UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE
RAČUNARSKE NAUKE

NASLOV

DIPLOMSKI RAD

- Osnovne akademske studije -

Kandidat
Dejan Predojević

Mentor
Žarko Živanov

Novi Sad, Septembar 2019

Sadržaj

Lista slika

Lista tabela

1 Uvod

1.1 *Commodore 1541*, *5.25"flopi disketa* i *D64 fajl*

Commodore 1541 je čitač *5.25"flopi disketa* korićenih u radu sa *Commodore 64* računarima. Razvijen je u Kanadi od strane kompanije *Commodore*, 1982. godine. Ovaj čitač imao je jednu glavu za čitanje i pisanje sa *Commodore 64* računarom i bio je povezan sa njim preko *serial bus-a*. Imao je dve lapmpice, zelenu koja je označavala da je uređaj uključen u struju i crvenu koja se palila kada uređaj radi [3]. Izgled čitača prikazan je na slici 1.

Slika 1: Commodore 1541

Originalna *5.25"flopi disketa* ili *miniflopi* bio je magnetni disk format koga je 1976. godine predstavio *Shugart Associates* kao zamenu za *8-inčnu disketu* koja se smatrala prevelikom za novije računare [4]. Izgled diskete prikazan je na slici 2.

Osnovni oblik ove diskete imao je 35 kružnih traka dok je postojala i verzija sa 40 traka na kojima su bili raspoređeni sektori od po 256 bajta. Trake su imale različit broj sektora i bile su raspoređene od spolja ka unutra, odnosno prva traka je bila bliža spoljašnjoj ivici i imala je najviše sektora,

dok je poslednja traka bila bliža unutrašnjem prstenu diskete sa najmanje sektora. Broj sektora po trakama je prikazan u tabeli 1.

Slika 2: 5.25"flopi disketa

Traka 18 predstavlja glavnu traku sa svojih 19 sektora. Nulti sektor ove trake čuva **BAM**(eng. Block Availability Map), ime i id flopi diskete dok ostali sektori sadrže podatke. **BAM** je struktura podataka koja prati koji sektori traka su slobodni a koji zauzeti. Sektor u kome je smesten **BAM** je formiran po posebnom šablonu. **BAM** za prvu traku smešten je od 4-7 bajta nultog sektora. Prvi bajt **BAM-a** govori ukupan broj slobodnih sektora, dok preostala tri govore koji sektori su slobodni a koji popunjeni. Svaka naredna 4 bajta predstavljaju **BAM** za narednu traku flopi diskete. 128 bajt čuva

Traka	Sektori/traka	Ukupno Sektora	Skladište u bajtima
1-17	21	357	7820
18-24	19	133	7170
25-30	18	108	6300
31-35	17	85	6020

Tabela 1: Broj sektora po traci originalne *5.25"flopi diskete*

ime flopi diskete dok 146 bajt čuva id diskete.

Podaci su uvek smešteni počevši od 1 sektora, ukoliko je reč o direktorijumima pomeranje se vrši za 3 sektora pa je nastavak na 4 sektoru dok se za smeštanje fajlova pomeranje vrši za 10 sektora i tada je nastavak na 10-tom sektoru. Sektori su popunjeni po šablonu, prva dva bajta sektora govore o lokaciji sledece trake/sektora. Kada dođemo do kraja direktorijuma vrednost prvog bajta je 00.

D64 fajl je najšire podržani i dobro definisan format koji predstavlja elektronsku byte predstavu *5.25"flopi diskete* za *Commodore 64* računare. Standardni *D64 file* ima 174848 bajtova, 683 sektora sa po 256 bajta i on oslikava originalnu *5.25"flopi disketu* sa 35 traka. Postoji i verzija ovog fajla sa 196608 bajtova,768 sektora sa po 256 bajta koja oslikava verziju *5.25"flopi diskete* sa 40 traka.

