UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

PROJEKAT

iz predmeta Razvoj softvera za embeded sisteme

Lavirint igirca i baza podataka svih igrača na lokalnom serveru

Student:

Jovan Slavujević

Profesor:

prof. dr Teodorović Predrag

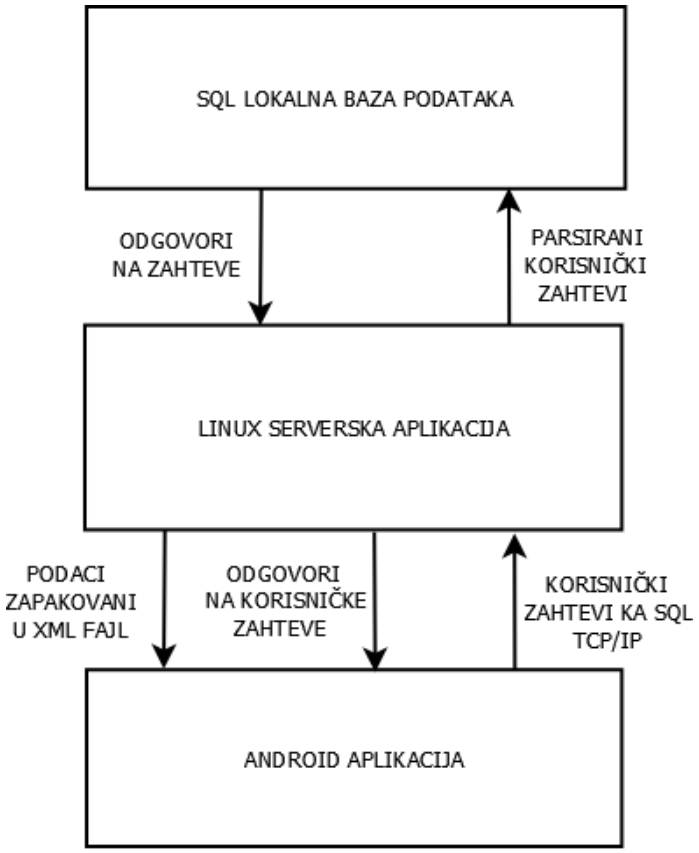
Novi Sad, 2020. god.

Sadržaj

1 Uvod 3
2 Lavirint igrica 4
2.1 Generisanje terena 5
3 Korisnički nalog 6
4 Algoritam komunikacije sa serverom 8
5 Server 10
6 Komunikacija između servera i SQL-a 11
7 Generisanje XML fajla 12
8 Algoritam serverske aplikacije 13
9 Zaključak 15
10 Literatura 16

1 Uvod

Zadatak projekta je podeljen u dva bloka. Prvi blok predstavlja kreiranje Android aplikacije koja sadrži jednostavnu lavirint igricu, korisnički interfejs i upravljanje podacima korisnika. Zadatak drugog bloka bio bi kreiranje *TCP/IP* (eng. *Transmission Control Protocol / Internet Protocol*) servera koji rukovodi (eng. *Handles*) komunikacijom sa Android klijentima otvaranjem *socket*-a. Server zapremljene podatke skladišti u *SQL* bazu podataka, odakle može da ih preuzme. Pored toga, server generiše *XML* fajl u kom su zapakovani podaci o top 10 igrača i njega je neophodno poslati ka svim aktivnim klijentima.



