ParanoYa

Lóránt Boráros, Filip Budáč, Martin Cehelský, Silvia Holecová

October 2019

1 Analysis

- **The current status of the application**

  Application named ParanoYa is used for statistical testing pseudone 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 frameworku Qt and used test suites are implemented in C.

- **Methods used in evaluate tests**

  porovnanie s odporucanymi/ interval odporucanych (pouzivnie pre ucely kryptografie na predmete) diehard test sa nedaju pouzit 10 MB

- **Used test sets**

  Testovanie je process vykonávania jedného alebo viacerých testovacích prípadov na základe stanovených podmienok, počas ktorého porovnávame aktuálne a očakávané správanie. V aplikácií sú implementované rôzne sady testov, medzi ktorými sú NIST, FIPS a Diehard.

  1. **NIST**
     NIST je štatistický balík testov, ktorý slúži na testovanie náhodnosti lubovoľne dlhých binárnych postupností vytvorených pomocou generátora náhodných alebo pseudonáhodných postupností. Tento balík pozostáva z 15 nasledujúcich testov:

     (a) **The Frequency (Monobit) Test**
     Cieľom tohto testu je zistiť, či pomer núl a jednotiek v danej postupnosti zodpovedá očakávanemu pomeru pre náhodnú postupnosť. Počet jednotiek a núl v postupnosti by mal byť približne rovnaký, čo tiež skúma daný test.

(b) **Frequency Test within a Block**
Tento test posudzuje pomer núl a jednotiek v M-bitových blokoch. Cieľom testu je určiť, či je ich frekvencia M-bitový blok približne M/2.

(c) **The Runs Test**
V tomto teste ide o celkový počet núl a jednotiek runov v celej postupnosti, kde run predstavuje nepretržitú sekvenciu rovnakých bitov.Run o dĺžke k znamená, že pozostáva z k identických bitov a je ohraničený pred a za s bitom, ktorý má opačnú hodnotu. Cieľom tohto testu je zistiť, či počet runov jednotiek a núl rozličnej dĺžky je taký ako sa očakáva pre náhodné postupnosti.Tento test sa používa najmä na posúdenie či je menlivosť medzi takýmito substringmi je príliš pomalá alebo príliš rýchla.

(d) **Tests for the Longest-Run-of-Ones in a Block**
Tento test sa zameriava na najdlhší run jednotiek v rámci M-bitových blokov.Jeho cieľom je zistiť, či dĺžka najdlhšieho runu jednotiek v testovanej postupnosti je konzistentná s dĺžkou najdlhšieho runu jednotiek, ktorá je očakávaná v náhodných postupnostiach. Nepravidelnosť v očakávanej dĺžke najdlhšieho runu jednotiek znamená, že existuje nepravidelnosť v očakávanej dĺžke najdlhšieho runu núl. Dlhé runy núl sa nevyhodnocujú separátne z dôvodu obáv zo štatistickej nezávislosti medzi testami.

(e) **The Binary Matrix Rank Test**
Test je zameraný na nesúvislé poradie podmatíc v celej postupnosti.Účel tohto testu je skontrolovať lineárnu závislosť u pevnej dĺžky podreťazcov pôvodnej postupnosti.

(f) **The Discrete Fourier Transform (Spectral) Test**
Ťažiskom tohto testu sú výšky vrcholov vo Fourierovej transformácii. Cieľom tohto testu je odhaliť periodické funkcie(napríklad opakujúce sa vzory, ktoré sa nachádzajú blízko seba) v testovanej postupnosti, ktoré by naznačovali odchýlku od predpokladu náhodnosti.

(g) **The Non-overlapping Template Matching Test**
Náhodná číselná sekvencia je rozdelená do nezávislých podreťazcov s dĺžkou M a počet výskytov šablóny B, kttorá predstavuje m-bitový run jednotiek v každom z podreťazcov.IfP-hodnota chi-kvadrátovej štatistiky je menšia ako úroveň signifikantnosti, test dospieva k záveru, že testovaná sekvencia sa javí ako náhodná. V opačnom prípade sa pri teste dospelo k záveru, že opakovaný test sa zdá byť náhodný. Priepustnosť je definovaná pomerom sekvencií, ktoré prešli testom.

(h) **The Overlapping Template Matching Test**
Tento test zisťuje počet výskytov vo vopred špecifikovaných cieľových reťazcoch. Test používa m-bitové okno na hľadanie špecifického m-bitového vzoru. Pokiaľ vzor nie je nájdený, okno sa posúva o

jednu bitovú pozíciu.Ak sa hľadaný vzor nájde, okno sa posúva len jeden bit pred obnovením hľadania.

(i) **Maurer's "Universal Statistical" Test**
Účelom tohto testu je odhaliť, či postupnosť môže byť výrazne komprimovaná bez straty informácií alebo nie. Príliš komprimovaná postupnosť sa považuje za nenáhodnú.

(j) **The Linear Complexity Test**
Cieľom tohto testu je určiť, či je sekvencia dostatočne zložitá na to, aby sa považovala za náhodnú.

(k) **The Serial Test**
Účelom tohto testu je zistiť, či je počet výskytov prekrývajúcich sa vzorov m-bitov približne rovnaký, ako by sa očakávalo v prípade náhodnej sekvencie.

(l) **The Approximate Entropy Test**
Test sa zameriava na frekvenciu všetkých možných prekrývaní m-bitových vzorov v celej sekvencii. Účelom tohto testu je porovnať frekvenciu prekrývajúcich sa blokov dvoch po sebe nasledujúcich alebo susediacich dĺžok (m a m+1) s očakávaným výsledkom pre náhodnú postupnosť.

(m) **The Cumulative Sums (Cusums) Test**
Tento test sa zameriava na maximálnu odchýlku (od nuly) náhodnej prechadzky(walk?????) definovanej kumulatívnym súčtom upravených (-1, +1) číslic v postupnosti. Cieľom testu je určiť, či kumulatívny súčet čiastkových sekvencií vyskytujúcich sa v testovanej sekvencii je príliš veľký alebo príliš malý relatívny k očakávanemu správaniu tejto kumulatívnej sumy pre náhodné sekvencie. Táto kumulatívna suma, môže byť považovaná za náhodnu walk. Pre náhodnú sekvenciu by mala byť odchýlka náhodnej walk blízko nuly. Pre určité typy náhodných sekvencií budú odchýlky tejto náhodnej walk väčšie od nuly.

(n) **The Random Excursions Test**
Test je zameraný na počet cyklov, ktoré majú presne K výskytov v kumulatívnom súčte náhodných krokov. Kumulatívny súčet môžeme zistiť ak sú čiastkové súčty (0, 1) sekvencie upravené na (-1, +1). Náhodná odchýlka náhodných krokov pozostáva zo sekvencie n krokov jednotkovej dĺžky. Cieľom testu je zistiť, či počet výskytov stavu s náhodnými krokmi prekračuje to, čo sa očakáva od náhodnej sekvencie.

(o) **The Random Excursions Variant Test**
Tento test skúma koľkokrát sa konkrétny stav vyskytne v kumulatívnom súčte náhodných krokov. Cieľom je odhaliť odchýlky od očakávaného počtu výskytov rôznych stavov v náhodných krokoch.

Tieto testy sa zaoberajú rôznymi typmi náhodností, ktoré by mohli

vzniknúť v postupnosti. Niektoré z testov by bolo možné rozložiť na rôzne subtesty. Poradie spustenia jednotlivých testov je lubovoľné, ale odporúča sa, aby bol Frequency test spustený ako prvý, pretože pokiaľ tento test zlyhá, pravdepodobnosť zlyhania ďalších testov je veľmi vysoká.

2. **FIPS** nist sp-822,fips 140-2 Test Federal Information Processing Standard predstavuje americký vládny bezpečnostný štandard, ktorý sa používa na shcválenie použitých kryptografických modulov. FIPS poskytuje rôzne druhy zabezpečenia a to na základe definovanej úrovne bezpečnosti. Takéto úrovne poznáme štyri:
   (a) **Úroveň 1** - najnižšia bezpečnostná úroveň, ktorá nevyžaduje špecifické mechanizmy fyzickej bezpečnosti, ale vyžaduje použitie aspoň jedného schváleného bezpečnostného algoritmu alebo funkcie
   (b) **Úroveň 2** - táto úroveň vyžaduje riadenie prístupu na základe rolí, vyžaduje tiež fyzické zabezpečenie
   (c) **Úroveň 3** - v tejto úrovni je poskytnutá autentifikácia založená na identite a fyzické zabezpečenie, mala by obsahovať mechanizmus na detekciu útoku a pokiaľ dôjde k hacknutiu systému, systém by mal byť schopný vymazať kritické bezpečnostné parametre
   (d) **Úroveň 4** - ide o najvyššiu úroveň zabezpečenia, okrem vyššie spomenutých náležitostí na systém sa v nej sprísňujú požiadavky fyzického zabezpečenia, je výhodná najmä pre prácu vo fyzicky nechránenom prostredí

   Validácia FIPS zahŕňa intenzívne testovanie na zistenie konkrétnych nedostatkov a slabín. Na to, aby systém spĺňal validáciu na základe FIPS, je potrebné, aby obsahoval kryptografické algoritmy a hashovacie funkcie.K trom najznámejším príkladom patrí AES, Triple DES a HMAC SHA-1

3. **Diehard**

   Diehard tests sú štatistické testy, ktoré slúžia na zhodnotenie miery kvality generátora náhodných čísel. Diehardová batéria testov pozostáva z rôznych, nezávislých štatistických testov. Výsledky týchto testov sa označujú ako p-hodnoty. Medzi Diehardové testy patria:
   (a) **The Birthday spacings test**
   V tomto teste sa najprv vyberá m narodenín v roku s n dňami, následne sa uvádza zoznam medzier medzi narodeninami. A nakoniec sa posudzuje asymptoticky Poissonove rozdelenie hodnoty j. Hodnota j predstavuje počet hodnôt, ktoré sa nachádzajú v uvedenom zozname medzier a pokiaľ sa v tomto zozname nachádza viackrát, tak j je asymptoticky Poissonovo rozdelené s priemerom $m^3/(4n)$. n musí byť dostatočne veľké, na porovnanie výsledkov s Poissonovým rozdelením.

