

Izveštaj primene alata za verifikaciju u okviru samostalnog praktičnog projekta na kursu Verifikacija Softvera Matematički fakultet

Milica Kleut, 1025/2022
milicakleut98@gmail.com

9. februar 2023.

Sažetak

U ovom radu biće dat detaljan prikaz analize projekta koji se nalazi na sledecoj adresi: https://github.com/matf-pp/2021_Podmornice. Projekat je nastao u okviru kursa Programske paradigme a tvorci su Sara Mišić (github nalog: <https://github.com/SaraMisic>) i Milena Filipović (github nalog: <https://github.com/Milena-99>). U nastavku biće navedeni svi korišćeni alati, opis i svrha njihovog korišćenja. Takođe biće navedeni svi zaključci koji su doneti. Primena alata će biti izvršena na main grani, nad komitom čiji je hash code sledeći: ec528ce43af9de94bf2fab308ce2d6270584881c.

Sadržaj

1	Uvod	2
2	Valgrind	2
2.1	Memcheck	2
2.2	Callgrind	4
3	Unit Test	6
4	LCOV	7
5	Zaključak	9

1 Uvod

Posmatrani projekat je igrice potapanja brodova. Položaji brodova na tabli se genrišu na slučajan način što daje nedeterminističko ponašanje programa što svakako nije pogodno za testiranje. Takođe koristi se Qt okruženje, odnosno veliki broj Qt biblioteka je prisutan koje nam otežavaju analizu koda pisanog od strane tvoraca projekta.

Ideja je da se kod popravi u skladu sa navedenim problemima. Zbog toga u okviru repozitorijuma postoji još jedan submodul koji sadrži izmenjen kod igrice. Naravno, kod je trebalo popraviti onoliko koliko je neophodno za testiranje. Ispravke koda se sastoje u korišćenju makroa. Odnosno kada želimo analizu i testiranje koda određene makroe uključimo inače se izvršava originalni kod. Glavni deo jeste dodata main funkcija za potrebe analize u kojoj se pozivaju samo funkcije koje su implementirali studenti kreatori projekta. Prema tome izbegnute su sve Qt biblioteke. Za potrebe testiranja treba uključiti dodatan makro MY_UKLONIRAND kojim je zamaskirano random postavljanje podmornica (zapravo je ovo delimično zamaskirano ali dovoljno za potrebe testiranja). Neke funkcije su skroz zamaskirane i uvedene su nove funkcije prilagođene potrebama analize ali sa istom logikom i ponašanjem.

Što se tiče makroa dovoljno ih je uključiti u okviru .pro fajla ukoliko želimo da radimo analizu.

Komanda u .pro fajlu je:

```
DEFINES+= "MY_TST_MAC=1" "MY_MAC_KIH=1" "MY_TST_MAC_KRENIIGRU=1" "MY_UKLONIRAND=1"
```

Za analizu i testiranje su posmatrani samo fajlovi kreniigru.cpp, kreniigru.h i main.cpp tako da za potrebe analize možemo zakomentarisati nepotrebne fajlove u .pro fajlu. Posmatramo samo ove fajlove jer je u njima implementirana cela logika igrice.

2 Valgrind

2.1 Memcheck

Prvi alat koji je korišćen je [valgrind alat](#) memcheck. Motivacija za korišćenje ovog alata je ta što u velikom kodu lako dolazi do curenja memorije. Teško je za veliki kod, kod koga ima mnogo različitih objekata, ispratiti na kom je mestu neki objekat kreiran (odnosto ostavljenja memorija za njega) i da li je on pravilno uklonjen i kada, da li je došlo do prekoračenja... To će za nas uraditi memcheck. Kako se koristi?

Prvo je potrebno napraviti izvršni fajl igrice Podmornica. Potrebno je pozicionirati se u direktorijum gde se nalazi igrice, otvoriti .pro fajl, dodati -g opciju, ugasiti optimizacije i dodati odgovarajuće makoe na već objašnjen način. Zatim u terminalu ukucati sledeće komande:

```
qmake PodmorniceGUI.pro
make
```

Nakon ovoga imamo izvršni fajl PodmorniceGUI. Zatim pokrećemo valgrind alat komandom `valgrind ./PodmorniceGUI`.