Na kraju *D64 fajla* mogu se naći dodatni bajtovi, koji govore koji sektori imaju grešku. Ukoliko se na kraju fajla ne pojave bajtovi to znači da su svi sektori ispravni. Svaki bajt je vezan za po jedan sektor odnosno koliko sektora flopi disketa ima toliko će imati i bajtova na kraju fajla ukoliko taj fajl ima grešku u nekom od sektora. U zavisnosti od broja traka i broj dodatnih bajtova se menja i to za 35 traka 689 bajtova dok je za 40 traka 768 bajtova. Ukoliko je vrednost bajta postavljena na 1 to znači da korespodentni sektor nema grešku [5]. Veličina fajlova u zavisnosti od dodatnih bajtova je prikazana u tabeli 2.

Tip diskete	Veličina(bajt)
35 traka, no errors	174848
35 traka, 689 error bajtova	175531
40 traka, no errors	196608
40 traka, 768 error bajtova	197376

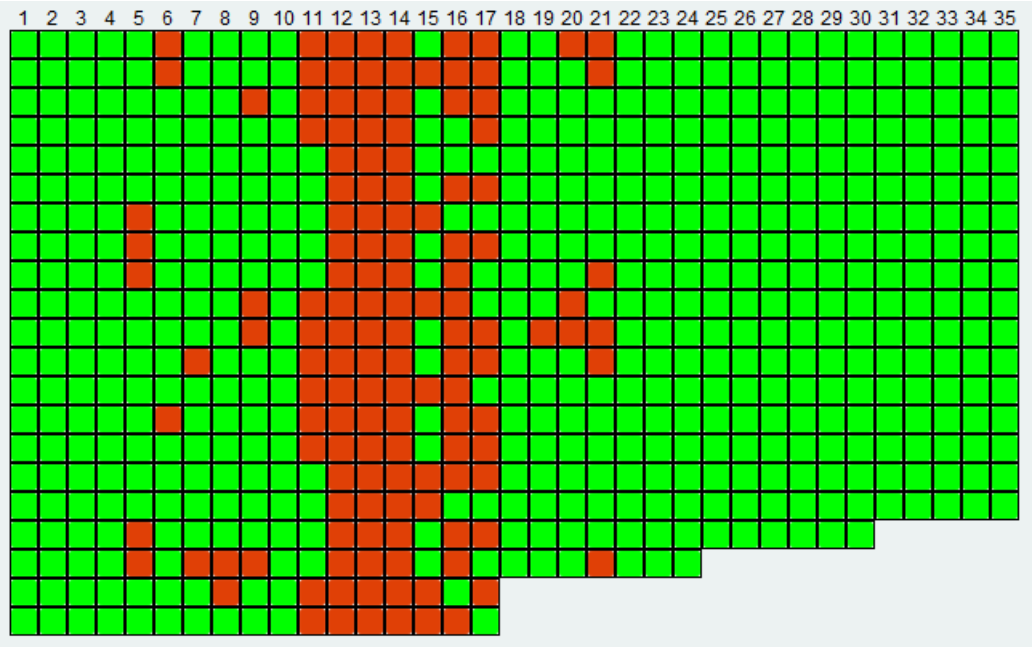
Tabela 2: Veličina fajla u zavisnosti od dodatnih bajtova

2 Rešavani problem

2.1 Opis problema

5.25"flopi disketa je istoriski primerak medijuma koji je korišćen za skladištenje podataka. Razvojem računarstva i postizanjem boljih performansi računara kao i potrebama za većom količinom memorije ovaj medium je izgubio svoje mesto na tržištu i preselio se u istoriju. Njegove karakteristike nisu mogle da ispune ni prostorne a ni performansne potrebe. Vreme je pregazilo ovaj medium a da bi se sačuvali podaci i zabeležio bitan deo istorije pribegavano je skaldištenju i čuvanju podataka u *D64 fajlovima.*

