TEHNIČKO VELEUČILIŠTE U ZAGREBU

SPECIJALISTIČKI STRUČNI STUDIJ INFORMATIKE

Tomislav Brabec

Android aplikacija za vježbanje matematike MathPath

Zagreb, srpanj 2018.

Sažetak

U ovom završnom radu bit će detaljno objašnjena izrada Android aplikacije koja će služiti za vježbanje osnovnih matematičkih zadataka namijenjenih osnovnoškolcima te karakteristike programskog jezika Kotlin u kojem je aplikacija pisana. Također bit će objašnjene sve tehničke karakteristike i tehnologije koje su korištene u razvoju aplikacije.

Sadržaj

Sadržaj....4
1. Uvod....9
2. Kotlin....10
2.1. Uvod u Kotlin....10
2.2. Povijest Kotlina....11
2.3. Sintaksa....12
2.4. Usporedba Kotlina i Jave....13
2.4.1. Sigurnost s null pokazivačima....13
2.4.2. Getter i setter metode....14
2.4.3. Polja....14
2.4.4. Funkcije višeg reda....15
2.4.5. Provjerene iznimke....15
2.4.6. Funkcionalnosti koje Java ima, a Kotlin ne....16
2.4.7. Funkcionalnosti koje Kotlin ima, a Java ne....16
3. MathPath....17
3.1. O aplikaciji....17
3.2. Dijagram slučajeva korištenja....18
4. Aktivnosti unutar aplikacije....19
4.1. Main....19
4.1.1. Aktivnost....19
4.1.2. Pogled....23
4.2. PopupDifficulty....24
4.2.1. Aktivnost....24
4.2.2. Pogled....25
4.3. CustomGame....25
4.3.1. Aktivnost....26
4.3.2. Pogled....28

4.4. Solving 29
4.4.1. Aktivnost 29
4.4.2. Pogled 34
4.5. Graph 36
4.5.1. Aktivnost 36
4.5.2. Pogled 38
4.6. BgMusic 38
4.6.1. Aktivnost 38
4.6. AnswerList 40
5. Modeli unutar aplikacije 41
5.1. Equation 41
5.2. EquationConfig 44
5.3. PetItem 44
6. Prilagođeni elementi 45
6.1. GoldView 45
6.2. KeyboardView 46
6.3. PetItemView 46
7. Ostalo 48
7.1. BackgroundSoundService 48
7.2. HomeWatcher 49
7.3. DbHelper 50
8. Korištene biblioteke 53
8.1. Anko SQLite 53
8.2. MVEL 55
9. Korišteni alati 56
10. Zaključak 57
11. Literatura 58

Popis slika

Slika 1. Kotlin logo 10
Slika 2. Dijagram slučajeva korištenja 18
Slika 3. Izgled početne aktivnosti 19
Slika 4. Skočni prozor za izbor težine zadataka 24
Slika 5. Izgled aktivnosti za ručno podešavanje zadataka 25
Slika 6. Izgled aktivnosti za rješavanje zadataka 29
Slika 7. Izgled aktivnosti za statistiku 36
Slika 8. Izgled AnswerList aktivnosti 40

Popis koda

Kod 1.Primjer kotlin koda 12
Kod 2. Primjer pozivanja servisa za pozadinsku melodiju 12
Kod 3. Primjer String varijable koja ne može poprimiti null vrijednost 13
Kod 4. Primjer String varijable koja može poprimiti null vrijednost 13
Kod 5. Dohvaćanje duljine stringova kod varijable a koja nije null i varijable b koja je null 13
Kod 6. Dohvaćanje i postavljanje vrijednosti u Kotlinu putem get i set metoda 14
Kod 7. Izvedba polja u Kotlinu 14
Kod 8. Primjer funkcije višeg reda 15
Kod 9. Primjer koda u Javi s provjerenom iznimkom 15
Kod 10. Primjer pozivanja servisa za pozadinsku melodiju 19
Kod 11. Primjer pozivanja baze 20
Kod 12. Dohvaćanje količine novca 20
Kod 13. Funkcija onResume koja se poziva prilikom svakog povratka na glavnu aktivnost 20
Kod 14. Interakcija sa stvarima i ljubimcem 21
Kod 15. Pokretanje aktivnosti statistika i uputa 21
Kod 16. Promijena boje trake i raspoloženje ljubimca ovisno o količini sreće 22
Kod 17. Pokretanje aktivnosti statistika i uputa 23
Kod 18. Primjer GoldView elementa unutar XML dokumenta 23
Kod 19. GeneriranjeEquationConfig objekta pritiskom na gumb "Lagano" 24
Kod 20. Poziv aktivnosti za ručno podešavanje zadataka 25
Kod 21. Primjer gumba za odabir lagane težine unutar XML pogleda 25
Kod 22. Primjer postavljanja elementa za raspon brojeva 26

Kod 23. Primjer provjere ispravnosti kod unosa broja koraka 27
Kod 24. Kreiranje EquationConfig objekta od unesenih postavki 27
Kod 25. Ručno postavljanje fontova 28
Kod 26. Primjer odjeljka i djela kategorije za postavljanje raspona brojeva 28
Kod 27. Funkcije koje se pozivaju unutar onCreate metode 30
Kod 28. Funkcija za postavljanje kreirane jednadžbe na TextView elemente 30
Kod 29. Funkcija za postavljanje savjeta 31
Kod 30. Postavljanje tipa igre s koracima 31
Kod 31. Funkcija koja se poziva na svakom koraku igre 32
Kod 32. Postavljanje tipa igre na vrijeme 32
Kod 33. Funkcija koja se poziva na svaki klik za sljedeći zadatak u tipu igre na vrijeme 32
Kod 34. Funkcija za odbrojavanje vremena 33
Kod 35. Prikazivanje elementa koji sadržava informaciju o zarađenom novcu ili krivom rješenju 33
Kod 36. Slučaj prikaza količine zarađenog novca kada je odgovor ispravan 33
Kod 37. Dodavanje odgovora u listu 34
Kod 38. Dodavanje broja točno ili netočno riješenih zadataka po operatoru u ejdnadžbi 34
Kod 39. TextView element za prikaz preostalog vremena 34
Kod 40. Tri elementa za prikaz jednadžbe i unos rješenja 35
Kod 41. Dio koda koji na PieChart postavlja omjer točnih i netočnih odgovora 37
Kod 42. Dohvaćanje i postavljanje podatka o rješivosti po operatoru zbrajanja 37
Kod 43. Promjena boje ProgressBar elementa ovisno o postotku 37
Kod 44. Primjer PieChart XML elementa 38
Kod 45. Pokretanje i zaustavljanje pozadinske muzike na određene okidače 39
Kod 46. Provjera jeli aplikacija u pozadini 39
Kod 47. Provjera rada servisa za glazbu 39
Kod 48. Konstruktor s članovima klase 41
Kod 49. Dobivanje nasumičnog broja operanada za trenutnu jednadžbu 41
Kod 50. Kreiranje indeksa za zagrade 41
Kod 51. Petlja unutar koje se dodaju operatori i operandi u jednadžbu 42
Kod 52. Dodavanje operatora prema postotku rješivosti 42
Kod 53. Odbacivanje zadnjeg znaka i evaluacija jednadžbe 43
Kod 54. Provjera ispravnosti jednadžbe 43
Kod 55. EquationConfig klasa 44
Kod 56. PetItem klasa 44

Kod 57. Funckija init unutar koje se prilagođuje veličina slike novčića s veličinom teksta .45
Kod 58. Uklanjanje slike s GoldView elementa 45
Kod 59. Implementacija klikova na tipke tipkovnice 46
Kod 60. Postavljanje PetItem objekta 47
Kod 61. Dio koda koji se izvršava pri kliku na stvar 47
Kod 62. Ispisivanje poruke ukoliko korisnik nema dovoljno novca 48
Kod 63. Kod koji se poziva prilikom kreiranja servisa za glazbu 48
Kod 64. Dio koda koji prati jeli korisnik pritisnuo home button 49
Kod 65. Sučelje čije metode korisnik može implementirati na pritiske home button-a 49
Kod 66. Nazivi svih tablica 50
Kod 67. Kreiranje tablice rezultata i postavljanje predefiniranekoličine novca 50
Kod 68. Kreiranje tablice za spremanje količine novca 50
Kod 69. Funkcija za dodavanje novca u bazu.....51
Kod 70. Primjer dodavanja stvari za ljubimca u bazu 51
Kod 71. Dohvaćanje prosjeka rješivosti po operatoru 52
Kod 72. Parser za pretvaranje prosjeka po operatoru u listu 52
Kod 73. Pozivanje instance SQLite baze 53
Kod 74. Asinkroni poziv instance baze 53
Kod 75. Kreiranje tablice.....54
Kod 76. Ubacivanje retka u tablicu 54
Kod 77. Primjer dohvaćanja korisnika iz baze 55
Kod 78. Primjer evaluacije izraza.....55

Popis tablica

Tablica 1. Popis Anko metoda za dohvaćanje podataka.....54

Popis kratica

MVEL - MVFLEX Expression Language

1. Uvod

U današnje vrijeme kada postoji već hrpa aplikacija gotovo za svaku priliku, teško je osmisliti nešto novo što će biti društveno korisno. Iako već sigurno postoji nekoliko sličnih aplikacija, cilj ove je da ostane jednostavna s korisničke i razvojne strane, a da istovremeno bude zabavna i edukativna. Ovaj seminar će se sastojati od dvije cjeline u kojima će biti opisan rad i razvoj aplikacije.

U prvom dijelu ovog seminara bit će opisan programski jezik Kotlin u kojem je aplikacija razvijena. Također, bit će objašnjena namjena lokalne baze podataka te nekoliko biblioteka koje su olakšale razvoj aplikacije.

U drugom dijelu bit će objašnjeni kod, dizajn i algoritmi pomoću kojih su razvijeni neki dijelovi aplikacije. Također, u ovom dijelu će biti opisani načini poboljšanja korisničkog iskustva prilikom korištenja aplikacije.