(b) **Overlapping permutations**
Tento test sleduje postupnosť jedného milióna 32-bitových náhodných celých čísel. Každá sada piatich po sebe idúcich celých čísel môže byť v jednom zo 120 stavov pre 5! možných usporiadaní piatich čísel.

(c) **Ranks of matrices**
Tento test sa vykonáva výberom určitého počtu bitov z určitého počtu náhodných čísel, aby sa vytvorila matica nad [0,1] a následne sa určí poradie matice. Počet radov by mal sledovať určité rozdelenie.

(d) **Monkey test**
Tiež nazývaný ako test bitových tokov. Tento test má svoj názov z nekonečnej "monkey theorem". Najlepšie sa dosiahne spracovaním sekvencií určitého počtu bitov ako slov" a spočítaním prekrývajúcich sa slov v pare. Počet slov", ktoré sa neobjavia, by mal nasledovať po známej distribúcii.

(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 x 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.

1.1 Spôsob riešenia

Základ aplikácie, teda sady testov, sú rezprezentované ako knižnice v programovacom jazyku C. Používateľské rozhranie je vo frameworku Qt a výstup aplikácie sa v súčasnosti realizuje zapísaním do súboru typu *.xls*, ktorý používateľ môže čítať v tabuľkových editoroch. Návrh riešenia spočíva vo viacerých krokoch:

1. **Aktualizácia projektu pre súčasné vývojové prostredia**

   Pre vytvorenie používateľského rozhrania sme sa rozhodli použiť jazyk Python. Pomocou knižnice CTypes sme zabezpečili prepojenie s knižnicami v jazyku C. Z knižníc sme vytvorili Shared Object File "*.so*". Tento Shared Object nám umožní dynamicky prepájať knižnicu s rôznymi programmi.

1.2 Funkcionálne požiadavky

- **Štatistické testovanie pseudonáhodných postupností** - používateľ si bude schopný štatisticky otestovať rôzne pseudonáhodné postupnosti pomocou implementovaných sád testov
- **Nastavovanie jednotlivých testov** - používateľovi bude umožnené nastavovať a upravovať jednotlivé testy, podľa jeho kritérií
- **Vyhodnocovanie testovaných postupností** - po testovaní vybraných postupností, si bude môcť používateľ prezrieť vyhodnotenie testov na základe zvolenej metodiky

2 Existujúce programy

1. **Ent** je konzolová aplikácia, ktorá slúži na vyhodnocovanie pseudnáhodných postupností pre štatistické vzorkovacie aplikácie, šifrovacie a kompresné algoritmy. Ent vykonáva nasledujúce testy:

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

Ent poskyutje viacerých možností na modifikovanie spracovania údajov ako aj formát výstupu:

- **-b**
  Vstupné dáta s spracované ako bity namiesto bajtov.
- **-c**
  Na štandarný výstup vytlačí tabuľku vyskytnutých znakov, ktoré sú zobrazené s ich desatinnými bajtovými hodnotami spolu s prislúchajúcim znakom zo sady ISO 8859-1 Latin-1.
- **-f**
  Veľké písmená sú zmenené na malé pred vykonaním testov.
- **-t**
  Výstup je vo forme tzv. *terse mode*, v ktorom výstupy sú oddelené čiarkou(CSV formát).
- **-u**
  Výpis informácií

2. **Dieharder** je vylepšená verzia programu *Diehard battery of tests* s vyčisteným zdrojovým kódom, implementované v jazyku C. Najdôležitejšie vylepšenia sú jeho rýchlosť a vďaka architektúry rozšíriteľnosť sady testov. Je open source projektom, program je voľne dostupný a stiahnuteľný na jeho web-stránke. Dieharder umožňuje otestovanie generátorov priamo, vstupom môže byť aj neobmedzený tok náhodných čísel.

3. **Practically random** je knižnica implementované v programovacom jazyku C++ a slúži na testovanie generátorov náhodných postupností-*RNG*