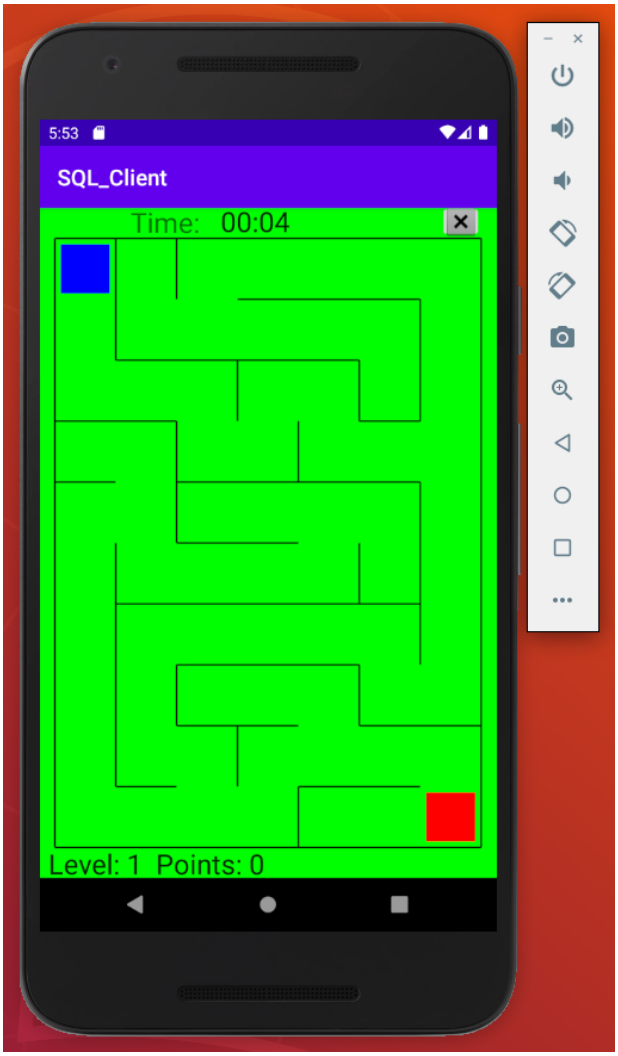
Slika 1. Blok šema celokupnog sistema

2 Lavirint igrica

Što se tiče načina na koji je igrica zamišljena, postoji 5 različitih nasumično (eng. *Random*) generisanih terena po kojima se igrač (plavi kvadratić) kreće dok ne nađe izlaz (crveni kvadratić). Pošto sam lavirint ne predstavlja veliki igrački izazov, odnosno lako je preći nivoe, napravljen je sistem bodovanja u koji je uključeno odbrojavanje vremena po sledećoj formuli:

$$Pts = \frac{Tmax' - (t - \dot{\Delta}t)}{Tmax' - Tmin'} 100$$

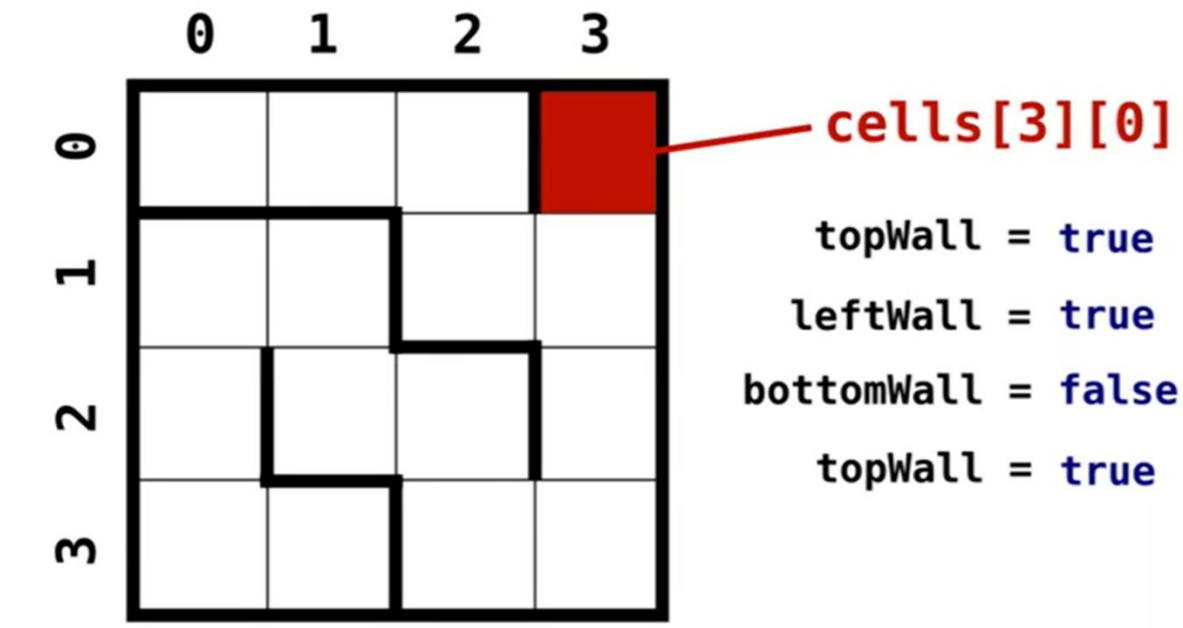
Gde je *Pts* broj poena po pređenom nivou koji se akumulira sa preostalim poenima, $Tmax'(Tmin')$ je zapravo petina od maksimalnog (minimalnog) dozvoljenog vremena igranja,a $\dot{\Delta}t$ je vreme koje je proteklo u trenutku prelaženja prošlog nivoa (inicijalno je nula).