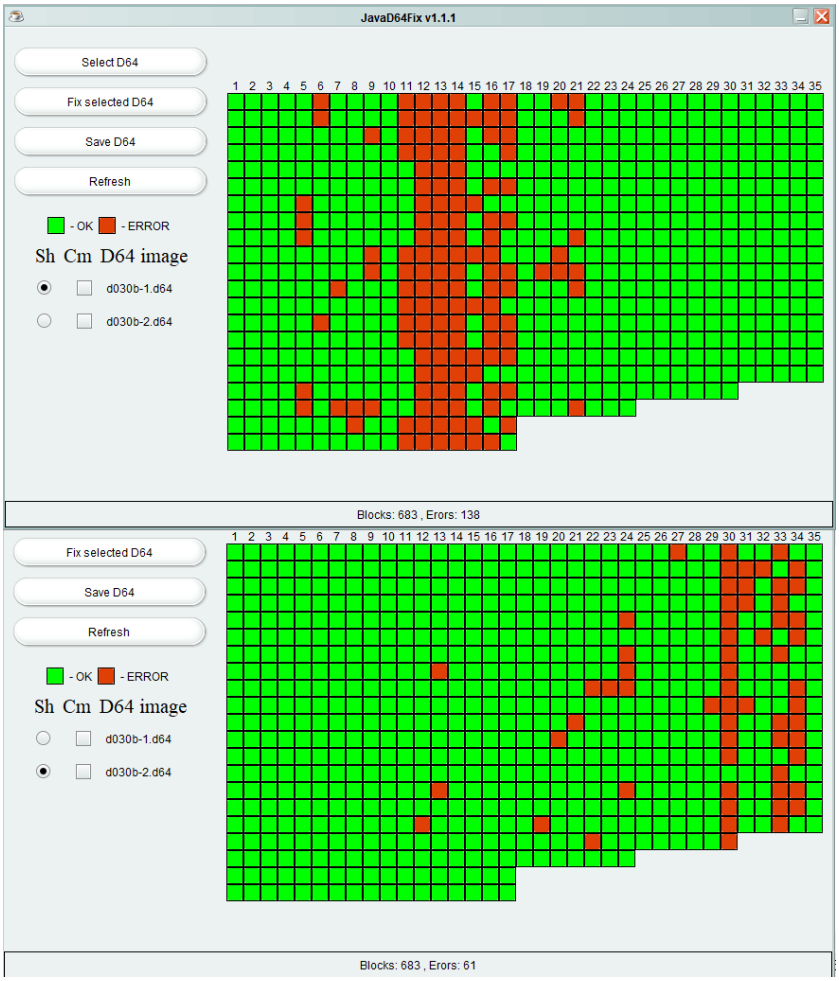
Prilikom čitanja *5.25"flopi diskete* i prebacivanja njenih podataka u elektronski oblik odnosno *D64 fajl* dolazi do pojave grešaka na nekim sektorima. Nastale greške prilikom čitanja se skladište u dodatnim bajtovima na kraju *D64 fajla* i dalja evidencija grešaka se vrši isključivo preko ovih dodatnih bajtova. Ovaj problem je predstavljao kamen spoticanja prilikom arhiviranja podataka sa ovih mediuma. Kao uzrok možemo navesti zastarelost tehnologije, kao i oštećenost mediuma. Primer jednog lošeg čitanja prikazan je na slici 3 gde crvena boja označava sektore sa greškom a zelena ispravne.



