Analiza projekta kroz upotrebu alata za verifikaciju softvera

Praktični seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Tamara Stojković, 1091/2022
tamara.stojkovic.1998@gmail.com

2. septembar 2023.

Sažetak

U okviru ovog rada biće predstavljena analiza projekta KrabbyPattySecretFromula rađenog u okviru kursa Razvoj softvera na Matematičkom fakultetu, koji se nalazi na adresi https://gitlab.com/matf-bg-ac-rs/course-rs/projects-2021-2022/10-KrabbyPattySecretFormula, gde se u okviru README.md fajla mogu pronaći i detaljnije informacije o igrici, instalacije, pokretanje, autori, kao i demo snimak. Ovaj rad će sadržati analizu tog projekta, odnosno alate za verifikaciju softvera koji su primenjeni, način njihove primene, rezultate, eventualne pronađene bagove i curenja memorije i zaključke izvedene iz ove analize.

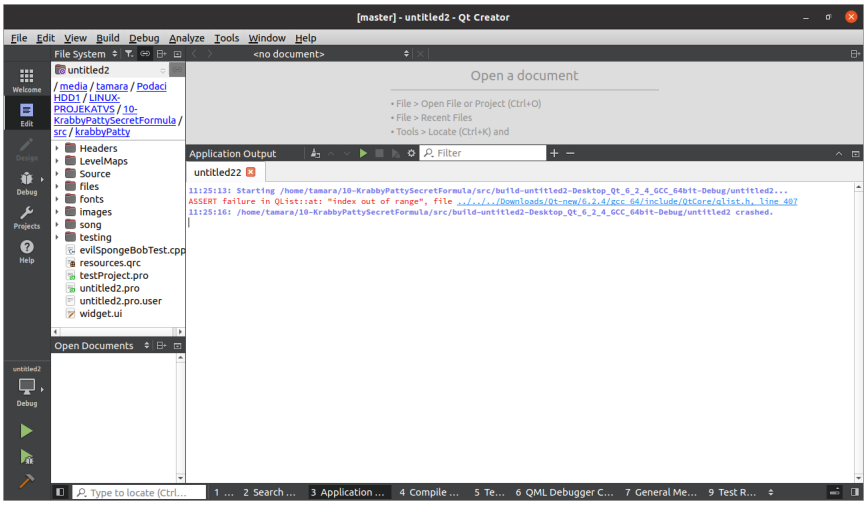
Sadržaj

1 Uvod

Projekat **KrabbyPattySecretFromula** je igrica koja prati Sunđer Boba na putu do pronalaska Tajnog recepta za Kebinu pljesku. On prolazi kroz 6 različitih nivoa, a završetkom svakog nivoa osvaja drugi sastojak za pljesku. Ukoliko Sunđer Bob ne uspe da završi nivo, Plankton je pobedio i osvojio tajni sastojak. Nakon izgubljena 3 života ne može da nastavi dalje svoj put. Igrica ima i teži i lakši režim igranja. Za projekat su korišćeni programski jezik C++ i biblioteka Qt >=6.2.2. - pri analizi koristila sam Qt 6.2.4. Na početku će biti prikazane početne greške koje su uočene prilikom pokretanja ovog projekta, debagovanje istih korišćenjem **Debbuger-a** i način na koji su greške rešene kako bi projekat mogao da se pokrene. Nakon toga biće prikazana primena alata: Clang-Tidy, Clazy, Clang-Format, Valgrind - Memcheck, Callgrind, Massif i GCov.

2 Debugger u okviru QtCreator-a

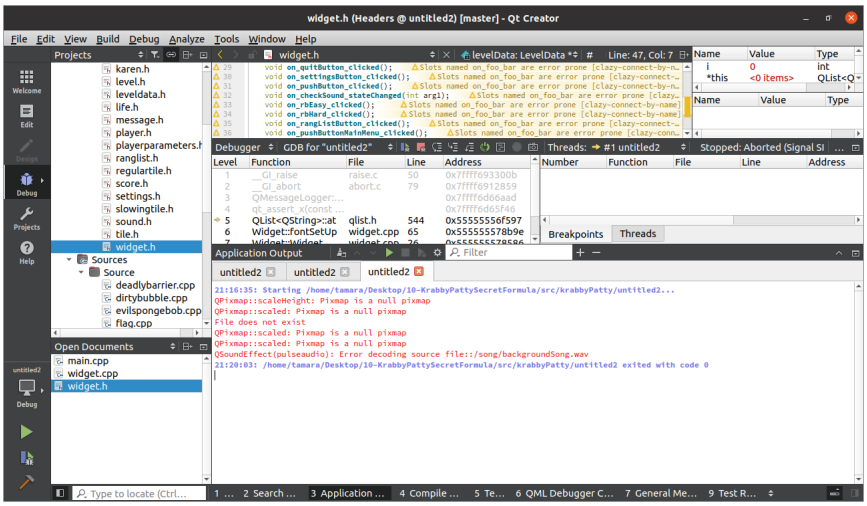
Projekat je preuzet i pokrenut po uputstvima iz README fajla sa linka. Učitan je .pro fajl, konfigurisan projekat u okviru Qt Creatora i nakon klika na dugme run dobijena je sledeća greška:



Slika 1: Početna greška

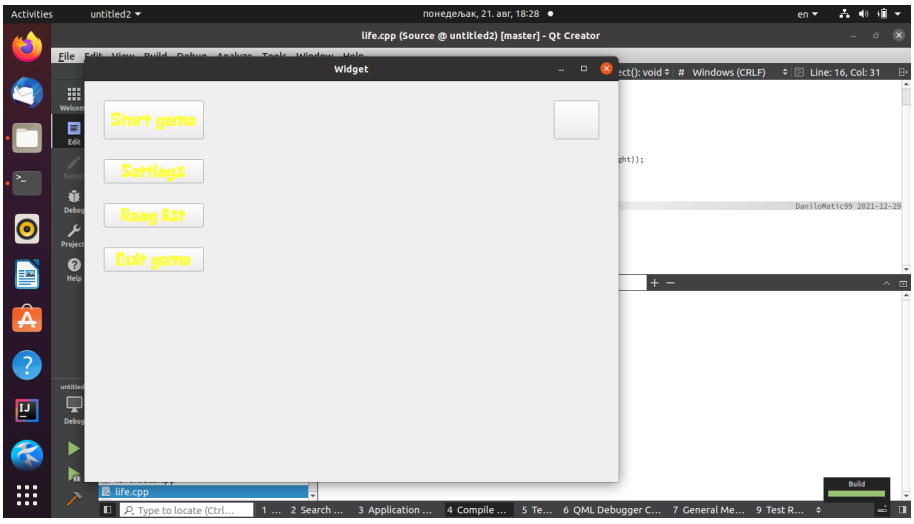
Iz ove greške može se zaključiti da se radi o pristupu elementu van granica liste, ali nije jasno koja lista, ni koji fajl pravi problem. Kako bi se oktrio fajl i struktura koja pravi problem, pokrenut je Debbuger u okviru Qt kreatora, postavljanjem breakpointa na prijavljenu liniju u okviru qlist.h fajla, odabirom opcije iz menija sa leve strane Debug –> Debugger i pokretanjem. Nakon čega je dobijeno da je greška u widget.cpp fajlu i funkciji fontSetUp i to na liniji gde je navedena putanja, što se može videti na slici 2.