Slika 2. Prikaz igrice

2.1 Generisanje terena

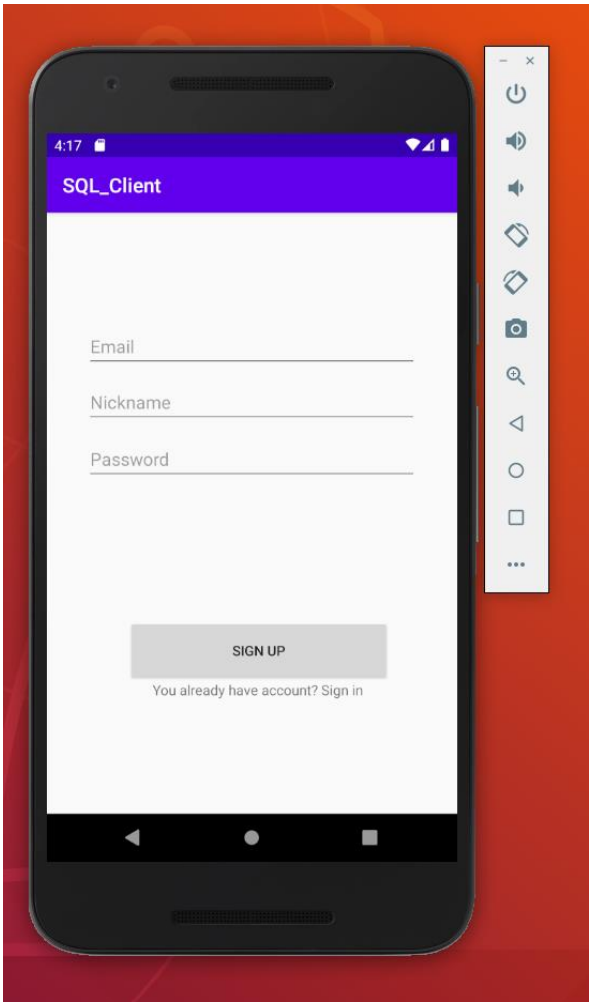
Teren se sastoji od ćelija (eng. *Cells*). Svaka od ćelija je objekat koji može da ima zid na svakoj od svoje četiri stranice. Na početku generisanja smatra se da svaka ćelija ima sve zidove, pa se prolaskom (iteracijom) kroz polje zidovi eliminišu posećivanjem komšija (susedne ćelije). Trenutna ćelija u iteraciji bira nasumičnim principom kojeg će od svojih komšija posetiti i ukloniti zid između njih dvoje i tako dok se ceo *stack* ne isprazni, odnosno svaka ćelija ne prođe kroz svoj „komšiluk“.



Slika 3. Princip generisanje terena

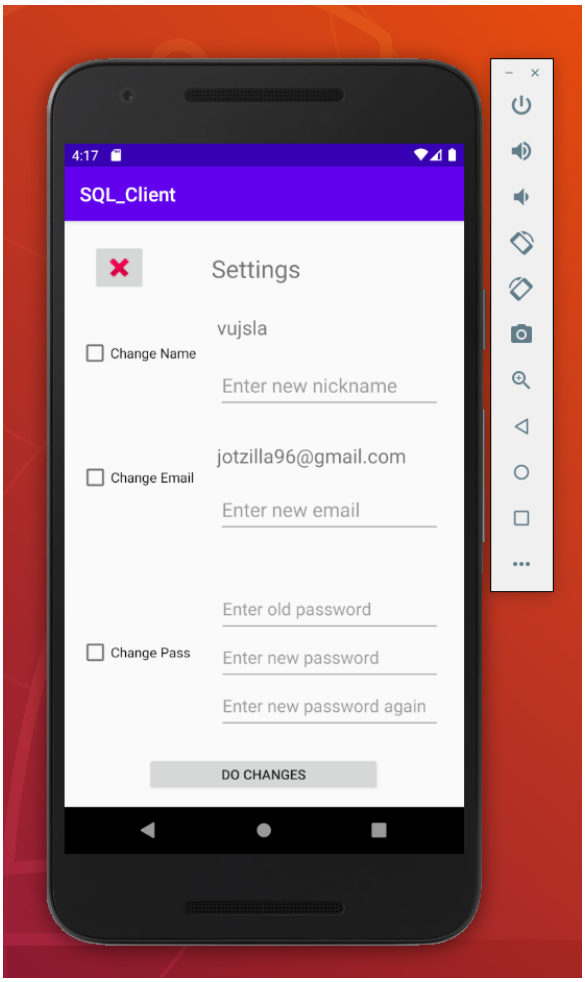
3 Korisnički nalog

Kao što je već navedeno na početku, android korisnik može da se registruje i svoje rezultate u igrici zabeleži u tabeli najboljih igrača. Da bi se igrač registrovao neophodno je da njegov android uređaj bude na istoj lokalnoj mreži kao i server, te da postoji konekcija između njih. Registracija se odvija po standardnom obrascu gde je neophodno da korisnik unese svoj e-mail, nadimak koji će da ima u okviru igrice, kao i lozinku kojom će moći da se prijavi. Kada korisnik unese neophodne podatke, šalje ih ka serveru, koji nakon toga odgovara pozitivnim odgovorom ukoliko je registracija uspešna ili nekim od negativnih odgovora ukoliko registracija nije uspela. Registracija može da bude neuspešna zbog različitih razloga kao što je nadimak ili e-mail koji već koristi drugi igrač, ali može doći i do greške na serverskoj strani o čemu će biti više reči na nekom od narednih poglavlja.