4. **TestU01** je knižnica implementované v programovacom jazyku ANSI C. Obsahuje funkcie na empirické štatistické testovanie generátorov náhodných postupností. Aplikácia

```
#==============================================================================#
#          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: Sada testov Dieharder

3 GUI-Grafické rozhranie

3.1 Vývojové prostredie Figma

3.2 Opening window

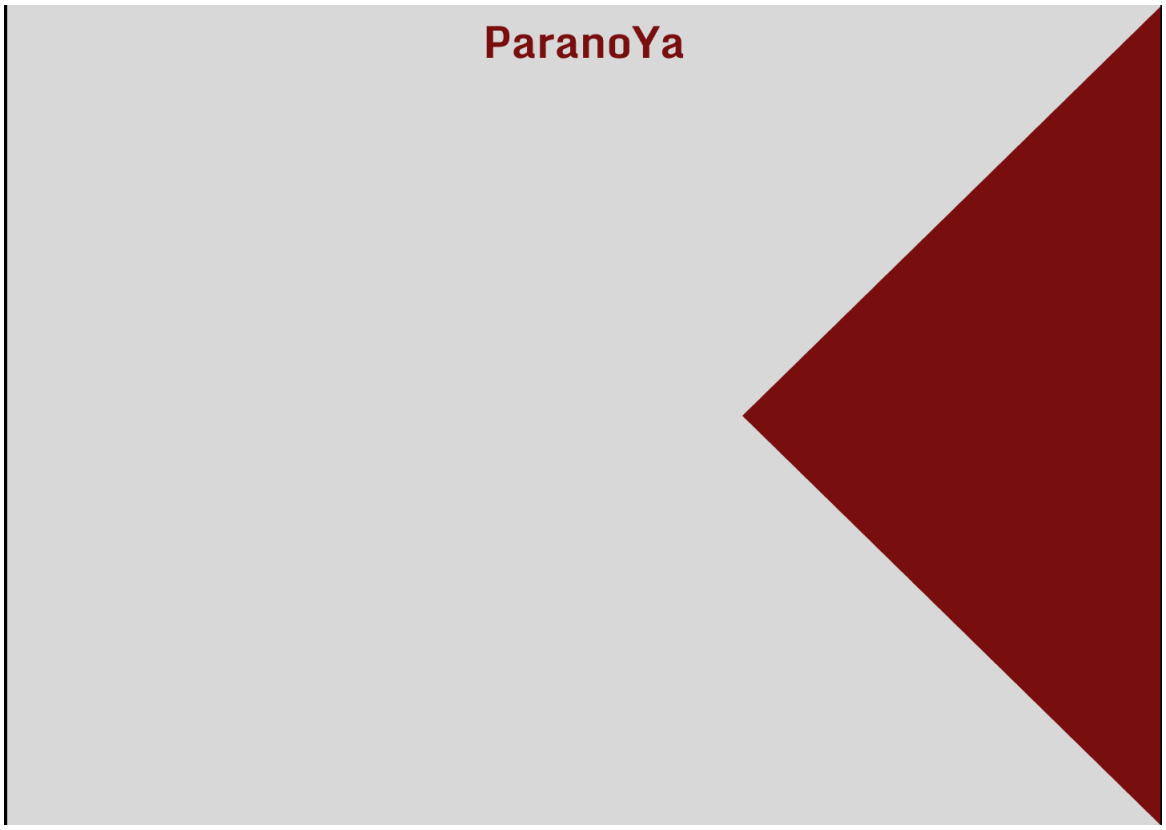

Figure 2: Opening window

Purpose:	Po spustení aplikácie užívateľ je uvítaný týmto oknom.
Navigation and User Interaction:	Po inicializácií aplikácia bude pripravený na používanie, program automaticky presmeruje na nasledujúce okno.

3.3 Main menu

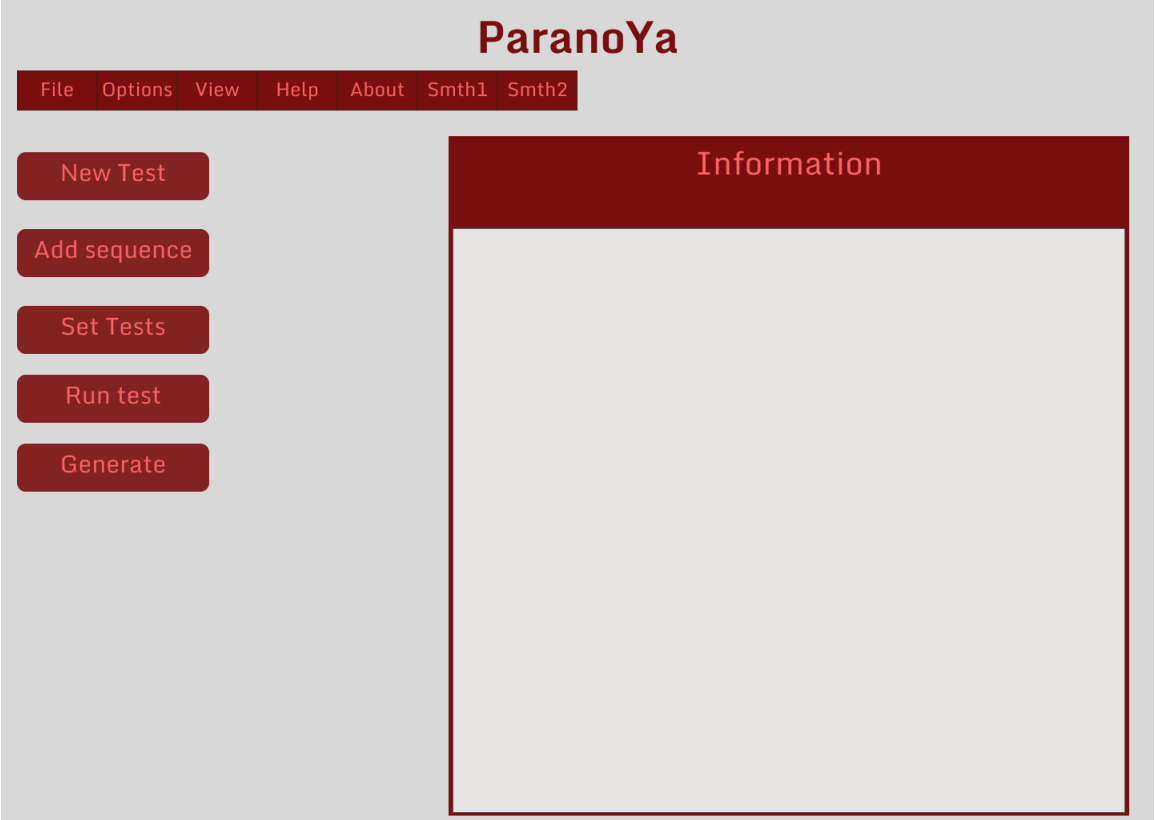

Figure 3: Main menu

Purpose:	Z tohto okna sú dostupné všetky funkcionality programu: • **Nastavenie testov** • **Pridanie postupností** • **Generovanie postupností** • **Spustenie testov**
Navigation and User Interaction:	Užívateľ tlačítkom zvolí akciu, ktorú chce vykonať. Aplikácia naviguje na príslušné okno.

3.4 New Test

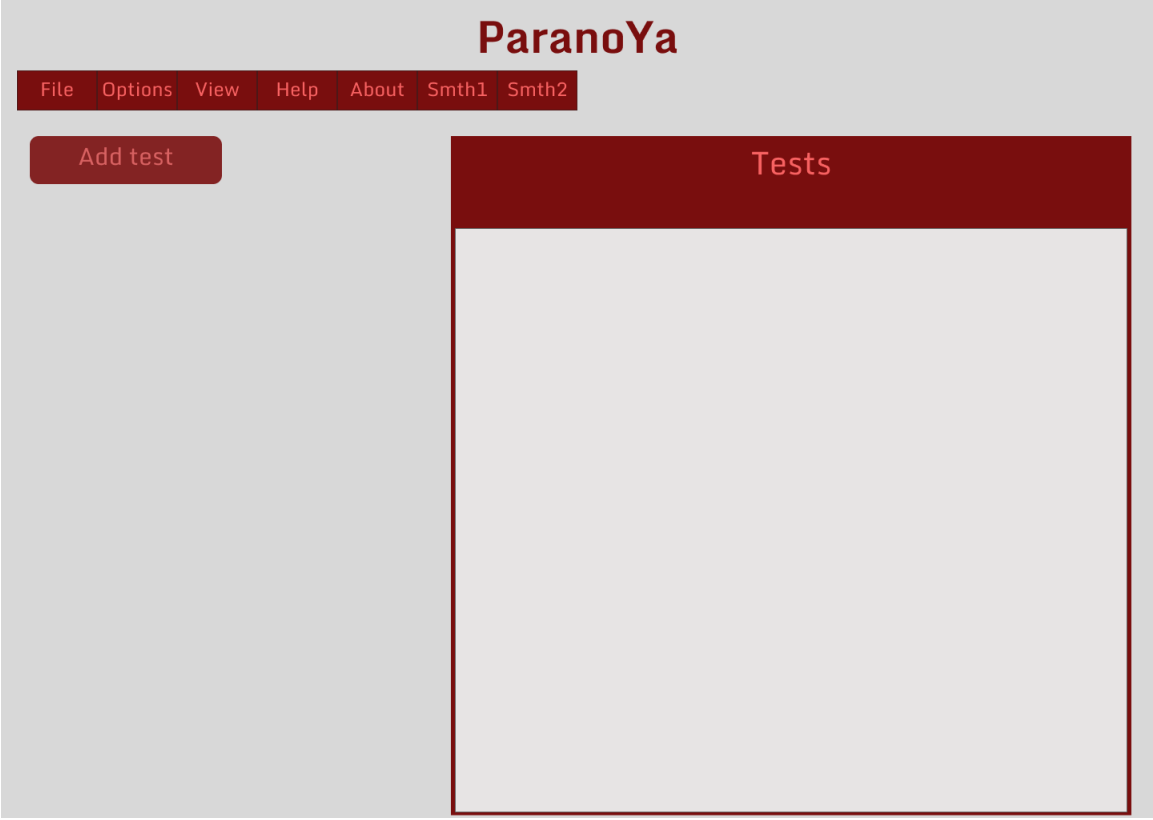

Figure 4: New Test

Purpose:	Aplikácia umožní užívateľovi zostrojiť novú sadu testov a uložiť ich, alebo načítať a použiť už existujúce sady testov
Navigation and User Interaction:	Užívateľ s tlačítkom **Add** môže pridať testy, ktorých vyberá zo zoznamu všetkých testov. Nastavenie parametrov jednotlivých testov sa robí vo vzláštnom okne. S tlačítkom **Save** je možné uložiť zostrojenú sadu testov. S tlačítkom **Load** je možné načítať vopred vytvorených sád testov.