Ovim je zaključeno da je problem to što se neispravno učitava putanja do fajla gde je font.Dodavanjem apsolutne putanje greška prestaje da se prikazuje, ali javljaju se upozorenja koja se mogu videti i na slici oblika QPixMap::scaled: Pixmap is a null pixmap ili File does not exist. Kada



Slika 2: Primena Debbuger-a

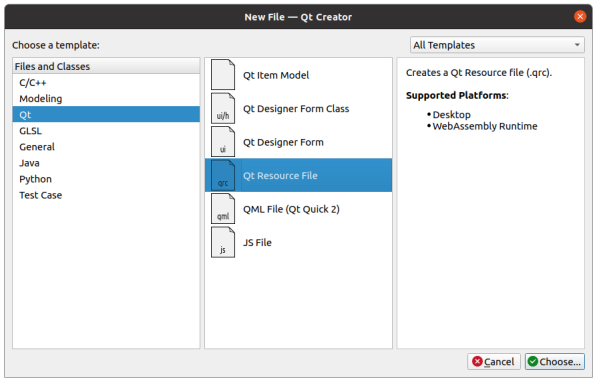
se ponovo pokrene program, uspešno se prikaže prozor igrice koji izgleda ovako:



Slika 3: Izgled prozora sa greškom

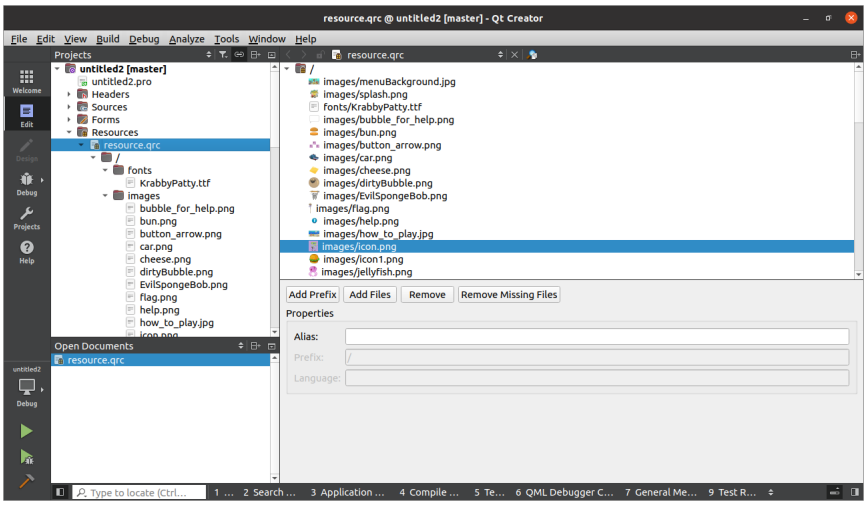
Na osnovu neispravnog izgleda prozora u kome je vidljiv samo tekst - formatiran nekim fontom i grešaka koje i dalje stoje kada se pokrene aplikacija (fajl ne postoji, Pixmap je null...) zaključuje se da je problem što su ostali fajlovi i slike i dalje nevidljivi iako je korišćen resources.qrc fajl koji bi trebalo da omogući nezavisnost ne samo od računara, već i od operativnog sistema (Na Windowsu sam uspešno pokrenula projekat, sa svim slikama i fajlovima vidljivim). Problem sam rešila brisanjem postojećeg resources.qrc fajla i kreiranjem novog fajla, kojima odgovaraju putanje

navedene u dokumentima. Koristila sam opciju desni klik na projekat, nakon toga Add New... i odabrala opciju sa slike:



Slika 4: Izor opcije za kreiranje novog fajla

Nakon ovoga kreirala sam koristeći Add Prefix i Add Files novi fajl resoure.qrc koji je prikazan na sledećoj slici:



Slika 5: Novi fajl - rešenje problema

3 Clang-Tidy i Clazy

Clang sa propratnim alatima (eng. *Clang Tools*) predstavlja jedan od najbitnijih delova LLVM projekta otvorenog koda. Clang je kompilator za jezike C, C++, Obejctive C... Zapravo, clang predstvalja frontend koji kao ulaz uzima kod koji je napisan u nekom od navedenih jezika i prevodi ga u međureprezentaciju tj. LLVM IR (to je ulaz za srednji deo - gde se vrše optimizacije nezavisne od jezika i arhitekture). Na kraju backend vrši

optimizacije vezane za konkretnu arhitekturu i prevodi kod na mašinski jezik. Implementiran je u C++ korišćenjem modernijih tehnologija.

Alat Clang-Tidy je deo Clang/LLVM projekta koji nam automatski refaktoriše C++ kod. Ova grupa alata se naziva linteri(linter). To su alati koji analiziraju kod i pronalaze programske i stilske greške u kodu. Motivacija za upotrebu dolazi sa uvođenjem novih standarda C++-a. Sa standardom C++11 unete su mnoge nove funkcionalnosti, kao što su na primer auto, override, lambda... Dosta programera i dalje ne koristi ove nove standarde u pisanju svojih kodova, pa postoji i dosta kodova nad kojima se može primeniti ovaj alat, tako što refaktoriše kod da koristi novije funkcionalnosti. Sve u svemu : Clang-Tidy omogućava dijagnostiku i pronalazi ispravke za tipične programske greške kao što su greške u radu sa interfejsima ili nekonzistentan stil prilikom pisanja koda. Deo ovog alata je statički analizator Clang.

Clazy je alat koji pomaže Clangu da razume Qt semantiku.On prikazuje upozorenja kompajlera vezana za Qt, počev od nepotrebne alokacije memorije do zloupotrebe API-ja. Ima akcije refaktorisanja za rešavanje nekih problema (eng. *issues*).

QtCreator integriše ove alate, pa će u nastvaku biti prikazana njiova upotreba u okviru njega.

