Android aplikacija za rješavanje sustava linearnih jednadžbi

Šarčević, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski sveučilišni studij računarstva

ANDROID APLIKACIJA ZA RJEŠAVANJE SUSTAVA LINEARNIH JEDNADŽBI

Diplomski rad

Tomislav Šarčević

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 23.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Tomislav Šarčević
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 887 R, 22.09.2017.
OIB studenta:	32975316739
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva:	Doc.dr.sc. Anita Katić
Naslov diplomskog rada:	Android aplikacija za rješavanje sustava linearnih jednadžbi
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Student će izraditi android aplikaciju za rješavanje sustava linearnih jednadžbi na mobitelima. Sumentor: Alfonzo Baumgartner
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Dovoljan (2)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 1 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 1 razina
Datum prijedloga ocjene mentora:	23.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTIRADA

Osijek, 27.09.2018.

Ime i prezime studenta:	Tomislav Šarčević
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 887 R, 22.09.2017.
Ephorus podudaranje [%]:	6%

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za rješavanje sustava linearnih jednadžbi**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ:

1. UVOD

Zadatak je diplomskog rada izrada Android aplikacije koja će riješavati sustav linearnih jednadžbi. Kako je Android vrlo raširena platforma u svijetu mobilnih aplikacija, aplikacije za Android uređaje mogu doprijeti do velikog broja učenika i pomoći im u svladavanju gradiva vezanog za ovaj problem, što je i glavna nit vodilja prilikom kreiranja ideje i realizacije ovog zadatka.

Za ispravnu funkcionalnost aplikacije potrebno je dobro poznavanje matematike, odnosno linearne algebre, principa izrade mobilnih aplikacija, kao i programskog jezika Java. Za rad na projektu korišteno je okruženje Android Studio, koje je nakon Eclipse-a postalo standard za izradu mobilnih aplikacija za navedenu platformu.

Aplikacija se sastoji od korisničkog dijela koji sadrži elemente za komunikaciju s korisnikom te programerskog dijela kojeg korisnik ne vidi, a u kojem se obavlja sva logika aplikacije. Sam rad sadrži poglavlja u kojima se opisuju tehnologije korištene za nastanak aplikacije, teoretska podloga svih elemenata aplikacije i, na posljetku, opis faza nastanka aplikacije.

1.1. Zadatak diplomskog rada

Zadatak je diplomskog rada izraditi android aplikaciju za rješavanje sustava linearnih jednadžbi na mobitelima.

2. ALATI

2.1. Android

Android je operacijski sustav otvorenog koda koji je u najvećoj mjeri namijenjen za mobilne uređaje, a osim za njih, zbog svoje izrazite prilagodljivosti, može se koristiti i u televizorima, satovima te automobilima. Razvijan je u strogoj tajnosti od 2003. godine, a razvijali su ga Rich Miner, Chris White, Nick Sears i Andy Rubin. Tvrtku je kupio Google 2005. godine i time je počeo proboj Androida na tržište mobilnih telefona.[1]

Android je baziran na Linuxu, a prvi mobilni telefon u kojeg je bio ugrađen Android operacijski sustav bio je HTC Dream. Verzije Android operacijskog sustava su, redoslijedom izdavanja, sljedeće: Petit Four (veljača 2009.), Cupcake (travanj 2009.), Donut (rujan 2009.), Eclair (listopad 2009.), Froyo (svibanj 2010.), Gingerbread (prosinac 2010.), Honeycomb (veljača 2011.), Ice Cream Sandwich (listopad 2011.), JellyBean (srpanj 2012.), KitKat (listopad 2013.), Lollipop (studeni 2014.), Marshmallow (listopad 2015.), Nougat (kolovoz 2016.), Oreo (kolovoz 2017.) te Android P (još nije službeno prezentirano). [2]

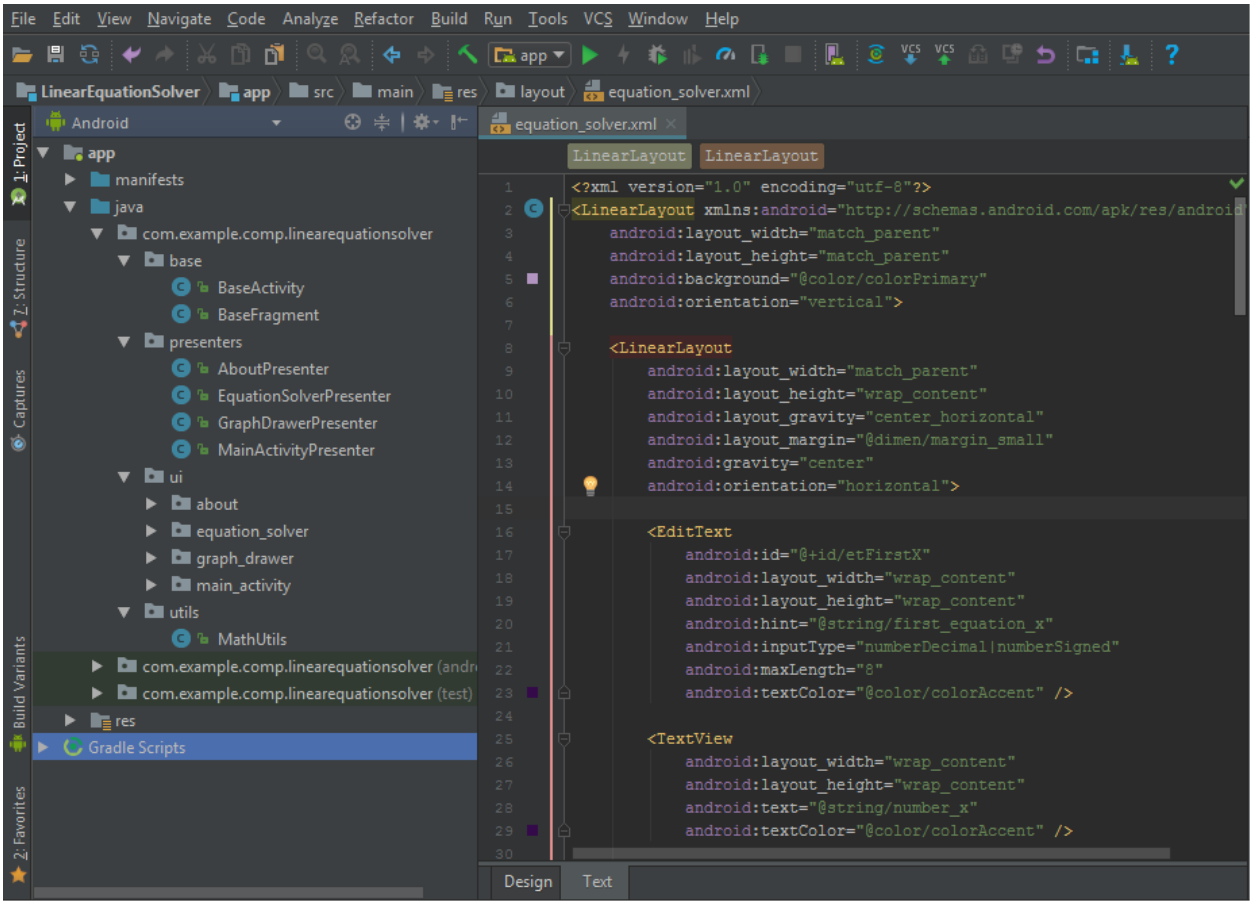
Što se tiče same arhitekture Androida, na dnu se nalazi jezgra pisana u Linuxu i ta jezgra sadrži drivere za međuprocesnu komunikaciju te power management driver. Iznad jezgre nalaze se biblioteke pisane u C i C++ programskom jeziku, a neke su od njih SQLite (upravljanje bazom podataka), Media Framework (snimanje i reprodukcija audio i video formata), SSL (sigurnosna komunikacija pomoću interneta) i mnoge druge. Iznad toga nalazi se sloj za pokretanje aplikacija, a iznad njega aplikacijski okvir koji dopušta upotrebu API-ja. Na kraju se nalaze aplikacije koje mogu biti raznih namjena, od osnovnih, poput aplikacija za slanje SMS poruka ili za prikaz e-mailova, do nekih izvedenih, tj. onih napisanih od strane određenih razvojnih programera, a koje se mogu skinuti u trgovini Google Play.[3]

2.2. Android Studio

Android Studio razvojno je okruženje za razvijanje Android aplikacija. Razvijen je 2013. godine kao zamjena za Eclipse Android Development Kit koji je do tada bio glavno razvojno okruženje. Zasnovan je na IntelliJ softveru razvijenom od strane JetBrainsa.