2. Kotlin

2.1. Uvod u Kotlin

Kotlin je statički programski jezik koji se izvršava na „Java virtual machine“ ili skraćeno JVM. Također, može biti kompajliran u JavaScript izvorni kod ili može koristiti LLVM infrastrukturu. Kotlin je razvila tvrtka JetBrains s timom razvijatelja u Petrogradu u Rusiji. Iako sintaktički nije kompatibilan s Javom, Kotlin je dizajniran tako da može koristiti i izvršavati postojeći Java kod te tako postoji mogućnost korištenja postojećih Java biblioteka. Kotlin koristi „agresivno“ sučelje tipova te tako lako određuje koje vrijednosti i izrazi su ostali još neinicijalizirani. Tako su izbjegnute mnoge greške koje se npr. u Javi često događaju, a to je prije svega NullPointerException. Nova verzija programskog okruženja Android Studio u potpunosti podržava programski jezik Kotlin te je također u potpunosti podržan od strane Google-a na Android operacijskim sustavima.

Slika 1. Kotlin logo

2.2. Povijest Kotlina

U srpnju 2011. godine, JetBrains je predstavio Projekt Kotlin, novi jezik za JVM, koji se do tada razvijao godinu dana. Voditelj razvoja Dmitry Jemerov je izjavio da većina jezika nema značajke koje su njima bile potrebne uz izuzetak programskog jezika Scala koja je nažalost imala problem sa sporošću kompajliranja. Jedan od primarnih ciljeva Kotlina je bio da se kompajlira brzo kao Java. 2012. godine, JetBrains je izdao otvoreni kod Kotlina pod Apache 2 licencom. Kotlin je naziv dobio po jednom otoku u blizini Petrograda. JetBrains se nada da će jezik poboljšati prodaju njihovih alata i olakšati posao mnogim tvrtkama i programerima. Prva stabilna verzija Kotlina bila je puštena 15. veljače 2016. godine, a 2017. godine Google je objavio podršku Kotlina za razvoj Android aplikacija. 28. studenog 2017. puštena je verzija Kotlina 1.2 u kojoj je omogućeno dijeljenje koda između JVM-a i omogućeno je pretvaranje Kotlin koda u Javascript.

Jedan od voditelja razvoja, Andrey Breslav je izjavio kako je Kotlin dizajniran da bude industrijski jak i objektno orijentiran programski jezik koji će biti bolji od Jave, ali da će i dalje biti u potpunosti interoperabilan s Javom tako da tvrtke mogu postepeno migrirati svoje aplikacije s Jave na Kotlin.

2.3. Sintaksa

Sintaksa Kotlina omogućuje bolju čitljivost koda od Jave i daje punu podršku funkcionalnom načinu programiranja. Kao i kod većine programskih jezika, ulazna točka Kotlin aplikacije je main funkcija kojoj se može proslijediti lista argumenata iz komandne linije. Također, podržana je interpolacija stringova kao u većini programskih jezika ili Perli i Unix skriptama. Kotlin za razliku od jave podržava i automatsku dodjelu tipa podataka (type inference).

```
// Hello, world! example
fun main(args: Array<String>) {

   val scope = "world"

   println("Hello, $scope!")

}
```

Kod 1.Primjer kotlin koda

Slično kao i u C# programskom jeziku, Kotlin omogućava proširivanje klasa s metodama bez dodatnog deriviranja istih. Proširivim metodama Kotlin dodaje novu funkcionalnost postojećim klasama tako da ih „zalijepi“ listu javnih metoda bez da je ona službeno unutar klase.

```
package MyStringExtensions


fun String.lastChar():  Char = this.get(this.length - 1)


>>> println("Kotlin".lastChar())

n
```

Kod 2. Primjer pozivanja servisa za pozadinsku melodiju

U ovom primjeru klasi *String* dodijeljena je proširiva funkcija *lastChar* koja vraća zadnje slovo zadanog stringa.

2.4. Usporedba Kotlina i Jave

2.4.1. Sigurnost s null pokazivačima

Sustav Kotlina je dizajniran tako da eliminira opasnost pojave grešaka s null pokazivačima. Jedna od najčešćih grešaka koja se javlja u mnogim programskim jezicima, kao i u Javi, je pokušaj referenciranja na član nekog nepostojećeg tj. neinicijaliziranog objekta što će kao rezultat dati NullPointerException.

U Kotlinu postoji samo nekoliko načina da se ta greška pojavi, a to su:

- Eksplicitni poziv na throw NullPointerException()
- Korištenje !! operatora kojim programer označava da objekt nema vrijednost null što ne mora biti točno
- Slučaj kada postoji nekonzistencija kod inicijalizacije
- Pozivanje Java koda koji može izazvati ovu grešku

```
var a: String = "abc"
a = null // compilation error
```

Kod 3. Primjer String varijable koja ne može poprimiti null vrijednost

Da bi omogućili dodjeljivanje null vrijednosti, tip varijable to mora omogućiti dodavanjem znaka ? na kraj tipa.

```
var b: String? = "abc"
b = null // ok
```

Kod 4. Primjer String varijable koja može poprimiti null vrijednost

Pri pokušaju dohvaćanja duljine stringa varijable a bili bi sigurni da ona nema null vrijednost pa ne bi dobili nikakvu grešku što nam nije slučaj kod varijable b.

```
val l = a.length // ok
val l = b.length // error: variable 'b' can be null
```

Kod 5. Dohvaćanje duljine stringova kod varijable a koja nije null i varijable b koja je null

2.4.2. Getter i setter metode

Getter is setter metode su unutar Kotlina predstavljeni kao članovi. Pristupne metode za Boolean varijable imaju prefiks is. Također, ako Java klasa ima samo setter metodu, tada ona neće biti prikazana kao član jer su za tu funkcionalnost unutar Kotlina potrebna getter i setter metoda.

```
import java.util.Calendar

fun calendarDemo() {
    val calendar = Calendar.getInstance()
    if (calendar.firstDayOfWeek == Calendar.SUNDAY) {  // call getFirstDayOfWeek()
        calendar.firstDayOfWeek = Calendar.MONDAY      // call setFirstDayOfWeek()
    }
    if (!calendar.isLenient) {                  // call isLenient()
        calendar.isLenient = true                 // call setLenient()
    }
}
```

Kod 6. Dohvaćanje i postavljanje vrijednosti u Kotlinu putem get i set metoda

2.4.3. Polja

Polja u kotlinu su invarijantna. Predstavljena su s Array klasom koja ima get i set funkcije koje preopterećenjem operatora pretvaraju listu u [].

```
class Array<T> private constructor() {
    val size: Int
    operator fun get(index: Int): T
    operator fun set(index: Int, value: T): Unit

    operator fun iterator(): Iterator<T>
    // ...
}
```

Kod 7. Izvedba polja u Kotlinu

2.4.4. Funkcije višeg reda

Kotlin za razliku od Jave podržava funkcije višeg reda. Iako i Java podržava lambde, one su u Javi implementirane kao apstraktne metode i ovise o sučeljima kao što su Callable i runnable dok su u Kotlinu implementirane kao first-class funkcije tj. mogu biti dodjeljene varijablama itd.

Funkcija višeg reda kao parametar prima funkciju ili vraća funkciju. Dobar je primjer idiom funkcionalnog programiranja za zbirke, koji uzima početnu vrijednost akumulatora i funkciju kombiniranja i gradi povratnu vrijednost zbrajanjem trenutne vrijednosti akumulatora sa svakim zbirnim elementom, zamjenjujući akumulator:

```
fun <T, R> Collection<T>.fold(
        initial: R,
        combine: (acc: R, nextElement: T) -> R
): R {
    var accumulator: R = initial
    for (element: T in this) {
        accumulator = combine(accumulator, element)
    }
    return accumulator
}
```

Kod 8. Primjer funkcije višeg reda

2.4.5. Provjerene iznimke

Za razliku od Jave, u Kotlinu ne postoje provjerene iznimke što znači da će se program kompajlirati iako npr. sadrži klasu koja može baciti iznimku. Ova funkcionalnost znatno povećava čitljivost koda jer try-catch blokovi više nisu obavezni, ali nam daje i znatnu fleksibilnost.

```
Appendable append(CharSequence csq) throws IOException;
```

Kod 9. Primjer koda u Javi s provjerenom iznimkom

2.4.6. Funkcionalnosti koje Java ima, a Kotlin ne

- Provjerene iznimke
- Primitivne tipove - Svi primitivni tipovi iz Jave slično se ponašaju i u Kotlinu osim što su definirani kao klase pa imaju razne dodatne funkcionalnosti.
- Statičke tipove - Kotlin nema statičke tipove, ali ako želimo identičnu funkcionalnost unutar klasa, možemo deklarirati companion objekte čiji članovi će se moći pozivati izvana kao da su statički.
- Privatne članove - U Kotlinu svaki član ima predefiniranu get i set funkciju koje mogu biti Privatne.
- Ternarne operatore - U Kotlinu se if može koristiti implicitno unutar deklariranja varijable i sl. pa ternarni operator nije potreban.

2.4.7. Funkcionalnosti koje Kotlin ima, a Java ne

- Lambda izrazi + Inline funkcije
- Proširive funkcije
- Null-sigurnost
- Pametno pretvaranje
- Predlošci za stringove
- Primarni konstruktori
- Zaključivanje tipova iz vrijednosti
- Singleton klase
- Projekcije tipova
- Izrazi raspona
- Preopterećenje operatora
- Companion objekti
- Podatkovne klase
- Različita sučelja za read-only klase and promjenjive kolekcije
- Korutine