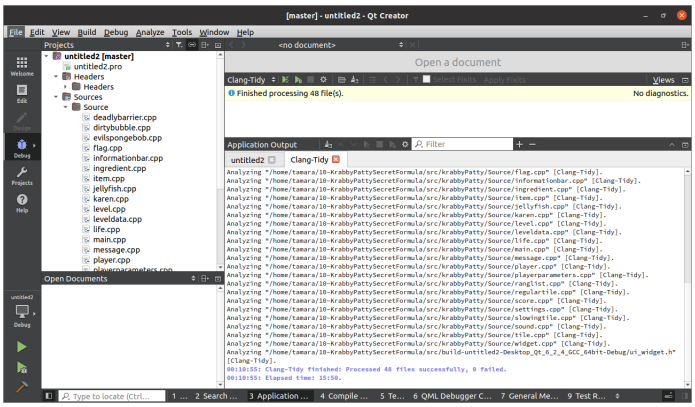
3.1 Pokretanje Clang-Tidy i Clazy

Da bih pokrenula Clang-Tidy ili Clazy za analizu trenutno otvorenog projekta primenila sam sledeče korake:

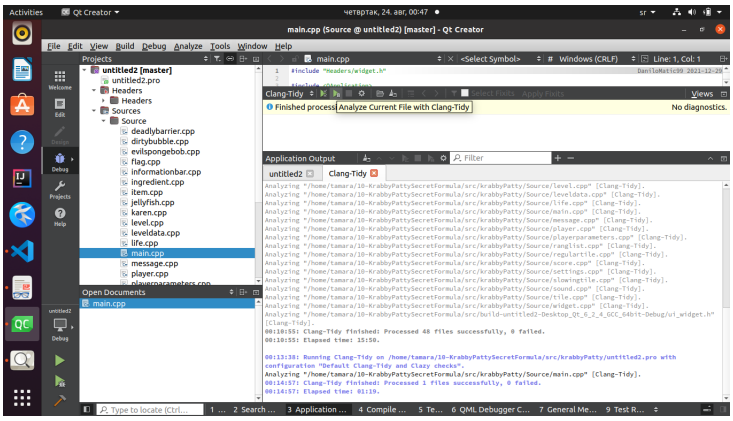
- Odabir **Analyze** i nakon toga iz padajućeg menija odabrala sam **Clang-Tidy ili Clazy**
- Izabrala sam fajlove nad kojima želim da izvršim statičku analizu (svi fajlovi u projektu ili određeni)
- Kliknula sam Analyze za počinjanje provera.
- Može se selektovati Debug sa strane (u mode selektoru) i onda odabrati Clang-Tidy/Clazy, nakon čega se mora pritisnuti start dugme, kako bi se otvorio prozor Files to Analyze.
- Na slici 6 se može videti prikaz Clang-Tidy (prikazuje probleme - issues, gde se dvostrukim klikom na issue može ući na taj problem u editoru).
- Na slici 7 se može videti kako se može odabrati opcija za primenu na samo jedan tekući otvoreni fajl iz projekta.
- Ovo podrazumeva pokretanje sa **Default Clang-Tidy and Clazy checks** konfiguracijom.

Takođe, mogu se konfigurisati određene(custom) dijagnostike globalno za Clang alate. Podešavanja sam uradila na sledeči način:

1. Odabir **Edit** –> **Preferences** –> **Analyzer** –> **Clang Tools**
2. U Clang-Tidy i Clazy poljima potrebno je da stoji putanja do odgovarajućih exe fajlova koji koriste (automatski učitano obično) (slika 8)
3. Clang Tools ne zahtevaju da se uradi **build** projekta pre analize, ali mogu da izbace neka upozorenja o nedostajućim fajlovima koji se generišu tokom builda. Za velike projekte je upravo zato ovo preporuka, pa sam izvršila build.

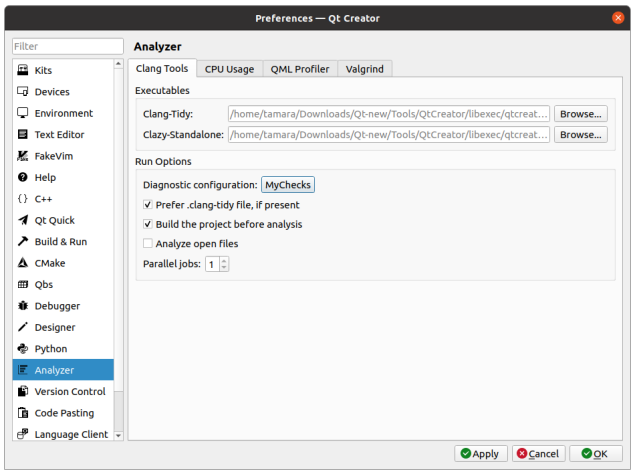


Slika 6: Clang-Tidy - custom checks

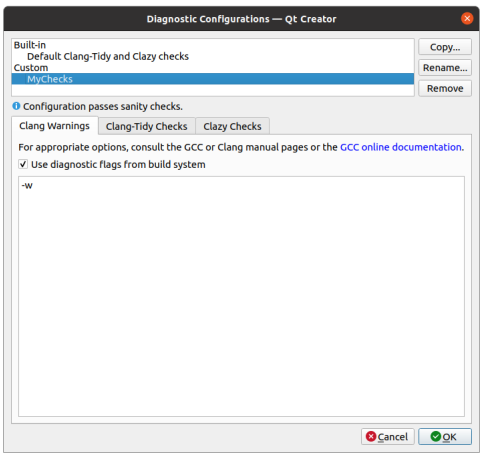


Slika 7: Clang-Tidy - analiza trenutnog fajla

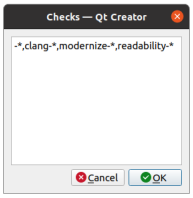
4. Potrebno je odčekirati i **Analyze open files** opciju, da se onemogući automatska analiza otvorenih dokumenta.
5. U **Diagnostic configurations** prozoru, prikazuje se check koji se izvršava i nakon toga klikom na vrednost otvara se prozor, gde se klikom na opciju Copy otvara prozor u kome sam unela ime konfiguracije u polju Diagnostic configuration name.
6. U **Clang Warnings** tabu, treba selektovati **Use diagnostic flags from the build system** da bi se prosledili flegovi, kao na primer warning flagovi, za prikaz anotacija u editoru. (slika 9)
7. U **Clang-Tidy checks** tabu, treba selektovati **Select checks** i tu sam odabrala provere koje želim da primenim, što se vidi na slici.(slika 10)
8. U **Clazy checks** tabu, treba selektovati nivo provera koji će se primeniti i tu sam odabrala provere level 0.(slika 11 gde se mogu videti provere za nivo)
9. Nakon toga treba ići na Apply i Ok, pa pokrenuti ove alate. Izlaz se vidi na slici. (slika 13 za Clang-Tidy i slika 12 za Clazy)