Slika 4. Registracija igrača

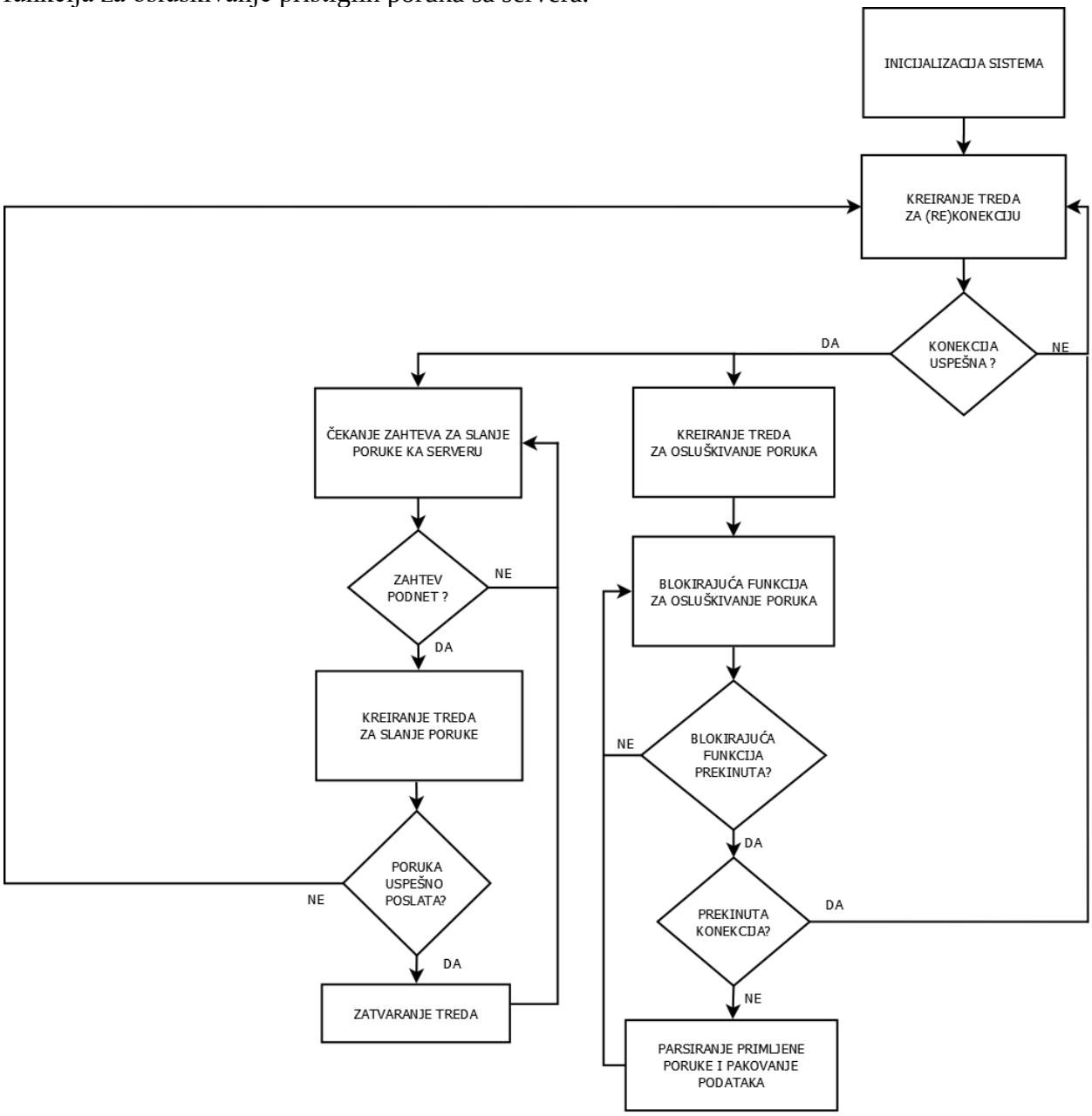
Ukoliko se igrač uspešno registrovao, ima mogućnost da naknadno promeni neki od unetih podataka, da preuzme listu najboljih 10 igrača od servera, ali i da igra. Opcija igranja igrice moguća je i u *Offline* modu, gde nije neophodno biti prijavljen ili povezan sa serverom.