Slika 3: Loše čitanje 5.25"flopi diskete

2.2 Analiza problema

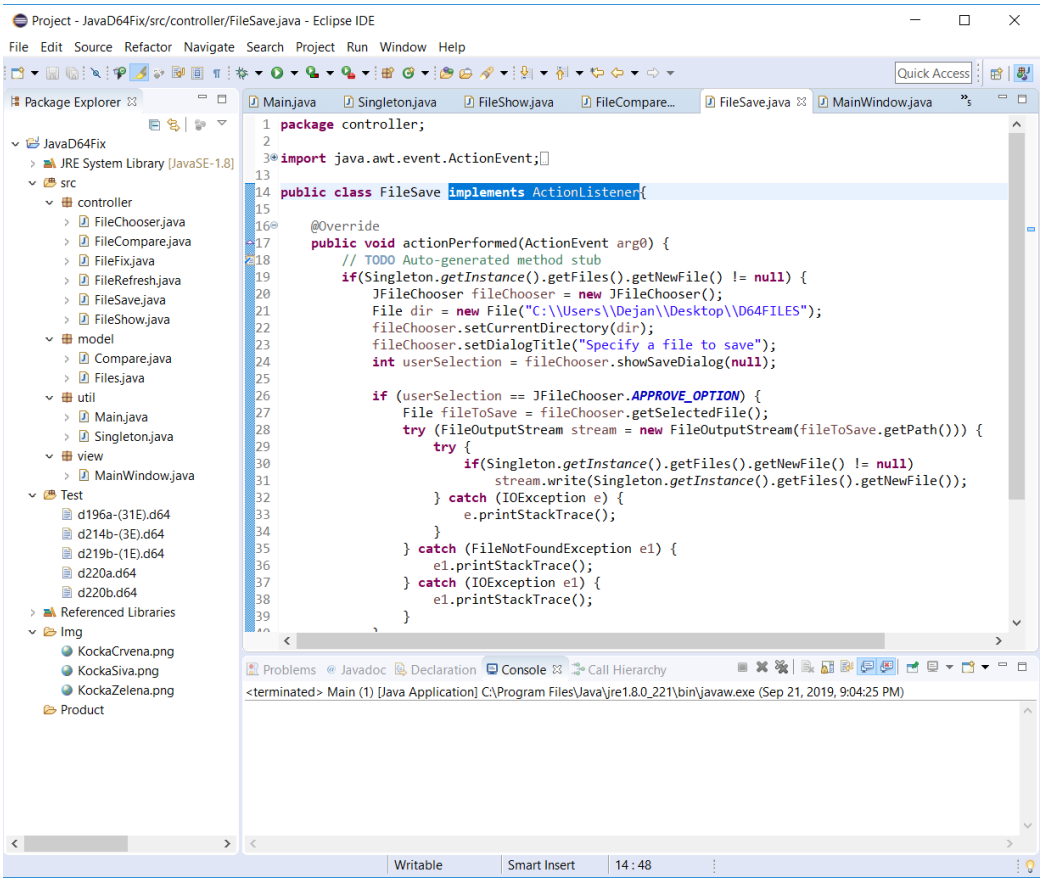
Kao što je već rečeno prilikom čitanja *5.25"flopi disketa* dolazi do neželjenih grešaka nad pojedinim sektorima, odnosno određeni sektori nisu dobro pročitani. Zbog analize ovog problema realizovano je programsko rešenje koje na osnovu dodatnih bajtova na kraju *D64 fajla* omogućuje vizuelnu predstavu sekotra. Ispravni sektori obeleženi su zelenom bojom dok su neispravni obeleženi crvenom. Analizom *D64 fajlova* ustanovljeno je da se prilikom višestrukog čitanja iste flopi diskete javljaju greške u različitim sektorima, odnosno da prilikom prvog i drugog čitanja nisu isti sektori pogrešno pročitani. Vizuelni prikaz dva *D64 fajla* iste *5.25"flopi diskete* prikazan je na slici 4.



Slika 4: Višestruko čitanje iste 5.25"flopi diskete

3 Opis korišćenih tehnologija i alata

Programsko rešenje je razvijeno uz pomoć *Java* programskog jezika i *Eclipse* programskog razvojnog okruženja prvenstveno namenjenog za razvoj *Java* aplikacija. Pored *Jave Eclipse* podržava i druge programske jezike [6]. Izgled *Eclipse* razvojnog okruženja je prikazan na slici 5.



Slika 5: Eclipse

Za rad sa grafičkim elementima korišćen je *Swing* radni okvir koji obezbedjuje skup grafičkih komponenti uz pomoć kojih realizujemo grafički korisnički interface u java programskom jeziku.