Za rad u razvojnom okruženju potrebno je računalo s operacijskim sustavom Windows 7 i većim, MAC operacijskim sustavom ili Gnome Linux operacijskim sustavom. Potrebno je minimalno tri GB radne memorije, što bi bilo najoptimalnije, a ako korisnik želi koristiti mogućnost emulatora, potreban je još jedan dodatni GB radne memorije, kao i Java Development Kit 8. [4]

Prilikom razvoja Android aplikacije postoji standardni zaslon razvojnog okruženja unutar kojeg se može mijenjati dizajn aplikacije, kao i sami kod unutar java datoteka. Na slici 2.1. može se vidjeti izgled zaslona razvojnog okruženja. Na njemu se s lijeve strane ekrana nalazi struktura projekta. Glavni dijelovi projekta nalaze se unutar java i res datoteka. Unutar java datoteka nalazi se sav kod koji je potreban za funkcioniranje aplikacije. Kod se može podijeliti na više datoteka, kako bi bilo jednostavnije snalaženje u klasama i međuovisnostima između klasa. Unutar res datoteke nalaze se layout datoteke u kojima se pravi dizajn same aplikacije. Tu se također nalazi i string datoteka unutar koje se upisuju tekstovi koji se koriste u aplikaciji, a prednost je takvog zapisa u tome što se na jednom mjestu može promijeniti neki tekst koji se nalazi na više lokacija unutar same aplikacije. Unutar res datoteke spremamo također i sve fotografije koje se koriste za samu aplikaciju. Uz navedeno, na slici se može vidjeti i kako izgleda razvoj dizajna aplikacije, a to se može postići na dva načina. Jedan je način pomoću opcije *drag-and-drop* gdje korisnik sam prenosi elemente koje želi koristiti unutar dizajna, a drugi je način pisanje XML koda koje nam daje veće mogućnosti prilikom određivanja međuovisnosti elemenata. XML kratica je za *ExtensibleMarkupLanguage*, jezik za označavanje podataka.



Sl. 2.1. Izgled Android Studio razvojnog okruženja

2.3. Java

Java je programski jezik kojega su 1990-ih godina razvili James Gosling i Patrick Naughton koji su radili u poduzeću Sun Microsystems. Četiri godine bile su potrebne za razvoj programskog jezika, tako da je objavljen tek 1995. godine. Java je objektno orijentirani programski jezik koji je nastao po uzoru na C programski jezik. Prednost je nad C programskim jezikom u tome što je za programe pisane u C-u bilo potrebno prilagođavati kod u ovisnosti o operacijskom sustavu na kojem se kod izvodi, dok je Javin programski kod mogao pokretati svaki operacijski sustav koji podržava JVM (eng. *Java VirtualMachine*).

Kako bi se razvio kod u Javi, potrebno je kreirati datoteku ekstenzije .java u koju se piše programski kod. Za razliku od C-a, kod Jave se sav kod mora pisati unutar klasa, a naziv klase mora biti jednak nazivu datoteke u koju je pisan kod.

Jedna od najbitnijih osobitosti Jave tipovi su podataka. Prilikom svakog inicijaliziranja varijabli mora se odabrati tip podatka za tu varijablu. Za spremanje određenog redoslijeda znakova odabire se tip podatka string, a ako se želi spremiti određeni broj, može se birati između nekoliko tipova podataka, kao što su integer, double i float. U tablici 2.1. može se vidjeti popis svih tipova podataka dostupnih u Java programskom jeziku, a uz to se može vidjeti i količina memorije koju zauzima svaki tip, kao i vrsta podatka koji se sprema u takav tip. Tipovi podataka u Javi slični su onima i u C-u, samo se neki od njih, primjerice boolean, drugačije pišu.

Tip podatka	Vrsta podataka	Memorija (bajtovi)
boolean	Logički podataka	1
byte	Cijeli broj	1
int	Cijeli broj	4
short	Cijeli broj	2
long	Cijeli broj	8
float	Decimalni broj	4
double	Decimalni broj	8
string	Slijed znakova	/
char	Jedan znak	2

Tab. 2.1. Prikaz tipova varijabli u Javi

Java podržava sve osnovne aritmetičke operacije poput zbrajanja, oduzimanja, množenja, dijeljenja, inkrementiranja, dekrementiranja, itd. Osim aritmetičkih, Java također podržava i relacijske operacije, a to su: veće od (>), manje od (<), veće ili jednako (>=), manje ili jednako (<=), jednako (==) i različito (!=). Također se mogu koristiti i logičke operacije nad elementima, a to su: logička negacija (!), logičko I (&&) i logičko ILI (||), kao i sve izvedenice ove 3 logičke operacije.

Relacijske i logičke operacije najčešće se koriste unutar petlji i uvjetnog grananja, koje je također podržano unutar Jave. Primjer korištenja petlje i uvjetnog grananja može se vidjeti može se vidjeti na slici 2.2. U *for* petlji postavljamo vrijednost varijable i na 0, a unutar petlje provjerava se uvjet je li *i* veći od 25. Ako jest, na ekran se ispisuje poruka „I je veci od 25“.

```
for(i=0;i<50;i++) {
      if(i>25) {
            printf(„I je veci od 25“);
      }
}
```

Slika 2.2. Primjer *for* petlje

Osim *for* petlje za kontrolu toka programa koriste se i naredbe poput *switch*, *while*, *do while*, *if*, *if-else*. *If* grananje koristi se kao uvjetno grananje, a izvodi se na način da se provjerava istinitost izraza unutar oblih zagrada. Ako je izraz istinit, programski prevoditelj izvršava kod koji se nalazi unutar vitičastih zagrada, a ako nije, tada programski prevoditelj nastavlja s izvršavanjem koda koji dolazi nakon *if* programskog dijela.

```
if(izraz) {
    //izvrši nekakav kod
}
```

Slika 2.3. Primjer *if* grananja

Osim običnog *if* grananja postoji i *if-else* grananje. Razlika između ta dva načina grananja u tome je što programski prevoditelj prelazi na *else if* izraz, ako je prvi *if* neizvediv. Unutar programskog koda programer može napisati beskonačan broj *elfe if* grananja.

```
if(izraz1) {
    //izvrši nekakav kod
} else if (izraz2) {
    //izvrši nekakav kod
} else if (izraz3) {
```

```
    //izvrši nekakav kod
}
...
   else {
    //izvrši nekakav kod
}
```