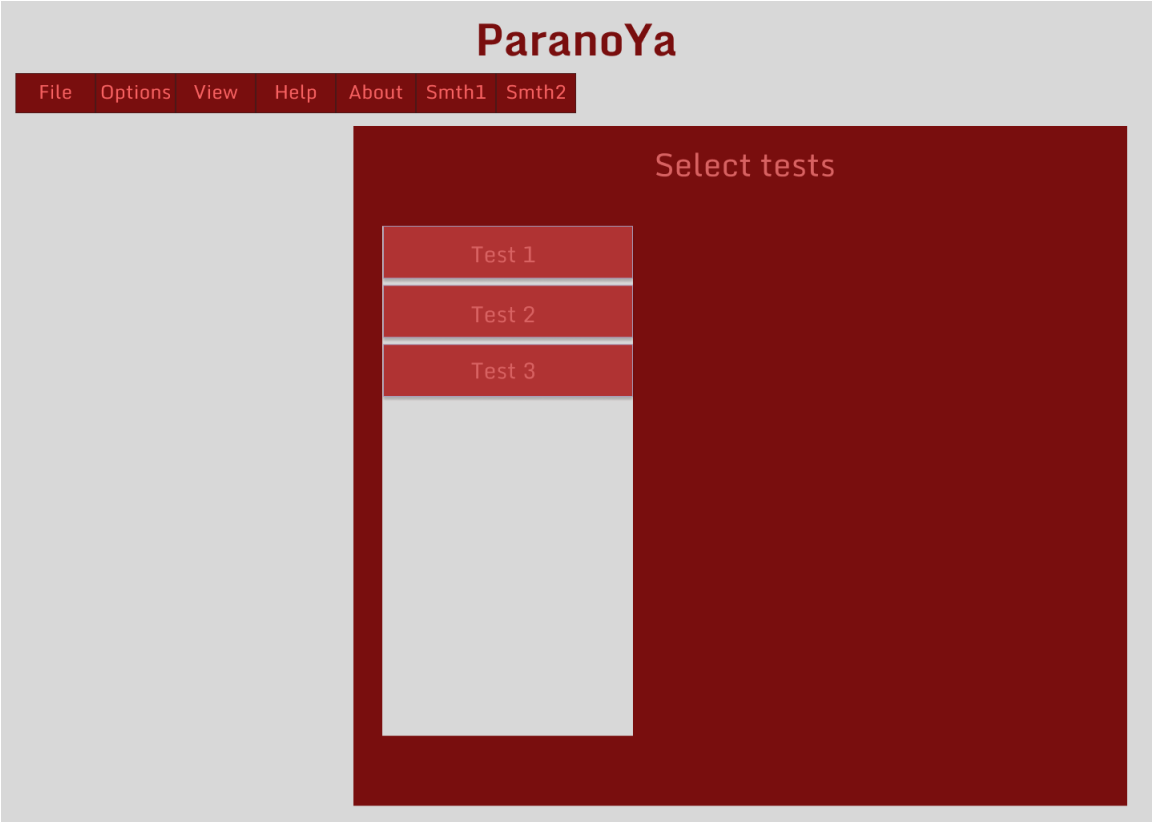

Figure 5: Add test

Purpose:	...
Navigation and User Interaction:	...

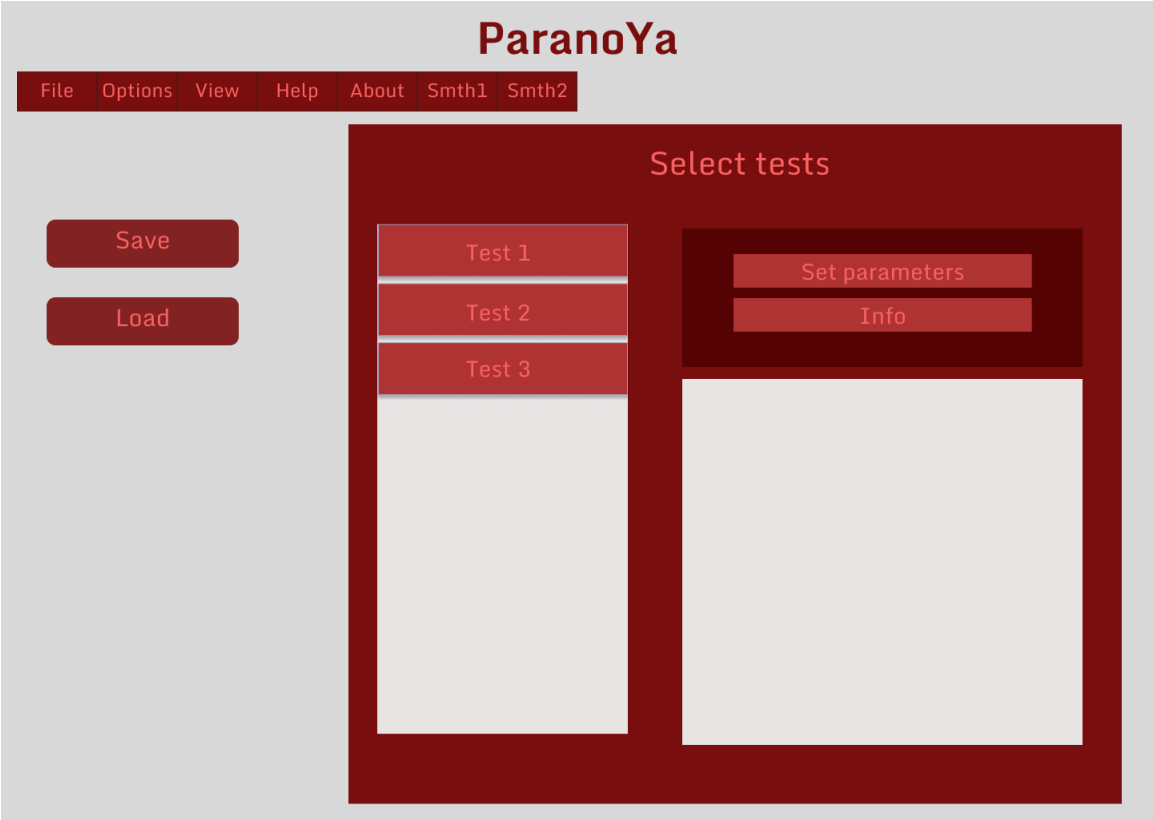

Figure 6: Add test

Purpose:	...
Navigation and User Interaction:	...

3.5 Sequences

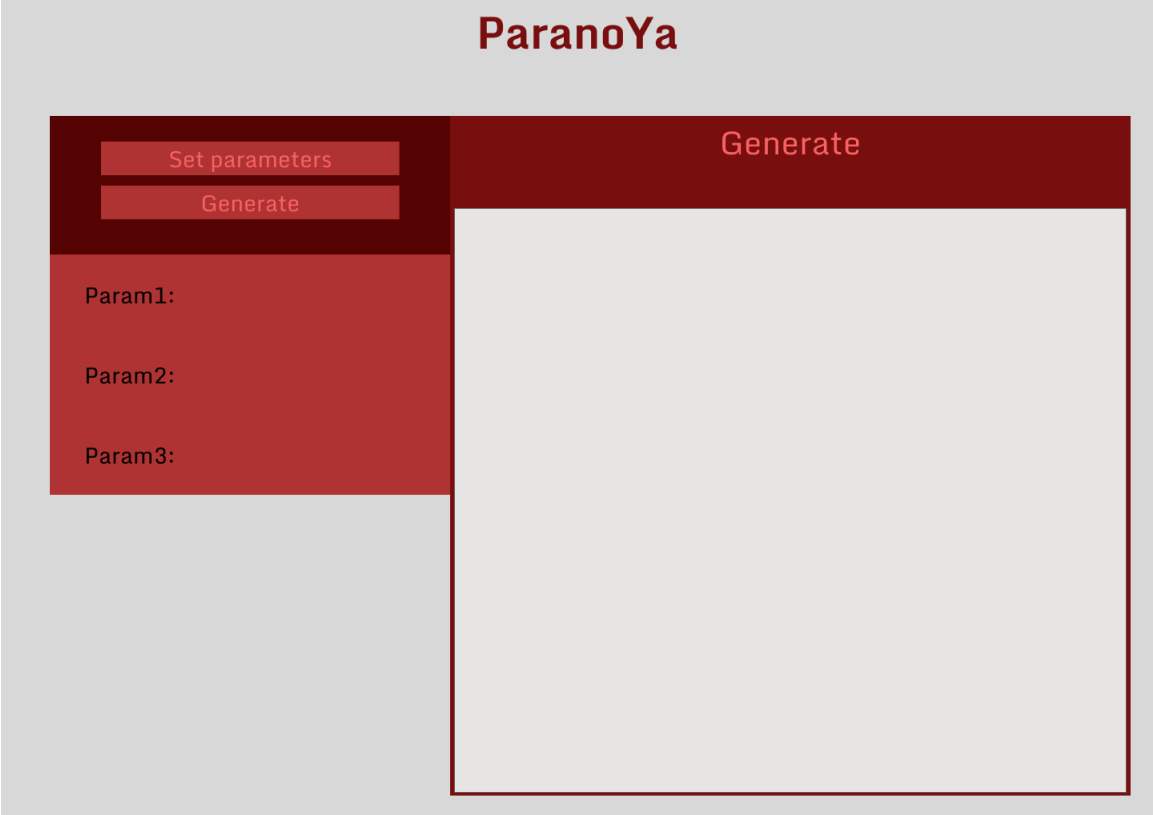

Figure 7: Generate

Purpose:	...
Navigation and User Interaction:	...

3.6 Results

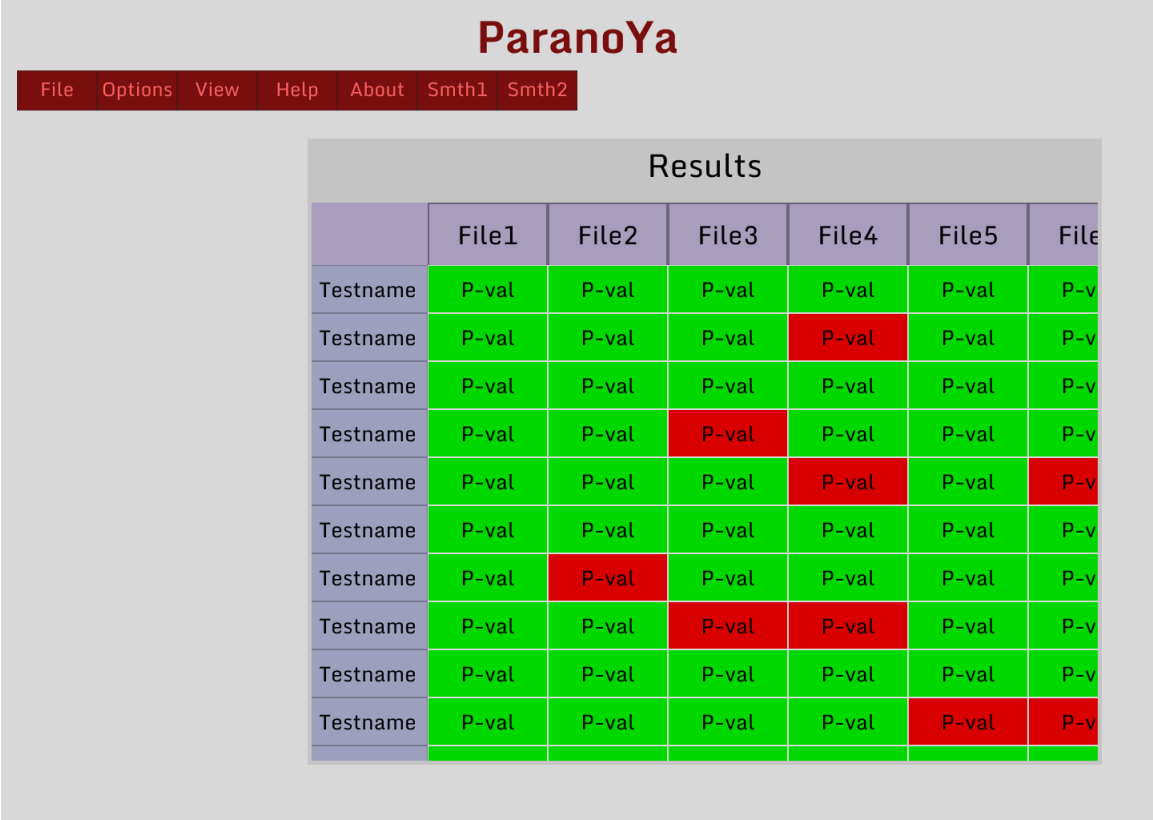

Figure 8: Results

Purpose:	...
Navigation and User Interaction:	...

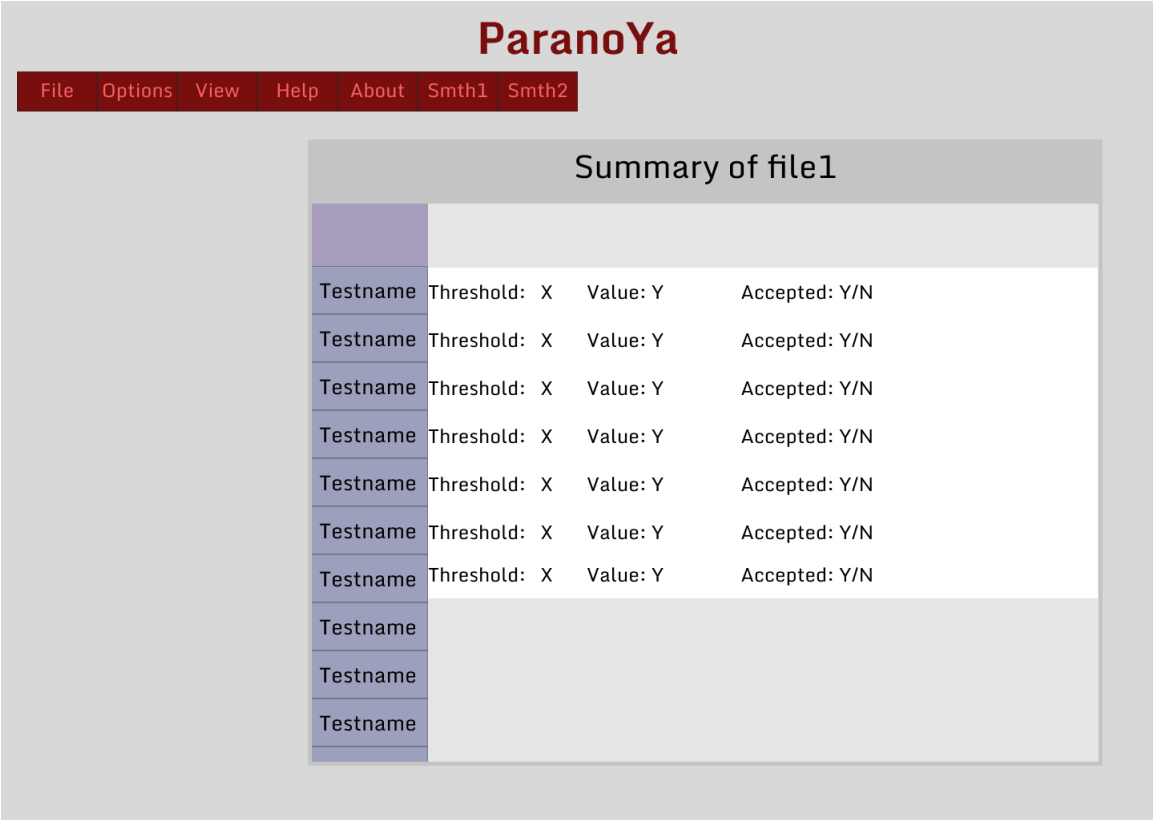

Figure 9: Detailed results

Purpose:	...
Navigation and User Interaction:	...

4 Implementácia

4.1 UML Diagramy

4.1.1 Use Case Diagram

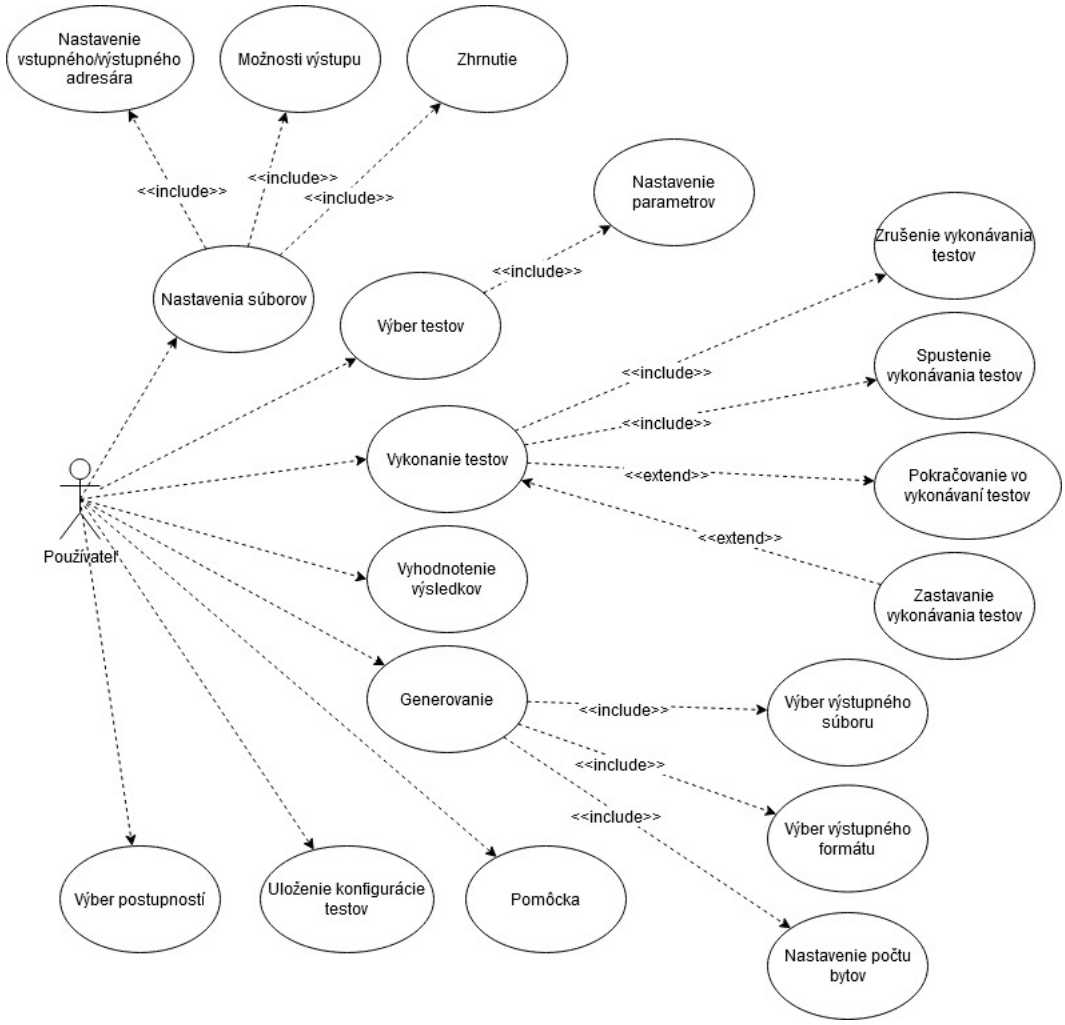

Figure 10: paranoYa - Use case diagram

4.1.2 Sequence Diagrams

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *Tasks* v navigačnej lište aplikácie, zvolí *Run*. Spustí sa testovanie pseudonáhodných postupností. Spusteniu predchádza nahranie postupnosti, zvolenie metodiky.

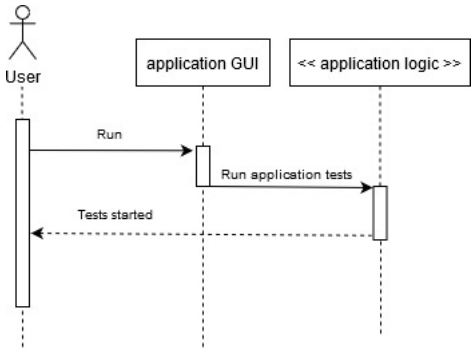

Figure 11: paranoYa - Sequence diagram (Run)

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *Tasks* v navigačnej lište aplikácie, zvolí *Continue*. Testovanie pseudonáhodných postupností pokračuje. Pokračovaniu testovania predchádza spustenie *Run* a pozastavenie testovania *Pause*.

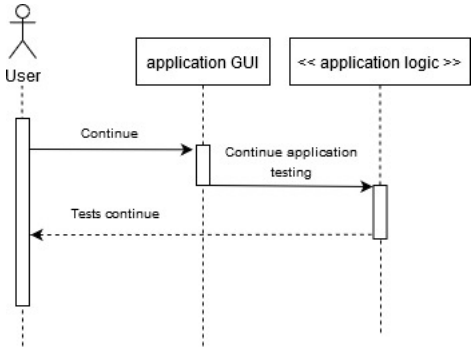

Figure 12: paranoYa - Sequence diagram (Continue)

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *Tasks* v navigačnej lište aplikácie, zvolí *Pause*. Preruší sa testovanie pseudonáhodných postupností. Prerušeniu predchádza spustenie testovania *Run*.

