

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

Milica Kleut, 1025/2022  
milicakleut98@gmail.com

26. januar 2023.

## Sažetak

U ovom radu biće dat detaljan prikaz analize projekta koji se nalazi na sledecoj adresi: [https://github.com/matf-pp/2021\\_Podmornice](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

<b>1</b>	<b>Valgrind</b>	<b>2</b>
1.1	Memcheck . . . . .	2
1.2	Callgrind . . . . .	3
<b>2</b>	<b>LCOV</b>	<b>5</b>
<b>3</b>	<b>Unit Test</b>	<b>7</b>
<b>4</b>	<b>Zaključak</b>	<b>9</b>

```

==11037== HEAP SUMMARY:
==11037==      in use at exit: 3,097,108 bytes in 27,792 blocks
==11037==    total heap usage: 745,402 allocs, 717,610 frees, 49,456,098 bytes allocated
==11037==
==11037== 1 bytes in 1 blocks are still reachable in loss record 1 of 5,805
==11037==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==11037==    by 0x6754E98: g_malloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
==11037==    by 0x676F153: g_strdup (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.6400.6)
==11037==    by 0xB0233A9: ??? (in /usr/lib/x86_64-linux-gnu/libgdk-3.so.0.2404.16)
==11037==    by 0x4011889: call_init.part.0 (dl-init.c:72)
==11037==    by 0x4011C90: call_init (dl-init.c:30)
==11037==    by 0x4011C90: _dl_init (dl-init.c:119)
==11037==    by 0x5D90894: _dl_catch_exception (dl-error-skeleton.c:182)
==11037==    by 0x401608E: dl_open_worker (dl-open.c:758)
==11037==    by 0x5D90837: _dl_catch_exception (dl-error-skeleton.c:208)
==11037==    by 0x40155F9: _dl_open (dl-open.c:837)
==11037==    by 0x665C34B: dlopen_doit (dlopen.c:66)
==11037==    by 0x5D90837: _dl_catch_exception (dl-error-skeleton.c:208)

```

Slika 1: Memcheck - deo izvrestaja

```

==11037== LEAK SUMMARY:
==11037==    definitely lost: 6,656 bytes in 26 blocks
==11037==    indirectly lost: 2,151 bytes in 101 blocks
==11037==    possibly lost: 4,144 bytes in 28 blocks
==11037==    still reachable: 3,015,045 bytes in 27,070 blocks
==11037==                of which reachable via heuristic:
==11037==                  length64          : 3,464 bytes in 62 blocks
==11037==                  newarray         : 2,064 bytes in 49 blocks
==11037==    suppressed: 0 bytes in 0 blocks
==11037==
==11037== For lists of detected and suppressed errors, rerun with: -s
==11037== ERROR SUMMARY: 29 errors from 29 contexts (suppressed: 0 from 0)

```

Slika 2: Memcheck - deo izvrestaja

# 1 Valgrind

## 1.1 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 kompilirati program sa -g opcijom i bez optimizacija a zatim pokrecemo valgrind alat komandom `$ valgrind ./a.out`. Podrazumevano se koristi memcheck alat, ali moglo je i stajati naglašeno da je alat memcheck opcijom `--tool=memcheck`.

