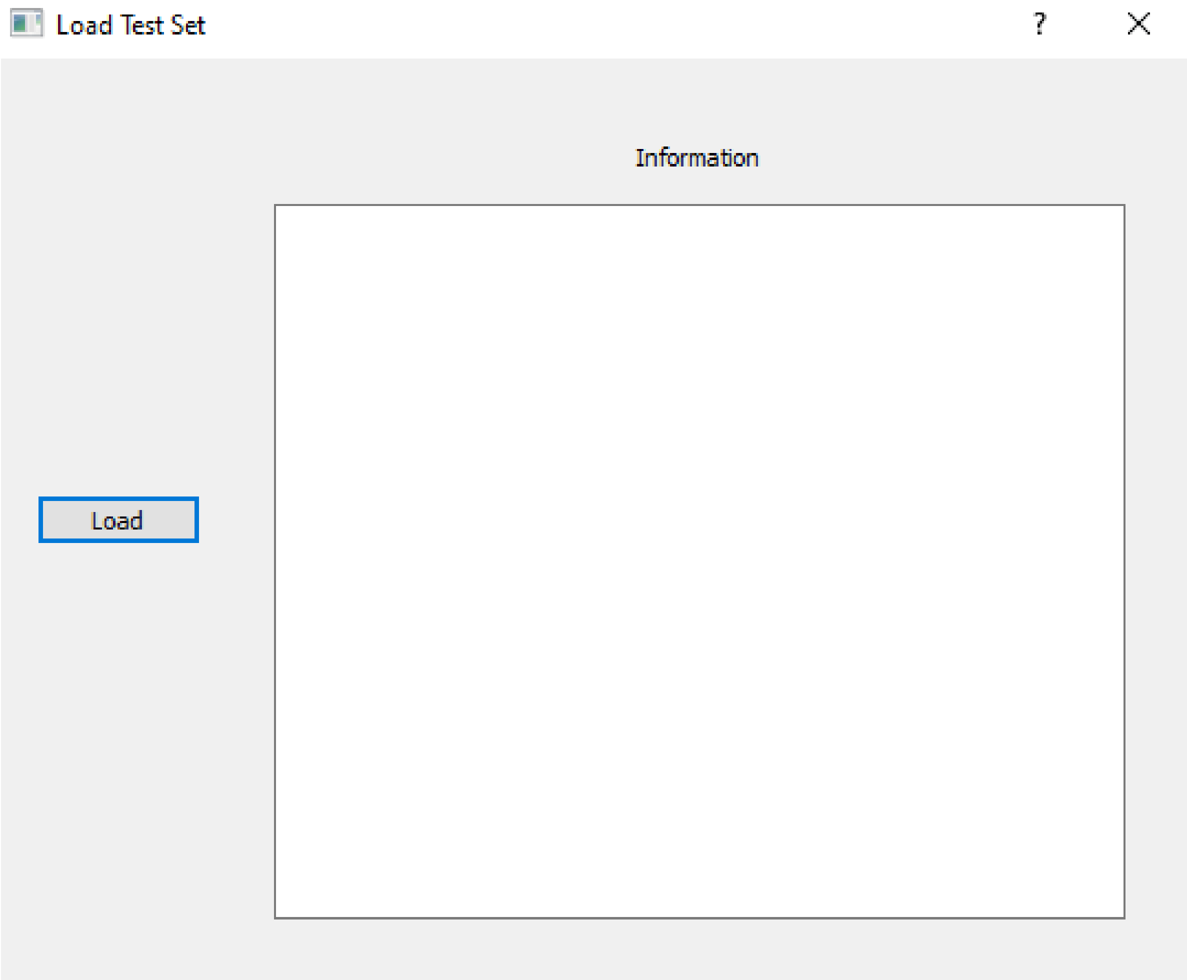

Figure 36: Set test

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

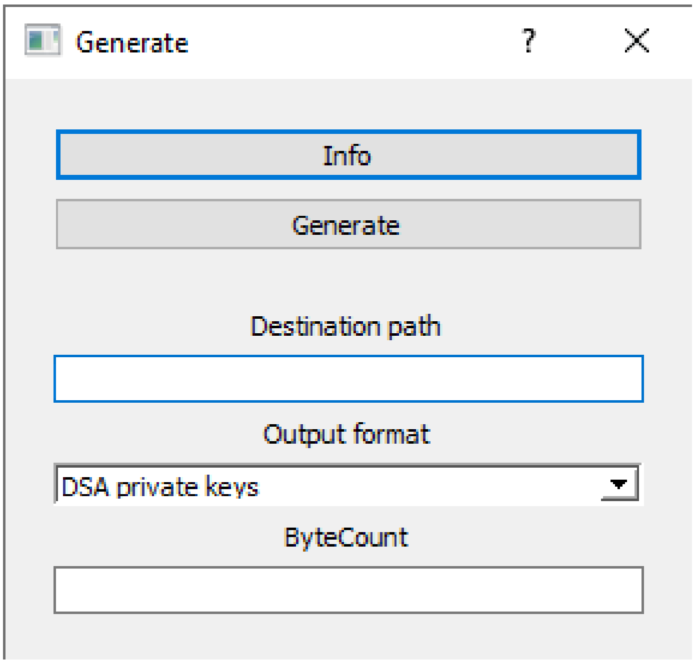