4 Rešenje problema

4.1 Opis rešenja problema

Na osnovu analize *D64 fajla* ustanovljeno je da se višestrukim čitanjem iste *5.25 flopi diskete* dobijaju fajlovi sa greškama na različitim sektorima. Na osnovu ovoga konstruisano je rešenje gore opisanog problema koje je zasnovano na kombinaciji više *D64 fajlova* sa ciljem da se kreira novi fajl koji bi imao manje ili u idealnom slučaju bio bez grešaka. Suština postupka bi bila da se sektori koji imaju grešku zamene sa ispravnim sektoirma iz drugih *D64 fajlova* iste *5.25 flopi diskete* ukoliko oni postoje.

4.2 Programsko rešenja problema

Pošto svi fajlovi koji učestvuju u procesu imaju greške, algoritam počinje traženjem fajla koji ima najmanje grešaka jer to dalje smanjuje broj pretraga prilikom zamene sektora sa greškama. Dat je kod koji se bavi ovim procesom, gde se prvo učitavaju svi bajtovi tekućeg fajla i zatim u zavisnosti od dimenzija vrši se prebrojavanje sektora pod greškama i ukoliko taj fajl ima manje grešaka od trenutno najboljeg on se postavlja za novi najbolji. Postupak se ponavlja po principu traženja minimuma odnosno fajla sa najmanje grešaka.

```
byte[] newFile = null;
int best = 1000;
for (File file : Singleton.getInstance().getCompare()
                                        .getCompare()) {
    byte[] fileContent = null;
    try {
        fileContent = Files.readAllBytes(file
                                        .toPath());
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

    if(fileContent.length == 175531) {
        int temp = 0;
        for(int i=fileContent.length-1 ;
         i > fileContent.length-684 ; i--) {
            if(fileContent[i] != 1) {
                temp++;
            }
```

```
        }
        if(temp <= best) {
                best = temp;
                newFile = fileContent;
        }
    }else if(fileContent.length == 197376){
        int temp = 0;
        for(int i=fileContent.length-1 ;
        i > fileContent.length-769 ; i--) {
            if(fileContent[i] == 1) {
            }else {
                temp++;
            }
        }
        if(temp <= best) {
                best = temp;
                newFile = fileContent;
        }
    }else if(fileContent.length == 196608
            || fileContent.length == 174848)
        newFile = fileContent;
        best = -1;
    }
    Singleton.getInstance().getCompare()
                    .setFiles(fileContent);
}
```

Po pronalasku fajla sa najmanje grešaka, prolaskom kroz bajtove koji govore o greškama nad sektorima, uzimajući redom sektore sa greškama, proveravamo da li postoji fajl u kome je taj sektor ispravan. Ukoliko postoji vrši se kopiranje bajtova ispravnog sektora u bajtove neispravnog. Ovaj postupak se nastavlja do poslednjeg sektora sa greškama i u idealnom slučaju rezultuje fajlom bez grešaka.Dat je kod koji se bavi ovim postpukom za fajlove koji oslikavaju *5.25 flopi disketu* sa 35 traka dok je za 40 traka postupak zansovan na istom principu samo sa različitim brojem dodatnih bajtova za greške.

```
for(int i=newFile.length-683
    ; i < newFile.length-1 ; i++) {
if(newFile[i] != 1) {
        for (byte[] b : Singleton
        .getInstance()
```

```
            .getCompare().getFiles()) {
                    if(b[i] == 1) {
                            for(int j = (i-174848)*256;
                            j < (i-174848)*256+256 ; j++) {
                                    newFile[j]
                                        = b[j];
                            }
                            newFile[i] = 1;
                            break;
                    }
            }
}
}
```