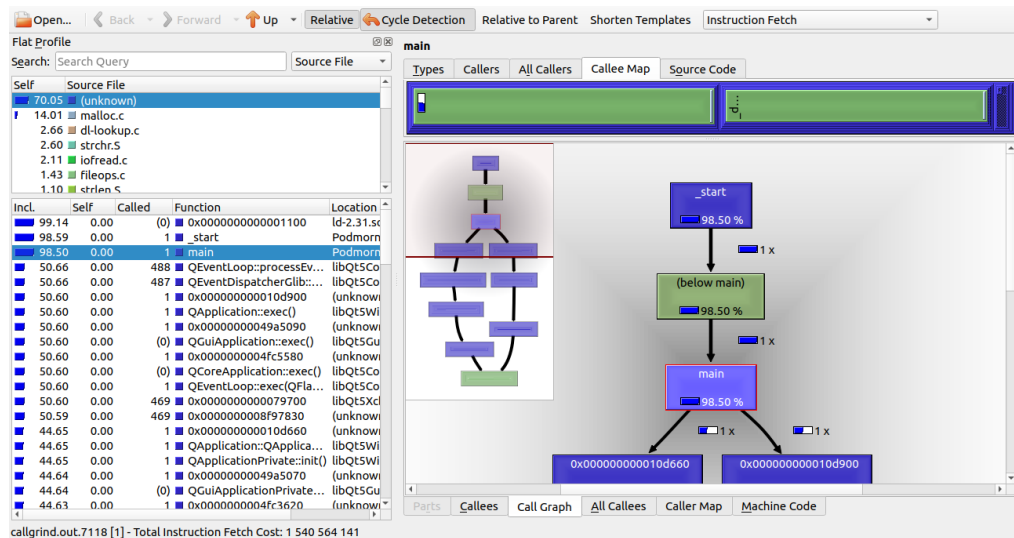
Dodatno opcije koje sam ja koristila su:

```

--track-origins=yes
--show-leak-kinds=all
--leak-check=full

```

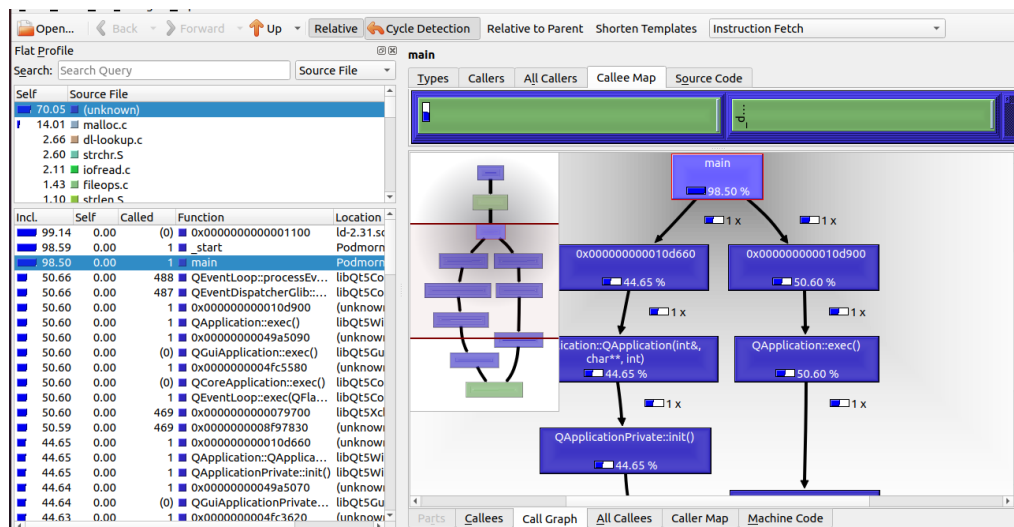
Nakon pokretanja alata kao izlaz smo dobili da postoji curenje memorije. Deo izvjestaja Memcheck-a prikazan je na slici 1 i 2. Kao što se može primetiti imamo više alociranja nego oslobađanja odakle sledi da je prisutno curenje memorije. Memcheck daje detaljan izveštaj koliko kog tipa curenja memorije je prisutno.



