Verifikacija softvera - Pacman

Projekat za analizu: Pacman

Adresa projekta: https://github.com/MATF-RS20/RS030-pacman
Alati za verifikaciu softvera će biti primenjeni commit sa hešom
22218d726c58f913ca10168f73bbd15aa47ceb98
Ovaj projekat predstavlja video-igru po uzoru na poznatu igru Pacman
Projekat je implementiran uz pomoć C++ i Qt okruženja.

1. Alat - Qt Unit Tests

Projekat na početku nije bio dobro napravljen, kako bi se mogli pokrenuti unit testovi. Problem je bio to što se u testovima ne treba uključivati main datoteka projekta koji se testira, međutim, u njoj su bile inicijalizovane globalne promenljive koje su se koristile dalje u programu. Ovaj problem rešen je tako što je napravljen extern_variables fajl koji inicijalizuje ove promenljive, kako nam ne bi bila potreban main fajl, već se inicijalizacija vrši iz tog novokreiranog fajla. Kod pre izmene je izgledao ovako:

```
20  Game * game;
21  int level_map;
22  //std::list<std::pair<std::string, int> > listOfScores;
23  //QString playersName = "";
24  QString player = "Player";
25
26
27
28  int main(int argc, char *argv[])
29  {
30      QApplication a(argc, argv);
31
32      level_map = 1;
33      game = new Game();
```

Sporne promenljive su game, level_map i player. Nakon izmene, main fajl izgleda ovako:

```cpp
#include "extern_variables.h"


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    level_map = 1;
    game = new Game();

    game->show();
    game->displayMainManu();


    return a.exec();
}

```

Dok extern_variables.h i extern_variables.cpp izgledaju ovako:

extern_variables.h

```cpp
#ifndef EXTERN_VARIABLES_H
#define EXTERN_VARIABLES_H

#include "game.h"

extern Game *game;
extern int level_map;
extern QString player;

void initializeVariables();

#endif // EXTERN_VARIABLES_H
```

extern_variables.cpp

```cpp
#include "extern_variables.h"

Game *game = nullptr;
int level_map = 1;
QString player = "Player";

void initializeVariables()
{
    game = new Game();
}

```

Klase koje su bile testirane su Ghost i Health, sa napomenom da su zbog testiranja funkcije sendToInitial() dodati seteri za privatna polja x1 i y1 klase Ghost.

```cpp
class Ghost: public QObject, public QGraphicsPixmapItem{
    Q_OBJECT
public:
    //Ghost ();
    Ghost(int x1, int y1, int id);
    int getX();
    int getY();
    void setX1(int x) { x1 = x; }
    void setY1(int y) { y1 = y; }
    void sendToInitial();
```

2. Alat - Valgrind Memcheck

Pre nego što se pokrene alat Memcheck, u Qt okruženje dodajemo linije koda -g i -O0 kako bi imali dobro korelisane linije na kojima su pronađene greške i linije koda u izvornom kodu, kao i da bi se eleminisale optimizacije koje potencijalno mogu zamaskirati neke probleme.

```
5  CONFIG += c++17
6
7  # Set optimization level to O0
8  QMAKE_CXXFLAGS_DEBUG += -O0
9
10 # Add -g option for debugging symbols
11 QMAKE_CXXFLAGS_DEBUG += -g
12
```

Nadalje, pokrećemo alat pozivom funkcije:

valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all --log-file="Memcheck_report" --track-origins=yes ./qt

Prilikom završetka rada programa, dobijamo fajl koji sadrži informacije o memoriji:

```
LEAK SUMMARY:
   definitely lost: 5,960 bytes in 240 blocks
   indirectly lost: 7,672 bytes in 239 blocks
     possibly lost: 768 bytes in 2 blocks
   still reachable: 2,887,377 bytes in 30,417 blocks
        suppressed: 0 bytes in 0 blocks
```

Ovde se
vidi da postoji jako mnogo propusta u pravljenju programa, ali kao primer, uzet je

```
8 bytes in 1 blocks are definitely lost in loss record 333 of 9,378
   at 0x4849013: operator new(unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
   by 0x124FCB: Game::gameOver(QString) (game.cpp:246)
```

jedan problem sa alokacijom memorije:

Dakle, u liniji 246 u funkciji gameOver, klase Game, dolazi do curenja memorije.

```
246         highScores[9] = std::pair<QString*,int>(new QString(player), sk);
247         QString *whichPlayer = new QString(&player + QString::number(howManyGames));
248         howManyGames++;
249         highScores[9] = std::pair<QString*,int>(whichPlayer, sk);
```