Podrazumevano se koristi memcheck alat, ali može i stajati naglašeno da je

```

1 ==30932== Memcheck, a memory error detector
2 ==30932== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
3 ==30932== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
4 ==30932== Command: ./PodmorniceGUI
5 ==30932== Parent PID: 4601
6 ==30932==
7 ==30932==
8 ==30932== HEAP SUMMARY:
9 ==30932==   in use at exit: 18,612 bytes in 6 blocks
10 ==30932== total heap usage: 28 allocs, 22 frees, 93,508 bytes allocated
11 ==30932==
12 ==30932== 4 bytes in 1 blocks are still reachable in loss record 1 of 6
13 ==30932==   at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
14 ==30932==   by 0x58018E3: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
15 ==30932==   by 0x5801F03: g_private_get (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
16 ==30932==   by 0x57D3460: g_slice_alloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
17 ==30932==   by 0x57A22E1: g_hash_table_new_full (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
18 ==30932==   by 0x57C57C2: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
19 ==30932==   by 0x4011B99: call_init.part.0 (dl-init.c:72)
20 ==30932==   by 0x4011CA0: call_init (dl-init.c:30)
21 ==30932==   by 0x4011CA0: _dl_init (dl-init.c:119)
22 ==30932==   by 0x4001139: ??? (in /usr/lib/x86_64-linux-gnu/ld-2.31.so)
23 ==30932==

```

Slika 1: Memcheck - deo izvrestaja

```

70 ==30932== 16,384 bytes in 1 blocks are still reachable in loss record 6 of 6
71 ==30932==   at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload
72 ==30932==   by 0x57BAE98: g_malloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0
73 ==30932==   by 0x57C57D3: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400
74 ==30932==   by 0x4011B99: call_init.part.0 (dl-init.c:72)
75 ==30932==   by 0x4011CA0: call_init (dl-init.c:30)
76 ==30932==   by 0x4011CA0: _dl_init (dl-init.c:119)
77 ==30932==   by 0x4001139: ??? (in /usr/lib/x86_64-linux-gnu/ld-2.31.so)
78 ==30932==
79 ==30932== LEAK SUMMARY:
80 ==30932==   definitely lost: 0 bytes in 0 blocks
81 ==30932==   indirectly lost: 0 bytes in 0 blocks
82 ==30932==   possibly lost: 0 bytes in 0 blocks
83 ==30932==   still reachable: 18,612 bytes in 6 blocks
84 ==30932==   suppressed: 0 bytes in 0 blocks
85 ==30932==
86 ==30932== For lists of detected and suppressed errors, rerun with: -s
87 ==30932== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Bracket match found on line: 62

```

Slika 2: Memcheck - deo izvrestaja

alat memcheck opcijom `--tool=memcheck` .

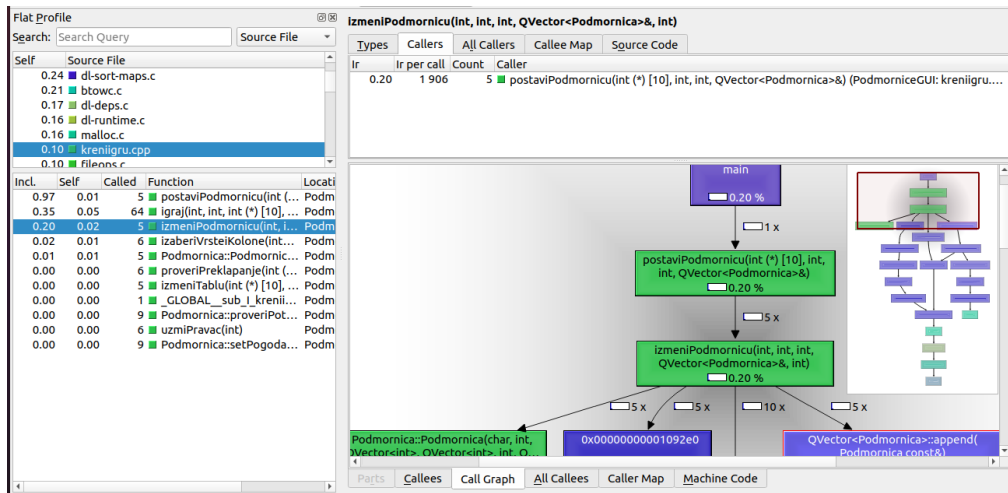
Datno opcije koje sam ja koristila su:

```

--track-origins=yes
--show-leak-kinds=all
--leak-check=full
--log-file=../../memCheck/logNew.txt