Figure 37: Generate

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

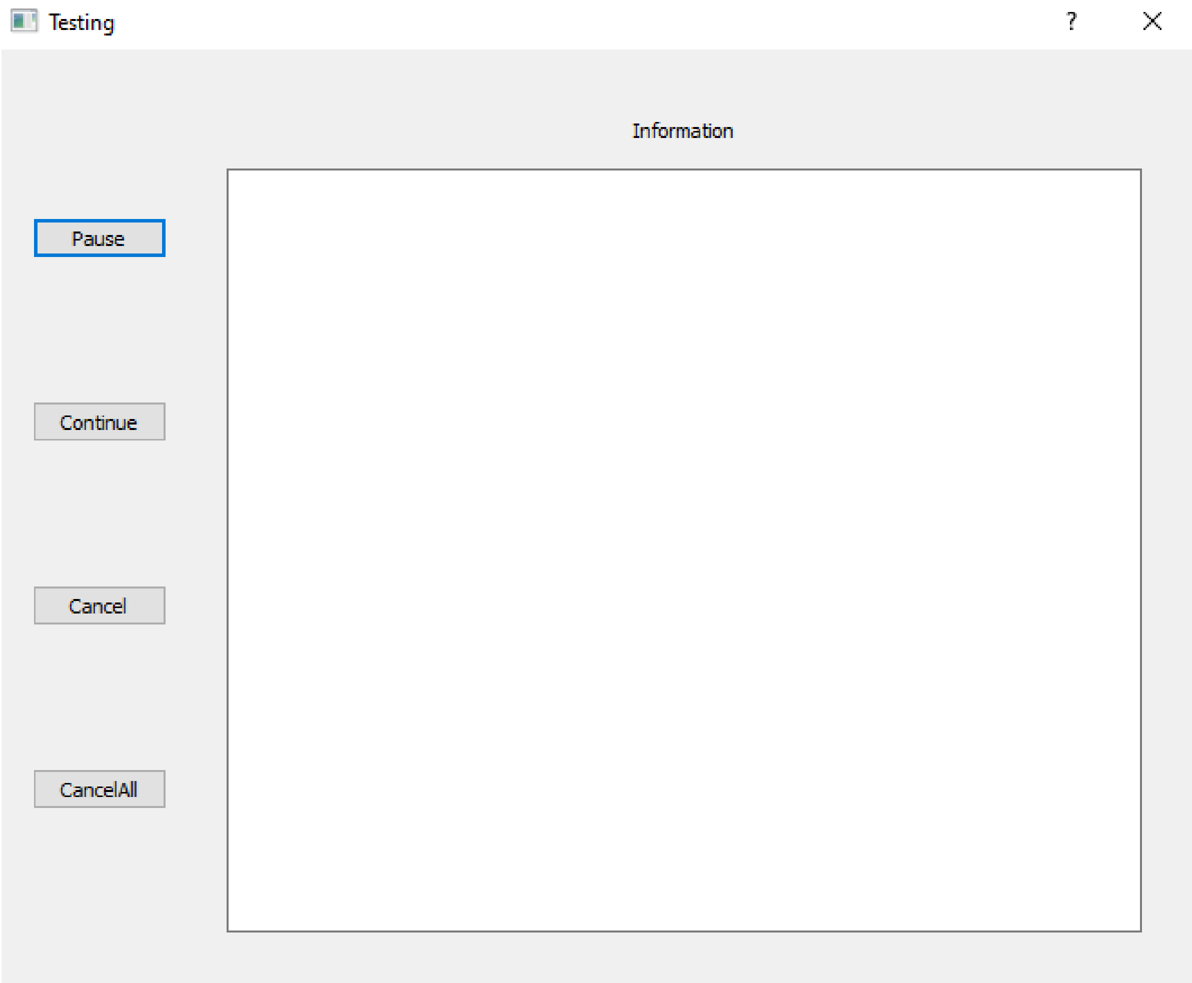

Figure 38: Run Test

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