Slika 2. 4. Primjer *if-else* grananja

Kako ne bi došlo do prekomjernog ponavljanja *else if* dijela koda, unutar Jave postoji i *switch* grananje. Prilikom korištenja *switch* naredbe predaje mu se varijabla čija se vrijednost provjerava unutar *case* ključnih riječi. Svaka *case* grana mora završavati ključnom riječju *break;* čime se automatski izlazi iz *switch* grananja i programski prevoditelj nastavlja s izvršavanjem koda izvan te naredbe. Na sici 2. 5. može se vidjeti primjer korištenja *switch* grananja. U ovom je slučaju predefinirana vrijednost varijable „broj“ na 5 i predaje se *switch* naredbi. Tada programski prevoditelj provjerava svaki *case,* to jest je li predani broj jednak broju koji je predviđen za izvršavanje koda unutar *case* komponente. Ako ni jedan slučaj ne odgovara predanom parametru, tada se izvršava predefinirana akcija. U ovom bi slučaju programski prevoditelj na ekran ispisao : „Broj je ili manji od 1, ili veći od 4“.

```
int broj = 5;

switch (broj) {
    case 1:
        println("Broj 1");
        break;
    case 2:
        println("Broj 2");
        break;
    case 3:
        println("Broj 3");
        break;
    case 4:
        println("Broj 4");
        break;
    default:
        println("Broj je ili manji od 1, ili
veći od 4");
        break;
}
```

Slika 2. 5. Primjer *switch* grananja

Osim *for* petlje mogu se koristiti i još dva načina za prolazak kroz petlju, a to su *while* i *do while* petlje. Razlika između te dvije vrste petlji i *for* petlje u tome je što je kod *for* petlje unaprijed

poznat broj ponavljanja petlje, dok se kod *while* i *do while* petlje ne zna točan broj ponavljanja. Kod *while* petlje provjerava se izraz unutar zagrade. Ako je on istinit, izvršava se kod unutar tijela funkcije, a ako nije, programski prevoditelj nastavlja s kodom nakon tijela funkcije *while* petlje. Na slici 2. 6. može se vidjeti jednostavan primjer *while* petlje - tijelo funkcije ponavljat će se sve dok varijabla „broj“ ne poprimi vrijednost veću od 4.

```
int broj = 0;

while ( broj < 5) {
    broj++;
}
```

Slika 2. 6. Primjer *while* petlje

Za razliku od *while* petlje koja se ne mora izvršiti nijednom, *do while* petlja izvršavat će se barem jedanput. Isto kao i kod *while* petlje, provjerava se izraz unutar zagrade, a ako je on istinit, tijelo *do while* petlje izvršit će se. Na slici 2. 7. može se vidjeti primjer kako se tijelo *do while* petlje izvršava jednom, iako je izraz unutar oblih zagrada neistinit. Taj primjer će jednom ispisati string „Tijelo funkcije“ i programski će prevoditelj nastaviti obrađivati kod nakon petlje.

```
int broj = 0;

do {
    println("Tijelo funkcije");
}while(broj != 0);
```

Slika 2. 7. Primjer korištenja *do while* petlje

Osim navedenih opcija, Java također podržava i korištenje funkcija. Funkcije se, kako bi se mogle koristiti, moraju definirati unutar klase. Same funkcije mogu pripadati klasi ili objektu te klase, ovisno o tome kako su definirane. Ako uz ime funkcije stoji ključna riječ *static*, tada funkcija pripada klasi i ne mora se instancirati objekt te klase za korištenje funkcije. Ako ne postoji ključna riječ *static*, tada funkcija pripada objektu kojeg je potrebno instancirati. Na slici 2. 8. može se vidjeti način definiranja funkcije, dok se na slici 2. 9. može vidjeti način pozivanja funkcije u ovisnosti o tome je li funkcija deklarirana kao *static* ili ne.

```
pravo_pristupa povratni_tip (static)
naziv_funkcije (tip_parametra naziv_parametra) {
    //tijelo funkcije

    return povratni_tip;
```

```
}
```

Slika 2. 8. Primjer funkcije

Oznaka pravo _pristupa može biti *public, private* i *protected*. Ako je pravo pristupa *public*, to znači da se toj funkciji može pristupiti iz svih klasa unutar projekta. Ako je pravo pristupa *private*, to znači da se toj funkciji može pristupiti samo unutar klase u kojoj je deklarirana funkcija. Ako je pravo pristupa *protected*, to znači da se toj funkciji može pristupiti unutar iste klase, naslijeđenih klasa i datoteke unutar koje se nalazi ta funkcija.

Oznaka povratni_tip može biti svaki od tipova podataka navedenih unutar Tab 2.1., a osim tih vrsta, može biti i *void*, to jest funkcija neće vratiti nikakvu vrijednost.

Oznaka naziv_funkcije mora započinjati slovima, prvi znak ne smije biti „_“ ili „$“, a naziv funkcije trebao bi što preciznije opisivati što ta funkcija radi.

```
class Brojevi {
    public static int uvecajBrojBezObjekta(int broj) {
            return broj++;
        }

    public int uvecajBrojSObjektom (int broj) {
            return broj++;
    }
}

public class Main () {
    int objektBroj = 1;
    int klasaBroj = 2;

    Brojevi brojObjekt = new Brojevi();

    Brojevi.uvecajBrojBezObjekta(klasaBroj);

    brojObjekt.uvecajBrojSObjektom(objektBroj);

      //objektBroj = 2 klasaBroj = 3
}
```

Slika 2. 9. Primjer korištenja *static* funkcija

U ovom isječku koda može se vidjeti način funkcioniranja *static* i običnih funkcija. Kako bi pozvali funkciju „uvecajBrojBezObjekta“, nije potrebno instancirati objekt klase „Brojevi“, a za pozivanje funkcije „uvecajBrojSObjektom“ potrebno je bilo instancirati objekt „brojObjekt“.

2.4. Sustav linearnih jednadžbi

Linearna jednadžba prvog reda ima oblik: $ax = c, a \neq 0$. Rješenje je linearne jednadžbe vrijednost $x = \frac{c}{a}$.

Sustav linearnih jednadžbi ima oblik:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$\vdots \quad \vdots \quad \ddots \quad \vdots \quad = \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

Rješenje sustava čine oni parovi $(x_1, x_2, \ldots, x_n)$ koji zadovoljavaju svaku navedenu linearnu jednadžbu, to jest, riječ je o točkama koje leže na svih ravninama određenim ovim jednadžbama.[5]

U primjeru korištenja ove aplikacije koriste se dvije linearne jednadžbe drugoga reda:

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$

Moguća su tri rješenja ovog sustava linearnih jednadžbi, a to su:

1. sustav linearnih jednadžbi ima beskonačno mnogo rješenja
2. sustav linearnih jednadžbi nema niti jedno rješenje
3. sustav linearnih jednadžbi ima točno jedno rješenje

Postoji nekoliko načina rješavanja sustava linearnih jednadžbi, a to su Cramerovo pravilo, Gaussova metoda eliminacije, metoda suprotnih koeficijenata i metoda supstitucije, a Kronecker-Cappelijev teorem opisuje strukturu rješenja sustava linearnih jednadžbi. Prilikom razvoja ove aplikacije korišteno je Cramerovo pravilo za rješavanje sustava linearnih jednadžbi.

2.4.1. Cramerovo pravilo

Prilikom rješavanja sustava linearnih jednadžbi Cramerovim pravilom potrebno je poznavati računanje determinanti. Ako bi se sustav jednadžbi zapisao u matričnom obliku, dobio bi se idući prikaz:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Rješenje su točke $x_i = \frac{D_i}{D}$, gdje D_i predstavlja determinantu čiji je *i*-ti stupac zamijenjen stupcem *b*, a *D* predstavlja determinantu lijeve strane sustava linearnih jednadžbi.

Ako je $D = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{vmatrix} = 0$, tada se provjerava D_i. Ako je $D_i = 0$ tada sustav jednadžbi ima beskonačno rješenja (*x, y*). Ako je $D_i \neq 0$, tada sustav jednadžbi nema niti jedno rješenje (*x, y*) koje zadovoljava navedeni sustav.

Ako je $D = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{vmatrix} \neq 0$, tada sustav ima samo jedno rješenje (*x, y*) koje zadovoljava sustav.