```

Nakon pokretanja alata kao izlaz smo dobili da ne postoji trajno curenje memorije. Deo izvestaja memcheck-a prikazan je na slici 1 i 2. Kao što se može primetiti imamo više alociranja nego oslobađanja. Memcheck daje detaljan izveštaj koliko kog tipa curenja memorije je prisutno. Iz izveštaja memcheck-a može se zaključiti da nema trajno izgubljene memorije. Svi delovi programa pisani od strane programera projekta gde je memorija alocirana, su pravilno oslobodeni. Imamo jos uvek dostupne bajtove (still reachable) ali to ne znači da je pristupno curenje memorije. Zaključujemo da ne postoji problem sa memorijom u projektu.



Slika 3: Callgrind

2.2 Callgrind

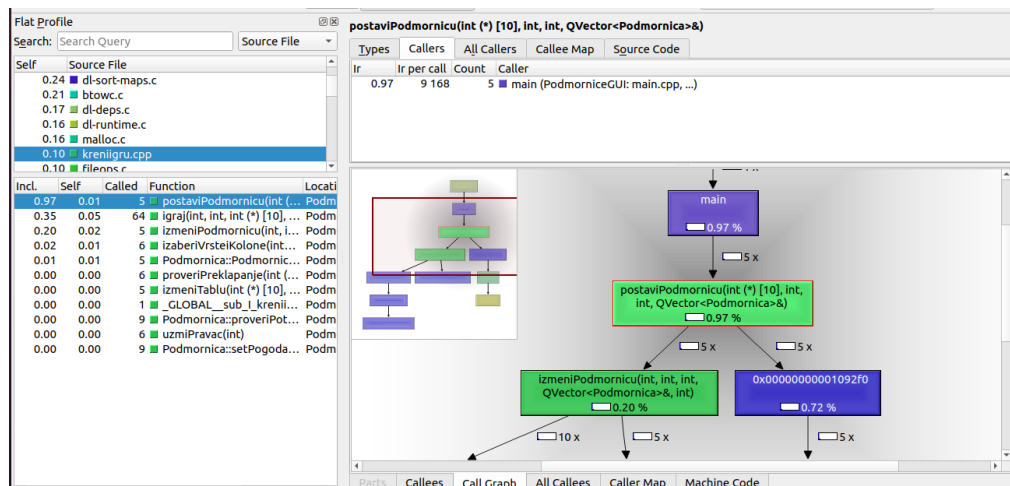
Callgrind je drugi alat koji je korišćen. On je takođe Valgrind alat. On generiše listu poziva funkcija u obliku grafa i računa cene funkcija. Cene funkcija se propagiraju. Sa grafa se lako može uočiti koje funkcije imaju najveću cenu. Za te funkcije treba razmotriti da li se mogu efikasnije implementirati kako bi se smanjila cena njihovog poziva. Prilikom kompilacije programa koji analiziramo nije potrebno gasiti optimizacije (ali ostavljamo `-g` opciju). Alat se poziva na sličan način kao i memcheck.