Slika 5. Promena podataka

4 Algoritam komunikacije sa serverom

Komunikacija sa serverom zasniva se na korišćenju niti (eng. *Threads*). Tredovi se koriste kako bi se komunikacija sa serverom paralelizovala sa korišćenjem *UI* interfejsa. Na početku postoji jedan *thread* koji se vrti u beskonačnoj petlji sve dok se ne uspostavi konekcija sa serverom. Kada klijent uspostavi konekciju sa serverom, odnosno uspešno se poveže na njegovu utičnicu (eng. *Socket*) izlazi se iz petlje i zatvara se *thread*, a pritom otvara se novi gde se koristi blokirajuća funkcija za osluškivanje pristiglih poruka sa servera.



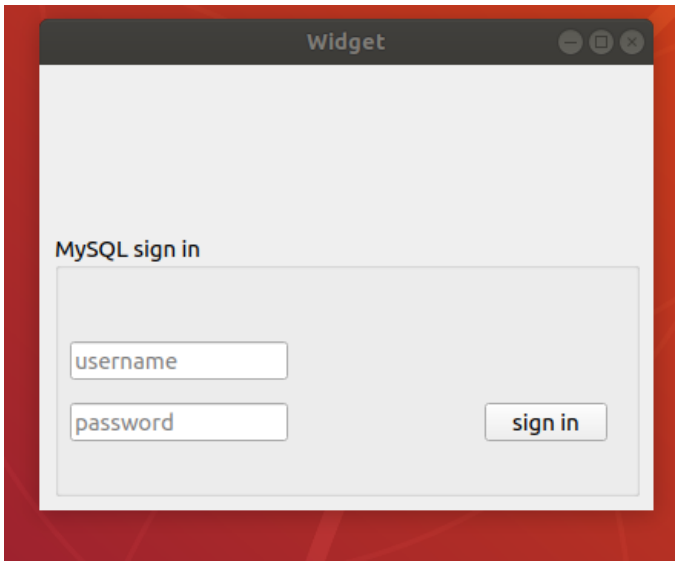
Slika 6. Flow chart komunikacije sa serverom

Ovaj *thread* trči sve dok postoji konekcija sa serverom. Ukoliko nekako dođe do diskonekcije (npr. serverska aplikacija se ugasi) ovaj tred se zatvara i izveštava klijenta o

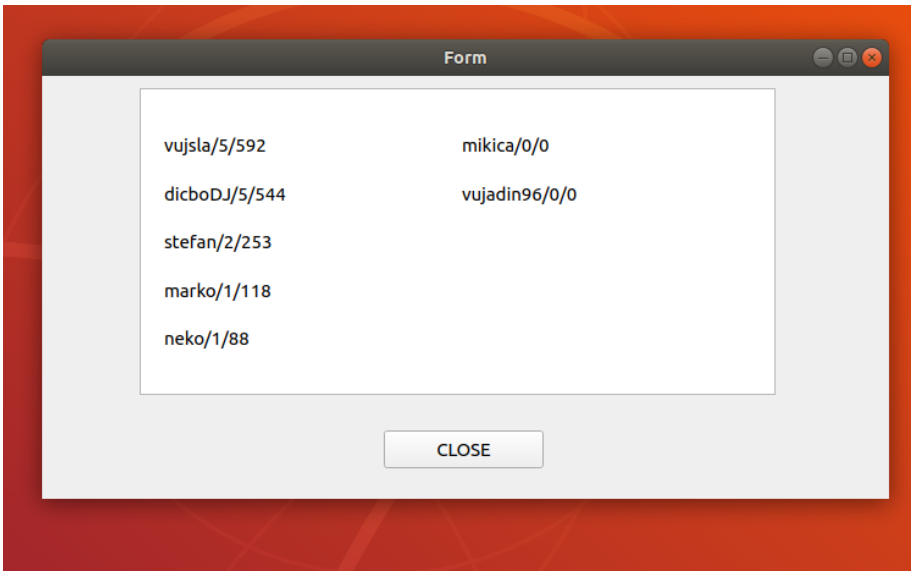
diskonekciji. Sve poruke koje pristignu sa servera bivaju zaprimljene upravo uz pomoć ovog dela. Poruke koje klijent može da primi od servera mogu biti podeljene u dve grupe. Prvu grupu predstavljaju jednostavni odgovori servera na neke od zahteva klijenta (logovanje, registracija, promena podataka, upis novih rezultata itd.) gde, kao što je već gore navedeno, server može odgovoriti pozitivno ili odrično. Druga grupa poruka su linije *XML* (eng. *Extensible Markup Language*) fajla u kom su zapakovani podaci o top 10 igrača. Razlika je u tome gde se primljena poruka dalje prosleđuje. Ukoliko se šalje fajl onda se taj podatak upisuje u novi fajl unutar Android uređaja. U oba slučaja prijem poruka počinje slanjem zahteva ka serveru. Zahtev se šalje, takođe, u vidu tekstualne poruke, tako što se privremeno otvara novi *thread* gde se šalje poruka.