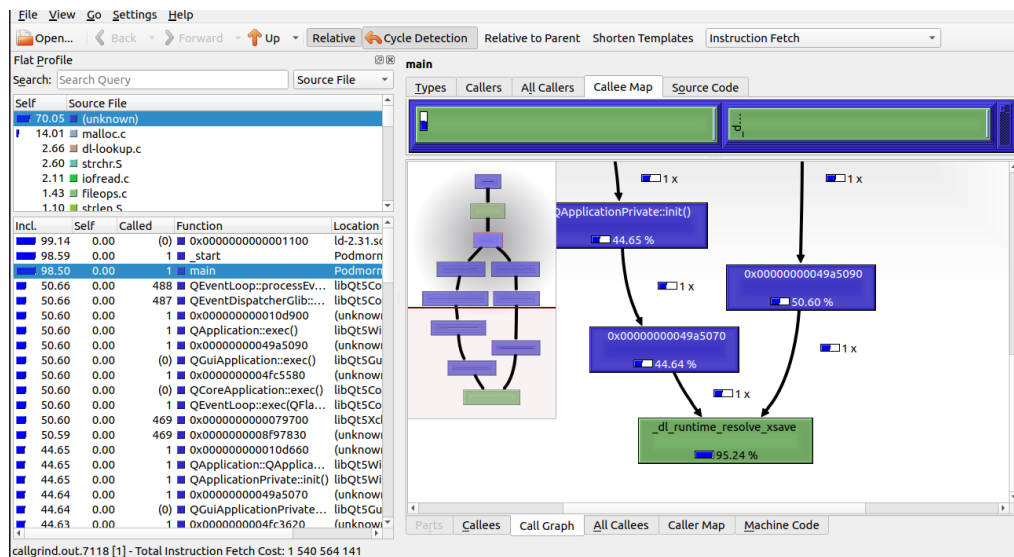
Slika 3: Callgrind

## 1.2 Callgrind

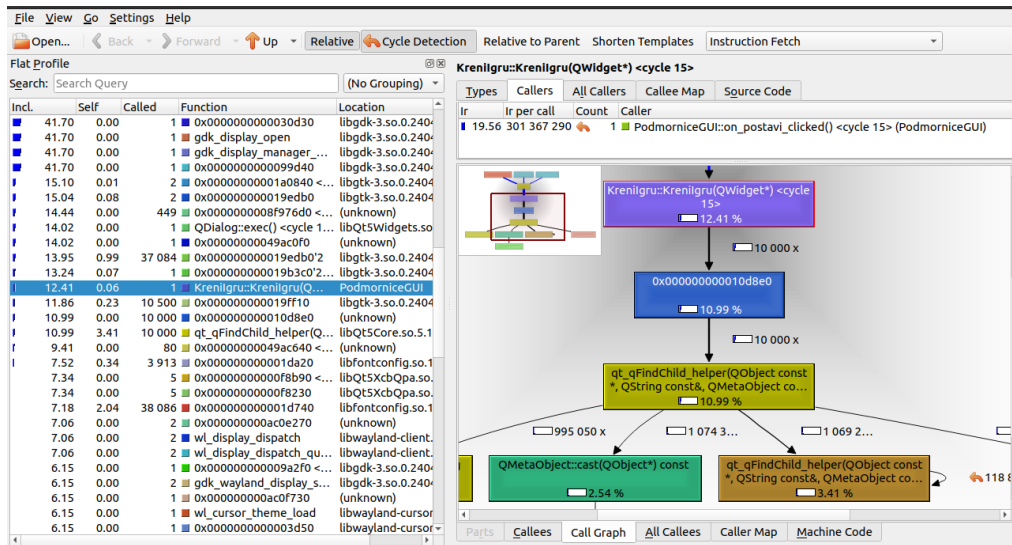
Callgrind je drugi alat koji je korišćen. On je takođe Valgrind alat. On generiše listu poziva funkcija u obliku grafa i računa cene funkcija. Cene funkcija se propagiraju. Sa grafa se lako može uočiti koje funkcije imaju najveću cenu. Za te funkcije treba razmotriti da li se mogu efikasnije implementirati kako bi se smanjila cena njihovog poziva. Alat se na sličan način kao i memcheck. Za grafičku reprezentaciju koristimo KCachegrind. Na slikama 3, 4, 5 i 6 se vidi deo rezultata callgrinda. Kao što se primećuje najviše vremena se troši na neke gotove funkcije na čiju popravku implementacije ne možemo uticati. To su uglavnom funkcije za gui. Funkcije koje su iz projekta koji posmatramo imaju mali udeo u odnosu na celokupno izvršavanje. Dosta funkcija nije ni prikazano jer imaju beznačajan udeo u odnosu na celokupno izvršavanje.