valgrind --tool=callgrind

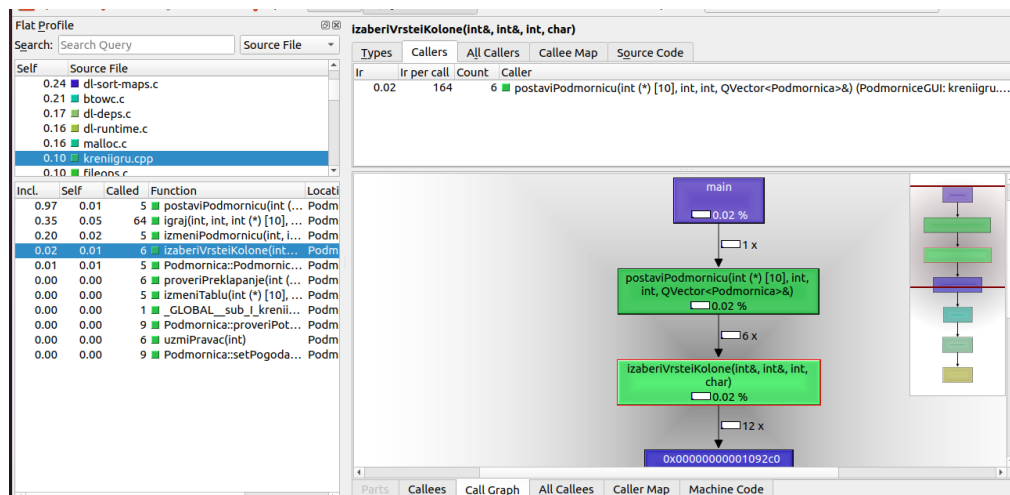
Mogu se koristiti i dodatne opcije (za ovaj konkretan primer ove opcije nisu korišćene) za analizu keša, praćenje upotebe grana itd. Za grafičku reprezentaciju koristimo KCachegrind.

Na slikama 3, 4, 5 i 6 se vidi deo rezultata callgrinda. Posmatrane su samo funkcije implementirane od strane programera projekata jer nas samo one zanimaju, njih eventualno možemo popraviti.

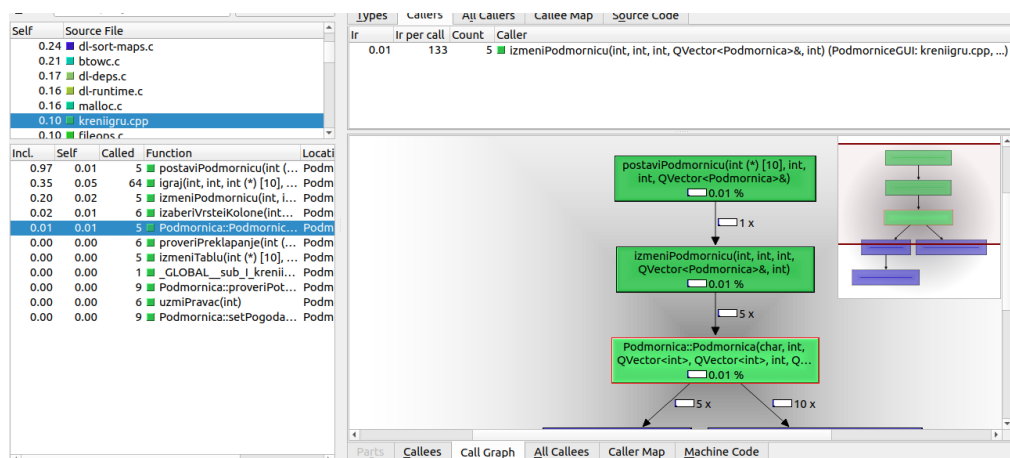
Posmatrane funkcije su dobro implementirane. Možemo primetiti da su korišćene reference na objekte u pozivima funkcija kad god je to moguće, čime je program rasterećen kopiranja objekata (pozivi copy konstruktora su skupi).



Slika 4: Callgrind



Slika 5: Callgrind



Slika 6: Callgrind

3 Unit Test

Sledeći alat koji je korišćen je testiranje. Testiranje ima za cilj povećanje pouzdanosti koda. Ne garantuje da je kod ispravan ali podiže nivo ispravnosti i sigurnost. Njega koristimo u kombinaciji sa alatom GCOV jer je bitna pokrivenost testova. Cilj je da test ima što bolju pokrivenost.