Primjer rješavanja sustava linearnih jednadžbi pokazati će se na primjeru sustava tri jednadžbe s tri nepoznanice:

$5x_1 + 3x_2 - 4x_3 = 5$

$-3x_1 + x_2 + 3x_3 = 13$

$4x_1 + 2x_2 - x_3 = 10$

Započinje se na način da se sustav prikaže u matričnom obliku.

$$\begin{bmatrix} 5 & 3 & -4 \\ -3 & 1 & 3 \\ 4 & 2 & -1 \end{bmatrix} = \begin{bmatrix} 5 \\ 13 \\ 10 \end{bmatrix}$$

Nakon toga računa se determinanta $D = \begin{vmatrix} 5 & 3 & -4 \\ -3 & 1 & 3 \\ 4 & 2 & -1 \end{vmatrix} = 5 * \begin{vmatrix} 1 & 3 \\ 2 & -1 \end{vmatrix} + 3 * \begin{vmatrix} 3 & -4 \\ 2 & -1 \end{vmatrix} + 4 * \begin{vmatrix} 3 & -4 \\ 1 & 3 \end{vmatrix} = 5 * (-1 - 6) + 3 * (-3 + 8) + 4 * (9 + 4) = 32 \neq 0$

Nakon što se provjeri nejednakost $D \neq 0$, može se izračunati D_i za svaku od nepoznanica.

$$D_{x1} = \begin{vmatrix} 5 & 3 & -4 \\ 13 & 1 & 3 \\ 10 & 2 & -1 \end{vmatrix} = 5 * \begin{vmatrix} 1 & 3 \\ 2 & -1 \end{vmatrix} - 13 * \begin{vmatrix} 3 & -4 \\ 2 & -1 \end{vmatrix} + 10 * \begin{vmatrix} 3 & -4 \\ 1 & 3 \end{vmatrix}$$

$$= 5 * (-1 - 6) - 13 * (-3 + 8) + 10 * (9 + 4) = 30$$

$$D_{x2} = \begin{vmatrix} 5 & 5 & -4 \\ -3 & 13 & 3 \\ 4 & 10 & -1 \end{vmatrix} = 5 * \begin{vmatrix} 13 & 3 \\ 10 & -1 \end{vmatrix} + 3 * \begin{vmatrix} 5 & -4 \\ 10 & -1 \end{vmatrix} + 4 * \begin{vmatrix} 5 & -4 \\ 13 & 3 \end{vmatrix}$$

$$= 5 * (-13 - 30) + 3 * (-5 + 40) + 4 * (15 + 52) = 158$$

$$D_{x3} = \begin{vmatrix} 5 & 3 & 5 \\ -3 & 1 & 13 \\ 4 & 2 & 10 \end{vmatrix} = 5 * \begin{vmatrix} 1 & 13 \\ 2 & 10 \end{vmatrix} + 3 * \begin{vmatrix} 3 & 5 \\ 2 & 10 \end{vmatrix} + 4 * \begin{vmatrix} 3 & 5 \\ 1 & 13 \end{vmatrix}$$

$$= 5 * (10 - 26) + 3 * (30 - 10) + 4 * (39 - 5) = 116$$

Prema jednadžbi $x_i = \frac{D_i}{D}$ može se izračunati x_1, x_2 i x_3

$$x_1 = \frac{30}{32} = 0.9375$$

$$x_2 = \frac{158}{32} = 4.9375$$

$$x_3 = \frac{116}{32} = 3.625$$

2.4.2. Gaussova metoda eliminacije

Drugi način rješavanja sustava linearnih jednadžbi je pomoću Gaussove metode eliminacije. Prilikom korištenja te metode potrebno je sustav linearnih jednadžbi pretvoriti u matrični prikaz, nakon čega se izvršavaju matematičke operacije nad retcima unutar matrica. Postoje tri osnovne vrste operacije nad retcima, a to su zamjena dva retka, množenje jednog retka skalarom $\alpha \neq 0$ te dodavanje nekog retka drugom retku. Glavni je cilj algoritma dobiti donju trokutastu matricu $\begin{bmatrix} a_1 & a_2 & a_3 \\ 0 & a_4 & a_5 \\ 0 & 0 & a_6 \end{bmatrix}$ gdje su $a_1, a_4, a_6 \neq 0$. Dobiva se pomoću osnovnih operacija nad retcima.

Primjer rješavanja:

$5x_1 + 3x_2 - 4x_3 = 5$

$-3x_1 + x_2 + 3x_3 = 13$

$4x_1 + 2x_2 - x_3 = 10$

Kako bi se jednostavnije razumio ovaj način rješavanja, prva linearna jednadžba označava se s L1, druga s L2, a treća s L3.

Prvo se zapisuje sustav u matričnom obliku, a nakon toga gleda se kako najjednostavnije dobiti donju trokutastu matricu.

$\begin{bmatrix} 5 & 3 & -4 & 5 \\ -3 & 1 & 3 & 13 \\ 4 & 2 & -1 & 10 \end{bmatrix} \rightarrow \frac{3}{5} * L1 + L2 \rightarrow L2$, također i $(-\frac{4}{5}) * L1 + L3 \rightarrow L3$. Objašnjeno riječima to izgleda ovako: redak L3 pomnožen sa skalarom 3 i zbrojen s retkom L2 prebacuje se u redak L2. Isti postupak obavlja se i za redak L1.

$$\begin{bmatrix} 5 & 3 & -4 & \\ 0 & \frac{14}{5} & \frac{3}{5} & 5 \\ & & & 16 \\ 0 & \frac{-2}{5} & \frac{11}{5} & 6 \end{bmatrix} \rightarrow \frac{1}{7} L2 + L3 \rightarrow L3$$

$$\begin{bmatrix} 5 & 3 & -4 & 5 \\ 0 & \frac{14}{5} & \frac{3}{5} & 16 \\ 0 & 0 & \frac{80}{35} & \frac{58}{7} \end{bmatrix}$$

Iz posljednjeg matričnog prikaza može se vidjeti da je dobivena donja trokutasta matrica. Iz nje se može ponovno zapisati sustav jednadžbi:

$5x_1 + 3x_2 - 4x_3 = 5$

$\frac{14}{5} x_2 + \frac{3}{5} x_3 = 16$

$\frac{80}{35} x_3 = \frac{58}{7} \rightarrow x_3 = 3.625$

Nakon što je izračunat x_3, može ga se uvrstiti u drugu jednadžbu kako bi se pronašao x_2 i nakon toga oba rezultata uvrstiti u prvu jednadžbu kako bi se pronašao x_1.

$$\frac{14}{5}x_2 + \frac{3}{5} * 3.625 = 16 \rightarrow x_2 = 4.9375$$

$$5x_1 + 3 * 4.9375 - 4 * 3.625 = 5 \rightarrow x_1 = 0.9375$$

2.4.3. Kronecker-Capellijev teorem

Kronecker-Capellijev teorem opisuje strukturu rješenja sustava linearnih jednadžbi **Ax = B**, a ta struktura ovisi o rangu matrice **A** i rangu proširene matrice (**A**|**B**), gdje **A** predstavlja matricu $\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$, **x** predstavlja matricu $\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, a (**A**|**B**) predstavlja proširenu matricu $\begin{bmatrix} a_{11} & \ldots & a_{11} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{11} & \vdots & a_{11} & b_n \end{bmatrix}$. [6]

Uzimajući u obzir rang matrice **A** i proširene matrice (**A**|**B**), kao i broj redaka(*m*) i stupaca(*n*) matrice **A**, mogu se izvesti tri zaključka:

1.ako je r**(A)** = *m*, rješenje sustava biti će jedinstveno ako je *m*=*n*, a ako je *m*<*n*, tada sustav neće imati niti jedno rješenje

2.ako je r**(A)** = *n*, sustav može imati ili jedinstveno rješenje ili biti bez rješenja, ovisno o rangu proširene matrice. Ako je r((**A**|**B**)) = *n+1*, sustav nema rješenja, a ako je r((**A**|**B**)) = *n*, sustav ima jedinstveno rješenje

3.ako je r(**A**) <*n*, tada sustav može imati ili beskonačno rješenja ili niti jedno rješenja. Ako je r((**A**|**B**)) > *r(A)*, sustav nema rješenja, a ako je r((**A**|**B**)) = *r(A)*, sustav ima beskonačno rješenja.