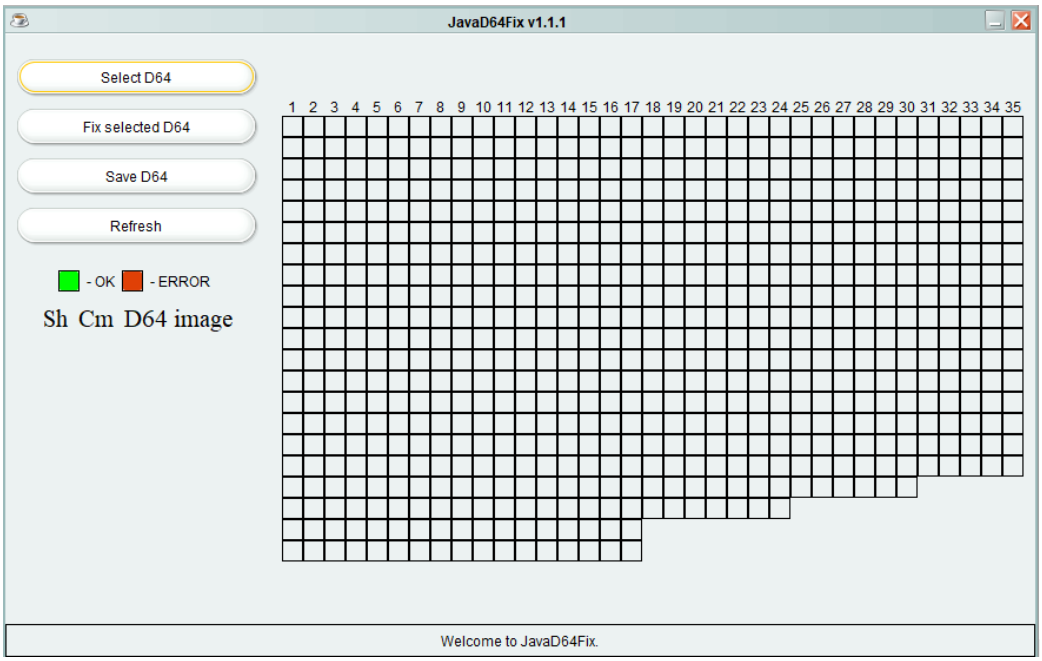
4.3 Interne strukture podataka

Za interno skladištenje putanja do selektovanih *D64 fajlova* korišćen je niz fajlova (java. File [] files). Takođe skladištenje *D64 fajlova* vršeno je i u vidu bajt predstave zbog kasnijeg poređenja i u tom slučaju je korišćena lista čiji elementi su bili nizovi bajtova (java. List<byte[]>). Putanja ka trenutno selektovanom fajlu za vizuelni prikaz čuvana je u fajlu (java. File), dok je bajt predstava na osnovu koje je vršeno iscrtavanje skladištena u nizu bajtova (java. byte[]).