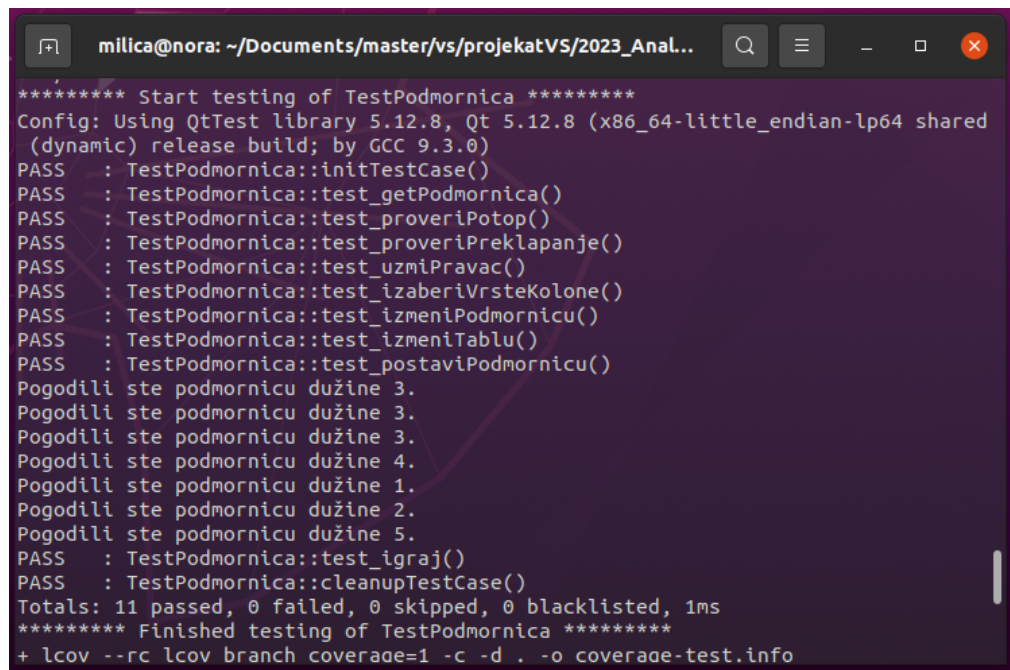
Za konkretan projekat čija se analiza radi, postoji mnogo funkcija koje se pozivaju radi pravljenja gui-a, pa su zato zanemarene jer nas interesuje funkcionalnost projekta, odnosno da li on radi u skladu sa očekivanjima. Zbog toga su uvedeni već pomenuti makroi koji menjaju tok programa. U .pro fajl su dodati sledeći makroi:

```
DEFINES+="MY_TST_MAC=1" "MY_MAC_KIH=1" "MY_TST_MAC_KRENIIGRU=1" "MY_UKLONIRAND=1"
```

Komande za prevođenje i pokretanje testa:

```
qmake TestPodmornica.pro
make
./TestPodmornica
```

Izlaz koji daje testiranje prikazan je na slici 7. Kao što se može primetiti iz izlaza testiranja testirano je 11 funkcija i sve su prošle testiranje. Testirane su sve funkcije koje su implementirane od strane kreatora igrice i koje služe za davanje logike projektu. Testiranju se samo funkcije biblioteke kreniigru.h jer ona sadrži celu logiku igrice. Biblioteke pobeda.h i gubitak.h sadrže funkcije koje generišu novi prozor u kome piše ishod igre pa su svi pozivi ovakvih funkcija u okviru kreniigru.h zamaskirani, kao i pojedini delovi koji koriste random generisanje. Random brojevi su zamenjeni konkretnim vrednostima kako bi ponašanje bilo determinističko i kako bi moglo da se zna šta se očekuje kao izlaz pojedinog funkcija.