5 Server

Ulogu servera igra C++ aplikacija koja je pokrenuta na Linux operativnom sistemu. Na početku, razvijena je kao konzolna aplikacija da bi naknadno bila unapređena u GUI (eng. *Graphical User Interface*) aplikaciju, razvijenu uz pomoć *QT Creator*-a i njegovih biblioteka. Server funkcioniše po *Multi-threading* principu gde za svaki novi otvoreni *socket* kreira novi *thread* gde neometano komunicira sa svakim klijentom posebno. Serverska aplikacija u ovom slučaju ima ulogu medijuma između *SQL* (eng. *Structured Query Language*) baze podataka i Android klijenata. Da bi pristupio bazi podataka, server koristi *MySQL* API (eng. *Application Programming Interface*). Kao što je gore navedeno, serverska aplikacija podatke o top 10 najboljih igrača pakuje u *XML* fajl koji posle toga šalje ka aktivnim klijentima. Da bi generisao *XML* fajl, server koristi *TinyXML* API.



Slika 7. Izgled prvog prozora GUI aplikacije



Slika 8. Izgled strimovanja tabele do 10 najboljih igrača

6 Komunikacija između servera i SQL-a

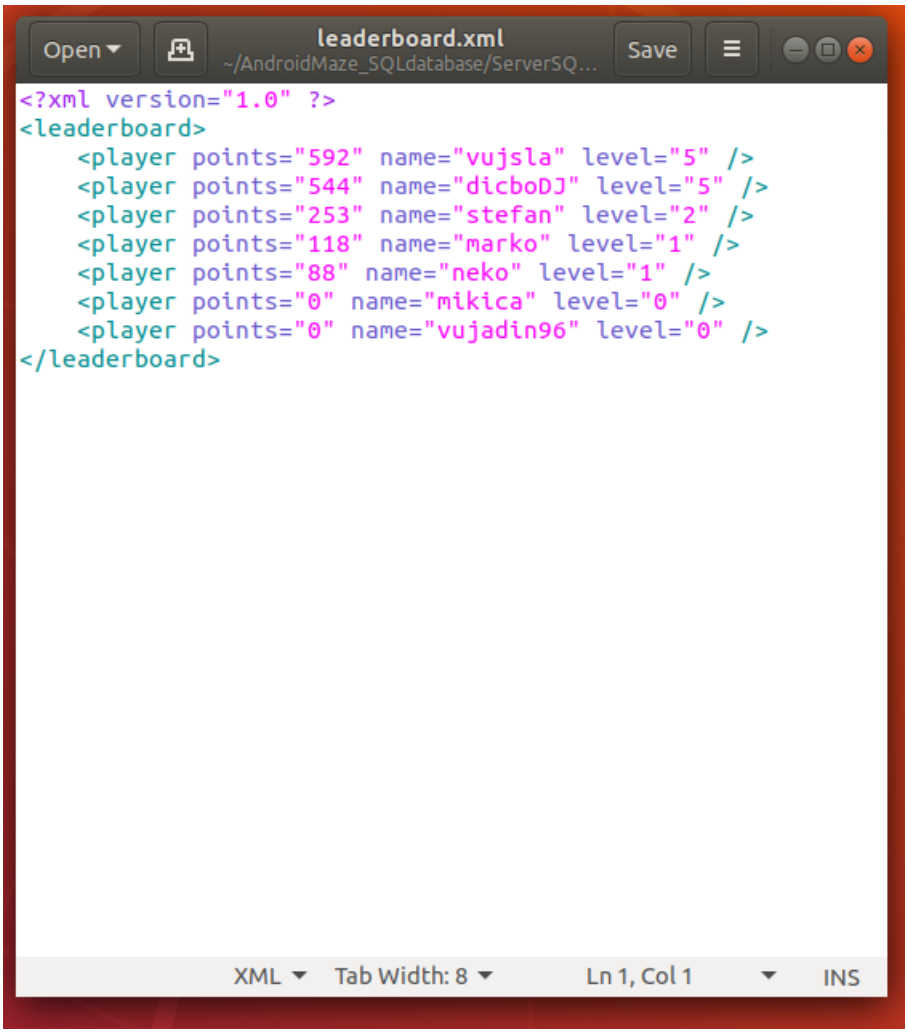
Kao što je već objašnjeno, komunikacija između ova dva bloka omogućena je uz pomoć *MySQL*-a. Ova C/C++ biblioteka traži bazu podataka na lokalnoj mreži i traži unos administratorovog imena i lozinke. Ukoliko baza podataka zaista postoji mogu se slati različiti zahtevi koje bi korisnik inače mogao unositi preko terminala.

Kada je jednom uspostavio komunikaciju sa bazom, server uz pomoć tajmera na jednom tredu na svakih pola sata *ping*-uje bazu podataka odnosno izvršava rekonekciju. Server izveštava aktivne klijente preko *Observer pattern*-a da li je došlo do promene na tabeli.