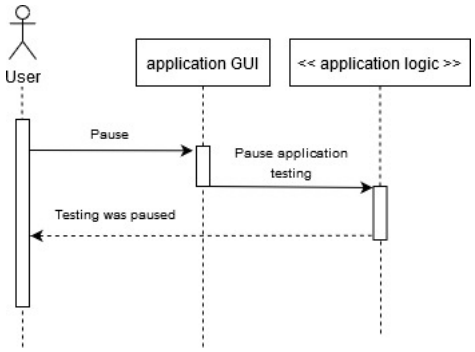

Figure 13: paranoYa - Sequence diagram (Pause)

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *Tasks* v navigačnej lište aplikácie, zvolí *Cancel*. Zastaví sa testovanie pseudonáhodných postupností. Zastaveniu predchádza spustenie testovania *Run*.

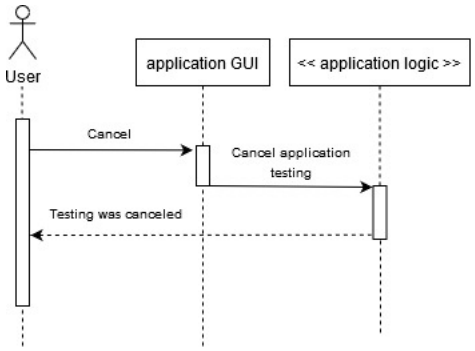

Figure 14: paranoYa - Sequence diagram (Cancel)

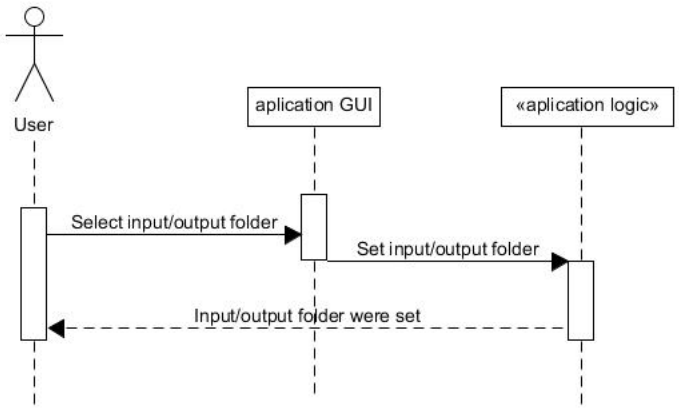

Figure 15: paranoYa - Sequence diagram (Set input/output folder)

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *File* v navigačnej lište aplikácie, zvolí *Batch process...*. Používateľovi sa zobrazí ďalšie okno, v ktorom sa mu zobrazí Source directory, Destination directory, Output options a Summary. Po kliknutí na tlačidlo *Set...*, pri možnosti Source directory si používateľ vyberie zdrojový priečinok a stlačí *OK*. Následne sa tento priečinok nastaví aj ako Destination directory. Pokiaľ by chcel používateľ zvoliť iný destination directory, nastaví ho obdobným spôsobom ako Source directory.

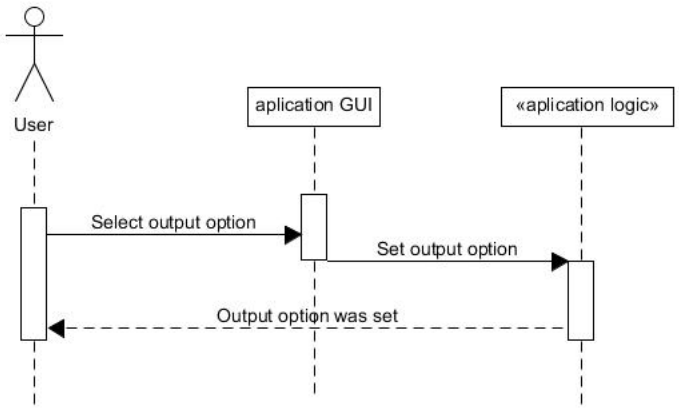

Figure 16: paranoYa - Sequence diagram (Select output option)

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *File* v navigačnej lište aplikácie, zvolí *Batch process....* Používateľovi sa zobrazí ďalšie okno, v ktorom sa mu zobrazí Source directory, Destination directory, Output options a Summary. V časti Output options si vyberie a zvolí jednu z nasledujúcich možností: XML, HTML, XML + HTML.