A screenshot of a terminal window with a dark background. The window title is 'milica@nora: ~/Documents/master/vs/projekatVS/2023_Anal...'. The terminal output shows the start of testing for 'TestPodmornica', followed by a configuration line for QtTest 5.12.8. A series of 'PASS' messages for various test functions are listed. Below these, several lines indicate 'Pogodili ste podmornicu dužine' followed by a number. The final summary shows 'Totals: 11 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms'. The terminal ends with 'Finished testing of TestPodmornica' and a line for lcov coverage analysis.

```
***** Start testing of TestPodmornica *****
Config: Using QtTest library 5.12.8, Qt 5.12.8 (x86_64-little_endian-lp64 shared
(dynamic) release build; by GCC 9.3.0)
PASS : TestPodmornica::initTestCase()
PASS : TestPodmornica::test_getPodmornica()
PASS : TestPodmornica::test_proveriPotop()
PASS : TestPodmornica::test_proveriPreklapanje()
PASS : TestPodmornica::test_uzmiPravac()
PASS : TestPodmornica::test_izaberiVrsteKolone()
PASS : TestPodmornica::test_izmeniPodmornicu()
PASS : TestPodmornica::test_izmeniTablu()
PASS : TestPodmornica::test_postaviPodmornicu()
Pogodili ste podmornicu dužine 3.
Pogodili ste podmornicu dužine 3.
Pogodili ste podmornicu dužine 3.
Pogodili ste podmornicu dužine 4.
Pogodili ste podmornicu dužine 1.
Pogodili ste podmornicu dužine 2.
Pogodili ste podmornicu dužine 5.
PASS : TestPodmornica::test_igraj()
PASS : TestPodmornica::cleanupTestCase()
Totals: 11 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms
***** Finished testing of TestPodmornica *****
+ lcov --rc lcov branch coverage=1 -c -d . -o coverage-test.info
```

Slika 7: Izlaz testiranja

4 LCOV

Uz GCC dolazi alat gcov za analizu pokrivenost izvornog koda i alat za profilisanje izraz-po-izraz. Želimo da vidimo kolika je moć naših testova a to ćemo znati na osnovu pokrivenosti. Iz izveštaja pokrivenosti mozemo videti tačno koji delovi programa nisu pokriveni testiranjem i ako je potrebno mogu se unaprediti testovi.

Gcov ne daje previše čitljiv izvestaj pa koristimo lcov. Prilikom kompilacije treba koristiti dodatne opcije kompajlera koje omogućavaju pamćenje koliko je puta koja linija, grana i funkcija izvršena. Ti podaci se čuvaju u datotekama ekstenzije .gcno za svaku datoteku sa izvornim kodom. One će kasnije biti korišćene za kreiranje izveštaja o pokrivenosti koda. Opcije koje se dodaju:

```
-fprofile-argc
-ftest-coverage
(umesto ove dve mozemo koristiti samo opciju --coverage)
-O0 (gasenje optimizacija jer one nisu pogodne kada se radi analiza izvornog koda)
```

Dakle, prvi korak bio je dodavanje odgovarajućih flegova u TestPodmornica.pro

```
QMAKE_CXXFLAGS_RELEASE_WITH_DEBUGINFO -= -O0
QMAKE_CXXFLAGS += --coverage
QMAKE_LFLAGS += --coverage$
```

Nakon izvršavanja programa, informacije o pokrivenosti prilikom izvršavanja će biti u sačuvane u datoteci tipa .gcda , za svaku datoteku sa izvornim kodom.

LCOV - code coverage report						
Current view: top level			Hit		Total	Coverage
Test: coverage-test.info			Lines:	692	774	89.4 %
Date: 2023-02-08 15:41:07			Functions:	49	54	90.7 %
			Branches:	724	1281	56.5 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
/home/milica/Documents/master/ys/projektVS/2023_Analysis_Podmornice/Podmornice	<div><div></div></div> 91.4 % 244 / 267	91.7 % 11 / 12	65.4 % 210 / 321
/home/milica/Documents/master/ys/projektVS/2023_Analysis_Podmornice/PodmorniceGUI	<div><div></div></div> 94.7 % 266 / 281	84.2 % 16 / 19	53.3 % 255 / 478
/home/milica/Documents/master/ys/projektVS/2023_Analysis_Podmornice/PodmorniceGUI_test	<div><div></div></div> 88.9 % 8 / 9	- 0 / 0	0.0 % 0 / 4
/usr/include/c++/9	<div><div></div></div> 100.0 % 21 / 21	100.0 % 2 / 2	56.2 % 9 / 16
/usr/include/c++/9/bits	<div><div></div></div> 50.0 % 2 / 4	- 0 / 0	0.0 % 0 / 2
/usr/include/x86_64-linux-gnu/bits	<div><div></div></div> 77.5 % 141 / 182	93.8 % 15 / 16	54.4 % 249 / 458
QtCore	<div><div></div></div> 100.0 % 10 / 10	100.0 % 5 / 5	50.0 % 1 / 2
QtTest			