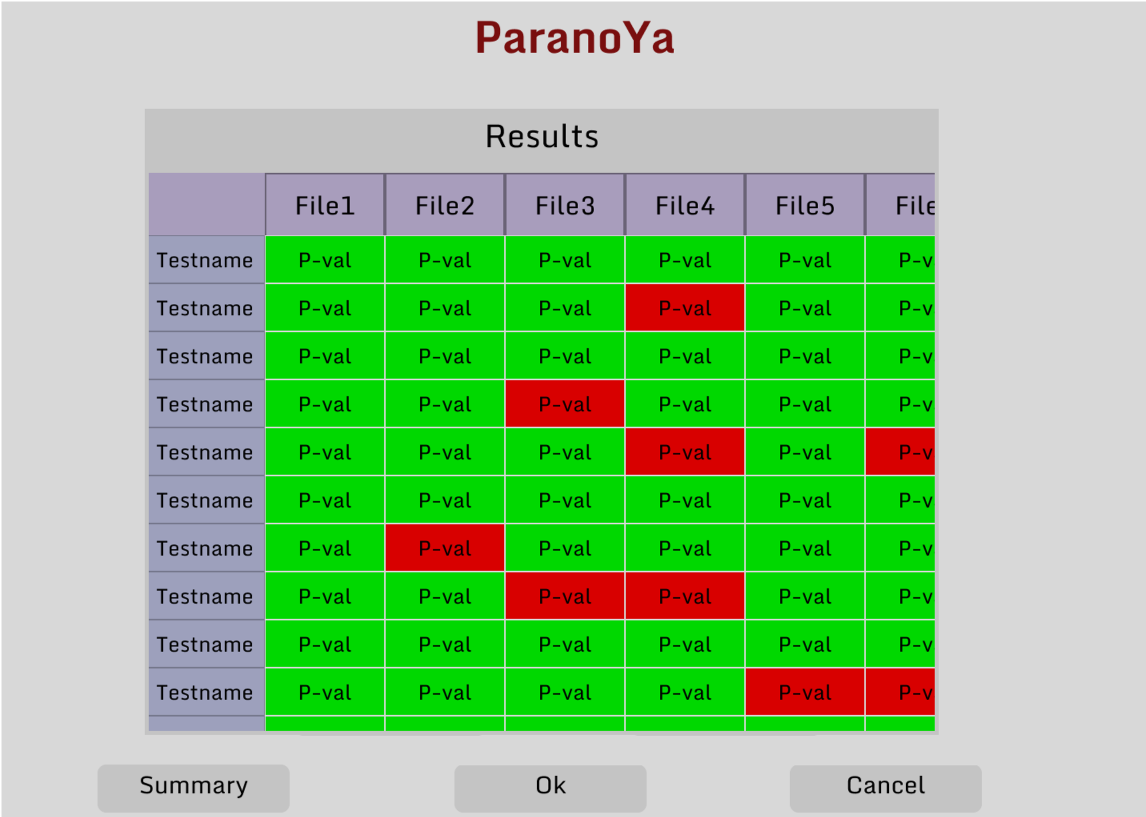

Figure 39: Results

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.

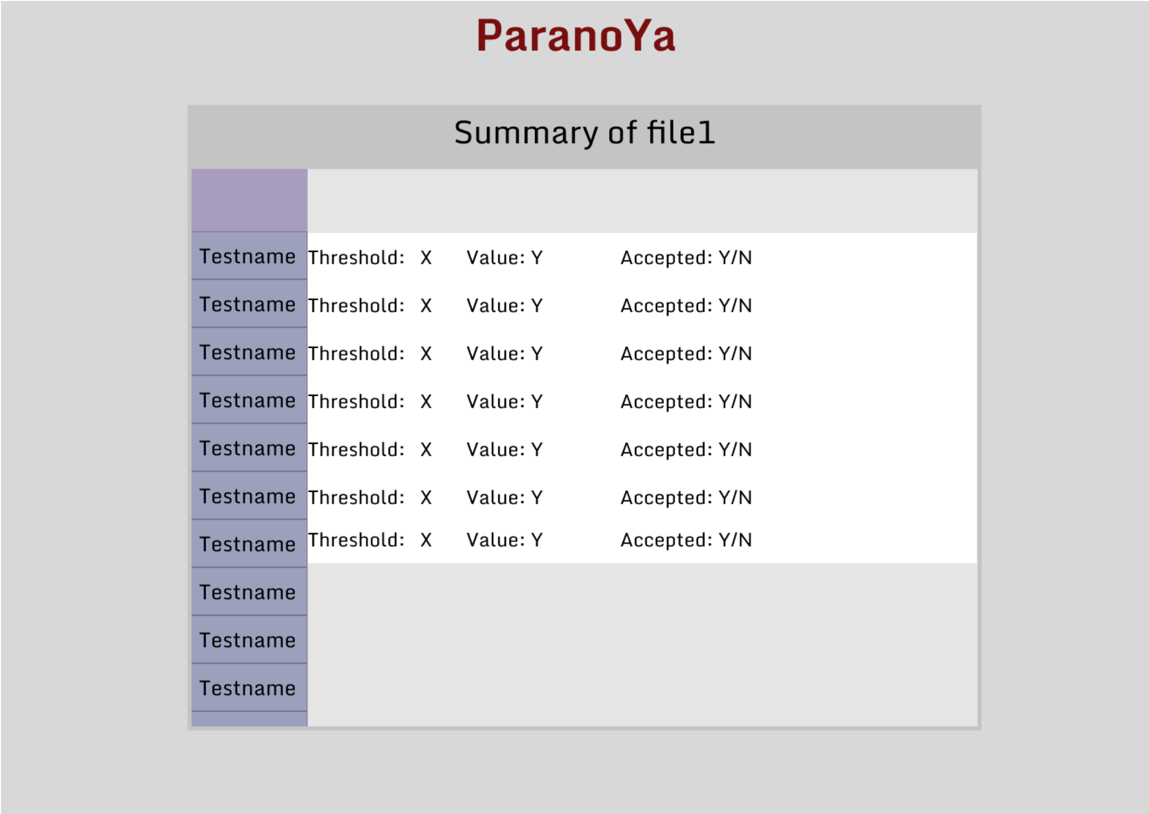

Figure 40: Detailed results

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 and libraries which we used for development and detailed process of implementation.