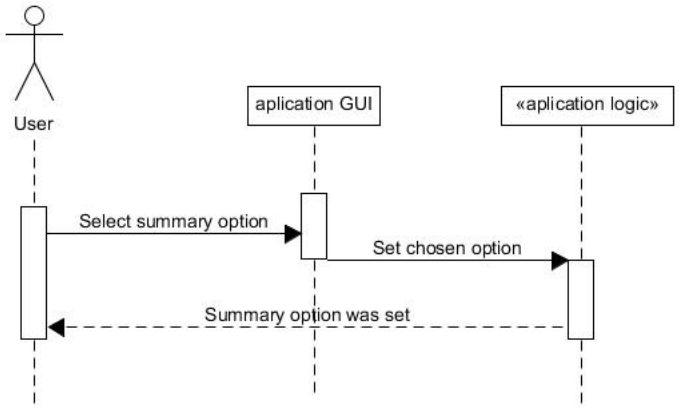

Figure 17: paranoYa - Sequence diagram (Select summary option)

Používateľ interaguje s grafickým rozhraním aplikácie. V záložke *File* v navigačnej lište aplikácie, zvolí *Batch process....* Používateľovi sa zobrazí ďalšie okno, v ktorom sa mu zobrazí Source directory, Destination directory, Output options a Summary. V časti Summary si vyberie a zvolí žiadnu, jednu alebo obidve z nasledujúcich možností: Generate summary HTML file, Prase P-values from all sequences..

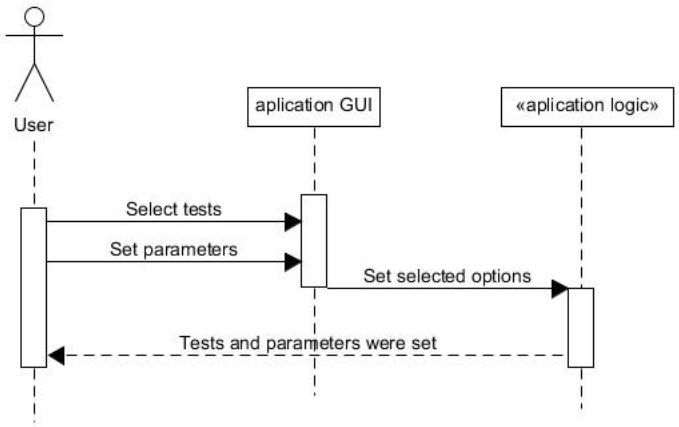

Figure 18: paranoYa - Sequence diagram (Set parameters for selected tests)

Používateľ interaguje s grafickým rozhraním aplikácie. V hlavnom okne aplikácie si zvolí test, ktorý chce vykonať. Pomocou tlačidla *Add new* ho pridá a následne mu vpisovaním môže nastavovať parametre, ktoré mu prislúchajú, ako napr. N - length of input string alebo M - length in bits of each block.

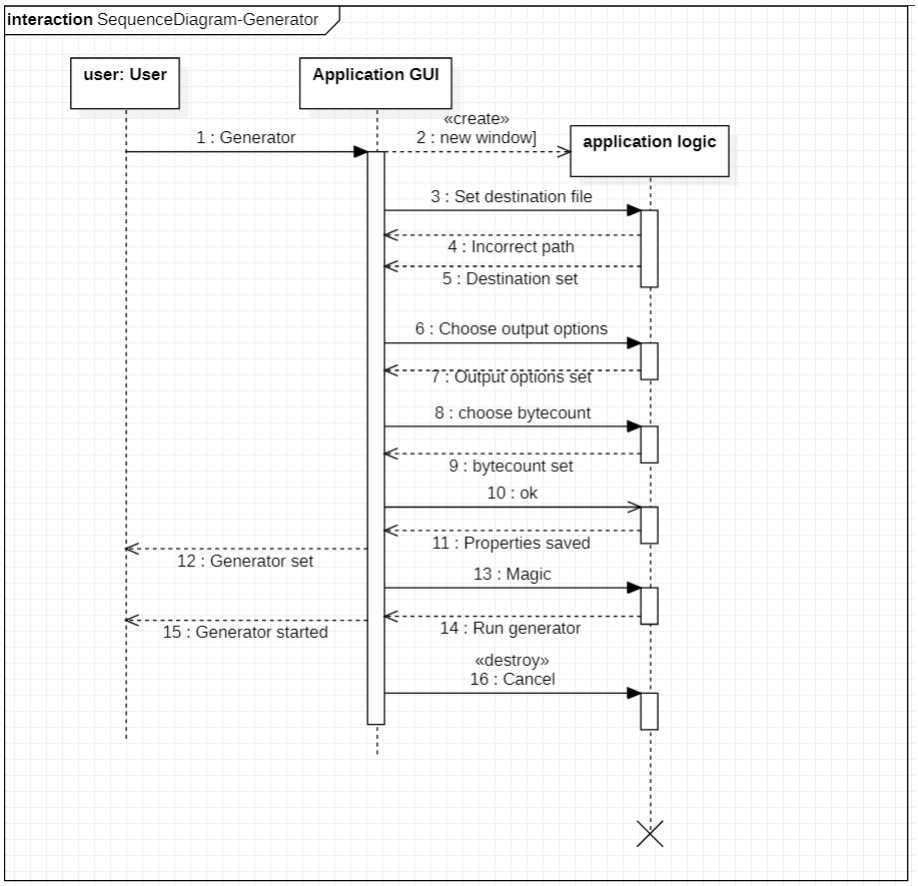

Figure 19: paranoYa - Sequence diagram (Generator)

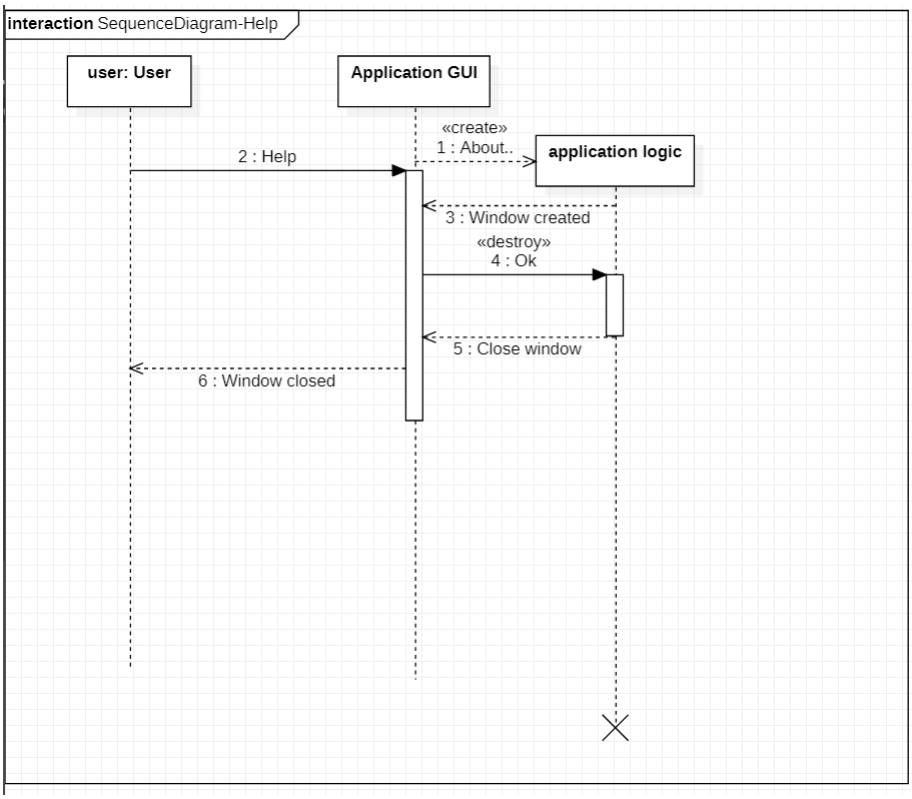

Figure 20: paranoYa - Sequence diagram (Help)

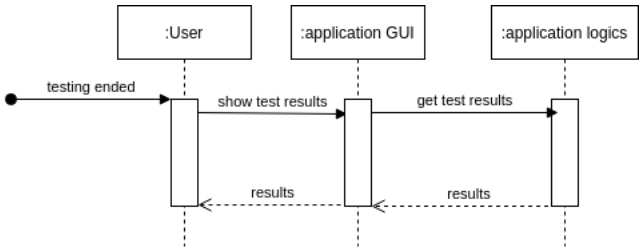

Figure 21: paranoYa - Sequence diagram (Show test results)

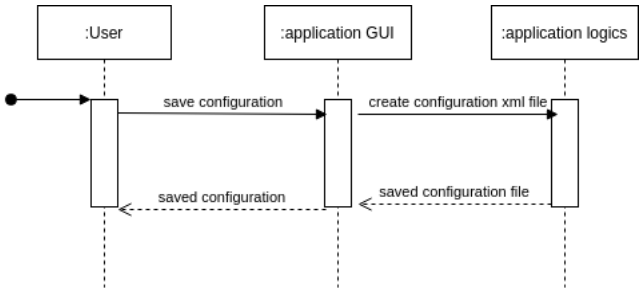

Figure 22: paranoYa - Sequence diagram (Save configuration)

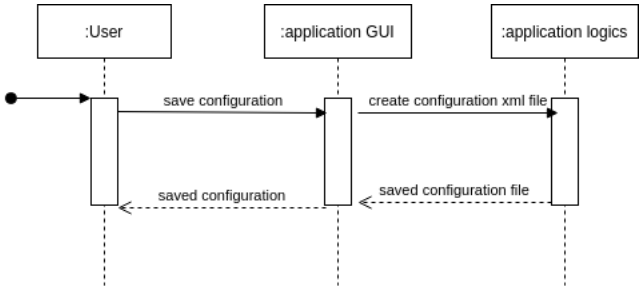

Figure 23: paranoYa - Sequence diagram (Load sequence)

4.1.3 Activity Diagrams

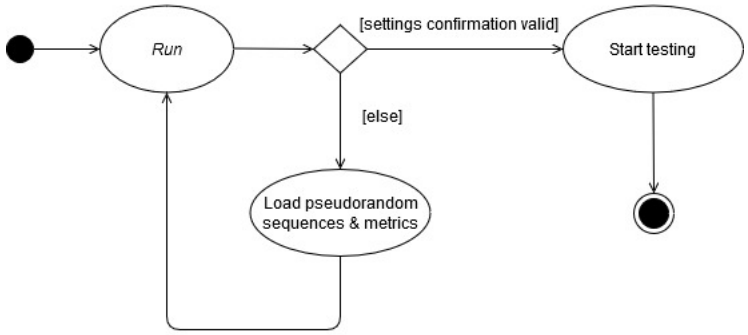

Figure 24: paranoYa - Activity diagram (Run)

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
```