Slika 4: Callgrind



Slika 5: Callgrind



Slika 6: Callgrind

## 2 LCOV

Uz GCC dolazi alat gcov za analizu pokrivenost izvornog koda i alat za profilisanje izraz-po-izraz. Gcov ne daje previše čitljiv izveštaj pa koristimo lcov. Pri likom 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 PodmorniceGUI.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. 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

LCOV - code coverage report					
Current view: top level			Hit	Total	Coverage
Test: coverage-filtered.info			Lines:	1299	1595
Date: 2023-01-23 21:05:15			Functions:	45	70
			Branches:	2127	4699
Directory	Line Coverage	Functions	Branches		
/usr/include/c++/9/bits	100.0 %	6 / 6	-	0 / 0	64.3 %
/usr/include/x86_64-linux-gnu/bits	50.0 %	1 / 2	-	0 / 0	0.0 %
/usr/include/x86_64-linux-gnu/qt5/QtCore	70.2 %	99 / 141	94.4 %	17 / 18	44.5 %
/usr/include/x86_64-linux-gnu/qt5/QtGui	100.0 %	5 / 5	-	0 / 0	50.0 %
/usr/include/x86_64-linux-gnu/qt5/QtWidgets	100.0 %	5 / 5	-	0 / 0	50.0 %
PodmorniceGUI	82.4 %	1183 / 1436	53.8 %	28 / 52	44.9 %

Generated by: LCOV version 1.14

Slika 7: izveštaj za prvi primer

LCOV - code coverage report					
Current view: top level - PodmorniceGUI			Hit	Total	Coverage
Test: coverage-filtered.info			Lines:	1183	1436
Date: 2023-01-23 21:05:15			Functions:	28	52
			Branches:	1440	3207
Filename	Line Coverage	Functions	Branches		
gubitak.cpp	0.0 %	0 / 7	0.0 %	0 / 3	0.0 %
krenigrui.cpp	49.8 %	157 / 315	70.6 %	12 / 17	26.1 %
krenigrui.h	100.0 %	1 / 1	-	0 / 0	0.0 %
main.cpp	100.0 %	5 / 5	100.0 %	1 / 1	50.0 %
moc_gubitak.cpp	0.0 %	0 / 11	0.0 %	0 / 4	0.0 %
moc_krenigrui.cpp	59.1 %	13 / 22	100.0 %	4 / 4	30.0 %
moc_pobeda.cpp	0.0 %	0 / 11	0.0 %	0 / 4	0.0 %
moc_podmornicegui.cpp	63.6 %	14 / 22	75.0 %	3 / 4	35.0 %
pobeda.cpp	0.0 %	0 / 8	0.0 %	0 / 3	0.0 %
podmornicegui.cpp	100.0 %	25 / 25	80.0 %	4 / 5	64.3 %
ui_gubitak.h	0.0 %	0 / 16	0.0 %	0 / 1	0.0 %
ui_krenigrui.h	100.0 %	861 / 861	100.0 %	2 / 2	50.0 %
ui_pobeda.h	0.0 %	0 / 25	0.0 %	0 / 2	0.0 %
ui_podmornicegui.h	100.0 %	107 / 107	100.0 %	2 / 2	50.0 %

Generated by: LCOV version 1.14

Slika 8: izveštaj za prvi primer - PodmorniceGUI

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. U nastavku biće prikazane dve različite pokrivenosti za igricu podmornice.

Prvi put je pokrenuta igrlica i nasumicno odigrano nekoliko koraka i zatim je igrlica napuštena. Dakle, nismo odigrali do kraja samim tim nismo je završili ni uspešno ni neuspešno. Očekivano pokrivenost je mala. Mali broj funkcija je pozvan i mali broj grana je prođen jer je napravljeno svega nekoliko koraka. Ukupan izveštaj za ovaj primer dat je na slici 7 tj. na slici 8 gde je prikazana statistika za PodmorniceGUI.

Sa slike se može primetiti da u neke od fajlova nije ni ulaženo pa im je pokrivenost 0.0% ali to smo i očekivali. Kako ovaj primer nije reprezentativan primer izvršavanja, odnosno toka programa, jer ne očekujemo da će neko pokrenuti igricu i ugastiti je pre nego što dođe do kraja (izgubi ili podebi) u nastavku razmatrano bolji primer.