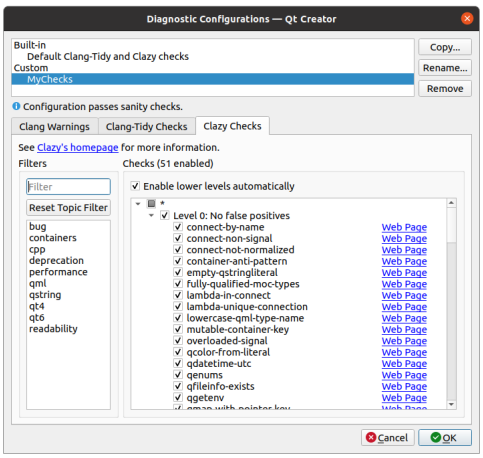
Slika 8: Pokretanje - početni prozor



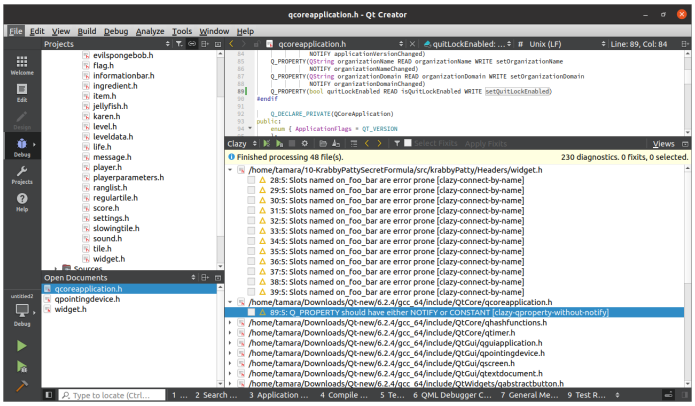
Slika 9: Selektovanje opcija



Slika 10: Odabrane provere



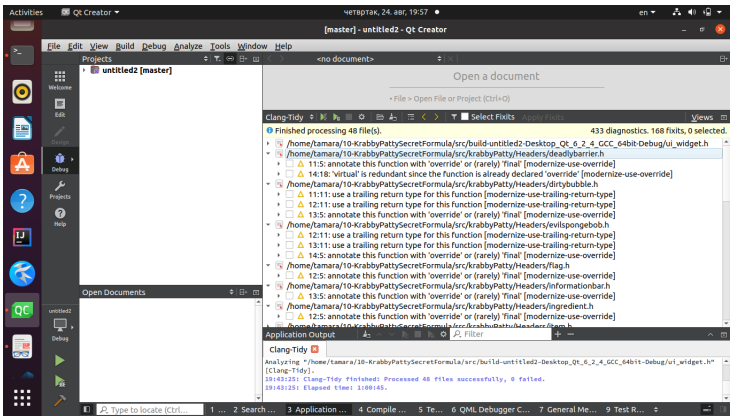
Slika 11: Odabrane provere za Clazy



Slika 12: Dobijeni izlaz - Clazy

10. Za kraj je iskorišćena naredba koja je i u fajl .clang-tidy stavila rezultate: **clang-tidy -checks=’-*,clang-*,modernize-*,readability-*’ -dump-config > .clang-tidy**

Sa slike za Clang-Tidy može se zaključiti da sam uvođenjem novih opcija u okviru podešavanja sada dobila upozorenja u vidu dijagnostika i ispravki, gde se prikazuju sve preporuke za refaktorisanje koda koje alat preporučuje za svaki fajl projekta. Moguće je primeniti sve ili samo neke od preporuka. Može se takođe uočiti da su uglavnom to preporuke za poboljšanje koda, u vidu anotacije funkcija sa override ili tipa povratne vrednosti, implicitne konverzije i slično. Zaključak je da u okviru projekta, iako je korišćen C++11, nisu i sve nove funkcionalnosti, koje se korišćenjem ovog alata mogu primeniti kako bi se kod refaktorisao i modernizovao. Sa slike za Clazy može se zaključiti da se javljaju neka upozorenja i dijagnostike uglavom u header fajlovima. Dva upozorenja koja su se javila se mogu videti na slici, prvo se odnosi na to da je korišćen "Go to slot"u okviru qt creatora da bi se kreirali slotovi automatski, a kako bi se izbeglo ovo upozorenje mogu se koristiti ručne signal-slot konekcije. U drugom



Slika 13: Dobijeni izlaz - Clang Tidy

upozorenju, notify signal je opcioni i dobija se ovo upozorenje, kojeg se možemo osloboditi na primer dodavaljem signala na sledeći način:

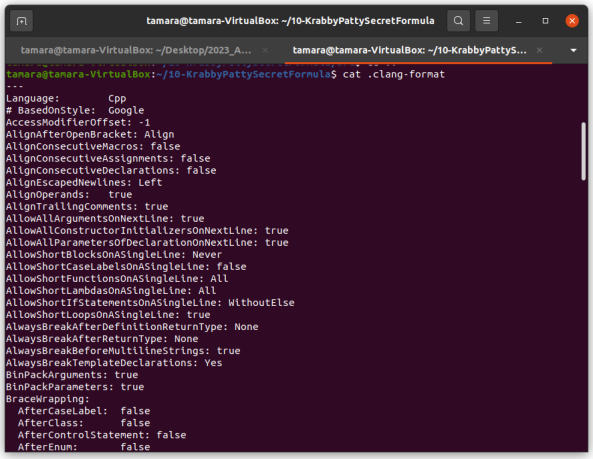
```
Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged
    )
//Deklarisanje signala, a iznad dodat NOTIFY
signals:
    void nameChanged(int newName);
```

4 Clang-Format

Alat Clang-Format omogućava automatsko formatiranje koda. Postoji više različitih dostupnih stilova npr. LLVM, Google, Chromium, Mozilla, WebKit, Microsoft, GNU. Može se iskoristiti bilo koji od ovih stilova, može se kreirati sopstveni stil, a može se i iskoristiti neki vec postojeći kao osnova koja se može menjati radi kreiranja sopstvenog stila. Ja sam za formatiranje izabrala Google format, uz neke minimalne izmene koje sam napravila. Odabir osnovnog stila vrši se komandom **clang-format -style=google -dump-config > .clang-format** . Time se dobija fajl .clang-format koji se može menjati komandom **gedit .clang-format** radi kreiranja sopstvenog stila.Deo .clang-format fajla može se videti na slici 14. Minimalne izmene koje sam napravila su:

- TabWidth: 8 na TabWidth: 4
- SpacesInParentheses: false na SpacesInParentheses: true
- SpacesinSquareBrackets: false na SpacesinSquareBrackets: true

Ovaj alat nema opciju run kojom bi se odmah primenio na ceo projekat(može na jedan fajl samo), pa sam kreirala python skriptu koja prolazi kroz sve fajlove .hpp i .cpp i primenljuje clang-format. Ova skripta se nalazi u direktorijumu za clang-format, a pokreće se komandom **python3 run-clang-format.py . file** Na ovaj način može se uočiti gledanjem fajlova, da je svaki fajl projekta uspešno formatiran, uz manje ili veće razlike, s tim što je uočeno da je kod dosta dosledno napisan, izmene su minimalne, pa je moguće da je kod ili napisan doslednim stilom ili korišćen već neki od alata za formatiranje koda.



Slika 14: Deo clang-format fajla za Google stil

5 GCov

GCov je alat koji služi za određivanje pokrivenosti koda prilikom izvršavanja programa (eng. *codecoverage*). Koristi se da bi se analizirao program i utvrdilo na koji način se može kreirati efikasniji program i da bi se utvrdila pokrivenost koda testovima, kao i koji delovi koda su koliko pokriveni.Zarad lepše reprezentacije rezultata koristi se **alat lcov**. Ovaj alat sam primenila na projekat, kao i na napisane testove koji već postoje u okviru projekta. Prvo ću navesti način primene ovog alata, a nakon toga i rezultate za oba slučaja i zaključke koje sam izvela.