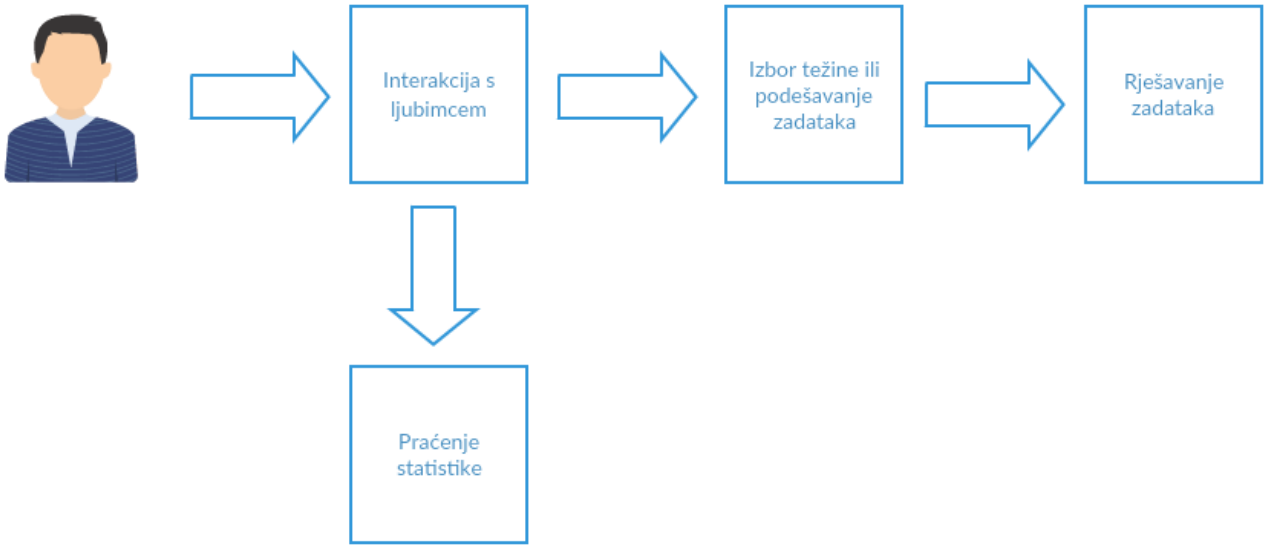
3. MathPath

3.1. O aplikaciji

U početku, aplikacija je bila zamišljena kao generator jednostavnih zadataka za osnovnoškolce. Da bi aplikacija bila zanimljivija i preinačena u igru, dodane su težine zadataka, tipovi igri(koraci i vrijeme) i kućni ljubimac o kojem korisnik mora voditi brigu. Ljubimcu se može kupovati hrana, piće i igračke s novcem koji se zarađuje rješavajući zadatke. Postoje tri predefinirane razine: lagano, srednje i teško, a moguće je i ručno podesiti generator zadataka gdje korisnik ima mogućnost izbora tipa igre, raspona brojeva, broja operanada po jednadžbi, pojavnost negativnih rezultata, pojavnost zagradi te unos na nasumičnom mjestu u jednadžbi. Prilikom rješavanja zadataka korisniku često iskaču savjeti u odnosu na zadatak, npr. ako zadatak sadržava zagrade, postoji mogućnost da se pojavi oblačić u kojem piše da se sve operacije unutar zagrada izvršavaju prije ostatka jednadžbe. Prilikom svakog točno riješenog zadatka na vrhu iskoči poruka s brojem osvojenih novčića za riješen zadatak, a ako je odgovor netočan, korisnik je također obaviješten o tome. Na vrhu aktivnosti se nalaze još informacije o timu igre, trenutno stanje novca, broj preostalih koraka ili preostalo vrijeme. Ispod jednadžbe nalazi se traka koja označava zadatak na kojem se nalazimo tj. koliko nam je vremena ostalo ako se radi o tipu igre na vrijeme.

Korisnik također ima mogućnost uvida u svoje statistike koje sadržavaju ukupan omjer točnih i netočnih rješenja te postotak rješivosti po operatorima.

3.2. Dijagram slučajeva korištenja

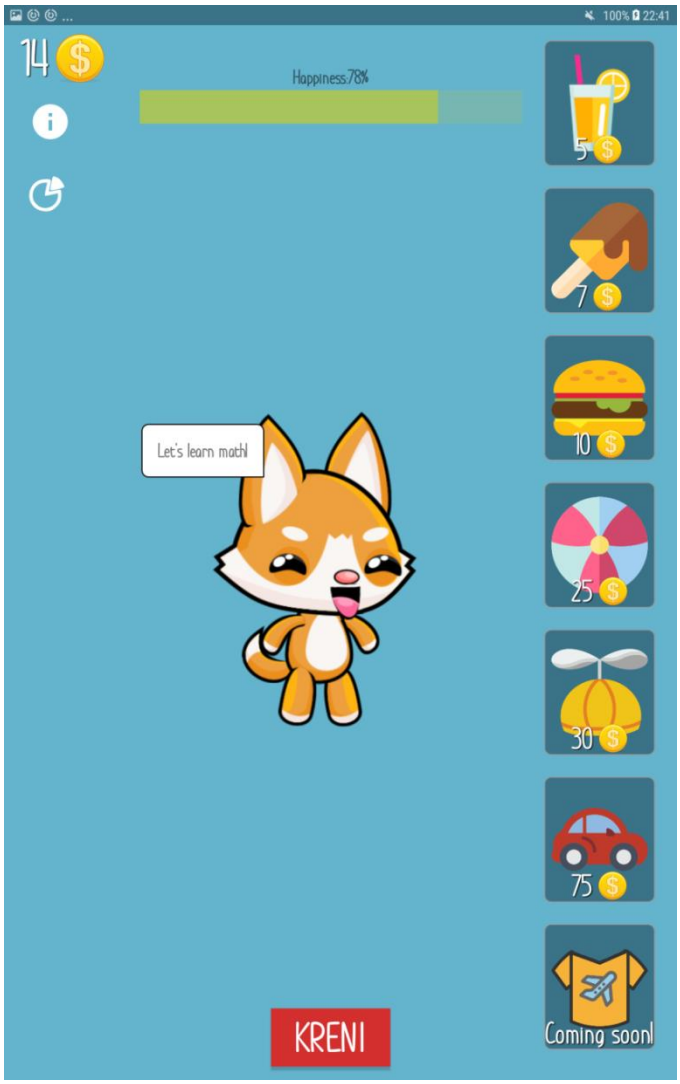


Slika 2. Dijagram slučajeva korištenja

1. Kao što je prikazano na slici, korisnik nakon pokretanja aplikacije na početnom zaslonu može izvršavati interakciju s ljubimcem tako što mu može kupovati razne stvari ili izazvati razne zvukove ili pokrete dodirom na ljubimca i njegove kupljene stvari.
2. Korisnik u ovom djelu može izabrati pregled statistike ili postavljati težinu za zadatke
    a. Postavljanje težina može se učiniti izborom tri predefinirane razine ili ručnim postavljanjem gdje korisnik svojevoljno podešava svaku postavku
    b. Unutar pregleda statistike korisnik može vidjeti omjer točnih i netočnih rješenja te postotak rješivosti po operatoru
3. Unutar aktivnosti za rješavanje zadataka korisniku se generiraju zadaci prema podešenim parametrima. Uz rješavanje zadataka korisniku iskaču razne obavijesti.

4. Aktivnosti unutar aplikacije

4.1. Main



Slika 3. Izgled početne aktivnosti

4.1.1. Aktivnost

Main ili glavna aktivnost pokreće se prilikom pokretanja aplikacije te se unutar nje pokreću servisi poput pozadinske melodije ili inicijalizacija baze podataka ako ne postoji. Ova aktivnost također se brine za svu interaktivnost s ljubimcem i njegovim stvarima.

```
// Start music
startService(Intent(this, BackgroundSoundService::class.java))
```

Kod 10. Primjer pozivanja servisa za pozadinsku melodiju

Baza podataka poziva se pri svakom pokretanju aktivnosti, ali kreira tablice samo ako ne postoje.

```
// Open Db and create tables if not existing
DbHelper(applicationContext).writableDatabas
```

Kod 11. Primjer pozivanja baze

U bazi se automatski pri prvom pokretanju postavlja broj novca na 0 tako da se vrijednost TextView elementa za prikaz novca dohvaća iz baze odmah nakon pozivanja. Ako je korisnik već koristio aplikaciju i zaradio određenu količinu novca, ona će biti prikazana.

```
val goldCurrent = DbHelper.getGoldValue(applicationContext).toString()
goldViewMain.text = goldCurrent
```

Kod 12. Dohvaćanje količine novca

Prilikom svakog povratka ili nastavka glavne aktivnosti provjerava se koje su od kupljenih stvari bile aktivirane i koje će biti odmah prikazane na zaslonu. U ovom trenutku se isto tako osvježava i popunjava lista sa stvarima za kupiti.

```
override fun onResume() {
    super.onResume()
    val petItems: List<PetItem> = DbHelper.getPetItems(applicationContext)

    for (petItem in petItems) {
        if (petItem.bindedElementId > 0) {
            val imgId = petItem.bindedElementId
            val relImg = (this as Activity).findViewById(imgId) as ImageView
            if (petItem.activated && petItem.bought) {
                relImg.visibility = View.VISIBLE
            } else {
                relImg.visibility = View.INVISIBLE
            }
        }
    }

    val adapter = PetItemAdapter(petItems)
    petItemList.adapter = adapter
    petItemList.layoutManager = LinearLayoutManager(this)
    checkHappiness()
}
```

Kod 13. Funkcija onResume koja se poziva prilikom svakog povratka na glavnu aktivnost

U glavnoj aktivnosti se također pridružuju sve funkcije za interakcije s ljubimcem i stvarima. U sljedećem kodu se može vidjeti postavljanje zvuka trube automobila, okretanje lopte i aviona te glasanje ljubimca na dodir.

```
imagePetCar.setOnClickListener {
    MediaPlayer.create(applicationContext, R.raw.car_horn).start()
}

imagePetBall.setOnClickListener {
    val rotate = RotateAnimation(0f, 720f, Animation.RELATIVE_TO_SELF, 0.5f,
Animation.RELATIVE_TO_SELF, 0.5f)
    rotate.duration = 2000
    rotate.interpolator = LinearInterpolator()
    imagePetBall.startAnimation(rotate)
}

imagePetPlane.setOnClickListener {
    MediaPlayer.create(applicationContext, R.raw.airplane).start()
    val rotate = RotateAnimation(0f, -350f, 15f, 15f)
    rotate.duration = 1000
    rotate.interpolator = LinearInterpolator()
    imagePetPlane.startAnimation(rotate)
}

imagePet.setOnClickListener {
    val sounds = listOf(R.raw.pet_sound1, R.raw.pet_sound2)
    MediaPlayer.create(applicationContext, sounds[rand.nextInt(sounds.size)]).start()
}
```

Kod 14. Interakcija sa stvarima i ljubimcem

Iz glavne aktivnosti je moguće pozvati aktivnost za prikaz statistike i aktivnost za upute dodirom na jednu od ikonica koje se nalaze uz lijevi rub prikaza.

```
iconChart.setOnClickListener {
    startActivity(Intent(applicationContext, GraphActivity::class.java))
}

iconInfo.setOnClickListener {
    startActivity(Intent(applicationContext, PopupInfoActivity::class.java))
}
```