Kroz tri primjera prikazat će se način funkcioniranja Kronecker-Capellijevog teorema.

Prvi primjer:

$$x_1 + x_2 = 5$$

$$x_1 + x_2 = 3$$

Ako se ovaj sustav zapiše u matričnom obliku, dobije se zapis $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$. Osnovnim matematičkim operacijama nad stupcima, potrebno je izračunati rang matrice. Kao i kod

Gaussove metode eliminacije, za jednostavniji zapis linearnih jednadžbi, koristit će se oznaka L1 za prvu jednadžbu i oznaka L2 za drugu jednadžbu. Prvo se računa rang matrice **A**.

$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} \rightarrow (-1) * L1 + L2 \rightarrow L2$. Nakon toga matrica **A** ima sljedeći izgled: $\begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}$, iz čega se može iščitati r(**A**)=1. Nakon toga se računa rang proširene matrice (**A**|**B**).

$\begin{bmatrix}1 & 1 & 5\\1 & 1 & 3\end{bmatrix} \rightarrow (-1) * L1 + L2 \rightarrow L2$. Nakon toga proširena matrica ima sljedeći izgled: $\begin{bmatrix}1 & 1 & 5\\0 & 0 & -2\end{bmatrix}$. Iz ove matrice se može iščitati r((**A**|**B**)) = 2. Ako se uzme u obzir 3. zaključak, može se vidjeti da ovaj sustav linearnih jednadžbi nema rješenja.

Drugi primjer:

$x_1 + x_2 = 5$

$2x_1 + 2x_2 = 10$

U matričnom prikazu sustav ima sljedeći izgled: $\begin{bmatrix}1 & 1\\2 & 2\end{bmatrix} = \begin{bmatrix}5\\10\end{bmatrix}$.

Prvo se računa r(**A**) uz pomoć Gaussovih transformacija, a nakon toga se provjerava rang proširene matrice.

$\begin{bmatrix}1 & 1\\2 & 2\end{bmatrix} \rightarrow (-2) * L1 + L2 \rightarrow L2$

$\begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}$ → iz ove matrice se može iščitati r(**A**) = 1. Nakon toga se računa rang proširene matrice.

$\begin{bmatrix}1 & 1 & 5\\2 & 2 & 10\end{bmatrix} \rightarrow (-2) * L1 + L2 \rightarrow L2$

$\begin{bmatrix}1 & 1 & 5\\0 & 0 & 0\end{bmatrix}$ → iz ove proširene matrice se može iščitati r((**A**|**B**)) = 1. S obzirom da je r(**A**) < *n*, provjerava se r((**A**|**B**)), koji je jednak r(**A**), čime se dolazi do zaključka da sustav ima beskonačno rješenja.

Treći primjer:

$2x_1 + 2x_2 = 4$

$2x_1 - 4x_2 = 10$

Kao i u prethodnim primjerima, sustav se zapisuje u matričnom obliku, nakon toga se računa rang matrice **A**, a nakon toga i rang proširene matrice (**A|B**).

$\begin{bmatrix} 2 & 2 \\ 2 & -4 \end{bmatrix} \rightarrow 2 * L1 + L2 \rightarrow L2$, nakon čega se dolazi do sljedećeg zapisa:

$\begin{bmatrix} 2 & 2 \\ 6 & 0 \end{bmatrix} \rightarrow (-\frac{1}{3}) * L2 + L1 \rightarrow L1$, nakon čega se dolazi do sljedećeg zapisa:

$\begin{bmatrix} 0 & 2 \\ 6 & 0 \end{bmatrix}$ čime se može doći do rješenja r(**A**) = 2.

Tada računamo rang proširene matrice:

$\begin{bmatrix} 2 & 2 & 4 \\ 2 & -4 & 10 \end{bmatrix} \rightarrow 2 * L1 + L2 \rightarrow L2$, nakon čega se dolazi do sljedećeg zapisa:

$\begin{bmatrix} 2 & 2 & 5 \\ 6 & 0 & 18 \end{bmatrix} \rightarrow \left(-\frac{1}{3}\right) * L2 + L1 \rightarrow L1$, nakon čega se dolazi do sljedećeg zapisa:

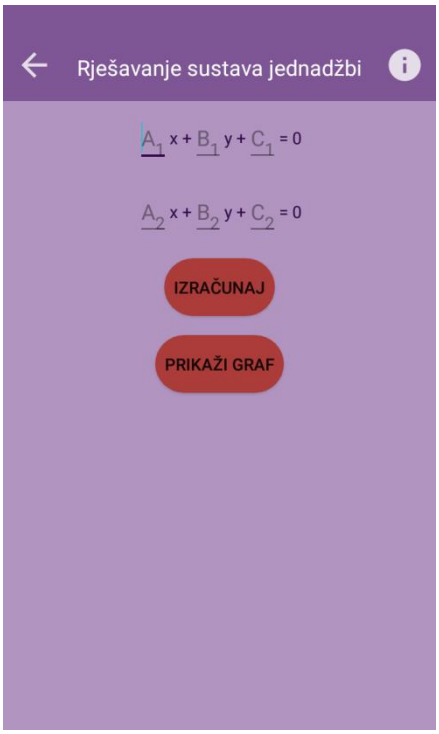
$\begin{bmatrix} 0 & 2 & 1 \\ 6 & 0 & 18 \end{bmatrix}$ →r((**A|B**)) = 2

Ako se uzme u obzir prvi zaključak, može se vidjeti da ovaj sustav ima jedinstveno rješenje.

3. APLIKACIJA

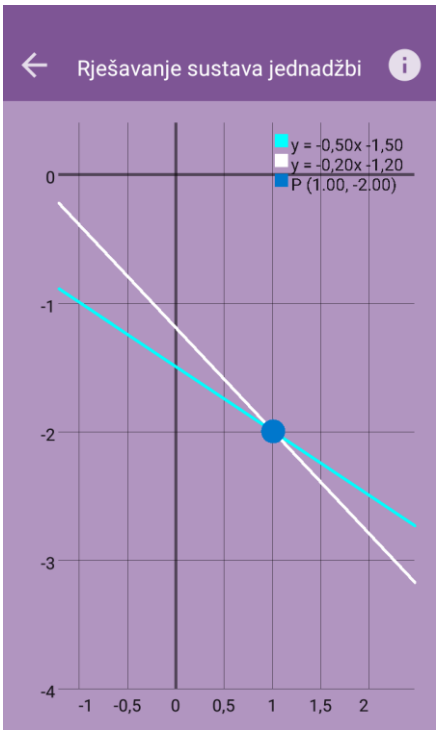
Za početak razvoja aplikacije morao se analizirati problem rješavanja sustava linearnih jednadžbi, a to je učinjeno uz pomoć dostupne literature.

Kako bi aplikacija bila privlačna korisniku, prvo se moralo analizirati kako će izgledati aplikacija, to jest gdje će se nalaziti dijelovi aplikacije za komunikaciju s korisnikom. Boje su birane uz pomoć aplikacije Adobe Color [7]. Na slici 3.1. može se vidjeti izgled početnog zaslona aplikacije, koji je ujedno i glavni zaslon. Na njemu se nalazi šest „edittext“ komponenti i tri „button“ komponente. Tu se nalaze još i dvije slike na koje se može kliknuti, jedna je za povratak na prethodni zaslon aplikacije, a druga za prikaz informacija o aplikaciji.