- Na početku u .pro fajl testova/projekta dodala sam sledeće linije:

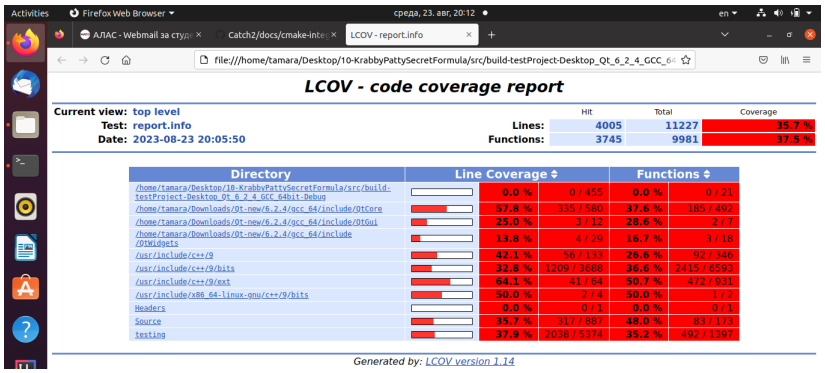
```
QMAKE_CXXFLAGS += --coverage
QMAKE_LFLAGS += --coverage
```

- A nakon build-ovanja testova/projekta pokrenula sledeće komande u build folderu, čime se generiše izveštaj report.info koji nije baš čitljiv:

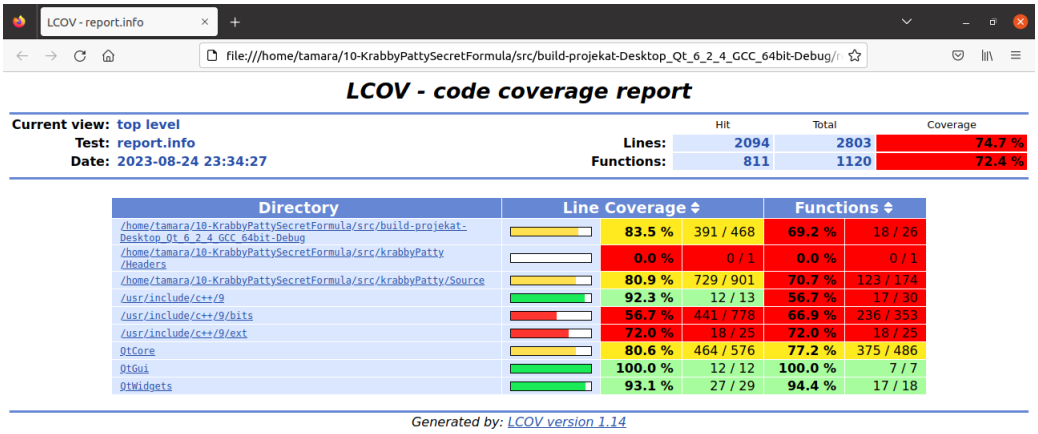
```
lcov --capture --directory . --output-file report.info
```

- Nakon ovoga uočavamo da su generisani .gcno i .gcda fajlovi na mestu gde se nalaze objektni fajlovi.Ti fajlovi se ne otvaraju direktno, već uz pomoć alata lcov koji ih analizira i generiše izveštaj.
- Nakon pokretanja komande **genhtml -o result report.info** u folderu result se nalazi izveštaj. Otvaramo taj index.html dokument koji se može otvoriti u browseru, što omogućava čitljivu i detaljnu analizu pokrivenosti koda.
- Rezultat za testove se može videti na slici 15, dok se rezultat za primenu na ceo projekat može videti na slici 16 i za .cpp fajlove na slici 17.

Na osnovu ovog izveštaja jasno se vidi da je pokrivenost testovima jako mala (za linije 35.7, a za funkcije 37.5) i da ima mesta za popravke



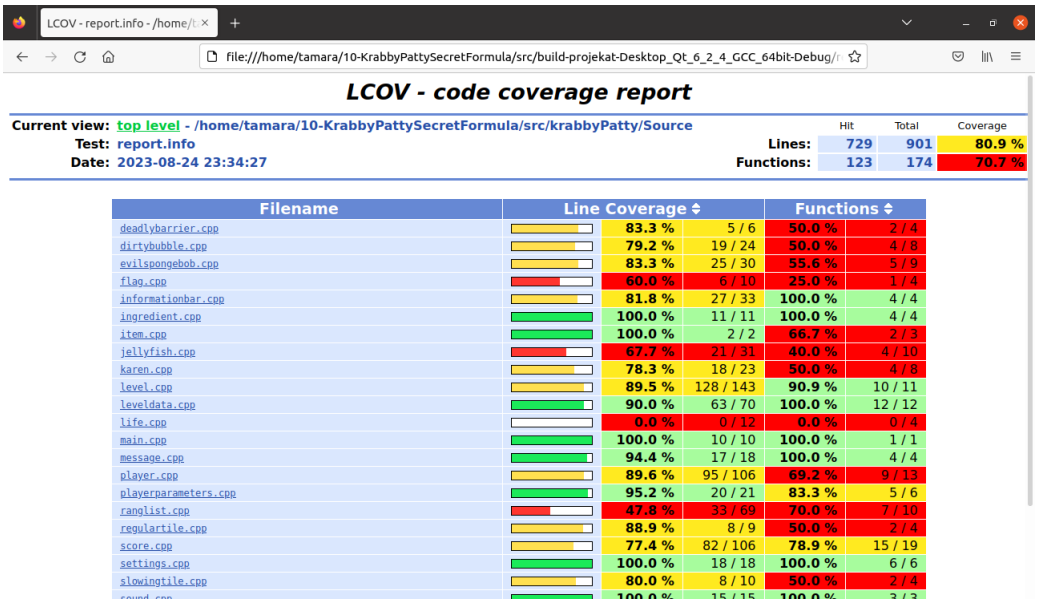
Slika 15: Izveštaj testovi



Slika 16: Izveštaj ceo projekat

testiranja i da bi trebalo napisati dodatne testove, kako bi projekat bio pokriveniji testovima.Postojanje testova ipak predstavlja vid poboljšanja, u odnosu na to kad ne bi postojali.

Što se tiče izveštaja za ceo projekat, može se uočiti da je pokrivenost prilično dobra, iako naravno treba težiti pokrivenosti koja je što bliža 100 procenata, nije moguće pri svakom pokretanju programa dostići sve fajlove i funkcije u njima. Procenti su ipak prilično dobri sa obzirom na to da su neke funkcije korišćene za provere, a da neke nisu ni dostignute. Na drugoj slici može se videti detaljni izveštaj i za cpp fajlove. U ovom slučaju odigran je jedan nivo igrice i izgubljeni su životi, usled čega je igrica prekinuta - nije se završila.Većina fajlova ovde pokazuje dobre procente pokrivenosti, a upravo navedeni razlozi su posledica manje pokrivenosti u fajlovima kao što je na primer ranglist.cpp - samo je pregledana ranglista, ali igrač se nije upisao na nju u ovom slučaju -nije samim tim pokrivena ta funkcija i linije.