7 Generisanje XML fajla

Generisanje *XML* fajla serverska aplikacija radi uz pomoć *TinyXML* API. *XML* fajl je vrsta datoteke koja se koristi za razmenu podataka između uređaja. Fajl je koncipiran tako da sadrži elemente i atribute kao glavne komponente gde dizajner fajla ima slobodu da pravi strukturalnu organizaciju po svom nahođenju.

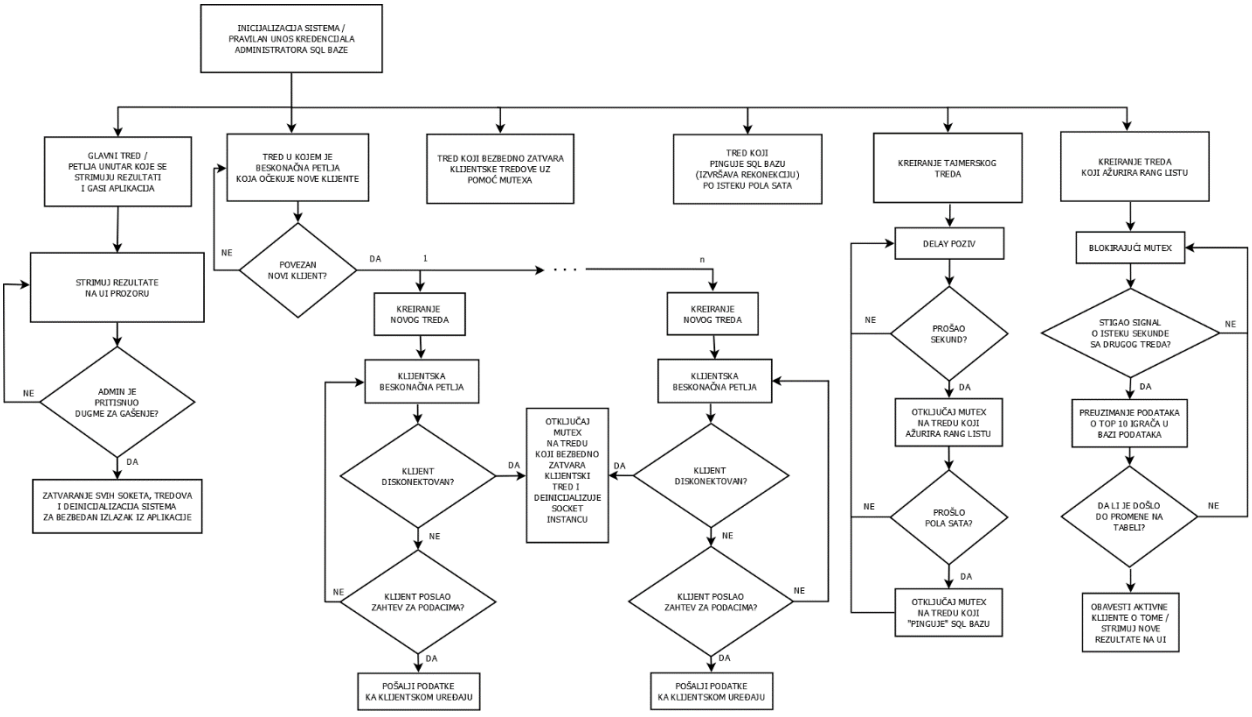
U konkretnom slučaju ove aplikacije struktura fajla je takva da postoji jedan *root* element sa nazivom <*leaderboard*> unutar kojeg se nalaze izvedeni (eng. *Child*) elementi <*player*>. *Root* element može da sadrži do 10 izvedenih elemenata. Svaki *child* element sadrži 2 atributa i to *"level"* i *"points"*. Pri promeni rezultata kreira se novi XML fajl koji *socket*-i mogu dalje da povlače.



Slika 9. Printscreen izgleda fajla

8 Algoritam serverske aplikacije

Pri pokretanju aplikacije odvija se inicijalizacija *SQL* objekta te se otvara *UI* prozor gde je neophodno uneti administratorske kredencijale za pristup bazi podataka. Ukoliko se unesu pogrešni podaci, proces se obnavlja dok se ne unesu tačni. Nakon toga kreće se sa inicijalizacijom *TCP/IP* dela, odnosno *Server* objekta gde se nalaže početak rada nekih tredova, među kojima je i jedan zadužen za osluškivanje konekcije sa klijentima. Naime, kada se neka klijentska aplikacija poveže, ona mora da gađa *IP* adresu serverske aplikacije i tačan komunikacioni port. Ukoliko se klijent uspešno povezao, server otvara novi tred za svakog novopristiglog klijenta gde komunicira samo sa njim. U slučaju da dođe do diskonekcije postoji tred koji je zadužen za deinicijalizaciju svih neaktivnih procesa (tredova).