Primer broj 2 je primer u kome je igrlica odigrana do kraja i u njoj je pobedeno. Sada očekujemo bolju pokrivenost. Znamo da je ishod pobeda pa samim tim znamo da je u programu pozvan veći broj funkcija nego u prošlom primeru. Nisu

LCOV - code coverage report							
Current view: <b>top level</b>				Hit		Total	Coverage
Test: coveragef0.info				Lines:	1429	1595	89.6 %
Date: 2023-01-24 00:59:48				Functions:	53	70	75.7 %
				Branches:	2278	4699	48.5 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
/usr/include/c++/9/bits	100.0 % 6 / 6	- 0 / 0	64.3 % 9 / 14
/usr/include/x86_64-linux-gnu/bits	50.0 % 1 / 2	- 0 / 0	0.0 % 0 / 2
/usr/include/x86_64-linux-gnu/gt5/QtCore	70.2 % 99 / 141	94.4 % 17 / 18	47.5 % 522 / 1100
/usr/include/x86_64-linux-gnu/gt5/QtGui	100.0 % 5 / 5	- 0 / 0	50.0 % 10 / 20
/usr/include/x86_64-linux-gnu/gt5/QtWidgets	100.0 % 5 / 5	- 0 / 0	50.0 % 178 / 356
PodmorniceGUI	91.4 % 1313 / 1436	69.2 % 36 / 52	48.6 % 1559 / 3207

Generated by: LCOV version 1.14

Slika 9: izveštaj za drugi primer

LCOV - code coverage report							
Current view: <b>top level - PodmorniceGUI</b>				Hit		Total	Coverage
Test: coveragef0.info				Lines:	1313	1436	91.4 %
Date: 2023-01-24 00:59:48				Functions:	36	52	69.2 %
				Branches:	1559	3207	48.6 %

Filename	Line Coverage ↕	Functions ↕	Branches ↕
gubitak.cpp	0.0 % 0 / 7	0.0 % 0 / 3	0.0 % 0 / 6
kreniigru.cpp	77.8 % 245 / 315	76.5 % 13 / 17	45.3 % 208 / 459
kreniigru.h	100.0 % 1 / 1	- 0 / 0	- 0 / 0
main.cpp	100.0 % 5 / 5	100.0 % 1 / 1	50.0 % 3 / 6
moc_gubitak.cpp	0.0 % 0 / 11	0.0 % 0 / 4	0.0 % 0 / 6
moc_kreniigru.cpp	59.1 % 13 / 22	100.0 % 4 / 4	30.0 % 6 / 20
moc_pobeda.cpp	81.8 % 9 / 11	75.0 % 3 / 4	50.0 % 3 / 6
moc_podmornicegui.cpp	63.6 % 14 / 22	75.0 % 3 / 4	35.0 % 7 / 20
pobeda.cpp	100.0 % 8 / 8	66.7 % 2 / 3	50.0 % 5 / 10
podmornicegui.cpp	100.0 % 25 / 25	80.0 % 4 / 5	64.3 % 9 / 14
ui_gubitak.h	0.0 % 0 / 16	0.0 % 0 / 1	0.0 % 0 / 24
ui_kreniigru.h	100.0 % 861 / 861	100.0 % 2 / 2	50.0 % 1184 / 2368
ui_pobeda.h	100.0 % 25 / 25	100.0 % 2 / 2	50.0 % 23 / 46
ui_podmornicegui.h	100.0 % 107 / 107	100.0 % 2 / 2	50.0 % 111 / 222

Generated by: LCOV version 1.14

Slika 10: izveštaj za drugi primer - PodmorniceGUI

obiđeni delovi programa koji se dešavaju kada je ishod gubitak pa je očekivano da je za takve funkcije pokrivesnost 0.0%. Izveštaj pogledati na slikama 9 i 10.