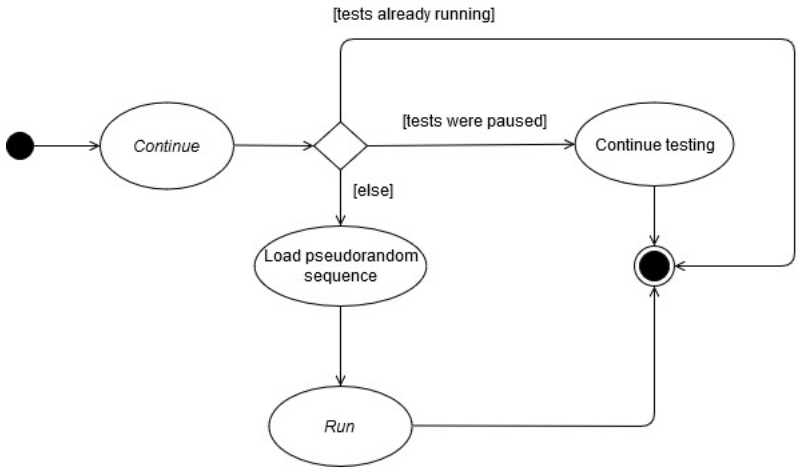

Figure 25: paranoYa - Activity diagram (Continue)

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
```

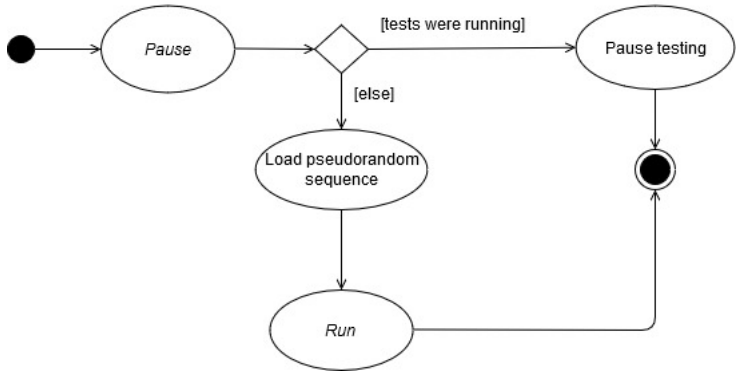

Figure 26: paranoYa - Activity diagram (Pause)

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
```

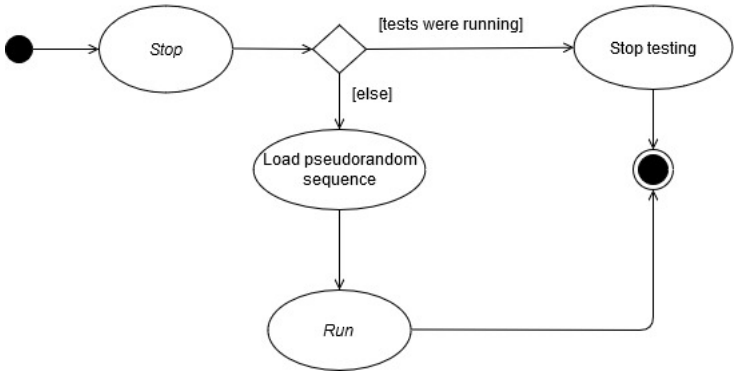

Figure 27: paranoYa - Activity diagram (Stop)

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

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

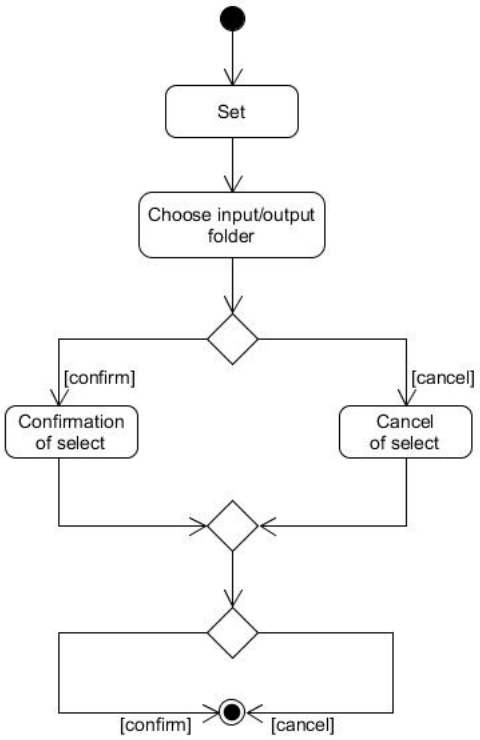

Figure 28: paranoYa - Activity diagram (Set input/output folder)

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
```

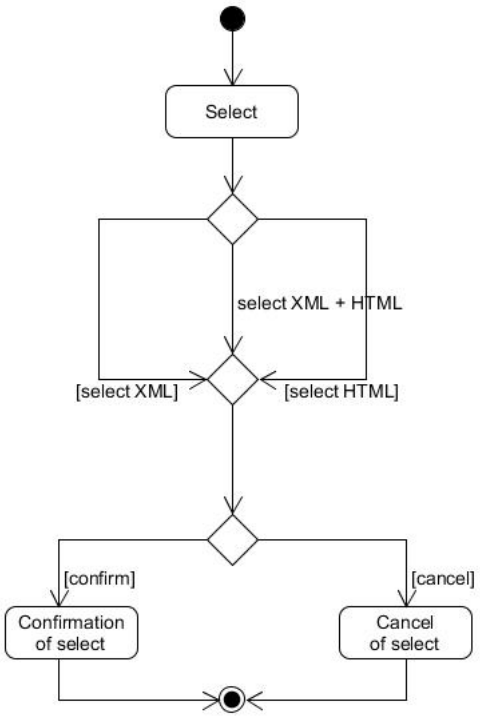

Figure 29: paranoYa - Activity diagram (Select output option)

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
```