Slika 17: Izveštaj samo za cpp fajlove u projektu

6 Valgrind alati

Valgrind je profajler otvorenog koda koji nadgleda funkcionisanje određenog programa i prijavljuje nepravilnosti u radu tog programa ako postoje. Distribucija Valgrinda sadrži sledeće alate: **Memcheck**(detektor memorijskih grešaka), **Massif** (praćenje rada dinamičke memorije), Callgrind (profajler funkcija), **Cachegrind** (profajler keš memorije), **Hellgrind i DRD** (detektori grešaka u radu sa nitima). Ovi alati funkcionišu po metodi bojenja vrednosti, gde zapravo svaki registar i memorijsku vrednost boje(zamenjuju) sa vrednošću koja nam govori nešto drugo o orginalnoj vrednosti. U nastvaku biće prikazana primena 3 Valgrindova alata : Memcheck, Callgrind i Massif.

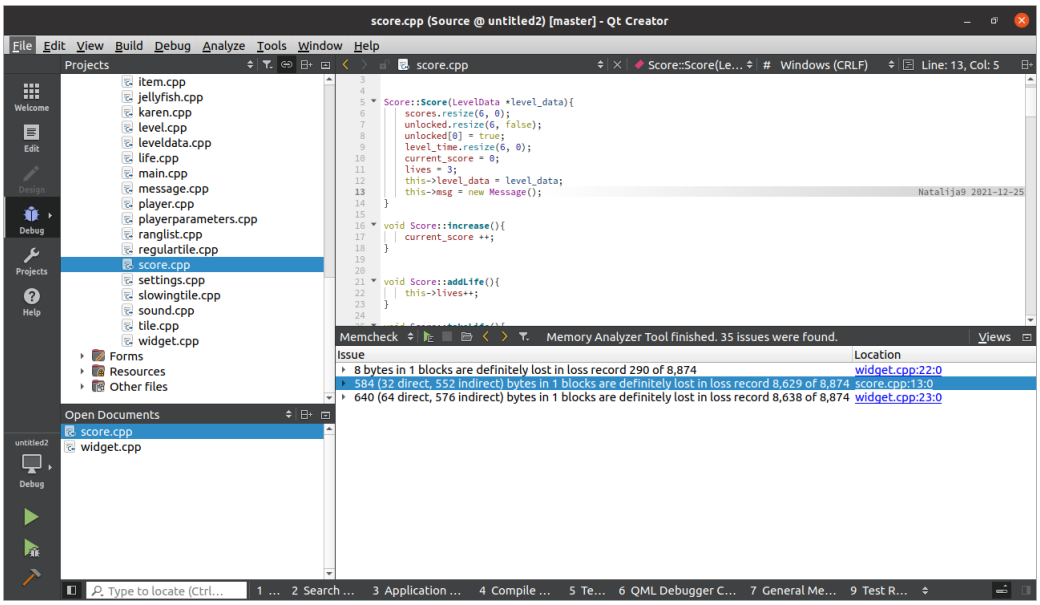
6.1 Memcheck

U slučaju kada se analizira Qt projekat, alat Memcheck može se pokrenuti direktno iz Qt Creatora ili komandom u terminalu. Ovde će biti prikazana prva opcija. Razvojno okruženje Qt Creator ima opciju Analyze kao padajući meni. Tu se odabere jedna od opcija - u ovom slučaju Valgrind Memory Analyzer za analizu koda. Time se otvara Memcheck alat i potrebno je kliknuti na Start za startovanje analize. Prvi rezultat analize koji sam dobila je dat na slici 18.

Iz prve analize može se izvući sledeći zaključak: problem curenja memorije se javlja u neredna 3 fajla što se vidi i sa slike 18.

```
widget.cpp
widget.cpp (druga greška)
score.cpp
```

Prvo je rešena greška u fajlu score.cpp i klikom na izbačenu grešku, vidi se da se radi o liniji 13, odnosno o msg objektu.Uočeno je da u destruktoru



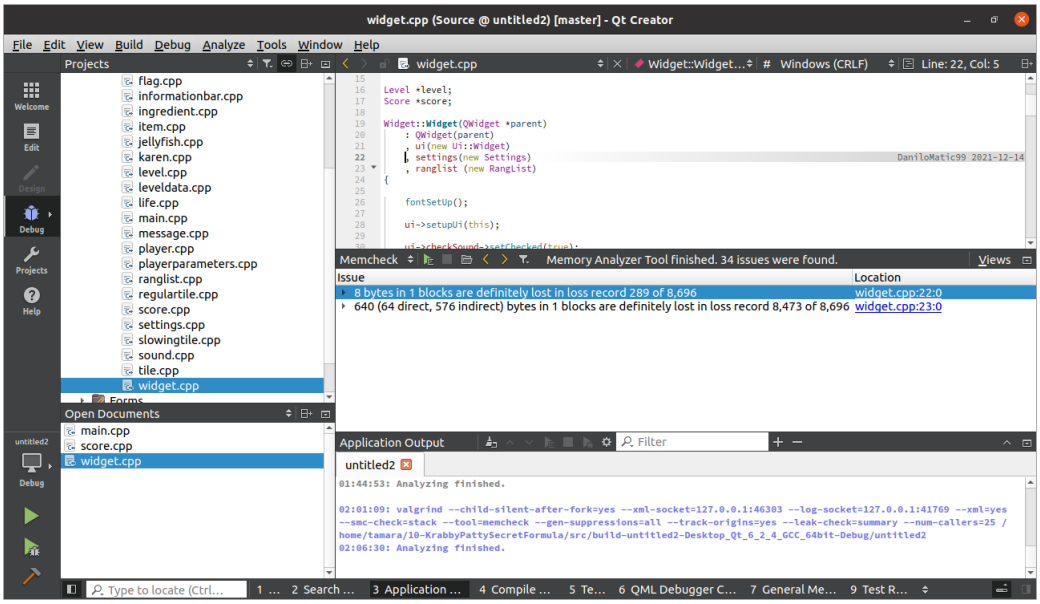
Slika 18: Memcheck - prva analiza

za klasu Score potrebno dealocirati taj objekat, pa je dodat poziv **delete msg** i na taj način je rešen problem curenja memorije za ovaj fajl. Ponovo je pokrenuta analiza na isti način i ostaju greške u fajlu widget.cpp, što se može videti sa slike 19.

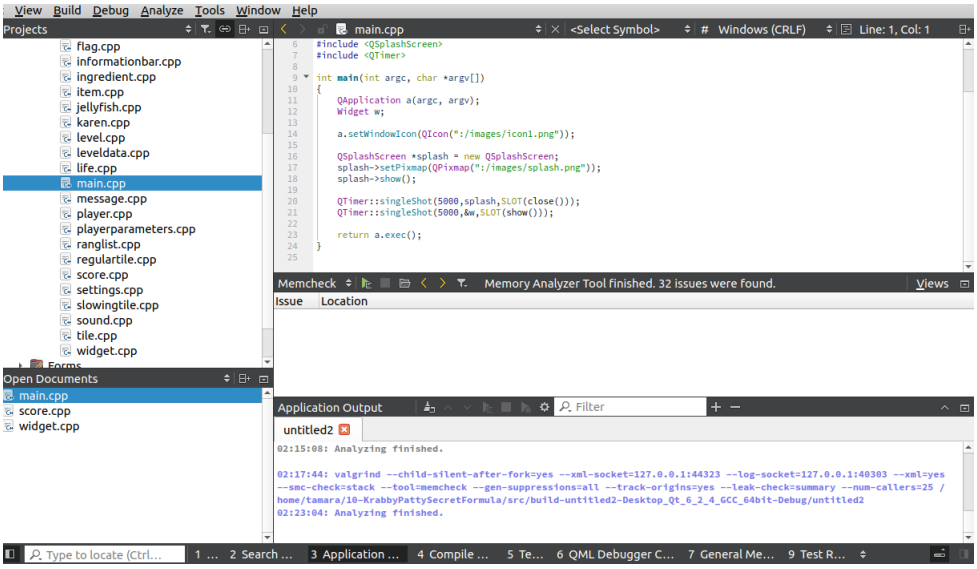
Klikom na izbačenu grešku u fajlu widget.cpp, vidi se da se radi o linijama 22 i 23, odnosno o objektima klase Settings i klase RangList. Uočeno je da u destruktoru ne postoji dealokacija memoriju za te objekte, pa je rešen ovaj problem dealokacijom ovih objekata u destruktoru na sledeći način:

```
1000 Widget::~Widget(){
         ...
1002     delete ranglist;
         delete settings;
1004 }
```