5.1 Test set

After analysis of the implemented testing tools online we decided to use NIST test set library developed by GINARTeam in Python in the github repository

```
https://github.com/GINARTeam/NIST-statistical-test
```

in the application. The main reason we chose this test set is that our application is also written in Python.
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

After analysis of requirements for our application we decided that the application language will be Python. Python has a great amount of testing libraries and UI tools which are both needed in our application. We started to develop the application following the principles of objected oriented programming.
Application consists of 4 folders. App, Gui, Tests and Results. The App folder is used to manage testing and connect application logic and GUI. Gui folder contains PyQt GUI files as well as it contains python files generated by those GUI files. These python files are representing the current GUI of the application. Tests folder consists of tests files and files necessary for test completion. Results folder is used to output test results into.

```
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. The application loads all of the tests inside Test folder and shows them in the dropdown menu. The chosen tests indices are represented by an array (list in Python). Whenever a test is added or removed, an index is added or removed from the array. By saving the newly created test set, the array of indices is added into the main window property TestSets.

Button Add sequence opens a new window which handles loading byte sequence file which is then being tested. The shown window contains a file input. By pushing the button, all the files with sequences are saved into main window property files. 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. By clicking Generate button a new file created pops up and user can choose the destination path.

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 diplayed 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. User is displayed all the saved test sets and by selecting one and pushing the Load Button the chosen test set is set to the main window property LoadedTestSet