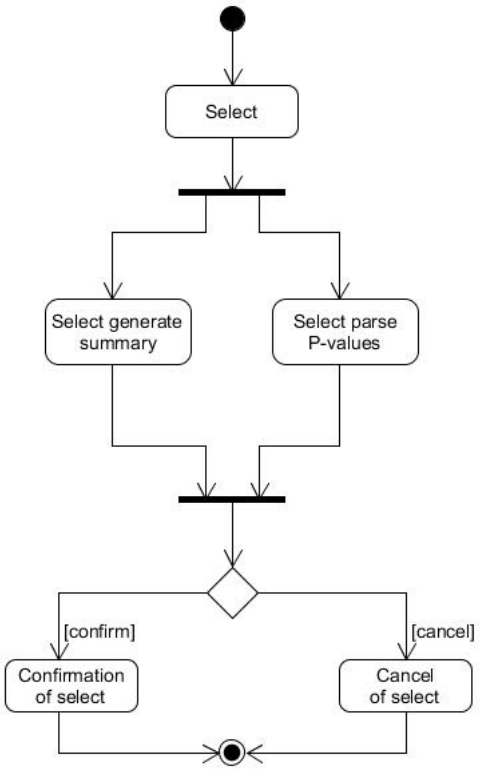

Figure 30: paranoYa - Activity diagram (Select summary option)

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

```
1  Function SelectSummary(event):
2      if isClickedGeneratet 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
```

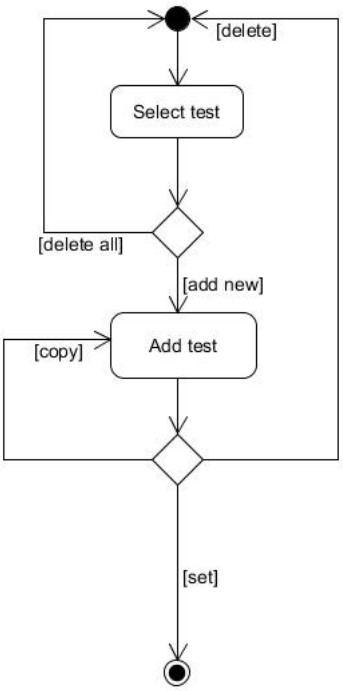

Figure 31: paranoYa - Activity diagram (Set parameters for selected tests)

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
```

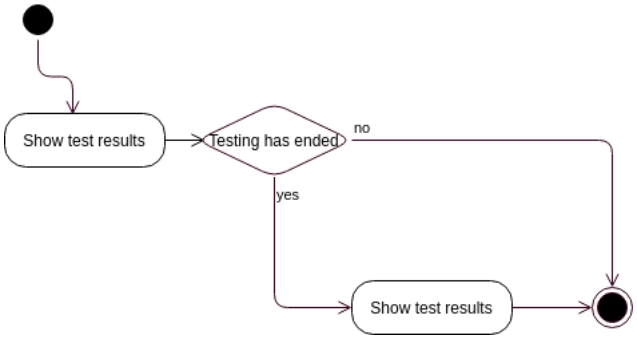

Figure 32: paranoYa - Activity diagram (Show test results)

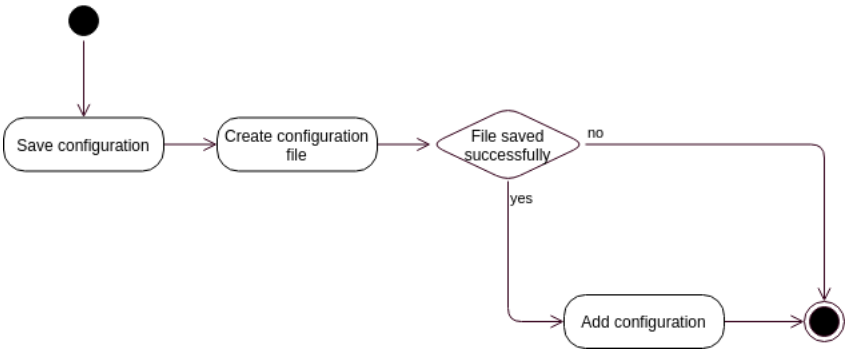

Figure 33: paranoYa - Activity diagram (Save configuration)

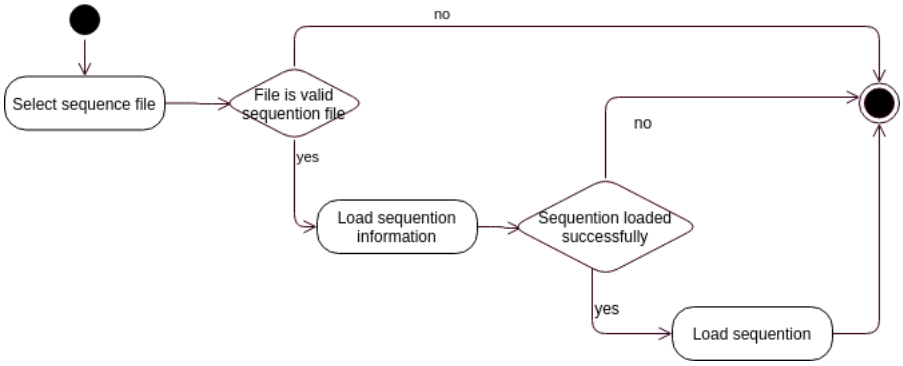

Figure 34: paranoYa - Activity diagram (Load sequence)

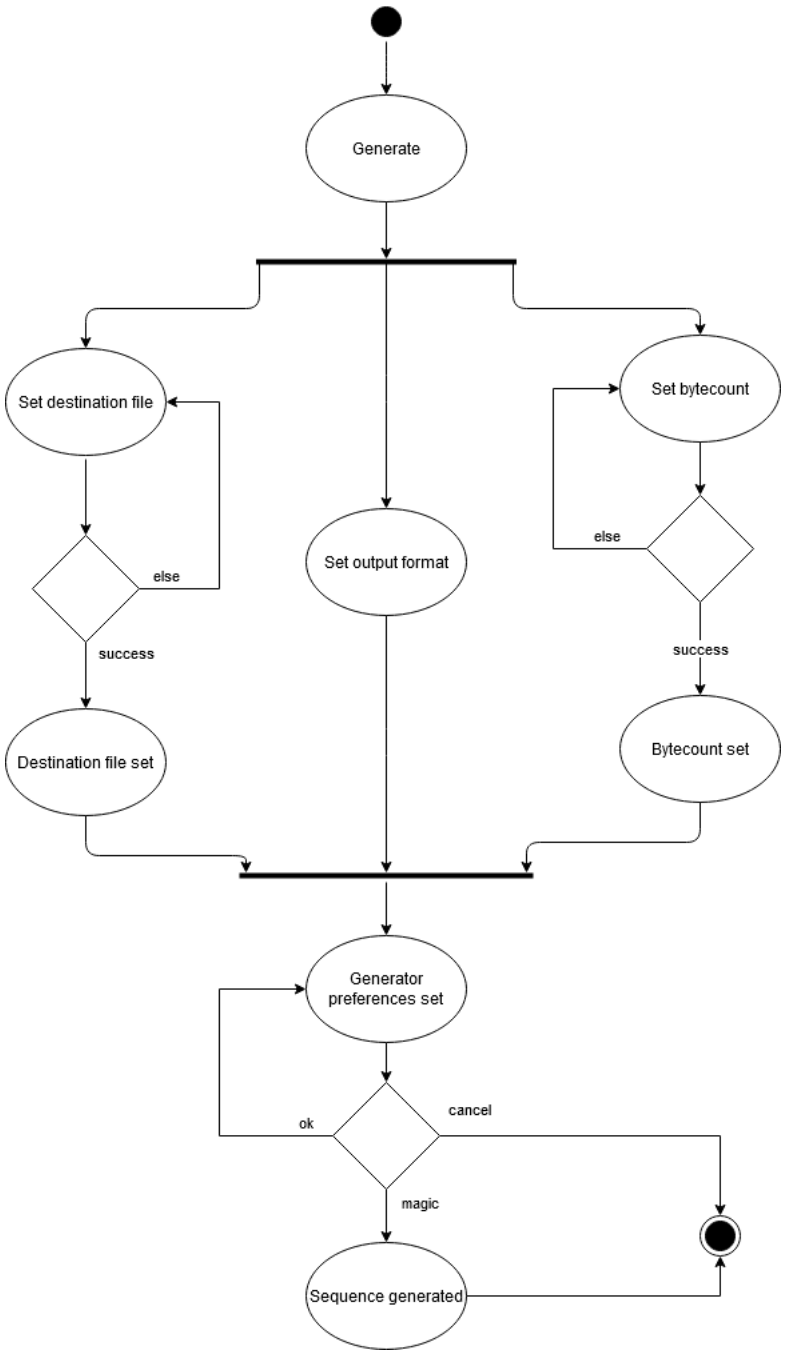

Figure 35: paranoYa - Activity diagram (Generate sequence)

Algorithm 9: Generate sequence into file.

```
1  Function Generate(event):
2      generate();
3      destinationFile_select();
4      if success then
5          destinationFile_set() ;
6      else
7          destinationFile_select() ;
8      end
9      OutputFormat_set();
10     byteCount_select();
11     if success then
12         byteCount_set() ;
13     else
14         byteCount_select() ;
15     end
16     generatorPreferences_select();
17     if Ok then
18         generatorPreferences_select();
19     end
20     if Magic then
21         Sequence_generate();
22     end
23     if Cancel then
24         Generate_quit();
25     end
```

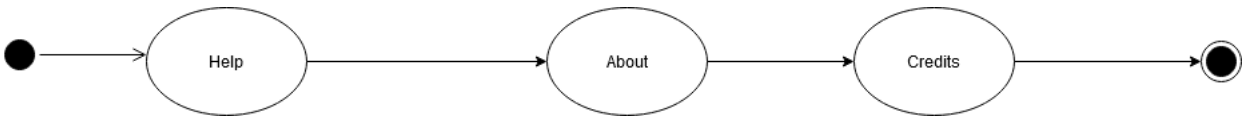

Figure 36: paranoYa - Activity diagram (Help)

Algorithm 10: Open help for inforamtion

```
1  Function Help(event):
2      help();
3      About_click();
4      Credits.show();
```

4.2 Acceptance tests

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	