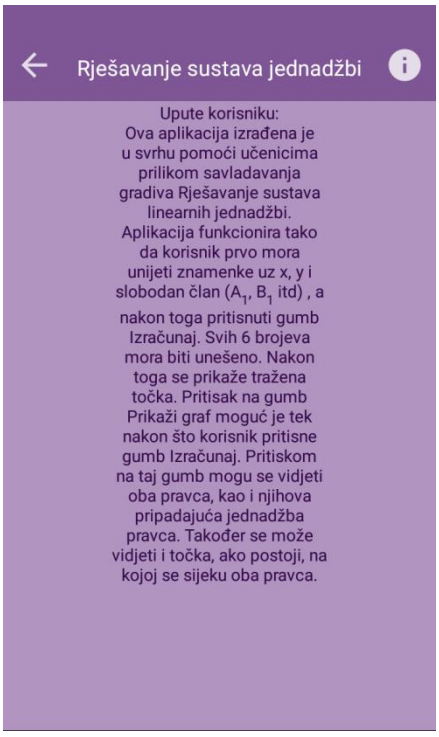


Sl. 3.1.Početni zaslon aplikacije

Na slici 3.2. može se vidjeti zaslon aplikacije nakon pritiska gumba „Prikaži graf“ na kojem se mogu vidjeti obje jednadžbe pravca koje su unesene, kao i točka u kojoj se pravci sijeku (ako se sijeku). Na slici 3.3. nalazi se opis aplikacije, to jest korisnik može dobiti informacije o načinu korištenja same aplikacije.

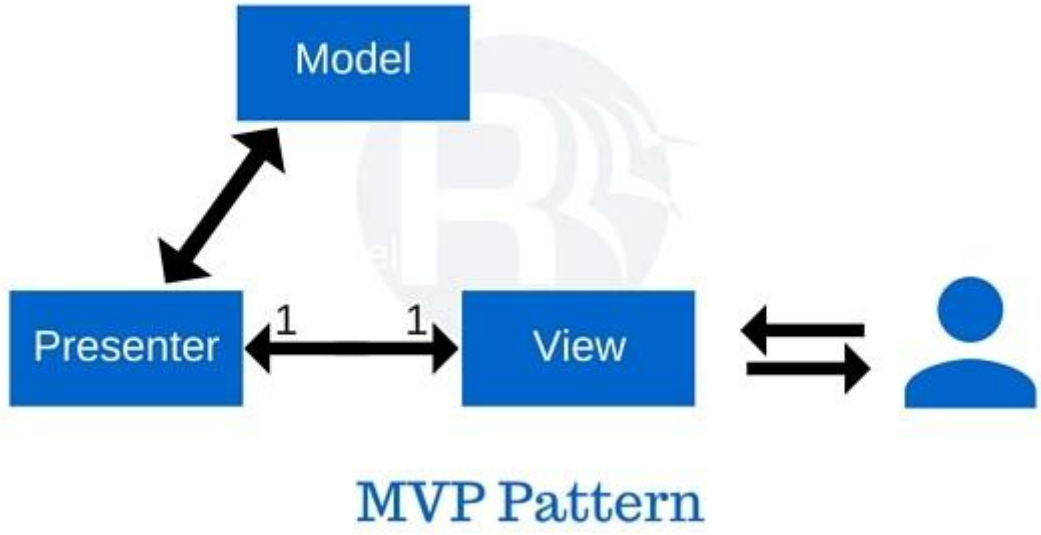


Sl. 3.2.Zaslon s prikazom grafa



Sl. 3.3. Zaslon s informacijama o aplikaciji

Za razvoj ove aplikacije korišten je *Model-View-Presenter* (MVP) *design pattern*. Na slici 3.4. može se vidjeti način na koji komuniciraju određeni dijelovi MVP *patterna*.



Sl. 3.4. MVP *design pattern*

Glavni razlog korištenja MVP-a pojednostavljivanje je testiranja aplikacije, a uz to dobivamo i puno pregledniji i jasniji kod. Korištenje MVP-a znači i korištenje tri klase za svaki pogled (eng. *activity*, *fragment*). Sam pogled (eng. *view*) treba biti što jednostavniji, u njemu bi se trebale nalaziti samo metode koje govore što je korisnik odabrao u aplikaciji, a nakon odabira u aplikaciji, pogled javlja *presenteru* što je korisnik odabrao. Za komunikaciju između pogleda i *presentera* koristi se *interface*. *Interface* predstavlja „ugovor“, a svaka klasa koja naslijedi određeni *interface* mora implementirati sve metode koje taj *interface* sadrži. Ideja je *presenter* klase da drži svu logiku aplikacije unutar klase. Također je bitno da se unutar *presenter* klase nalaze samo biblioteke vezane uz Java programski jezik te je potrebno izbjegavati biblioteke Androida. S druge strane, *presenter* komunicira s modelom, iako u ovoj aplikaciji modela nema, s obzirom da se ne radi ni s lokalnom bazom podataka, ni s podacima s interneta. Kao i u slučaju odnosa između pogleda i *presentera*, u slučaju *presentera* i modela također postoji komunikacija pomoću *interfacea*.

Glavni dio programa izvodi se pritiskom na gumb „Izračunaj“. Na slici 3.5. može se vidjeti dio koda koji javlja *presenteru* da je pritisnut gumb s oznakom „btn_calculate“. Taj dio koda nalazi

se unutar „View“ klase „EquationSolver“ fragmenta. Poziva se metoda „onCalculateClicked“ i predaje joj se svih šest brojeva pretvorenih u string, bez ikakvih provjera unosa.

```
@OnClick(R.id.btn_calculate)
public void calculateNumbers() {
    presenter.onCalculateClicked(etFirstX.getText().toString(), etSecondX.getText().toString(),
            etFirstY.getText().toString(), etSecondY.getText().toString(),
            etFirstZ.getText().toString(), etSecondZ.getText().toString());
}
```

Sl.3.5. Dio koda koji se poziva klikom na gumb „Izračunaj“

Nakon što se kliknulo na gumb, prelazi se u *presenter* klasu„EquationSolver“. Na slici 3.6. može se vidjeti koje se funkcije pozivaju unutar *presentera*. To su: „checkIfNumberInputsAreFilled“, koja provjerava jesu li svi „edittext“ elementi uneseni, „checkIfNumberInputsAreZeros“, koja provjerava jesu li svi brojevi 0 (ništa se ne može izračunati ako su svi unosi 0) te „checkIfXAndYAreZeros“, koja provjerava jesu li brojevi uz X i Y 0, ako se postavi da je neki od njih nula.

```
@Override
public void onCalculateClicked(String x1, String x2, String y1, String y2, String z1, String z2) {
    if (checkIfNumberInputsAreFilled(x1, x2, y1, y2, z1, z2) && !checkIfNumberInputsAreZeros(x1, x2, y1, y2, z1, z2)
            && !checkIfXAndYAreZeros(x1, x2, y1, y2)) {
```

Sl. 3.6. Provjera unesenih brojeva

Na slici 3.7.može se vidjeti dio koda koji provjerava jesu li X i Y elementi jednaki 0. Na početku se postavlja zastavica u false, što bi značilo da ni jedan element nije 0. Tada se provjerava pomoću ILI logičkog sklopa je li koji od unesenih X i Y elemenata jednak 0. Ako je barem jedan element jednak 0, tada zastavica xAndYAreZeros mijenja svoju vrijednost u true.

```
private boolean checkIfXAndYAreZeros(String x1, String x2, String y1, String y2) {
    boolean xAndYAreZeros = false;

    if ((Double.parseDouble(x1) == 0) || (Double.parseDouble(x2) == 0) || (Double.parseDouble(y1) == 0)
            || (Double.parseDouble(y2) == 0)) {
        view.showXAndYAreZerosToast();
        xAndYAreZeros = true;
    }

    return xAndYAreZeros;
}
```