5.3 Tests

The NIST test set contains a total of 15 tests. 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.

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. P-value is calculated and then compared with critical value. If P-value is greater or equal than critical value, hypothesis is accepted. If P-value is smaller than the critical value, hypothesis is rejected.

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.

Test results are being output into *.csv* files. If multiple files are being tested at the time, the result file is named "result-[index of the file]-[name of the test]". The files have specific form of output for each different test.

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. WALKER, John. *Ent - A Pseudorandom Number Sequence Test Program* [online]. 2008 [visited on 2020-06-16]. Available from: `https://www.fourmilab.ch/random/`.
2. BROWN, Robert G. *Dieharder: A Random Number Test Suite* [online]. Duke University Physics Department [visited on 2020-06-16]. Available from: `https://webhome.phy.duke.edu/~rgb/General/dieharder.php`.
3. *Practically Random* [online]. 2010 [visited on 2020-06-16]. Available from: `http://pracrand.sourceforge.net/`.
4. L'ECUYER, Pierre. *TestU01* [online]. Université de Montréal, 2009 [visited on 2020-06-16]. Available from: `http://simul.iro.umontreal.ca/testu01/tu01.html`.
5. III, Lawrence E Bassham (comp.). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* [online]. NIST Special Publication 800-22 Revision 1a. United States of America: National Institute of Standards and Technology [visited on 2019-10-25]. Available from: `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf`.
6. RUKHIN, Andrew (comp.). *A STATISTICAL TEST SUITE FOR RANDOM AND PSEUDORANDOM NUMBER GENERATORS FOR CRYPTOGRAPHIC APPLICATIONS: NIST Special Publication 800-22 (with revisions dated May 15, 2001)* [online]. NIST Special Publication 800-22. National Institute of Standards and Technology, 2001 [visited on 2019-10-25]. Available from: `https://www.random.org/statistics/SP800-22b.pdf`.