Kod 15. Pokretanje aktivnosti statistika i uputa

Iznad ljubimca se nalazi traka koja prikazuje količinu sreće ljubimca koja se smanjuje svakih 6 minuta ili povećava ako se ljubimcu nešto kupi. Ovisno o količini sreće, mijenja se boja trake i raspoloženje ljubimca, a ta je funkcionalnost izvedena sljedećim kodom:

```
fun checkHappiness() {
    val curHappiness = DbHelper.getHapiness(this)
    textHappiness.text = getString(R.string.happiness) + curHappiness.toString() + "%"
    progressHappiness.progress = curHappiness
    when (curHappiness) {
        in 66..100 -> {
            progressHappiness.progressDrawable.setColorFilter(
                    ContextCompat.getColor(this, R.color.colorBgGreen),
                    android.graphics.PorterDuff.Mode.SRC_IN)
            textCloud.text = getString(R.string.cloud_happy)
            imagePet.setImageResource(R.drawable.pet_happy)
        }
        in 34..65 -> {
            progressHappiness.progressDrawable.setColorFilter(
                    ContextCompat.getColor(this, R.color.colorBtnYellow),
                    android.graphics.PorterDuff.Mode.SRC_IN)
            textCloud.text = getString(R.string.cloud_average)
            imagePet.setImageResource(R.drawable.pet)
        }
        in 0..33 -> {
            progressHappiness.progressDrawable.setColorFilter(
                    ContextCompat.getColor(this, R.color.colorBtnRed),
                    android.graphics.PorterDuff.Mode.SRC_IN)
            textCloud.text = getString(R.string.cloud_unhappy)
            imagePet.setImageResource(R.drawable.pet_unhappy)

        }
    }
}
```

Kod 16. Promijena boje trake i raspoloženje ljubimca ovisno o količini sreće

Kao što se može vidjeti, svaka trećina sreće mijenja sliku raspoloženja ljubimca, boju trake za sreću i tekst poruke unutar oblačića.

Prilikom svake kupnje pojavi se oblačić s porukom zahvalnosti iznad ljubimca koja ostaje na zaslonu dvije sekunde. Ta funkcionalnost izvedena je uz pomoć Timer klase unutar koje se još na dretvi korisničkog sučelja poziva provjera sreće ljubimca.

```
fun textCloudThankYou() {
    textCloud.text = getString(R.string.thank_you)
    timer.schedule(object : TimerTask() {
        override fun run() {
            runOnUiThread { checkHappiness() }
        }
    }, 2000)
}
```

Kod 17. Pokretanje aktivnosti statistika i uputa

4.1.2. Pogled

XML pogled glavne aktivnosti je dosta minimalističan. Sadrži kontejner ljubimca i njegovih stvari, ikonice statistike i informacija o aplikaciji, element za prikaz količine novca, listu sa stvarima za kupnju i gumb za izbor težine zadataka nakon koje počinje rješavanje. Kontejner ljubimca se poziva se iz druge datoteke s oznakom include, ikonice za statistiku i informacije su slike (ImageView), lista je tipa RecyclerView, a element za prikaz stanja novca je tipa GoldView koji je ručno izrađen.

```
<com.example.cromat.mathpath.view.GoldView
    android:id="@+id/goldViewMain"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/GoldView"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Kod 18. Primjer GoldView elementa unutar XML dokumenta

4.2. PopupDifficulty

Prije početka rješavanja zadataka potrebno je odabrati težinu zadataka. To je omogućeno skočnim prozorom na kojem je moguće odabrati jednu od tri predefinirane težine ili odabrati ručno podešavanje zadataka.

Slika 4. Skočni prozor za izbor težine zadataka

4.2.1. Aktivnost

Prilikom pritiska na jedan od tri gumba za izbor težine, generira se predefinirani objekt klase EquationConfig koji se prosljeđuje u SolvingActivity tj. aktivnost za rješavanje zadataka.

```
btnEasy.setOnClickListener {
    val equationConfig = EquationConfig(
            maxNumOperands = 2,
            minNumOperands = 2,
            maxNum = 10,
            minNum = 0,
            operators = listOf("+", "-"),
            gameType = GameType.STEPS.toString(),
            timeSec = 60,
            stepsNum = 10,
            negativeRes = false,
            randomizeInput = false
    )
    val intent = Intent(applicationContext, SolvingActivity::class.java)
    intent.putExtra("equationConfig", equationConfig)
    startActivity(intent); finish();}
```

Kod 19. GeneriranjeEquationConfig objekta pritiskom na gumb "Lagano"

Što se tiče gumba za ručno podešavanje, prilikom dodira na njega poziva se aktivnost za ručno podešavanje zadataka.

```
btnCustom.setOnClickListener {
    startActivity(Intent(applicationContext, CustomGameActivity::class.java))
    finish()
}
```

Kod 20. Poziv aktivnosti za ručno podešavanje zadataka

4.2.2. Pogled

Pogled za ovu aktivnost jako je jednostavan. Četiri guba su unutar LinearLayouta vertikalno poredani jedan ispod drugog.

```
<Button
    android:id="@+id/btnEasy"
    style="@style/BtnGreen"
    android:layout_width="wrap_content"
    android:minWidth="200dp"
    android:layout_height="wrap_content"
    android:text="@string/easy"
    android:layout_margin="10dp" />
```

Kod 21. Primjer gumba za odabir lagane težine unutar XML pogleda

4.3. CustomGame

Slika 5. Izgled aktivnosti za ručno podešavanje zadataka

4.3.1. Aktivnost

Unutar ove aktivnosti izvršava se postavljanje karakteristika za generiranje zadataka. Moguće je odabrati tip igre koji može biti na vrijeme ili na korake koje korisnik može dodatno podesiti, a zadane vrijednosti su 60 sekundi za vrijeme i 10 koraka ako je odabran tip igre na korake. Nakon toga postoji mogućnost izbora raspona brojeva koji će ući u jednadžbe. Zadani raspon brojeva koji će se pojaviti u zadacima je između 0 i 10. Sljedeća postavka je izbor operatora koji će doći u jednadžbe, a moguće je odabrati operatore za zbrajanje, oduzimanje, dijeljenje i množenje. Ispod postavke za operatore, nalazi se postavka za izbor broja operanada koji će ući u jednadžbu. Zadan je raspon da nasumično ulaze 2 do 3 operanda u jednadžbu. Na rasponu je moguće postaviti najmanje dva operanda, a najviše 5. Ako se označi postavka za korištenje zadanog broja operanada, tada je moguće odabrati fiksni broj operanada kojeg će svaka generirana jednadžba sadržavati. U zadnjem odjeljku postavki nalaze se tri moguće opcije za označiti, a to su mogućnost pojavnosti negativnih rezultata, mogućnost unosa broja na nasumičnom mjestu i mogućnost pojavnosti zagrada u jednadžbama. Nakon definiranja svih postavka i klika na gumb za pokretanje, generira se objekt EquationConfig koji se prosljeđuje u aktivnost za rješavanje zadataka koja se također istovremeno i pokreće.

Kao što je već navedeno, unutar ove aktivnosti korišteni su elementi za raspone. To nisu originalni android elementi nego spadaju u biblioteku simplerangeview koja je ručno dodana u projekt.

```
// RangeView listeners
rangeNumbers.onTrackRangeListener = (object : SimpleRangeView.OnTrackRangeListener {
    override fun onStartRangeChanged(@NotNull rangeView: SimpleRangeView, start: Int) {
        editRangeNumStart.setText(start.toString())
    }

    override fun onEndRangeChanged(@NotNull rangeView: SimpleRangeView, end: Int) {
        editRangeNumEnd.setText(end.toString())
    }
})
```

Kod 22. Primjer postavljanja elementa za raspon brojeva

Prije pokretanja aktivnosti za rješavanje, sva polja prolaze kroz funkciju koja potvrđuje ispravnost unesenih postavki.

```
// Steps validation
if (radioSteps.isChecked && editSteps.text.toString().toInt() < 1) {
    Toast.makeText(applicationContext,
        applicationContext.resources.getString(R.string.valid_steps),
        Toast.LENGTH_SHORT).show()
    return false
}
```

Kod 23. Primjer provjere ispravnosti kod unosa broja koraka

Nakon provjere ispravnosti, od unesenih postavki se kreira EquationConfig klasa i pokreće se aktivnost za rješavanje zadataka.

```
...
var gameType = GameType.STEPS.toString()
if (radioTime.isChecked)
    gameType = GameType.TIME.toString()
val timeSec = editTime.text.toString().toInt()
val stepsNum = editSteps.text.toString().toInt()
val negativeRes = checkNegativeRes.isChecked
val randomizeInput = checkRandomInputs.isChecked
val braces = checkBraces.isChecked

val equationConfig = EquationConfig(maxNumOperands, minNumOperands, maxNum,
        minNum, operators, gameType, timeSec, stepsNum, negativeRes,
        randomizeInput, braces = braces)
val intent = Intent(applicationContext, SolvingActivity::class.java)
intent.putExtra("equationConfig", equationConfig)
startActivity(intent)
```

Kod 24. Kreiranje EquationConfig objekta od unesenih postavki

CheckBox elementi ne preuzimaju globalne postavke fontova pa je na njima potrebno ručno postaviti font.