Sl. 3.7. Provjera jesu li X i Y jednaki 0

Ako kod kroz sva tri uvjeta prođe uspješno, tada se izvršava kod na slici 3.8. Napravljena je klasa „MathUtils“ koja je pomoć u rješavanju determinanti. Na slici 3.9. može se vidjeti način računanja determinante. Postoji nekoliko uvjeta koji se provjeravaju, budući da je odabrana Cramerova metoda rješavanja sustava. Nakon što *presenter* izračuna sve potrebne elemente, javlja pogledu da je sve izračunato te ispisuje korisniku poruku na ekran, odnosno rješenje njegovog sustava.

```
if (MathUtils.calculateDeterminant(this.x1, this.x2, this.y1, this.y2) == 0) {
    if (MathUtils.calculateDeterminantX(this.y1, this.y2, this.z1, this.z2) == 0) {
        view.showInfinitySolutionsToast();
        view.showInfinitySolutionsText(MathUtils.calculateEquationWithX(this.x1, this.y1, this.z1));
    } else {
        view.showNoSolutionsToast();
        view.showNoSolutionText();
    }
} else {
    pointX = MathUtils.calculateDeterminantX(this.y1, this.y2, this.z1, this.z2) /
            MathUtils.calculateDeterminant(this.x1, this.x2, this.y1, this.y2);
    pointY = MathUtils.calculateDeterminantY(this.x1, this.x2, this.z1, this.z2) /
            MathUtils.calculateDeterminant(this.x1, this.x2, this.y1, this.y2);

    isOnlyOnePointSelected = true;

    view.showXAndY(pointX, pointY);
}
```

Sl. 3.8. Kod za računanje rješenja sustava

```
public static Double calculateDeterminant(double x1, double x2, double y1, double y2) {
    return (x1 * y2) - (x2 * y1);
}

public static Double calculateDeterminantX(double y1, double y2, double z1, double z2) {
    return (-z1 * y2) - (-z2 * y1);
}

public static Double calculateDeterminantY(double x1, double x2, double z1, double z2) {
    return (x1 * -z2) - (x2 * -z1);
}
```

Sl.3.9. Kod za računanje determinanti

Ako postoji samo jedna točka, poziva se metoda „showXAndY“, ako postoji beskonačno mnogo točaka, poziva se metoda „showInfinitySolutionsText“, a ako ne postoji ni jedna točka, onda se poziva metoda „showNoSolutionsText“.

Osim tog dijela koda odlučeno je da će se prilikom svakog unesenog broja provjeravati jesu li svi elementi uneseni i jesu li dobro uneseni. Taj dio koda izvršava se uz pomoć Android biblioteke TextWatcher [8]. Ona radi na način da prilikom svake promjene unutar bilo koje „edittext“ komponente provjerava sve unesene element unutar svih „edittext“ komponenti. Taj dio koda može se vidjeti na slici 3.10. Nakon što se promijeni tekst unutar jedne komponente, poziva se metoda „onTextChanged“ u *presenteru*.

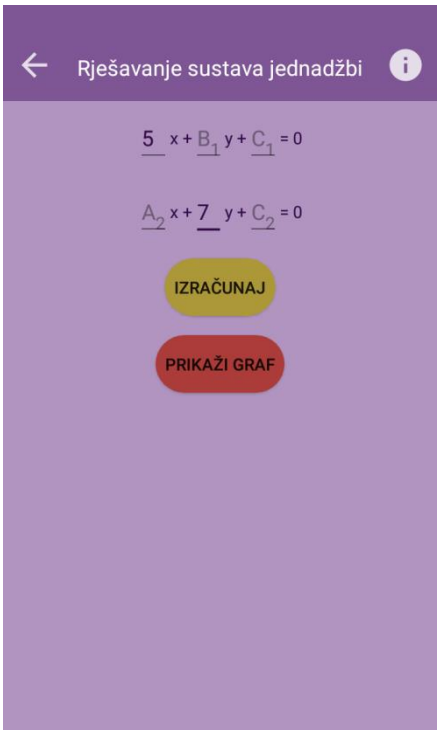
```
@Override
public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
    if (charSequence.hashCode() == etFirstX.getText().hashCode()) {
        presenter.onTextChanged(etFirstX.getText().toString(), etSecondX.getText().toString(),
                etFirstY.getText().toString(), etSecondY.getText().toString(),
                etFirstZ.getText().toString(), etSecondZ.getText().toString());
```

Sl. 3.10. Kod koji se izvršava prilikom promjene teksta

Na slici 3.11. može se vidjeti koje se metode pozivaju prilikom promjene teksta. Ako nijedan element nije prazan i ako X i Y elementi nisu jednaki 0, tada se javlja pogledu da promijeni boju gumba „Izračunaj“ u zeleno. Ako su svi elementi prazni, tada se javlja pogledu da promijeni boju gumba u crveno. Naposljetku, ako niti jedan od ta dva uvjeta nije ispunjen, tada se javlja pogledu da promijeni boju gumba u žuto. Na slici 3.12. može se vidjeti kako funkcionira aplikacija kada nisu uneseni svi brojevi. To je napravljeno kako bi korisniku bilo jednostavnije shvatiti kada uneseni brojevi zadovoljavaju uvjete, a kada ne.

```
@Override
public void onTextChanged(String x1, String x2, String y1, String y2, String z1, String z2) {
    if (!x1.isEmpty() && !x2.isEmpty() && !y1.isEmpty() && !y2.isEmpty() && !z1.isEmpty() && !z2.isEmpty()
            && !checkIfXAndYAreZeros(x1, x2, y1, y2)) {
        view.showAcceptedCalculateColor();
    } else if (x1.isEmpty() && x2.isEmpty() && y1.isEmpty() && y2.isEmpty() && z1.isEmpty() && z2.isEmpty()) {
        view.showErrorCalculateColor();
    } else {
        view.showOnHoldCalculateColor();
    }
}
```

Sl. 3.11. Provjere nakon svakog unesenog znaka



Sl. 3.12. Izgled aplikacije u međukoraku

Ako su ispunjeni svi uvjeti, gumb „Prikaži graf“ prebacuje se u zelenu boju koja indicira da je moguće kliknuti na taj gumb. Pritiskom na gumb mijenja se fragment „EquationSolver“ fragmentom „GraphDrawer“. Osim mijenjanja fragmenta, također se u novi fragment šalju i određeni argumenti, to su šest unesenih brojeva, kao i boolean vrijednost koja govori je li odabrana samo jedna točka koja siječe dva pravca.

```
public static GraphDrawerView newInstance(double x1, double x2, double y1, double y2, double z1,
                                          double z2, boolean isOnlyOnePointSelected) {
    Bundle args = new Bundle();

    args.putDouble("x1", x1);
    args.putDouble("x2", x2);
    args.putDouble("y1", y1);
    args.putDouble("y2", y2);
    args.putDouble("z1", z1);
    args.putDouble("z2", z2);
    args.putBoolean("IS_ONLY_ONE_POINT_SELECTED", isOnlyOnePointSelected);

    GraphDrawerView fragment = new GraphDrawerView();
    fragment.setArguments(args);

    return fragment;
}
```