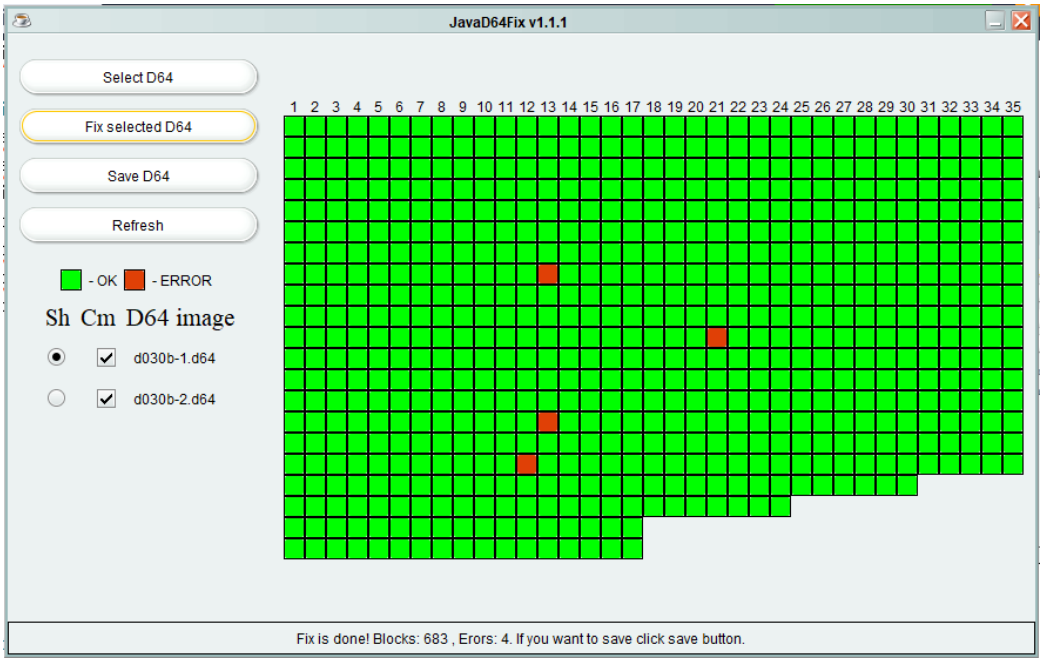
4.4 Prikaz implementiranog rešenja

Početni izgled aplikacije prikazan je na slici 6. Klikom na dugme *Select D64* vršimo izbor fajlova. Klikom na dugme *Fix selected D64* pokreće se gore opisani algoritam za popravku selektovanih fajlova i interno kreiranje novog sa minimalnim brojem grešaka. Klikom na dugme *Save D64* omogućava se čuvanje prethodno popravljenog i prikazanog fajla na lokaciji koju korisnik odabere. Klikom na dugme *Refresh* vrši se deselekcija svih selektovanih fajlova kao i selekcija fajlova za prikaz.

Selekcija fajlova se vrši na osnovu čekera u levom donjem delu aplikacije a prikaz na osnovu radio dugmadi. Brojevi kod vizuelnog prikaza govore o kojoj traci *5.25 flopi diskete* je reč dok donja traka služi za prikaz svih informacija bitnih za korisnika [7]. Izgled ovog dela aplikacije prikazan je na slici 7.



Slika 6: Izgled aplikacije



Slika 7: Izgled aplikacije

5 Zaključak

U ovom radu predstavljeno je rešenje za problem lošeg čitanja *5.25"flopi disketa* i njegovo pretvaranje u elektronski *D64 fajl*. Opisan je kompletan postupak kojim je to postignuto kao i sve potrebne informacije za razumevanje problema i njegovo rešenje.

Rešenje je proizišlo iz detaljne analize D64 fajla kao i procesa čitanja podataka sa *5.25"flopi diskete* gde je oučeno da se višestrukim čitanjem javljaju greške u različitim sektorima. Ova informacija bila je ključna prilikom otklanjanja ovog problema u većoj ili manjoj meri.

Proširenje datog programskog rešenja može se kretati u smeru vizualizacije izmenjenih sektora sa greškama u vidu prikaza podataka i ručnom kontrolom izmene istih, gde bi korisnik ovaj automatizovan proces mogao kontrolisati. Takođe prilikom nailaska na sektore koji ne mogu biti popravljeni korisnik bi dobio mogućnost da izabere i sa posmatranim sektorom zameni sektor sa najmanje oštećenja ili sa sektorom čiju grešku smatra najpovoljnijom iz njegovog aspekta gledanja.

6 Literatura

[1] Commodore 1541 Img. https://assets.catawiki.nl/assets/2018/9/2/c/e/9/ce9bdd48-747e-467e-b6a6-e8b76614f1ed.jpg. Pristupljeno: 21.9.2019.

[2] 5.25"flopi disketa. https://d2t1xqejof9utc.cloudfront.net/screenshots/pics/44b7851875d4549de01cf35ffc6d9fb7/large.jpg. Pristupljeno: 22.9.2019.

[3] Commodore 1541. https://www.c64-wiki.com/wiki/Commodore_1541. Pristupljeno: 21.9.2019.

[4] 5.25"flopi disketa. https://obsoletemedia.org/5-25-inch-minifloppy-disk/. Pristupljeno: 22.9.2019.

[5] D64 (Electronic form of a physical 1541 disk). http://unusedino.de/ec64/technical/formats/d64.html. Pristupljeno: 20.9.2019.

[6] Eclipse. https://en.wikipedia.org/wiki/Eclipse_(software). Pristupljeno: 22.9.2019.

[7] Kod projekta. https://github.com/DejanPredojevic/Diplomski-rad. Pristupljeno: 23.9.2019.