Generated by: LCOV version 1.14

Slika 8: Izveštaj o pokrivenosti koda

Sada smo dobili odgovarajuće .gcda i .gcno fajlove. Pokrenimo alat lcov da bismo dobili čitljiviju reprezentaciju rezultata:

```
lcov --rc lcov.branch_coverage=1 -c -d . -o coverage-test.info
```

Hocemo iz ovog izvrestaja da uklonimo informacije o datotekama čija nas pokrivenost ne zanima. To radimo sledećom komandom:

```
lcov --rc lcov.branch_coverage=1 -r coverage-test.info '/usr/*'
'/opt/*' '*.moc' -o coverage-filtered.info
```

Na kraju generisemo graficki prikaz, odnosno odgovarajuće .html datoteke komandom:

```
genhtml --rc lcov.branch_coverage=1 -o Reports coverage-filtered.info
```

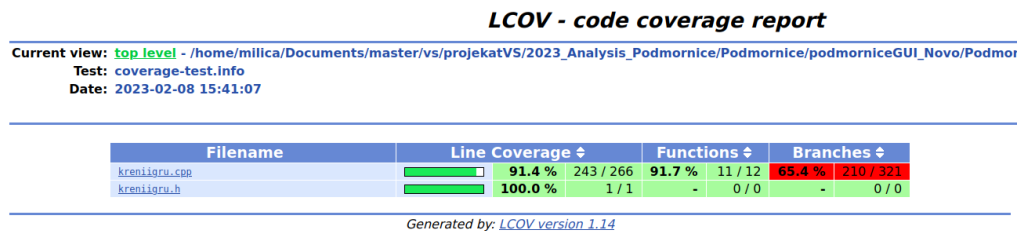
Html verziju izveštaja možemo videti otvaranjem index.html fajla koji se nalazi u Reports direktorijumu.

Sve komande koje su potrebne za prevođenje, pokretanje i analizu pokivenosti TestPodmornica-e nalaze se u skriptu analyse.sh. Dovoljno je u terminalu uku-

cati:

```
./analyse.sh
```

Na slikama 8 i 9 prikazana je pokrivenost testiranja. Cilj je bilo testiranje PodmornicaGUI koje i ima veliku pokrivenost, dakle testovi pokrivaju gotovo celo izvršavanje ovog programa. Kada kliknemo na prvo polje table sa slike 8 ono nam otvara novu stranu sa tablom sa slike 9 gde vidimo da je kreniigr.h skroz pokriven i da je kreniigr.cpp odlično pokriven sem kada su u pitanju grane. Grane koje nisu pokrivene su uglavnom neke if-else iste grane koje su uvedene za podmornice svih mogućih dužina (podmornice dužine 1,2,3,4,5). Odnosno naredbe koje se ponavljaju mnogo puta svuda po kodu a retko se dešavaju i imaju trivijalnu logiku koju nema potrebe testirati.



Slika 9: Izveštaj o pokrivenosti koda

5 Zaključak

Posmatrani projekat nema krucijalnih propusta. Suštinski je dobro implementiran.

Preporuka je da se za klasu Podmornica implementiraju geteri i da se uvedu provere vrednosti promenljivih klase Podmornica prilikom instanciranja objekata. Na primer provera da li je prosleđena dužina podmornice izmedju 1 i 5. Takođe bilo bi lepo kada bi funkcije koje koriste random generator primale dodatan argument koji označava da li da funkcija koristi random ili fiksne vrednosti. Time bi, na primer, dobili lakše testiranje koda.

Ceo projekat je uglavnom lepo podeljen na klase i funkcije tako da svaka funkcija obavlja tačno jednu funkcionalnost. Postoje neki delovi gde bi mogla da se uvede veća granularnost ali je trenutna granularnost korektna.

Svuda su u funkcije prosleđivane reference na objekte. Implementirani su odgovarajući destruktori.