```
// Checkboxes font family
val font = ResourcesCompat.getFont(this, R.font.im_wunderland_cro)
checkRandomInputs.typeface = font
checkNegativeRes.typeface = font
checkFixedNumOperands.typeface = font
checkBraces.typeface = font
```

Kod 25. Ručno postavljanje fontova

4.3.2. Pogled

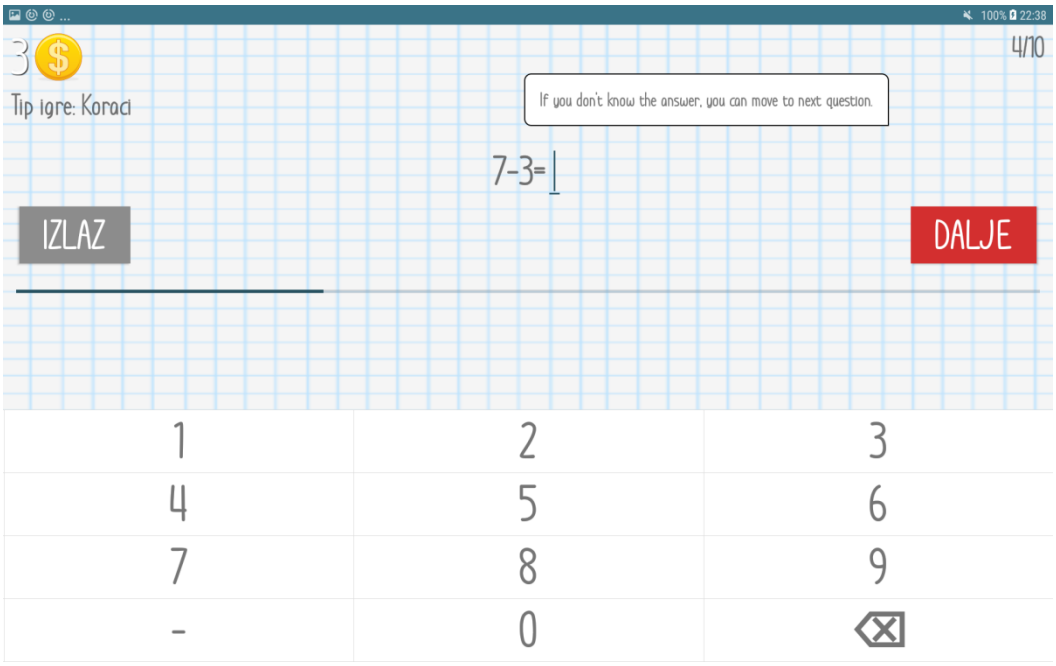
Zbog veće količine elemenata unutar aktivnosti, glavni element je ScrollView koji omogućuje sklizanje na zaslonu.

Zbog bolje preglednosti postavke su kategorizirane i odijeljene linijom koja je improvizirana View elementom, a svaka kategorija postavki smještena je unutar zasebnog RelativeLayout elementa.

```
<View android:id="@+id/line1" android:layout_width="fill_parent"
    android:layout_height="2dp"
    android:layout_below="@+id/relativeGameType"
    android:layout_marginLeft="@dimen/activity_horizontal_margin"
    android:layout_marginRight="@dimen/activity_horizontal_margin"
    android:layout_marginTop="@dimen/activity_horizontal_margin"
    android:background="#c0c0c0" />

<!--Numbers range-->
<RelativeLayout android:id="@+id/relativeNumRange"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/line1"
    android:padding="20dp">

    <TextView android:id="@+id/textNumRange"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/numbers_range"
        android:textSize="@dimen/default_text" />

    <me.bendik.simplerangeview.SimpleRangeView xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/rangeNumbers"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/padding_margin_40dp"
        app:count="100"
        app:end="10"
        app:endFixed="100"
        app:showFixedLine="true"
        app:start="0"
        app:startFixed="0" />
```

Kod 26. Primjer odjeljka i djela kategorije za postavljanje raspona brojeva

4.4. Solving



Slika 6. Izgled aktivnosti za rješavanje zadataka

4.4.1. Aktivnost

SolvingActivity, odnosno aktivnost za rješavanje zadataka, sadržava glavnu svrhu cijele aplikacije jer unutar nje se generiraju zadaci koje korisnik rješava i tako je nagrađivan novcem. Ovisno o tipu igre, korisnik ima određeno vrijeme ili određen broj koraka(zadataka) za rješavanje. Uvijek ima samo jedno polje za unos cijelog broja. Polje za unos se ne mora nužno nalaziti iza znaka jednakosti i rezultat može biti također negativan, a sve to ovisi o prethodnim postavkama. Nakon unosa broja ili ostavljanja polja praznim, korisnik može pritisnuti gumb „dalje“ nakon kojega će iskočiti obavijest o zarađenoj količini novca za točno riješen zadatak ili poruka o krivom rješenju te će se generirati novi zadatak ako vrijeme nije isteklo odnosno ako ima još koraka. Osim gumba za nastavak na sljedeći zadatak, s lijeve strane nalazi se gumb za izlaz nakon čijeg pritiska se gasi aktivnost za rješavanje i vraća korisnika na glavnu aktivnost s ljubimcem. Nakon što korisniku istekne vrijeme ili dođe do zadnjeg koraka, iskače podatak o broju točno riješenih zadataka u odnosu na ukupan broj generiranih zadataka. Uz taj podatak, korisnik može otvoriti i listu za prikaz svih zadataka s njihovim točnim rješenjima koje može usporediti sa svojima. Tijekom rješavanja zadataka, korisniku mogu iskakati razni savjeti koji se odnose na trenutni tip zadatka.

Zbog bolje organizacije koda unutar onCreate metode pozivaju se funkcije za inicijalizaciju elemenata, generiranje zadatka i postavlja se tip igre.

```
// Initialize SolvingActivity view
initView()

// Create first equation
nextEquation()

// Next
when (equationConfig.gameType) {
    GameType.STEPS.toString() -> {
        setStepGameType()
    }
    GameType.TIME.toString() -> {
        setTimeGameType()
    }
}
```

Kod 27. Funkcije koje se pozivaju unutar onCreate metode

Unutar funkcije nextEquation kreira se objekt klase Equation kojem se predaje objekt klase EquationConfig koji sadržava svu konfiguraciju zadataka te se prema njemu generira jednadžba. Zatim se postavlja jednadžba kao sadržaj TextView elemenata unutar kojih se naknadno zamjenjuju znakovi za množenje i dijeljenje s pravilnim matematičkim znakovima. Na kraju funkcije se poziva funkcija za postavljanje savjeta.

```
private fun nextEquation() {
    equation = Equation(equationConfig, this)
    if (equationConfig.randomizeInput) {
        val strings = equation.splitAtOperandIndex()
        txtViewEquationFirst.text = strings[0].replace("/", "\u00F7")
                .replace("*", "\u02E3")
        txtViewEquationSecond.text = strings[1].replace("/", "\u00F7")
                .replace("*", "\u02E3")
    } else {
        val userEquation = equation.toString().replace("/", "\u00F7")
                .replace("*", "\u02E3")
        txtViewEquationFirst.text = "$userEquation="
    }
    setHint(equation.toString())
}
```

Kod 28. Funkcija za postavljanje kreirane jednadžbe na TextView elemente

Funkcija za postavljanje savjeta u svoju listu savjeta predefinirano postavlja savjet da korisnik prijeđe na sljedeći zadatak ako trenutni ne može riješiti. Ako u jednadžbi postoje operacije dijeljenja i množenja u listu savjeta će također biti dodan savjet o tome da operacije množenja i dijeljenja imaju veći prioritet od operacija zbrajanja i oduzimanja. Ako

u jednadžbi postoje zagrade bit će dodan još i savjet o tome da se sve operacije unutar zagrada izvršavaju prije svega ostalog. Nakon svega toga se nasumično odabire jedan savjet iz liste koji na kraju ima trećinu šanse da bude prikazan na zaslonu.

```
private fun setHint(equation: String) {
    var hints: List<String> = listOf()
    hints += getString(R.string.hint_next)
    if ("/" in equation || "*" in equation)
        hints += getString(R.string.hint_div_mul)
    if ("(" in equation)
        hints += getString(R.string.hint_braces)

    if (Random().nextInt(3) == 1) {
        textHint.visibility = View.VISIBLE
        val index = Random().nextInt(hints.size)
        textHint.text = hints[index]
    }
    else
        textHint.visibility = View.INVISIBLE
}
```

Kod 29. Funkcija za postavljanje savjeta

Ako se radi o tipu igre s koracima, sve informacije na vrhu aktivnosti će biti vezane za taj tip igre i postavit će se brojač koraka te će klik na gumb za sljedeći zadatak prilagoditi tipu igre s koracima.

```
private fun setStepGameType() {
    stepperText.text = taskNum.toString() + "/" + equationConfig.stepsNum.toString()
    progressBarSolving.max = equationConfig.stepsNum
    btnNext.setOnClickListener {
        stepGame()
    }
}
```

Kod 30. Postavljanje tipa igre s koracima

Kod tipa igre s koracima, na svaki klik na sljedeći zadatak povećava se broj koraka koji se može vidjeti na vrhu aktivnosti među informacijama i u traci ispod jednadžbe koja se puni svakim novim korakom.

```
private fun stepGame() {
    stepperText.text = (taskNum + 1).toString() + "/" + equationConfig.stepsNum.toString()
    val userAns = edtViewAnswer.text.toString()
    checkAnswer(userAns)

    if (taskNum < equationConfig.stepsNum) {
        edtViewAnswer.text = SpannableStringBuilder("")
        nextEquation()
        taskNum++
    } else {
        stepperText.visibility = View.GONE
        setSolvingFinishedView()
    }
    progressBarSolving.incrementProgressBy(1)
}
```

Kod 31. Funkcija koja se poziva na svakom koraku igre

Ako se radi o tipu igre na vrijeme, postavljaju se informacije o tome i pokreće se štoperica koja odbrojava vrijeme koje je postavljeno u konfiguraciji.

```
private fun setTimeGameType() {
    stepperText.visibility = View.GONE
    timerText.visibility = View.VISIBLE
    progressBarSolving.max = equationConfig.timeSec
    timerText.text = equationConfig.timeSec.toString()
    startTimer()
    btnNext.setOnClickListener {
        timeGame()
    }
}
```

Kod 32. Postavljanje tipa igre na vrijeme

Kada je igra na vrijeme pokrenuta, može se vidjeti odbrojavanje na vrhu zaslona te na traci koja se svake sekunde puni. Funkcija koja se poziva pri svakom pritisku na gumb za sljedeći zadatak u ovom slučaju ne provjerava kada je kraj jer to radi funkcija koja odbrojava vrijeme.

```
private fun timeGame() {
    val userAns = edtViewAnswer.text.toString()
    checkAnswer(userAns)
    edtViewAnswer.text = SpannableStringBuilder("")
    nextEquation()
    taskNum++
}
```

Kod 33. Funkcija koja se poziva na svaki klik za sljedeći zadatak u tipu igre na vrijeme

```
private fun startTimer() {
    val countDownTimer = object : CountDownTimer((equationConfig.timeSec * 1000).toLong(),
1000) {
        override fun onTick(millisUntilFinished: Long) {
            timerText.text = (millisUntilFinished / 1000).toString()
            progressBarSolving.incrementProgressBy(1)
        }

        override fun onFinish() {
            timerText.text = "0"
            setSolvingFinishedView()
        }
    }.start()
}
```

Kod 34. Funkcija za odbrojavanje vremena

Kod oba tipa igre poziva se funkcija checkAnswer koja služi za provjeru ispravnosti rješenja ali i dodavanja analitike koja kasnije koristi za prikaz statistike rješivosti po operatorima. Unutar te funkcije nalazi se još jedna lokalna funkcija showTextGoldAdded koja služi za prikazivanje informacije o količini zarađenom novcu ili netočnom odgovoru u duljini od jedne sekunde.