Izvršena je finalna analiza, nakon čega se može videti da su rešeni navedeni problemi sa curenjem memorije u programu na slici na narednoj strani 20.



Slika 19: Memcheck - druga analiza

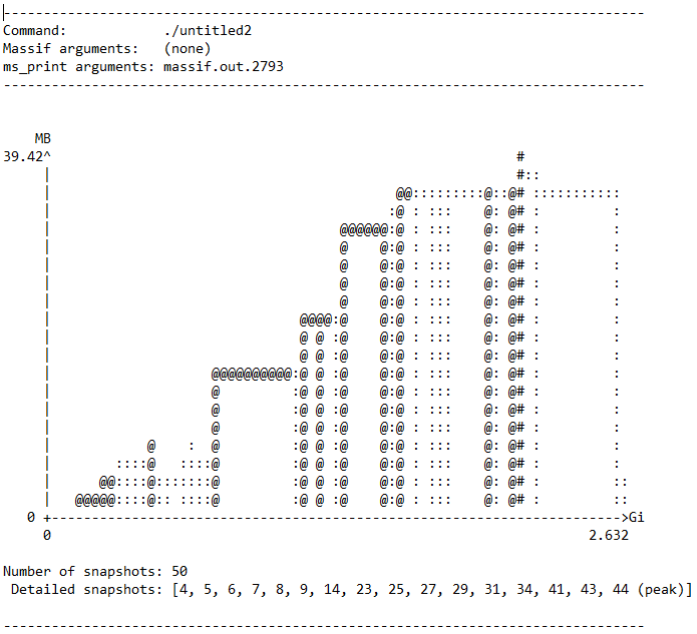


Slika 20: Memcheck - finalna analiza

6.2 Massif

Alat Massif kao što je već rečeno, služi za praćenje rada dinamičke memorije, odnosno za analizu hip memorije i takođe će biti primenjen na projekat. Služi za otkrivanje druge vrste curenja memorije, odnosno slučaj kada referenca na neki objekat i dalje psotoji, ali se taj objekat ne koristi, što dovodi do bespotrebnog trošenja velike količine memorije ovakvih programa. Da bih pokrenula Massif, prethodno sam izgradila projekat, i u terminalu izvršila sledeće komande:

- **valgrind -tool=Massif ./untitled2**
- Kao rezultat sam dobila nečitljiv fajl **massif.out.2793** pa sam pokrenula sledeću komandu kako bih dobila čitljiviju reprezentaciju u txt formatu: **ms_print massif.out.2793 > massif.txt**



Slika 21: Graf massif

Sa slike se može zaključiti da je alat Massif napravio 50 preseka stanja (eng. *Number of snapshots*). Takođe, možemo ispod toga uočiti da je izdvojio neke određene (eng. *Datiled snapshots*). U ovoj listi se u zagradi vidi da se pik u potrošnji memorije dostiže u preseku 44.

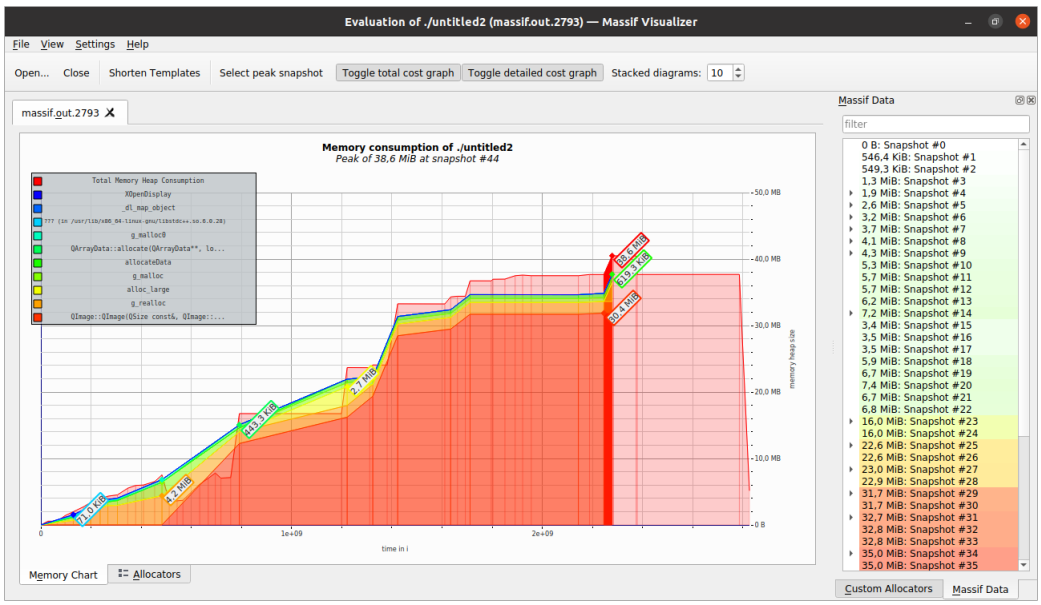
Poredeći ovih par preseka (22 , 23) sa presekom gde se desio pik, možemo zaključiti da potrošnja memorije postepeno raste i da je očekivano najveća u preseku 44, kada se i dostiže pik.Podaci o ostalim presecima se mogu naći u priloženom fajlu. I ostali preseci mogu se videti u fajlu .txt u direktorijumu za massif. Može se zaključiti da memorija u programu postepeno raste. Kako bih dobila bolji vizuelni prikaz, koristila sam Massif Visualizer, u koji sam učitala generisani massif.out.2793 fajl i dobila prikaz na slici 24 gde se vidi već zaključeno i takođe da je pik na 44 preseku i iznosi 38.6 MB. Dodata je u folder slika pokretanja nad testovima ovog alata, gde se vidi značajno manji utrošak memorije kad se eliminiše neki deo za Qt.