7. SCARFONE, Karen (comp.). *Technical Guide to Information Security Testing and Assessment: Recommendations of the National Institute of Standards and Technology* [online]. NIST Special Publication 800-115. Gaithersburg: National Institute of Standards and Technology [visited on 2019-10-25]. Available from: `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf`.
8. RUKHIN, Andrew L. (comp.). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* [online]. Special Publication (NIST SP) - 800-22. National Institute of Standards and Technology, 2001 [visited on 2019-10-25]. Available from: `https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic-0`.

9. *FIPS 140-2 and SED Technology: Frequently Asked Questions* [online]. Seagate [visited on 2019-10-26]. Available from: https://www.seagate.com/files/www-content/solutions-content/security-and-encryption/en-us/docs/faq-fips-sed-mb605-2-1302us.pdf.
10. *Compliance FAQs: Federal Information Processing Standards (FIPS)* [online]. National Institute of Standards and Technology, 2018 [visited on 2019-10-26]. Available from: https://www.nist.gov/standardsgov/compliance-faqs-federal-information-processing-standards-fips.
11. *What is FIPS 140-2?* [online]. THALES [visited on 2019-10-26]. Available from: https://cpl.thalesgroup.com/faq/key-secrets-management/what-fips-140-2.
12. LEWIS, Nate. What You Need To Know About FIPS 140-2 Validation. 2018, vol. 28, no. 9. Available also from: https://blog.ipswitch.com/what-you-need-to-know-about-fips-140-2-validation.
13. *SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES: Federal Information Processing Standards Publication 140-2* [online]. FIPS PUB 140-2. Gaithersburg: National Institute of Standards and Technology, 2001 [visited on 2019-10-26]. Available from: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf.
14. *How the randomness tests work* [online]. PortSwigger [visited on 2019-10-26]. Available from: https://portswigger.net/burp/documentation/desktop/tools/sequencer/tests.
15. KIM, Song-Ju. IP Core of Statistical Test Suite of FIPS 140-2. 2003, no. 1. Available also from: https://www.researchgate.net/publication/294760536_IP_Core_of_Statistical_Test_Suite_of_FIPS_140-2.
16. ANDERSON, Walter (comp.). *Dieharder test descriptions: Original Die Hard tests* [online]. A study of entropy [visited on 2019-10-27]. Available from: https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions.