```
fun showTextGoldAdded() {
    textGoldAdded.visibility = View.VISIBLE
    timer.schedule(object : TimerTask() {
        override fun run() {
            runOnUiThread { textGoldAdded.visibility = View.INVISIBLE }
        }
    }, 1000)
}
```

Kod 35. Prikazivanje elementa koji sadržava informaciju o zarađenom novcu ili krivom rješenju

Prilikom prikazivanja količine zarađenog novca mijenja se boja teksta i pojavljuje se zvuk.

```
if (equation.isCorrect(userAns)) {
    score++
    addVal = 1
    textGoldAdded.text = "+" + equationConfig.goldPerTask.toString()
    textGoldAdded.setTextColor(ContextCompat.getColor(this, R.color.colorBtnGreen))
    MediaPlayer.create(applicationContext, R.raw.right).start()
    showTextGoldAdded()
```

Kod 36. Slučaj prikaza količine zarađenog novca kada je odgovor ispravan

U ovom dijelu funkcije se može primijetiti da se poziva funkcija isCorrect klase Equation u kojoj se izvršava cijeli izraz i provjerava ispravnost jednadžbe uz pomoć MVEL biblioteke. Ova funkcija će biti detaljnije opisana u djelu s Equation modelom.

Nakon svakog zadatka, jednadžba se zajedno s rješenjem dodaje u listu odgovora koja se na kraju aktivnosti može prikazati.

```
listAnswers.add("${equation.toUserEquationString()};$userAns;${equation.getRightAns()}")
```

Kod 37. Dodavanje odgovora u listu

Također, u bazu se dodaje informacija o tome jeli za određeni operator u jednadžbi rješenje ispravno. Ta informacija kasnije služi za statistiku te novo generiranje zadataka gdje će veće šanse za generiranje imati jednadžbe s operatorom koji ima manji postotak riješenosti.

```
val values: MutableMap<String, Int> = mutableMapOf("+" to 0, "-" to 0, "/" to 0, "*" to 0)
…
for (key in values.keys) {
    if (equation.toString().contains(key))
        values[key] = addVal
}

DbHelper.addOperations(values, applicationContext)
```

Kod 38. Dodavanje broja točno ili netočno riješenih zadataka po operatoru u ejdnadžbi

4.4.2. Pogled

Na samom vrhu aktivnosti s lijeve strane nalazi se nekoliko TextView i jedan GoldView element. TextView elementi sadržavaju informacije o trenutnom tipu igre, a GoldView ima istu ulogu kao i na glavnoj aktivnosti, a to je prikaz trenutnog stanja novca.

Na gornjoj desnoj strani pogleda, nalazi se informacija o trenutnom broju koraka ili vremenu igre.

```
<TextView
    android:id="@+id/timerText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginEnd="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:visibility="gone"
    android:textSize="@dimen/default_text"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Kod 39. TextView element za prikaz preostalog vremena

Na sredini pogleda nalaze se tri dva TextView i jedan EditText element. Ako je postavljeno da se polje za unos nalazi isključivo iza znaka jednakosti, tada se desni TextView ne koristi nego je tekst cijele jednadžbe postavljen unutar lijevog TextView elementa. Ako se polje za unos može naći na bilo kojem drugom mjestu, tada se jednadžba rastavlja na dva stringa koji su podijeljeni u lijevi i desni TextView.

```
<TextView
    android:id="@+id/txtViewEquationFirst"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:textSize="@dimen/text_size_solving" />

<EditText
    android:id="@+id/edtViewAnswer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_toEndOf="@+id/txtViewEquationFirst"
    android:layout_toRightOf="@+id/txtViewEquationFirst"
    android:gravity="center"
    android:imeOptions="flagNoExtractUi"
    android:inputType="numberSigned"
    android:paddingTop="0dp"
    android:textSize="@dimen/text_size_solving" />

<TextView
    android:id="@+id/txtViewEquationSecond"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toEndOf="@id/edtViewAnswer"
    android:layout_toRightOf="@id/edtViewAnswer"
    android:gravity="center"
    android:textSize="@dimen/text_size_solving" />
```

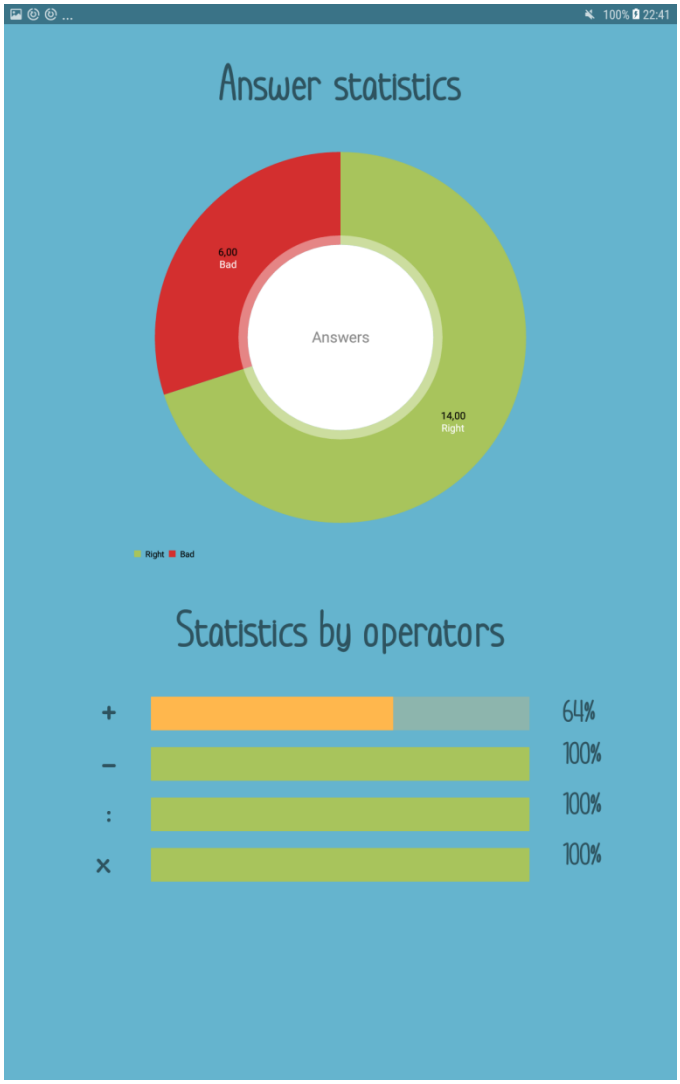
Kod 40. Tri elementa za prikaz jednadžbe i unos rješenja

Ispod tih elemenata nalaze se gumbovi za izlaz i za nastavak na sljedeći zadatak, svaki uz jedan rub ekrana.

Na samom dnu nalazi se traka koja se puni ovisno o tipu igre te je već objašnjena u prethodnom djelu.

4.5. Graph

GraphActivity služi za prikaz jednostavnih statistika kao što su ukupan omjer točnih i netočnih odgovora i postotak rješivosti po operatorima.



Slika 7. Izgled aktivnosti za statistiku

4.5.1. Aktivnost

Aktivnost za prikaz statistika koristi MPAndroidChart library za prikaz grafa u obliku torte i ProgressBar elemente za prikaz postotka rješivosti po operatorima. Aktivnost dohvaća podatke iz baze koji su zabilježeni tijekom rješavanja zadataka i prikazuje ih u obliku grafova.

```
val sumRight = results.sumBy { result -> result.score }
val sumBad = results.sumBy { result -> result.numAns } - sumRight
val entries = ArrayList<PieEntry>()
entries.add(PieEntry(sumRight.toFloat(), getString(R.string.right)))
entries.add(PieEntry(sumBad.toFloat(), getString(R.string.bad)))
```

Kod 41. Dio koda koji na PieChart postavlja omjer točnih i netočnih odgovora

Osim postavljanja podataka na graf, potrebno je postaviti labele, boje i neke sitne konfiguracije koje utječu na krajnji izgled grafa.

Postavljanje ProgressBar elemenata izvršava se nakon povlačenja gotovih podataka o postotku rješivosti po svakom operatoru pa je potrebno samo postaviti progress atribut i TextView element na broj dobiven iz baze.

```
val perByOperators = DbHelper.getPercentageByOperation(this)
progressPlus.progress = perByOperators[0]
…
textPercentPlus.text = perByOperators[0].toString() + "%"
```

Kod 42. Dohvaćanje i postavljanje podatka o rješivosti po operatoru zbrajanja

Da bi podaci u trakama bili izraženiji, dodane su boje koje ovise o postotku.

```
in 66..100 -> {
    progressPlus.progressDrawable.setColorFilter(
            ContextCompat.getColor(this, R.color.colorBgGreen),
            android.graphics.PorterDuff.Mode.SRC_IN)
}
in 34..65 -> {
    progressPlus.progressDrawable.setColorFilter(
            ContextCompat.getColor(this, R.color.colorBtnYellow),
            android.graphics.PorterDuff.Mode.SRC_IN)
}
in 0..33 -> {
    progressPlus.progressDrawable.setColorFilter(
            ContextCompat.getColor(this, R.color.colorBtnRed),
            android.graphics.PorterDuff.Mode.SRC_IN)

}
```

Kod 43. Promjena boje ProgressBar elementa ovisno o postotku

4.5.2. Pogled

Iako pogled nije kompleksan, sadrži dosta elemenata, a posebice zato što su ProgressBar elementi okruženi TextView elementima s obje strane koji prikazuju o kojem se operatoru radi i koliki postotak rješivosti ima u brojkama.

