Analiza projekta MATFWars

Projekat iz Verifikacije softvera

Milica Kužet 1054/2024

September 16, 2025

Sadržaj

1 Uvod

MATFWars projekat predstavlja implementaciju matematičkih igrica koje na zabavan način podstiču razumevanje i učenje funkcija i njihovih grafova. Moguće je izabrati jednu od dve igrice: War game i Guessing game. War game predstavlja mulitplayer igricu gde je cilj odgovarajućom matematičkom funkcijom pogoditi protivnika uz prepreke. Guessing game predstavlja singleplayer igricu različitih težina, gde je cilj pogoditi iscrtanu funkciju u određenom vremenu. Za potrebe kursa *Verifikacija softvera*, izvršena je analiza projekta MATFWars tako što su primenjeni alati:

- Cppcheck: alat za statičku analizu koda koji pomaže u otkrivanju potencijalnih grešaka, problema sa stilom, neiskorišćenih promenljivih i drugih problema u kodu pre nego što se on kompajlira i izvrši.
- Memcheck: alat u okviru Valgrind-a koji služi za detektovanje problema sa memorijom u C i C++ programima.
- Massif: alat u okviru Valgrinda koji analizira korišćenje heap memorije u programu.
- ClangFormat: alat koji automatski sređuje kod prema unapred zadatim pravilima. Koristi se u raznim programskim jezicima kako bi kod bio lakši za čitanje i održavanje.

2 Cppcheck

Cppcheck predstavlja alat za statičku analizu koda, specijalizovan je za programske jezike C i C++. Za razliku od samog kompajlera koji proverava samo sintaksu i osnovna pravila jezika, Cppcheck vrši dubinsku analizu koda i oktriva potencijalne greške u upravljanju memorijom koje ne bi prijavio kompilator.
Neke od glavnih karakteristika alata Cppcheck su:

- Otkrivanje grešaka: otkriva probleme kao što su neinicijalizovane promenljive, nekompatibilni tipovi, greške u petljama i uslovima, curenje memorije...
- Analiza čitljivosti koda: neiskorišćene promenljive i funkcije, dupliranje koda, ideje za bolju organizaciju samog koda...
- Štednja vremena i smanjenje troškova: rano otkrivanje grešaka u razvoju smanjuje vreme koje bi bilo potrebno za debagovanje kasnije, što doprinosi efikasnijem i sigurnijem softveru.

2.1 Pokretanje alata

Nakon instalacije alata Cppcheck, potrebno je pozicionirati se u direktorijum Cppcheck i pokrenuti skriptu koja sadrzi komandu primene datog alata na projekat MATFWars. Rezultat ce biti upisan u fajl *result.txt*, čija će analiza biti izvršena u nastavku.

2.2 Analiza rezultata alata

Vrste grešaka koje je detektovao alat Cppcheck su:

2.2.1 Konstruktor sa jednim argumentom nije eksplicitan

Ova greška govori da postoje konstruktori koji imaju samo jedan argument i da ih je moguće koristiti za implicitne konverzije tipova. Radi boljeg i bezbednijeg koda, preporuka je koristiti ključnu reč `explicit` i na taj način obezbediti da ne dođe do neželjene implicitne konverzije.
Primer problematičnog koda je:

```
class PlayerNode : public QGraphicsItem
{
public:
  PlayerNode(Player *player);
};
```

Ispravnije bi bilo:

```
class PlayerNode : public QGraphicsItem
{
public:
  excplicit PlayerNode(Player *player);
};
```

2.2.2 Metod može biti const

Greška govori da postoje funkcije koje ne menjaju stanje objekta, ali nisu označene kao konstantne. Pravilnije bi bilo da se doda ključna reč `const`.
Primer koda gde se javlja ovaj problem:

```
QVector<double> Function::getXCoords(double start, double end, int num) {
    QVector<double> xCoords(num);

    if (num == 0) return xCoords;

    if (num == 1) {
        xCoords[0] = start;
        return xCoords;
    }

    double delta = (end - start) / (num - 1);

    for (int i = 0; i < num - 1; i++)
        xCoords[i] = start + delta*i;
    xCoords[num - 1] = end;
```

```
    return xCoords;
}
```

Dakle, ispravnije bi bilo dodati da data funkcija bude `const`, jer ne menja nijedan objekat, već samo vraća vrednost članova klase.