### 3 Unit Test

Poslednji alat koji je korišćen je testiranje. Njega koristimo u kombinaciji sa prethodnim alatom jer je bitna pokrivenost testova. Cilj je da test ima što bolju pokrivenost. Za konkretan projekat čija se analiza radi, testiranje ne pokriva funkcije koje se pozivaju radi pravljenja gui-a. Samim tim pokrivesnot je loša obzirom da je u pitanju igrica koja koristi mnogo takvih funkcija. Testiranje je sprovedeno na funkcije koje se tiču samih postavki i korišćenja podmornica jer su one najpodložnije greškama. Takvih je 9 funkcija i sve su prošle testiranje. Formalno testiranje bi trebalo sprovesti na sve funkcije na koje može da se primeni i na sve grane. Međutim to u ovom slučaju nije urađeno jer ima dosta trivijalnih funkcija. Pokrivenost je urađena na kraju postupkom iz prethodnog odeljka a rezultati su smesteni u ReportsTestPodmornicaa. Rezultati testiranja su prikazani na slici 11. Pokrivenost je prikazana na slikama 12 i 13.

```

Application Output
Clang-Tidy and Clazy
TestPodmornica
(Red Hat 10.3.1-1)), ubuntu 20.04
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::cleanupTestCase()
Totals: 9 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms
***** Finished testing of TestPodmornica *****
05:27:49: /home/milica/Downloads/vs/2021_Podmornice-main/podmorniceGUI_Novo/PodmorniceGUI/build-TestPodmornica-
Issues 2 Search R... 3 Applicati... 4 Compile ... 5 QML Deb... 7 Version C... 8 Test Res...

```

Slika 11: Rezultati testiranja

LCOV - code coverage report

Current view: [top level](#) - /home/milica/Downloads/vs/2021\_Podmornice-main/podmorniceGUI\_Novo/PodmorniceGUI/TestPodmornica

Test: testPodmornicaafilterd.info

Date: 2023-01-25 05:50:33

Function

Branch

Filename	Line Coverage		Functions		Branches	
main.cpp	<div><div></div></div> 100.0 %	3 / 3	100.0 %	1 / 1	50.0 %	2 / 4
test_testpodmornica.cpp	<div><div></div></div> 99.4 %	167 / 168	91.7 %	11 / 12	52.3 %	227 / 434

Generated by: [LCOV version 1.14](#)

Slika 12: Pokrivenost testa

Current view: [top level](#) - /home/milica/Downloads/vs/2021\_Podmornice-main/podmorniceGUI\_Novo/PodmorniceGUI

Test: testPodmornicaafilterd.info

Date: 2023-01-25 05:50:33

Lines: 1981410

Functions: 936

Branches: 1743313

Filename	Line Coverage	Functions	Branches
gubitak.cpp	<div><div></div></div> 0.0 %0 / 7	0.0 %0 / 3	0.0 %0 / 8
kreniigrv.cpp	<div><div></div></div> 56.6 %198 / 350	52.9 %9 / 17	32.3 %174 / 539
pobeda.cpp	<div><div></div></div> 0.0 %0 / 8	0.0 %0 / 3	0.0 %0 / 12
podmornicegui.cpp	<div><div></div></div> 0.0 %0 / 26	0.0 %0 / 5	0.0 %0 / 32
ui_gubitak.h	<div><div></div></div> 0.0 %0 / 20	0.0 %0 / 2	0.0 %0 / 46
ui_kreniigrv.h	<div><div></div></div> 0.0 %0 / 865	0.0 %0 / 2	0.0 %0 / 2368
ui_pobeda.h	<div><div></div></div> 0.0 %0 / 25	0.0 %0 / 2	0.0 %0 / 60
ui_podmornicegui.h	<div><div></div></div> 0.0 %0 / 109	0.0 %0 / 2	0.0 %0 / 248

Generated by: [LCOV version 1.14](#)

Slika 13: Pokrivenost testa



## 4 Zaključak

Posmatrani projekat nema krucijalnih propusta. Suštinski je dobro implementiran. Neophodne popravke su popravke u radu sa memorijom odnosno neophodno je sanirati curenje memorije.