Promenljiva highScores je par koji sadrži pokazivač na objekat QString i int. Generalno nije dobra praksa koristiti pokazivače za jednostavne tipove kao što je QString, a u ovom slučaju memorija koju highScores[9] čuva nije dealocirana pre nove dodele. Ovaj kod bi trebao da izgleda na primer, ovako:

```
246         if (highScores[9].first != nullptr) {
247             delete highScores[9].first;
248         }
249         QString whichPlayer = player + QString::number(howManyGames);
250         highScores[9] = std::pair<QString*, int>(new QString(whichPlayer), sk);
251         howManyGames++;
```

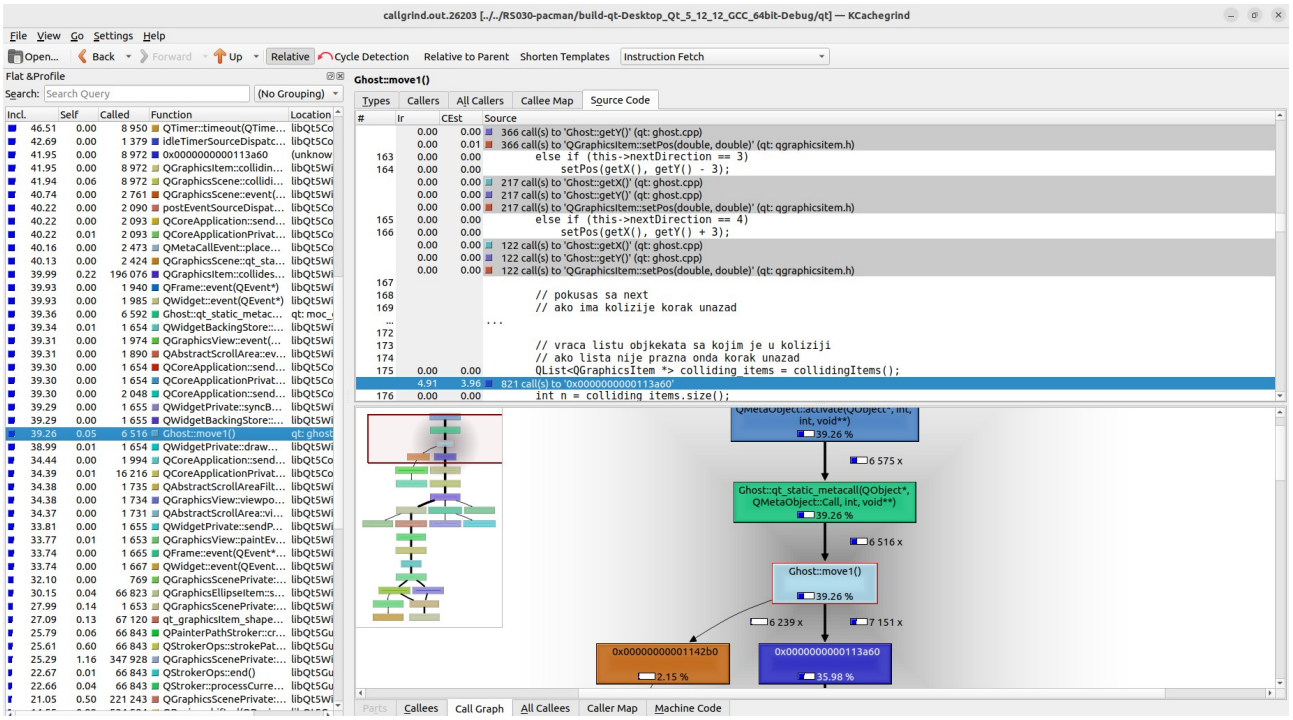
3. Alat - Valgrind Callgrind

Alat se pokreće pozivom funkcije:

```
valgrind --tool=callgrind --cache-sim=yes --branch-sim=yes ./qt
```

Vizuelnu reprezentaciju poziva funkcija dobijamo pokretanjem alata za prikaz komandom:

```
kcachegrind callgrind.out.26203
```



Ovde se vidi da je funkcija move1() klase Ghost vrlo visoko u stablu pozivanja, tj. poziva se jako često i samim tim je kandidat čija optimizacija bi trebala da se razmotri.

Ova funkcija obuhvata 240 linija koda koje se izvršavaju svaki put kad se ova funkcija poziva, te je refaktorisanje definitivno potrebno. Ona obuhvata različite celine koje treba da se izvrše, kao što su: postavljanje slike u polje na mapi, izračunavanje narednog koraka, proveravanje da li je došlo do kolizija, iscrtavanje na mapi... Funkcija se treba razdvojiti u više manjih funkcija od kojih se neke, po mogućstvu neće pozivati, ili se logika treba potpuno promeniti.