Slika 10. Flow chart serverske aplikacije

U paraleli sa tim, otvara se novi *UI* prozor gde se nalazi tabela sa top 10 igrača i ona se konstantno strimuje na tom prozoru. Ovaj *streaming* moguć je zahvaljujući *Observer* obrascu objektno-orijentisane implementacije klasa. Kao i unutar *Server* klase postoji *multi-threading* upravljanje procesima i unutar *SQL* klase. Glavnu ulogu u tome ima tajmer koji odbrojava vreme i obaveštava ostale tredove da izvršavaju svoje procese po isteku specifičnog vremena. Pri isteku svake sekunde tajmer "otključava" blokirani *mutex* na drugom tredu gde se izvršava povlačenje podataka o top 10 igrača te poređenje da li je došlo do promene. Ukoliko je to slučaj, neophodno je obavestiti *Observer* klase, odnosno *Server* klasu i klasu zaduženu za upravljanjem korisničkog prozora da ažuriraju svoje podatke. U tom trenutku izvršava se i generisanje *XML* fajla o kojem je bilo reči u prošlom poglavlju. U tom slučaju na *Server* strani setuje se *flag* o ažurnosti svakog povezanog klijenta na *false*, te mu se pri sledećem zahtevanju rang liste šalje *XML* fajl, a u suprotnom se šalje poruka da je klijent *up-to-date*. Pri isteku svakih pola sata tajmer "otključava" *mutex* na tredu za *"ping*-ovanje", odnosno rekonekciju aplikacije sa bazom podataka.

Komunikacija sa klijentom se odvija u beskonačnoj petlji gde akcija započinje blokirajućom funkcijom za osluškivanje poruka sa specifičnog *socket*-a. Kada primi novu poruku, izvršava se njeno parsiranje i odgovarajući podaci se prosleđuju na odgovarajuća mesta. Ta mesta su zapravo argumenti metoda *SQL* klase. Postoji grupa funkcija preko kojih se prosleđeni podaci prepakuju i šalju ka bazi podataka a povratna vrednost je odgovor baze, koji se šalje nazad ka *socket*-u. Postoji izuzetak u ovoj komunikaciji gde se jedan zahtev ne šalje direktno ka *SQL* bazi podataka nego se šalje generisani *XML* fajl, gde baza podataka ima posredni uticaj.

9 Zaključak

Aplikacija čiji je cilj bio da primeni klijent-server komunikaciju na kreativan način je ovim uspešno realizovana. U sklopu ovog projekta korišćeno je objektno-orijentisani princip na obe strane, kako na klijentskoj (java) tako i na serverskoj (C++) aplikaciji. Obezbeđeno je paralelno izvršavanje velikog broja procesa uz pomoć tredova, *mutex*-a, *unique lock*-ova i uslovnih promenljivih (eng. *Conditional variable*). Optimizacija rada sistema je na visokom nivou, što je prvenstveno omogućeno pravilnom primenom *STL* (eng. *Standard Template Libraries*) kontejnera kao što je vektor, te se maksimalno izbegava kopiranje konstruktora, već se prosleđuju originalne instance uz zaštitu originala sa atributom *const*. Kreirana je *GUI* aplikacija na serverskoj strani umesto standardne konzolne, što je realizovano putem *QT Creator*-a. Implementacija klasa koje su nezavisne od vida okruženja (da li je aplikacija konzolna ili grafička) je napisana i generisana kao samostalna (eng. *Standalone*) dinamička biblioteka, koja se naknadno može linkovati unutar GUI programa. *Build* okruženje koje je korišćeno za pravljenje biblioteke je *CMake*, koji generiše *Makefile* sa svim neophodnim stavkama.

Neka od eventualnih poboljšanja bila bi svakako unapređivanje korisničkog interfejsa na *Android* strani. Ovaj deo bi uključivao *front-end* razvoj (eng. *Development*), koji ne predstavlja veliki izazov pa stoga fokus nije bio na njemu. Takođe bi bilo poželjno zaštiti enkripcijom kako im se ne bi moglo neovlašćeno pristupati.

10 Reference

[1] https://github.com/jovanSlavujevic96/MazeGame_Database-RSZES - *Github* repozitorijum projekta, pristupljeno: april 2020.

[2] https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-18-04 - Instalacija *MySQL* baze i kreiranje lokalne baze podataka, pristupljeno: maj 2020.

[3] https://developer.android.com/studio – Android studio, zvanična stranica, pristupljeno: maj 2020.

[4] https://developer.android.com/ - Android Studio Developers, zvanična stranica, pristupljeno: jun 2020.

[5] https://www.qt.io/ - QT, zvanična stranica, pristupljeno: jul 2020.