2.2.3 Parametar funkcije se prosleđuje po vrednosti

U ovom slučaju, postoje funkcije čiji se parametri prosleđuju po vrednosti tj. pravi se kopija objekta svaki put kada se data funkcija pozove. Ovo može da predstavlja problem u situacijama kod kopiranja velikih objekata, jer može da dođe do nepotrebne potrošnje memorije. Rešenje bi bilo prosleđivati parametre kao `const &`.
Na primer u sledećoj funkciji, ne bit trebalo promenljivu `functionString` proslediti po vrednosti:

```
Function::Function(std::string functionString,  double startX, double endX, int numX) {
    m_parser.DefineVar("x", &m_varX);
    m_parser.SetExpr(functionString);
    setPoints(startX, endX, numX);
}
```

2.2.4 Razlikovanje imena argumenata u deklaraciji i definiciji

Ovo govori da se imena argumenata funkcije razlikuju u deklaraciji (.h) i definiciji (.cpp), što više predstavlja stilski problem u kodu koji otežava njegovu čitljivost. Bilo bi poželjno da imena budu ista, radi lakšeg snalaženja i čitanja koda. Primer takve funkcije:

```
// funkcija u result.h
bool eventFilter(QObject *watched, QEvent *event) override;

// funkcija u result.cpp
bool Result::eventFilter(QObject *obj, QEvent *event){
```

Bilo bi lakše da se u oba fajla promenljiva zove `watched` ili eventualno `obj`.

2.2.5 Funkcija se ne nikada ne koristi u kodu

Ova greška upozorava na to da se funkcija koja postoji u kodu nikada ne poziva. Ukazuje na to da nepotrebne funkcije samo otežavaju održavanje koda i da ih je moguće jednostavno ukloniti. U MATFWars projektu, postoji dosta takvih funkcija koje pripadaju muparser biblioteci, koja je korišćena u kodu. Međutim, postoji mnogo funkcija iz date biblioteke koje se uopšte nisu pozivale, pa ih iz tog razloga alat prijavljuje kao problem.

3 Valgrind

Valgrind predstavlja alat za dinamičku analizu programa i najčešće se koristi za C i C++ projekte. Koristi se kao pomoć u rešavanju problema sa memorijom i nitima. Njegova primarna funkcija je kao profajler i debager, ali je takođe i framework za pravljenje novih alata za dinamičku analizu.

3.1 Memcheck

Memcheck prestvalja najpoznatiji alat unutar Valgrinda, koji detektuje memorijske greške korisničkog programa. On pokreće program, prati pristup memoriji i svaki poziv funkcija za alokaciju i oslobađanje memorije. Nakon izvršavanja programa, generiše se detaljan izveštaj sa problemima koje je pronašao. Program koji radi pod kontrolom Memcheck-a je čak 20 do 100 puta sporiji nego kad se izvršava samostalno.
Njegova glavna uloga je da pronađe greške kao što su:

- Curenje memorije: kada se memorija alocira, ali se ne oslobodi
- Korišćenje neinicialozovane memorije: čitanje vrednosti iz memorije koja nije prethodno postavljena
- Neispravno oslobađanje memorije: npr. dvostruko oslobađanje memorije
- Greške pri pristupu memoriji: npr. pristup van granica niza, korišćenje oslobođene memorije

3.1.1 Pokretanje alata

Nakon instalacije Valgrind-a i pozicioniranja u direktorijum alata Memcheck, potrebno je pokrenuti skriptu *memcheck.sh*, koja poziva alat nad izvršnim fajlom projekta MATFWars.
Takođe je dodat parametar: `--suppressions=valgrind_supressions.supp` uz pomoć kog će se izbeći generisanje upozorenja koja nisu vezana za sam korisnički kod, već više za sistemske biblioteke i Qt okruženje.
Primer slučajeva koje sadrži fajl valgrind_suppressions.supp su funkcije `strncmp`, `expand_dynamic_string_token`, `_dl_map_object`, `dl_open_worker`, kao i funkcije iz modula `QtCore`, `QtGui` i `QtWidgets`.

3.1.2 Analiza rezultata alata

U izveštaju *result.txt*, prikazan je rezultat rada alata Memcheck. Na početku, u odeljku **HEAP SUMMARY**, se vidi da određen broj memorijskih blokova nije oslobođen pre završetka izvršavanja programa. Količina memorije koja je ostala zauzeta nije preterano velika, već predstavlja interne strukture Qt-a koje ostaju aktivne do samog gašenja procesa, nakon čega ih operativni sistem svakako oslobodi.
U delu **LEAK SUMMARY**, nalaze se informacije o curenju memorije. Broj