Sl. 3.13. Instanca „GraphDrawer“ fragmenta s argumentima

Nakon što su brojevi proslijeđeni, pogled javlja *presenteru* da su brojevi primljeni. Nakon toga *presenter* izračunava 200 točaka, od -100 do 100. Funkciji se predaje x i z komponente jednadžbe, a nakon toga se računa točka y uz pomoć vrijednosti uz varijable x i z. Nakon što se izračuna y za svaku od jednadžbi, *presenter* javlja pogledu vrijednosti x i y kako bi se mogao nacrtat graf. Na slici 3.14. može se vidjeti dio koda gdje se unutar for petlje računaju obje vrijednosti te ih se šalje pogledu. U pogledu se iscrtava taj graf, a dio koda koji iscrtava graf može se vidjeti na slici 3.15. Funkciji predajemo vrijednost varijable i, vrijednost varijable y, a predaje se također i najveći broj točaka koji se može nacrtati, a u ovom je slučaju to 200.

```
for (int i = -100; i < 101; i++) {
    firstEquationY = x1 * i + z1;
    secondEquationY = x2 * i + z2;

    view.setFirstEquationData(i, firstEquationY);
    view.setSecondEquationData(i, secondEquationY);
}
```

Sl. 3.14. Kod koji izračunava vrijednost *y*

```
@Override
public void setFirstEquationData(int i, double firstEquationY) {
    firstFunctionGraph.appendData(new DataPoint(i, firstEquationY), scrollToEnd: true, maxDataPoints: 200);
}
```

Sl. 3.15. Kod koji crta graf na zaslon

Kao što se može vidjeti iz navedenih isječaka koda, za funkcioniranje cjelokupne aplikacije potrebno je dobro poznavanje Jave i logike koja se nalazi u pozadini izračunavanja točaka iz sustava linearnih jednadžbi. Također se moraju znati i osnovne stvari iz Androida, kako bi se mogla povezati funkcionalnost Androida s programskim jezikom Jave, pomoću kojega se može napisati logika za rješavanje sustava.

4. ZAKLJUČAK

Cilj je ovog diplomskog rada bio napraviti Android aplikaciju koja će pomoći učenicima prilikom savladavanja gradiva „Sustav linearnih jednadžbi“ te je isti uspješno ostvaren. Glavna ideja rada bila je pomoći učenicima koji se pripremaju za ispite iz navedenog gradiva te im omogućiti da uz pomoć ove aplikacije mogu provjeriti svoje rezultate. Glavna prepreka bila je odabrati metodu pomoću koje će se izračunavati rješenje sustava. Nakon što je odlučen način na koji će aplikacija raditi, objašnjene su teorijske podloge. Što se programskog dijela, tiče prvo je opisan Android kao operacijski sustav, potom Android Studio koji se koristi za izradu aplikacija za navedeni sustav i na posljetku Java, programski jezik u kojemu je aplikacija napisana. Uz to, opisana je i matematička teorija vezana za aplikaciju, odnosno sustav linearnih jednadžbi te metode rješavanja. Jedna od tih metoda, Cramerovo pravilo, koristi se kao podloga za računanje u aplikaciji.

Sama aplikacija izrazito je jednostavna za korištenje, jer je uz pomoć *hintova* korisniku apsolutno pojednostavljeno korištenje iste (zelena boja „buttona“ kada se nešto može izračunati, žuta boja kada korisnik nije sve potrebno unio u „edittext“). Aplikacija je dostupna svakoj osobi koja posjeduje .apk datoteku i nije izrađena u komercijalne svrhe. Testirana je na više uređaja s Android operacijskim sustavom, na svakom od njih adekvatno radi i daje iste, točne rezultate. Naravno, aplikacija uvijek može biti kvalitetnija te bi se tako mogao poboljšati izgled sučelja i dodati računanje pomoću ostalih metoda.

LITERATURA

[1] Wikipedia, Android (operacijski sustav), https://hr.wikipedia.org/wiki/Android_(operacijski_sustav) (stranica posjećena: 20. lipnja 2018.)

[2] Wikipedia, Android VersionHistory, https://en.wikipedia.org/wiki/Android_version_history (stranica posjećena: 20. lipnja 2018.)

[3] MVP Android, https://antonioleiva.com/mvp-android/ (stranica posjećena: 21. lipnja 2018.)

[4] B. Bates, K. Sierra, Head First Java, O'Reilly Media, SAD, 2005.

[5] B. Dakić, N. Elezović, S. Banić, Matematika 1, Element, Zagreb, 2013.

[6] Kronecker-Capellijev teorem, http://lavica.fesb.hr/mat1/predavanja/node38.html (stranica posjećena: 25. lipnja 2018.)

[7] Adobe Color, https://color.adobe.com/create/color-wheel/

[8] StackOverflow, Textwatcher, https://stackoverflow.com/questions/4283062/textwatcher-for-more-than-one-edittext/4283532 (stranica posjećena: 22. lipnja 2018.)

SAŽETAK

Osnovni je cilj ovog rada bio napraviti funkcionalnu aplikaciju koja će pomoći učenicima prilikom rješavanja sustava linearnih jednadžbi. Aplikacija omogućava korisniku unos vrijednosti uz nepoznanice x, y i z. Nakon toga korisnik ima mogućnost izračunavanja rješenja sustava, a nudi mu se i mogućnost pogleda na grafove jednadžbi pravca koje su izračunate od vrijednosti koje je korisnio unio. Također se prikazuje i točka u kojoj se grafovi sijeku, ako se sijeku. Korisnik također ima mogućnost obrisati upisane vrijednosti uz nepoznanice pritiskom na gumb „Reset“. Aplikacija je razvijena u Android studiu, a namijenjena je isključivo mobitelima i tabletima s Android operacijskim sustavom. Razvijena je za Android iz razloga što većina ciljanih korisnika u Hrvatskoj, učenici i studenti, posjeduju mobitel s Android operacijskim sustavom. U teoretskom dijelu rada opisan je razvoj Android Studia, temeljna obilježja i odlike Jave kao programskog jezika, a također je obrađena i tema iz algebre naziva „Rješavanje sustava linearnih jednadžbi“.

Ključne riječi: Android Studio, Java, sustav linearnih jednadžbi

ABSTRACT

The main goal of this assignment was to make a functional application that will help students with solving linear equations. Application enables user entering the values of unknowns x, y and z. After that, user has the possibility of solving the equation, while he can also see the graphs of the linear equations, which are calculated from the values user entered. It also shows the point where two linear equation graphs intersect, if they intersect. Moreover, user has the possibility to delete the values of the unknowns with the click on the button „Reset“. The application was developed in Android Studio, and it is intended for cell phones and tablets with Android operating system. It was developed for Android because most of targeted users in Croatia are using Android phones. In theoretical part the development of Android Studio was described, alongside with its basic characteristics and qualities of Java as a programming language. Finally, it has processed the „Linear equation solving“ topic as well.

Keywords: Android Studio, Java, linear equation solving

ŽIVOTOPIS

Tomislav Šarčević rođen je 20. svibnja 1994. godine u Đakovu, Hrvatska. 2001. godine počinje s osnovnoškolskim obrazovanjem u OŠ Vladimir Nazor u Đakovu. Nakon završetka osnovnoškolskog obrazovanja, 2009. se upisuje u Gimnaziju Antuna Gustava Matoša u Đakovu, smjer Opća gimnazija. 2013. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija, smjer Računarstvo, koji još uvijek pohađa. Dobitnik je stipendije za deficitarna zanimanja Nacionalne zaklade za potporu učeničkom i studentskom standardu za akademsku godinu 2015./2016, 2016./2017., 2017./2018. Također posjeduje određeno znanje u govoru, čitanju i pisanju engleskog jezika, vozačku dozvolu B kategorije. Posjeduje i određeno znanje u radu s Microsoft Office alatima, osnovno znanje opisnog jezika HTML, osnovno znanje programskih jezika C, C++ i C#, znanje o razvijanju Android aplikacija u programskom jeziku Java, kao i osnovno znanje PHP programskog jezika.

Tomislav Šarčević

PRILOZI

DVD

- Android Studio projekt
- Rad u .docx i .pdf formatu

- rad na Githubu https://github.com/tsarcevic/LinearEquationSolver