Jedini nestandardni element u ovom pogledu je PieChart koji nema nikakvih posebnih karakteristika unutar XML-a pošto se većina stvari na njemu podešava dinamično.

```
<com.github.mikephil.charting.charts.PieChart
    android:id="@+id/pieChart"
    android:layout_width="@dimen/pie_chart_size"
    android:layout_height="@dimen/pie_chart_size"
    android:layout_below="@id/textTitlePieChart"
    android:layout_centerHorizontal="true"
    android:backgroundTint="@color/colorBg"
    app:backgroundTint="@color/colorBg" />
```

Kod 44. Primjer PieChart XML elementa

4.6. BgMusic

4.6.1. Aktivnost

BgMusicActivity je aktivnost koja se brine o servisu pozadinske glazbe i nasljeđuju je sve aktivnosti u aplikaciji.

Aktivnost upravlja servisom na nekoliko ciklusnih metoda. Na početak i nastavljanje svake aktivnosti pokreće se servis za pozadinsku glazbu, a na izlaz iz aplikacije ili slanjem aplikacije u pozadinu, zaustavlja se servis za glazbu.

```
override fun onStop() {
    onExit()
    super.onStop()
}

override fun onResume() {
    if (!isMyServiceRunning(BackgroundSoundService::class.java)) {
        startService(Intent(applicationContext, BackgroundSoundService::class.java))
    }
    super.onResume()
}

private fun onExit() {
    if (isApplicationSentToBackground(applicationContext))
        stopService(Intent(applicationContext, BackgroundSoundService::class.java))
}
```

Kod 45. Pokretanje i zaustavljanje pozadinske muzike na određene okidače

Za provjeru jeli aplikacija poslana u pozadinu, koristi se sljedeći kod:

```
private fun isApplicationSentToBackground(context: Context): Boolean {
    val am = context.getSystemService(Context.ACTIVITY_SERVICE) as ActivityManager
    val tasks = am.getRunningTasks(1)
    if (!tasks.isEmpty()) {
        val topActivity = tasks[0].topActivity
        if (topActivity.packageName != context.packageName) {
            return true
        }
    }
    return false
}
```

Kod 46. Provjera jeli aplikacija u pozadini

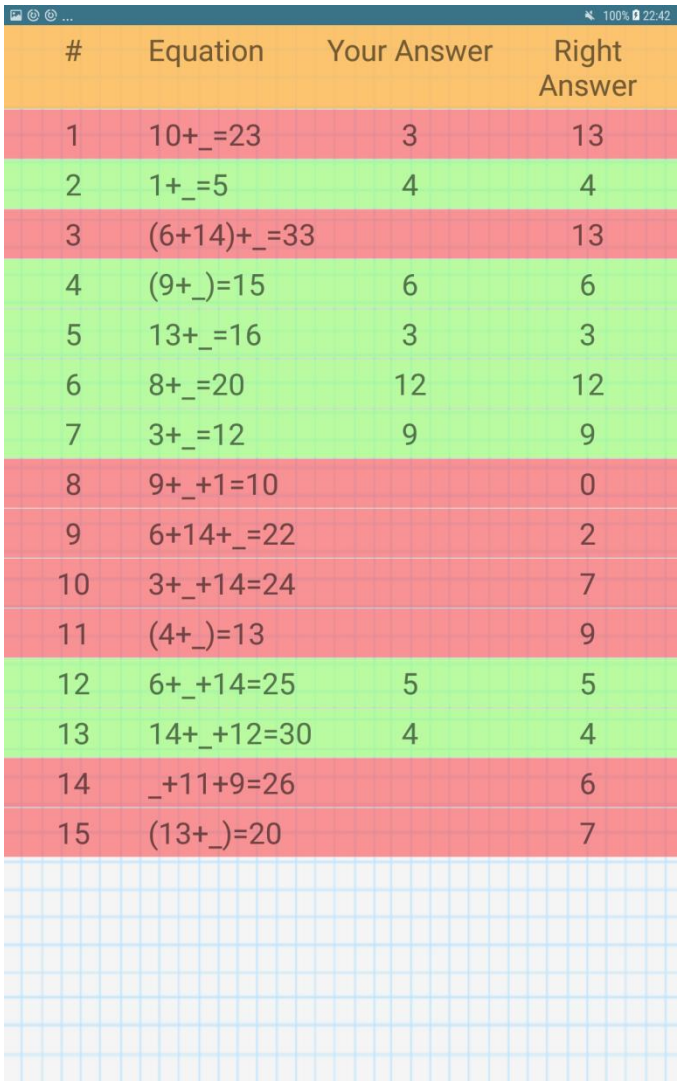
Kod nastavljanja rada aplikacije, potrebno je znati jeli servis za glazbu već pokrenut.

```
private fun isMyServiceRunning(serviceClass: Class<*>): Boolean {
    val manager = getSystemService(Context.ACTIVITY_SERVICE) as ActivityManager
    for (service in manager.getRunningServices(Integer.MAX_VALUE)) {
        if (serviceClass.name == service.service.className) {
            return true
        }
    }
    return false
}
```

Kod 47. Provjera rada servisa za glazbu

4.6. AnswerList

Ova jednostavna aktivnost služi za prikazivanje jednadžbi s njihovim točnim i unesenim rješenjima od korisnika. Sastavljena je od jednostavnog ListViewa koji se popunjava podacima koji su dobiveni putem Intent objekta iz aktivnosti za rješavanje. Zbog bolje preglednosti zaglavlje tablice obojeno je u narančastu boju. Točni odgovori su obojeni zeleno, a netočni crveno. Za dodavanje pojedinog retka u tablicu brine se AnswerListAdapter u kojem je smještena sva logika.



Slika 8. Izgled AnswerList aktivnosti

5. Modeli unutar aplikacije

5.1. Equation

Equation model predstavlja jednadžbu unutar aplikacije. Svaki objekt klase Equation sadržava listu operatora i operanada, reprezentaciju u stringu, prvi i drugi dio jednadžbe ako se rastavlja na dva djela, točan odgovor i indekse na kojima se pojavljuju otvorena i zatvorena zagrada.
Konstruktor Equation klase prima objekt EquationConfig klase i kontekst.

```
class Equation(private var eConfig: EquationConfig, private val ctx: Context) {
    private var operands = listOf<String>()
    private var operators = listOf<String>()
    private var equationStr = ""
    private val rand = Random()
    private var result: Int? = null
    private var firstString = ""
    private var secondString = ""
    private var rightAns = ""
    private var braceOpen = -1
    private var braceClose = -1
```

Kod 48. Konstruktor s članovima klase

Prilikom instanciranja objekta ove klase, automatski se generira jednadžba. Generiranje jednadžbe izvršava se tako da se koriste podaci EquationConfig objekta. Ako nije definiran stalan broj operanada, odabire se nasumičan broj za trenutnu jednadžbu.

```
var randNumOperands = eConfig.maxNumOperands
if (eConfig.maxNumOperands > eConfig.minNumOperands) {
    randNumOperands = rand.nextInt(eConfig.maxNumOperands + 1
            - eConfig.minNumOperands) + eConfig.minNumOperands
}
```

Kod 49. Dobivanje nasumičnog broja operanada za trenutnu jednadžbu

Ako je uključena opcija za pojavljivanje zagrada, generiraju se indeksi otvaranja i zatvaranja istih.

```
if (eConfig.braces && rand.nextInt(2) == 1) {
    braceOpen = rand.nextInt(randNumOperands - 1)
    braceClose = rand.nextInt(randNumOperands - braceOpen - 1) + 1 + braceOpen}
```

Kod 50. Kreiranje indeksa za zagrade

Nakon zagrada, generiraju se operandi i operator za jednadžbu prolazeći kroz petlju. Ako je prethodni operator u petlji bio onaj za dijeljenje, tada se sljedeći operand prilagođava da izraz bude jednak cijelom broju i da ne dođe do dijeljenja s nulom.

```
for (i in 0 until randNumOperands) {
    if (operator == "/") {
        val lastNum = randNum
        randNum = 1
        if (lastNum > 1) {
            for (j in 2 until lastNum) {
                if (lastNum % j == 0) {
                    randNum = j
                    break
                }
            }
        }
    } else {
        randNum = rand.nextInt(eConfig.maxNum - eConfig.minNum) + eConfig.minNum
    }
```

Kod 51. Petlja unutar koje se dodaju operatori i operandi u jednadžbu

Nakon što se doda nasumični operand u jednadžbu, slijedi dodavanje operatora koji se dodaje ovisno o postotku rješivosti. Npr. ako korisnik ima lošiji prosjek rješavanja zadataka s operatorom za množenje, veće su šanse da će mu u jednadžbu doći operator za množenje.

```
// Choose operator by percentage solving
val perByOperators = DbHelper.getPercentageByOperation(ctx)
val mapPerByOp = mutableMapOf<String, Int>()
mapPerByOp["+"] = perByOperators[0]
mapPerByOp["-"] = perByOperators[1]
mapPerByOp["/"] = perByOperators[2]
mapPerByOp["*"] = perByOperators[3]

val sortedPerByOp = mapPerByOp.toList()
        .sortedBy { (_, value) -> value }.toMap()

operator = "+"
val foo = rand.nextInt(401 - sortedPerByOp.values.sum())
var sum = 0
for (key in sortedPerByOp.keys) {
    sum += 100 - sortedPerByOp[key]!!.toInt()
    if (foo < sum) {
        operator = key
        break
    }}
```

Kod 52. Dodavanje operatora prema postotku rješivosti

Nakon što je jednadžba izgenerirana, odbacuje se zadnji znak jer je to operator koji je višak i izračunava se točan rezultat da bi se s njim poslije moglo uspoređivati korisničko rješenje.

```
equationStr = equationStr.dropLast(1)
operators = operators.dropLast(1) + ""
val evaluated = MVEL.evalToString(equationStr)
result = evaluated.toDouble().toInt()
rightAns = result.toString()
```

Kod 53. Odbacivanje zadnjeg znaka i evaluacija jednadžbe

Ova klasa također sadržava metodu za razdvajanje jednadžbe u dva stringa koja služi kada se polje za unos može pojaviti na bilo kojem mjestu. Za rastavljanje stringova koristi se slična metoda kao i kod kreiranja cijele jednadžbe. Prolazi se dvjema petljama, prvom do nasumičnog indeksa na kojem će se nalaziti polje za unos. Druga petlja prolazi od tog indeksa pa sve do kraja originalne jednadžbe. Iz lista operatora i operanada se redom uzimaju znakovi i povezuju da bi kreirali dva stringa.