bajtova koji su definitivno izgubljeni je jednak nuli, što znači da nema situacija gde se gubi referenca na zauzeti memorijski blok. Takođe, alat Memcheck prijavljuje da nema ni indirektno izgubljene memorije, tako da nema ni posrednih gubitaka preko nekih složenijih struktura.
Postoji 1872 bajta za koje nije potpuno sigurno da li su i dalje dostupni ili izgubljeni, ali svakako je ovo mala količina memorije koja najverovatnije potiče iz rada biblioteka. Najveći deo memorije je označen kao still reachable, što znači da je program i dalje imao pokazivače ka tim blokovima, ali ih nije eksplicitno oslobodio. Ova situacija je česta kod Qt aplikacija, gde veliki broj objekata ostaje u memoriji do samog gašenja programa.
U **ERROR SUMMARY** delu, navedeno je da nije pronađena nijedna konkretna greška u korišćenju memorije. Prisutno je samo nekoliko upozorenja koja se nalaze u suppresed fajlu, tako da nisu vezana za sam korisnički kod.
Zaključak je da je rezultat pokretanja alata Memcheck nad MATFWars projektom vrlo povoljan, ne postoje ozbiljna curenja memorije na koja je potrebno obratiti pažnju.

3.2 Massif

Alat Massif predstavlja deo Valgrind skupa alata, koristi se za profilisanje memorije programa sa fokusom na hip memoriju. Kada se program pokrene sa ovim alatom, beleže se detaljne informacije o alokaciji memorije tokom vremena, kao što su:

- kada se alokacije dešavaju
- koliko memorije je zauzeto
- kada se oslobađa memorija

Rezultat alata se čuva u posebnom fajlu, koji se konvertuje u čitljiv tekstualni prikaz uz pomoć komande `ms_print`. Tako se dobija grafički prikaz zauzeća memorije kroz vreme, kao i detalji o funkcijama koje su alocirale najviše memorije.
Primena Massif-a je najviše korisna za optimizaciju programa koji intenzivno koriste memoriju, kao i za otkrivanje problema sa curenjem memorije koji mogu ostati neprimećeni pri standardnom testiranju. Ovaj alat je ključan za analizu i optimizaciju memorijskog ponašanja programa, prokazujući detaljan uvid u korišćenje hip memorije.

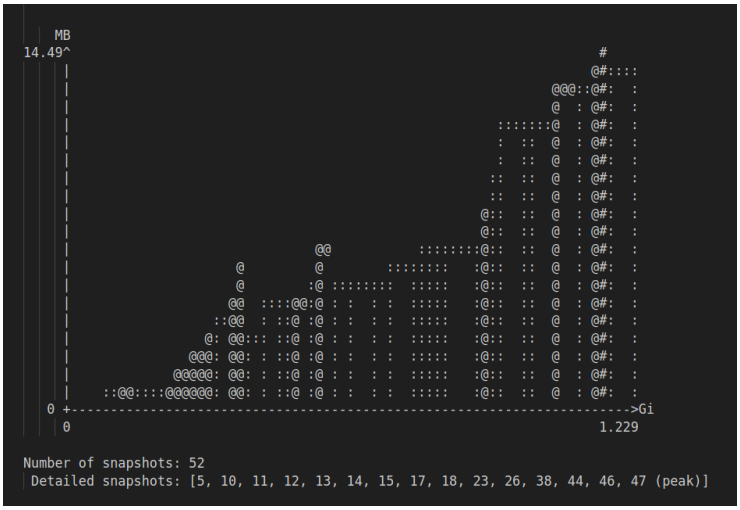
3.2.1 Pokretanje alata

Za primenu Massif alata nad projektom MATFWars, potrebno je pozicionirati se u odgovarajući direktorijum i pokrenuti *massif.sh* skruptu. Tako će se pokrenuti odgovarajuća komanda koja će generisati fajl *result.txt*.
Nakon toga se pokreće: `ms_print $massif_file > $ms_print_file`, uz pomoć koje se upisuje izlaz alata u fajl `$ms_print_file`, koji je mnogo lakši za razumevanje i čitanje rezultata.

3.2.2 Analiza rezultata alata

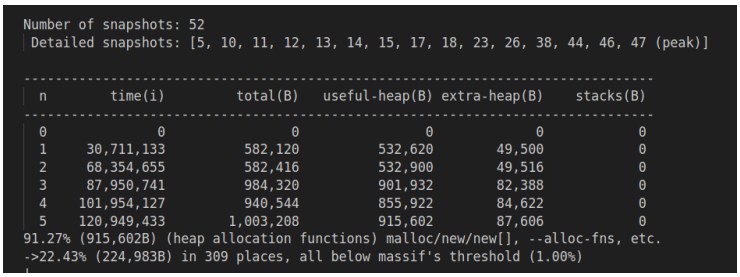
Na osnovu dobijenog izlaza alata, mogu se uočiti nekoliko karakteristika u vezi se potrošnjom memorije programa. Praćenje je izvršeno u 52 snimka, a detaljan uvid je dat samo za ključne tačke izvršavanja među kojima je i tačka najveće potrošnje memorije.
Na osnovu grafa sa slike 1, može se primetiti da memorijska potrošnja tokom vremena raste postepeno. Na x osi je predstavljeno vreme u broju izvršenih instrukcija, a na y osi se nalazi količina zauzete memorije na heap-u. Najveći rast memorije se dostiže u kasnijim fazama izvršavanja, a memorijski maksimum je oko 14.49 MB. Mogu se uočiti dve faze rasta memorije:

- na početku izvršavanja, kada se vrši inicijalizacija i prve alokacije
- kasnije tokom izvršavanja, kada se kreiraju dodatni resurski kao što su teme, fontovi...



Slika 1: Izlaz iz Massif alata (graf)

U tabelarnom prikazu u fajlu *result_ms_print.txt* (slika 2), vidi se da se ukupna potrošnja memorije kreće od inicijalnih 582120 bajta, pa sve do preko 1 MB. Korisna memorija koja predstavlja realno zauzeće memorije od npr. podataka i struktura koje program koristi, zauzima najveći deo ukupne memorije. Dodatna količina memorije koja predstavlja interne strukture i pomoćne podatke, prisutna je u dosta manjoj meri. Od svih alokacija u programu je 92.27% povezano sa heap funkcijama kao što su malloc, new, new[]...
Veliki broj alokacija dolazi od mnogo manjih objekata, koji pojedinačno zauzimaju manje od 1% ukupne memorije. To znači da ne postoji jedna velika alokacija, već više sitnih. Ovakva situacija je standardna za grafičke programe,



Slika 2: Izlaz iz Massif alata (tabela)

jer brojni objekti zahtevaju manju količinu memorije, a u zbiru daju veću vrednost.
Potrošnja memorije u programu je relativno stabilna, može se primetiti da nema indikacija da postoje ozbiljna curenja memorije. Takođe, kao što je već navedeno, aplikacija dostiže maksimum potrošnje memorije oko 14.29 MB, što je očekivano za grafičku aplikaciju razvijenu u Qt-u. Većina te memorije potiče iz GUI-a i sistemskih biblioteka, a ne konkretne logike igre. Takođe, standardna C++ biblioteka isto doprinosi dodatnoj potrošnji, a najviše u fazi inicijalizacije.

4 ClangFormat

ClangFormat je alat za automatsko formatiranje koda čiji je cilj da eliminiše ručno formatiranje. On obezbeđuje dosledan stil pisanja koda, tako što prema unapred definisanim pravilima prilagođava razmake, uvlačenja, raspored zagrada, komentare, poravnanja... Podržava razne programske jezike kao što su C, C++, Java, JavaScript, C# i drugi. Neke od njegovih glavnih karakteristika su:

- Obezbeđivanje konzistentnog stila programiranja u projektu
- Povećanje čitljivosti i preglednosti koda
- Smanjenje konflikta u sistemima za verzionisanje (npr. Git)
- Ušteda vremena programerma, nema potrebe za ručnim nameštanjem koda
- Radi na osnovu sintaksnog stabla, tj. razume strukturu koda i ne posmatra ga samo kao tekst

4.1 Pokretanje alata

Nakon instalacije alata ClangFormat, potrebno je generisati fajl *.clang_format* u kome se definiše stil formatiranja. Neke od stvari koje je potrebno podesiti:

- broj razmaka za uvlačenje

- gde idu zagrade
- maksimalna dužina linije
- razmak oko operatora

Uz pomoć ovog fajla je omogućena konzistentnost, kod projekta će izgledati isto kod svih članova tima koji rade nad njim. Zatim je potrebno pokrenuti skriptu koja pokreće alat ClangFormat, prolazi kroz sve fajlove projekta i vrši formatiranje.

4.2 Analiza rezultata alata

Izmene se vrše na osnovu *.clang_format* fajla, koji kaže alatu da koristi GNU C/C++ stil, odnosno:

- uvlačenje se sastoji od 2 razmaka
- zagrade se otvaraju u novom redu
- linija koda ima limit od 80 karaktera
- postoje razmaci oko operatora

Primer koda pre primene alata:

```
double Canvas::scaleToCanvas(double len) {
    return len*this->width()/gridWidth();
}
```

Primer koda nakon primene alata:

```
double
Canvas::scaleToCanvas (double len)
{
  return len * this->width () / gridWidth ();
}
```

Primećuje se da je kod nakon primene alata više prostran i čitljiviji. Otvorene zagrade se nalaze u novom redu, takođe postoje razmaci oko operatora i funkcija.