n	time(i)	total(B)	useful-heap(B)	extra-heap(B)	stacks(B)
35	1,795,948,841	37,462,960	36,718,741	744,219	0
36	1,802,687,955	37,716,096	36,960,155	755,941	0
37	1,850,573,922	37,738,376	36,981,570	756,806	0
38	1,891,269,814	38,325,632	37,495,503	830,129	0
39	1,921,150,658	38,419,672	37,583,431	836,241	0
40	1,955,615,816	38,335,088	37,499,790	835,298	0
41	2,143,479,052	38,335,088	37,499,790	835,298	0

Slika 22: Massif neki preseci

n	time(i)	total(B)	useful-heap(B)	extra-heap(B)	stacks(B)
44	2,278,203,642	41,333,080	40,496,889	836,191	0

Slika 23: Massif -pik presek



Slika 24: Massif Visualizer prikaz

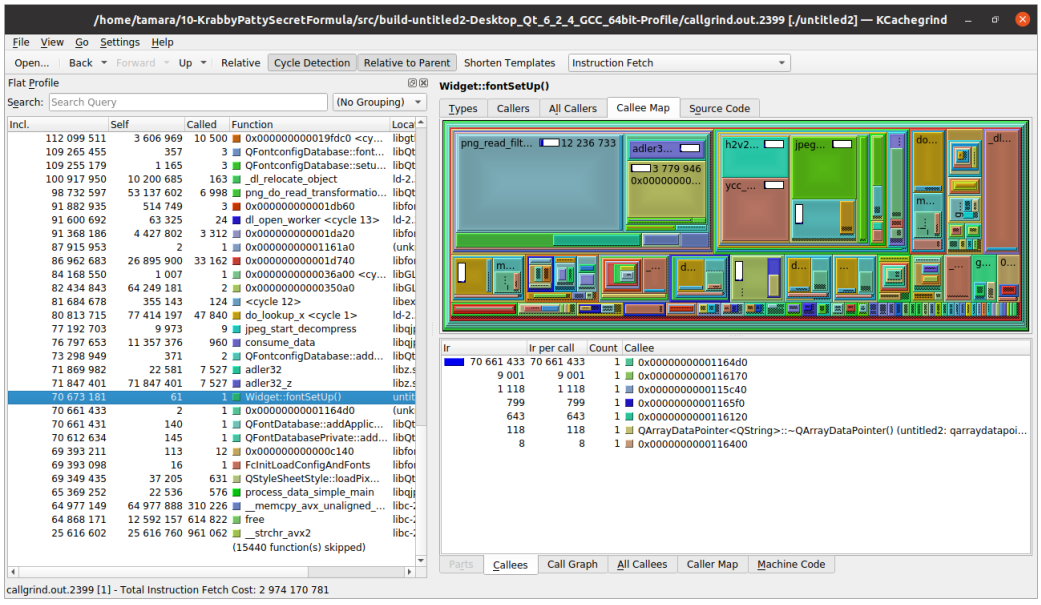
6.3 Callgrind

Callgrind je alat za profajliranje koji čuva istoriju poziva funkcija u programu kao graf poziva. Neke od informacija koje možemo dobiti za dati program su:

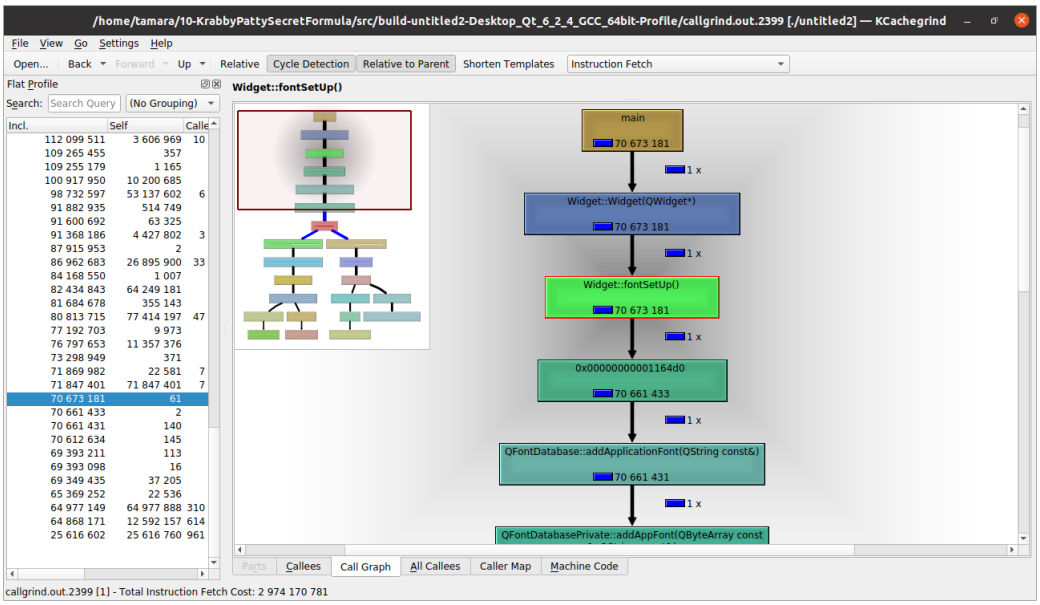
- broj izvršenih instrukcija
- odnosi izvršenih instrukcija sa odgovarajućim linijama koda
- caller/callee odnos između funkcija (sa frekvencijama)
- informacije o keširanju (promašaji, pogodci, ...)

Osnovna ideja iza profajlera je da prikaže koji delovi koda (npr. funkcije) najviše utiču na performanse programa (vreme, memorija, ...). Kada bi se uočilo da se veliki procenat izvršavanja programa svodi na jednu funkciju, onda je to znak da je u pitanju dobro mesto za optimizaciju.

Kako bih primenila ovaj alat za početak generisala sam profile verziju programa prilikom izgradnje. Nakon toga pozvala sam komandu u tom direktorijumu :
valgrind –tool=callgrind –log-file="report-callgrind"./untitled2
Nakon toga, kao rezultat dobila sam fajl callgrind.out.2399 gde je PID=2399 i označava identifikator procesa. KCachegrind je pomoćni alat za vizuelizaciju podataka dobijenih iz callgrind analize. Koristila sam ga kako bih otvorila generisani fajl i vizuelno videla rezultate. Na slici 25 i 26 sam prikazala neke od informacija za konkretnu funkciju fontSetUp.



Slika 25: Prikaz Callee Map i Callee za fontSetUp funkciju



Slika 26: Prikaz Call Graph za funkciju fontSetUp

7 Zaključak

U okviru ovog projekta izvršena je detaljna analiza projekta KrabbyPattySecret formula i uočeni su oni dobri delovi, kao i delovi koji se mogu poboljšati i unaprediti. Takođe, uočena su moguća unapređenja koda, u skladu sa novim standardima, curenja memorije i njihovo rešavanje i to su svi problemi koje treba uočiti i rešiti, jer neki mogu imati neželjene posledice po projekat.Primenjeni su alati za razvoj softvera, koji su krucijalni segment razvoja softvera. Svaki deo sadrži opis alata, način i korake primene, kao i rezultate koji su dobijene, moguća rešenja i izvedene zaključke. Svi dodatni fajlovi i slike mogu se naći u okviru ovog projekta, u posebnom direktorijumu za svaki alat.