17. *Diehard tests* [online]. Wikipedia [visited on 2019-10-27]. Available from: https://en.wikipedia.org/wiki/Diehard_tests.

18. AHMED AL-MASRI, Kevin Curran. In: AHMED AL-MASRI, Kevin Curran. *Smart Technologies and Innovation for a Sustainable Future: Proceedings of the 1st American University in the Emirates International Research Conference — Dubai, UAE 2017*. Switzerland: Springer International Publishing, 2019, pp. 222–223. Advances in Science, Technology & Innovation. ISBN 9783030016593. Available also from: `https://books.google.sk/books?id=MueCDwAAQBAJ`.

19. AHMED AL-MASRI, Kevin Curran. *Smart Technologies and Innovation for a Sustainable Future: Proceedings of the 1st American University in the Emirates International Research Conference — Dubai, UAE 2017*. Switzerland: Springer International Publishing, 2019. Advances in Science, Technology & Innovation. ISBN 9783030016593. Available also from: `https://books.google.sk/books?id=MueCDwAAQBAJ`.

Appendix

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.

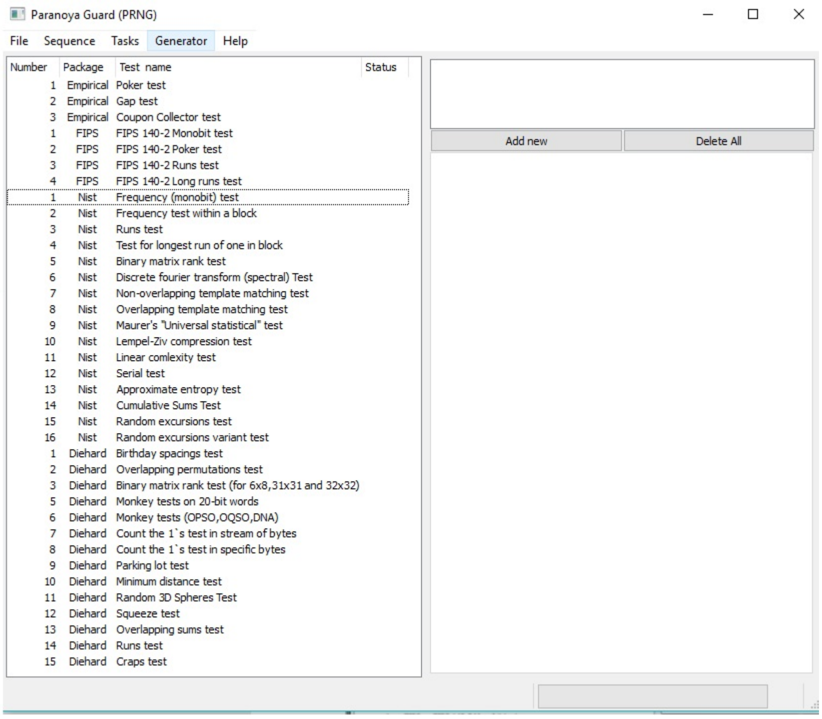

Figure A.1: ParanoYa - Main window

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.

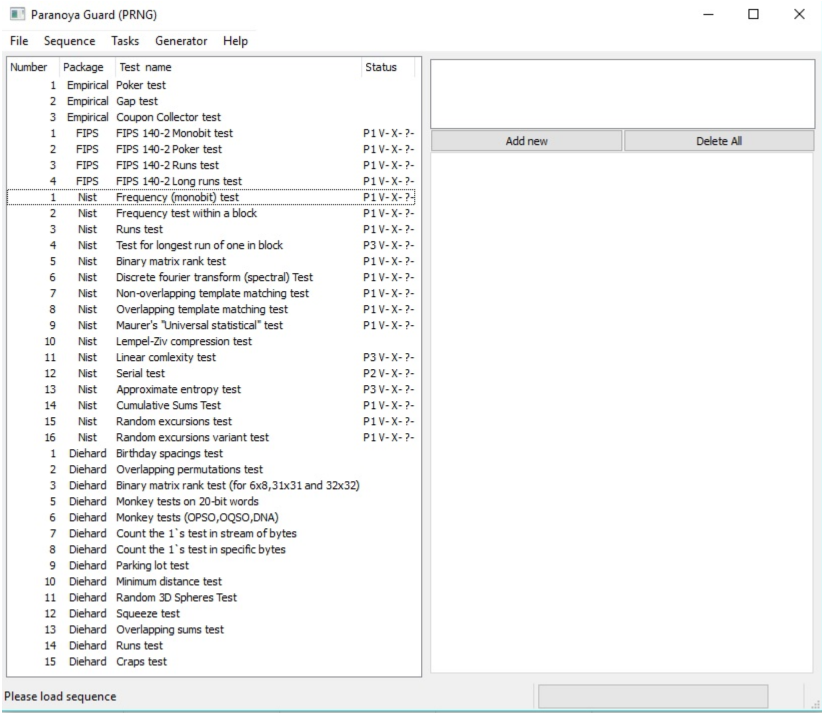

Figure A.2: ParanoYa - Tests are set

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

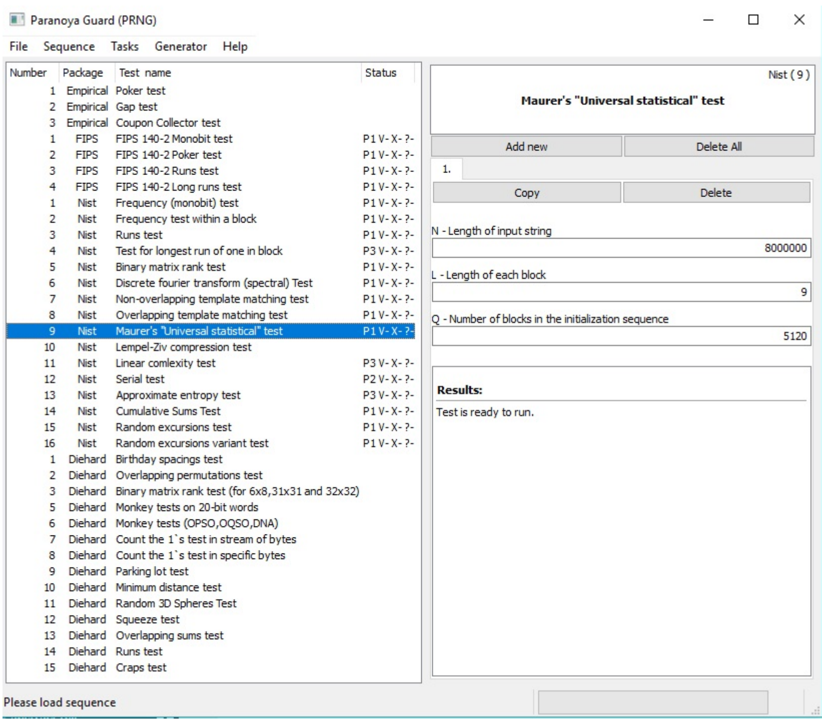

Figure A.3: ParanoYa - Additional modification of parameters

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.

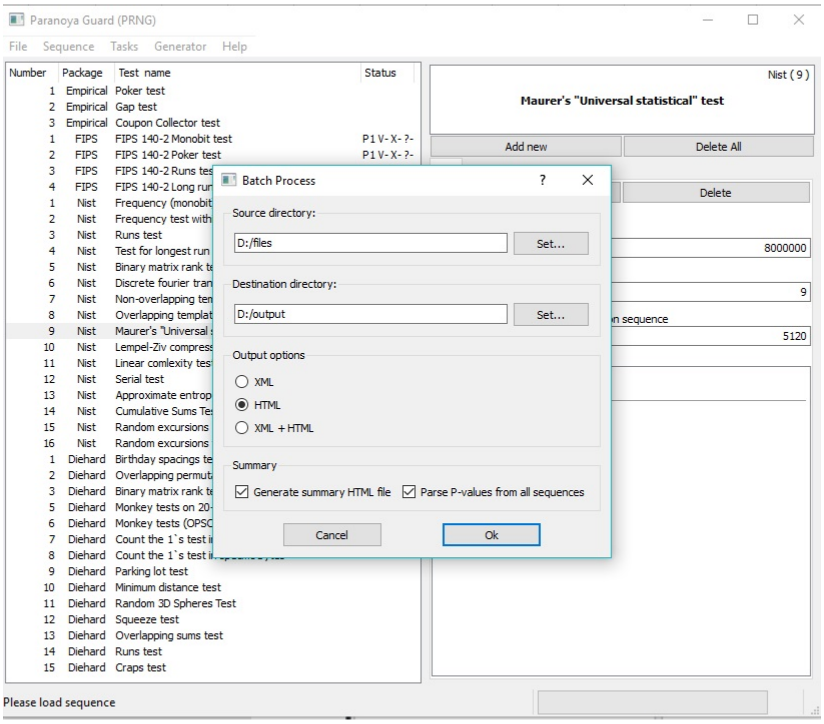

Figure A.4: ParanoYa - Set the output options

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

Figure A.5: ParanoYa - pval.html file

B Remote repository structure

Remote repository can be found on the following link:
https://github.com/matusjokay/paranoYa

\documentation\ParanoYa.pdf

project documentation

\src\app

application business logic

\src\results

results of statistical tests

\src\gui\generated

application GUI

\src\tests\NIST

battery of 15 statistical NIST tests

\.gitignore

\LICENSE

\README.md

application manual

\requirements.txt

required packages for application proper start