Zadnja bitna funkcija ove klase je ona za provjeru ispravnosti jednadžbe nakon unesenog broja. Funkcija prima string koji predstavlja uneseni broj te se putem MVEL biblioteke evaluira rezultat s tim brojem i provjerava poklapa li se on sa stvarnim točnim riješenjim.

```
fun isCorrect(number: String): Boolean {
    if (!number.isBlank() && number.matches(Regex("^-?\\d+(\\.\\d+)?\$"))) {
        if (eConfig.randomizeInput) {
            val eqToTry = ("$firstString$number$secondString").split("=")[0]
            val evaluated = MVEL.evalToString(eqToTry)
            if (evaluated.toDouble().toInt() == result) {
                rightAns = number
                return true
            }
        } else {
            val evaluated = MVEL.evalToString(equationStr)
            if (evaluated.toDouble().toInt() == number.toInt()) {
                rightAns = number
                return true
            }
        }
    }
    return false
}
```

Kod 54. Provjera ispravnosti jednadžbe

Na početku funkcije provjerava se jeli je unesen broj te se nakon toga, ovisno o parametru o nasumičnoj pojavi polja za unos, povežu vrijednosti TextView elemenata u jednadžbu koja se evaluira. Ako se dobiveni rezultat poklapa s unesenim, funkcija vraća istinitost tj. true.

5.2. EquationConfig

EquationConfig klasa sadrži sve postavke koje će se koristiti kod generiranja jednadžbi i u sebi sadržava samo zadani konstruktor.

```
class EquationConfig(var maxNumOperands: Int = 2,
                     var minNumOperands: Int = 2,
                     var maxNum: Int = 10,
                     var minNum: Int = 0,
                     var operators: List<String> = listOf<String>("+", "-"),
                     var gameType: String = GameType.STEPS.toString(),
                     var timeSec: Int = 60,
                     var stepsNum: Int = 10,
                     var negativeRes: Boolean = false,
                     var randomizeInput: Boolean = false,
                     var goldPerTask: Int = 1,
                     var braces: Boolean = false
) : Serializable
```

Kod 55. EquationConfig klasa

5.3. PetItem

PetItem klasa predstavlja bilo koju stvar koju je moguće kupiti ljubimcu. Sadržava zadani konstruktor koji prima naziv, cijenu, sliku, vezani element, količinu sreće koju donosi te podatke o tome jeli stvar moguće kupiti samo jednom ili više puta, jeli aktivirana i jeli kupljena.

```
class PetItem(
        var name: String,
        var price: Int,
        var permanent: Boolean,
        var bought: Boolean,
        var activated: Boolean,
        var picture: Int,
        var bindedElementId: Int,
        var happiness: Int
) : Parcelable
```

Kod 56. PetItem klasa

6. Prilagođeni elementi

Prilagođeni ili korisnički elementi su ručno kreirani elementi koji nasljeđuju postojeće elemente i služe za bolju organizaciju, lakšu čitljivost koda i efikasniju uporabu unutar aplikacije.

6.1. GoldView

GoldView je element namijenjen za prikaz trenutnog stanja novca. Nasljeđuje TextView element te mu je dodana slike s desne strane od teksta. Pošto GoldView može biti različitih veličina, potrebno je sliku prilagoditi veličini teksta. Najlakši način je da se dohvati postojeća slika i od nje kreira nova koja će biti veličine teksta.

```
private fun init(attrs: AttributeSet?) {
    // Load attributes
    val a = context.obtainStyledAttributes(attrs, R.styleable.GoldView)
    val img = ContextCompat.getDrawable(context, R.drawable.gold)

    val displayMetrics = DisplayMetrics()
    (context as Activity).windowManager.defaultDisplay.getMetrics(displayMetrics)
    val size: Int = (lineHeight * 0.85).toInt()

    val bitmap = (img as BitmapDrawable).bitmap
    val coinImage = BitmapDrawable(resources, Bitmap.createScaledBitmap(bitmap, size, size, true))
    setCompoundDrawablesWithIntrinsicBounds(null, null, coinImage, null)
    compoundDrawablePadding = 10
    setTypeface(ResourcesCompat.getFont(context, R.font.im_wunderland_cro), Typeface.BOLD)
    setShadowLayer(1.6f, 1.5f, 1.3f, Color.BLACK)
    setTextColor(Color.WHITE)
    gravity = Gravity.CENTER_VERTICAL or Gravity.RIGHT
    a.recycle()
}
```

Kod 57. Funckija init unutar koje se prilagođuje veličina slike novčića s veličinom teksta

U nekim slučajevima unutar aplikacije bilo je potrebno ukloniti sliku novčića pa je to postignuto sljedećim kodom

```
fun removeImage(){
    setCompoundDrawablesWithIntrinsicBounds(null, null, null, null)
    invalidate()
}
```

Kod 58. Uklanianie slike s GoldView elementa

6.2. KeyboardView

KeyboardView je prilagođena tipkovnica koja se koristi kod rješavanja zadataka. Prednosti ove tipkovnice nad predefiniranom su što je uvijek aktivna, sadrži samo brojeve, znak minus i tipku za brisanje. Pogled joj je sačinjen od tablice popunjene TextView elementima. Aktivnost koja se poziva prilikom klika na svaku tipku je jednostavno implementirana. Ako se radi o broju, taj isti broj se nadopiše na polje, a u slučaju pritiska tipke za brisanje briše se zadnji znak.

```
override fun onClick(v: View) {
    if (edtViewAnswer!!.text.length < 7) {
        if ((v as TextView).tag != null && "number_button" == v.tag) {
            if (!edtViewAnswer!!.text.isBlank() && edtViewAnswer!!.text[0] == '0')
                edtViewAnswer!!.setText("0")
            else
                edtViewAnswer!!.append(v.text)
            return
        }
    }
    when (v.id) {
        t9_key_backspace.id -> {
            val editable = edtViewAnswer!!.text
            val charCount = editable.length
            if (charCount > 0) {
                editable.delete(charCount - 1, charCount)
            }
        }
    }
}
```

Kod 59. Implementacija klikova na tipke tipkovnice

6.3. PetItemView

PetItemView je element koji vizualno predstavlja svaku stvar koju je moguće kupiti. Sadrži GoldView i ImageView elemente koji služe za prikazivanje cijene i slike određene stvari. Kod kreiranja elementa potrebno je dodijeliti objekt klase PetItem nakon čega se postavlja inicijalni izgled elementa.

```
var petItem: PetItem? = null
    set(petItem) {
        field = petItem
        goldView = this.findViewById(R.id.petItemGold)
        imgView.setImageDrawable(ContextCompat.getDrawable(context, petItem!!.picture))
        if (petItem.price == 9999) {
            goldView.text = context.getString(R.string.coming_soon)
            goldView.removeImage()
        }
        else
            goldView.text = petItem.price.toString()
        if (petItem.bindedElementId > 0) {
            val imgId = petItem.bindedElementId
            val relImg = (context as Activity).findViewById(imgId) as ImageView
            if (petItem.activated && petItem.bought) {
                alpha = .5f
                relImg.visibility = View.VISIBLE
            } else {
                alpha = 1f
                relImg.visibility = View.INVISIBLE
            }

            if (petItem.bought) {
                goldView.visibility = View.INVISIBLE
            }
        }
        invalidate()
    }
```

Kod 60. Postavljanje PetItem objekta

Prilikom klika na neku od stvari pokreće se onClick metoda koja provjerava mogućnost kupovine trenutne stvari. Dohvaća se trenutna količina novca i ako ga ima dovoljno na raspolaganju, stvar se kupuje i mijenja se njegov izgled ako se radi o igrački (jer se može samo jednom kupiti).

```
    if (!petItem!!.bought)
        if (!DbHelper.addGold(petItem!!.price * -1, context))
            Toast.makeText(context, context.getString(R.string.no_gold), Toast.LENGTH_LONG).show()
        else {
            addHappiness()
            val imgId = petItem!!.bindedElementId
            val relImg = (context as Activity).findViewById(imgId) as ImageView
            petItem!!.bought = true
            goldView.visibility = View.INVISIBLE
            if (petItem!!.activated) {
                alpha = .5f
                relImg.visibility = View.VISIBLE
            } else {
                alpha = 1f
                relImg.visibility = View.INVISIBLE
            }
```

Kod 61. Dio koda koji se izvršava pri kliku na stvar

Ako korisnik nema dovoljno novca za kupnju stvari, na zaslon se ispisuje odgovarajuća poruka.

```
if (!DbHelper.addGold(petItem!!.price * -1, context))
    Toast.makeText(context, context.getString(R.string.no_gold), Toast.LENGTH_LONG).show()
else
    addHappiness()
```

Kod 62. Ispisivanje poruke ukoliko korisnik nema dovoljno novca

7. Ostalo

U ovom poglavlju bit će opisani servisi i klase za upravljanje pozadinskom glazbom, bazom podataka i preostali dijelovi aplikacije.

7.1. BackgroundSoundService

Ovaj servis zadužen je za pozadinsku glazbu koja svira dok god je aplikacija upaljena i dok nije u pozadini.

Prilikom kreiranja ovog servisa inicijalizira se varijabla tipa MediaPlayer kojoj se dodjeljuje glazbena datoteka. Također se kreira objekt tipa HomeWatcher koji poziva određeni dio koda ako korisnik pritisne home button.

```
override fun onCreate() {
    super.onCreate()
    player = MediaPlayer.create(this, R.raw.bkground)
    player.isLooping = true
    player.setVolume(100f, 100f)

    val mHomeWatcher = HomeWatcher(this)
    mHomeWatcher.setOnHomePressedListener(object : OnHomePressedListener {
        override fun onHomePressed() {
            stopSelf()
        }

        override fun onHomeLongPressed() {
            stopSelf()
        }
    })
    mHomeWatcher.startWatch()
}
```

Kod 63. Kod koji se poziva prilikom kreiranja servisa za glazbu

7.2. HomeWatcher

Pošto android nema implementiran jednostavan način za praćenje pritiska home button-a, potrebna je implementacija vlastitog rješenja tj. HomeWatcher klase. Klasa prati pozive pritiske gumbova te ako je pritisnut home button tada korisnik može implementirati funkciju sučelja OnHomePressedListener.

```
internal inner class InnerRecevier:BroadcastReceiver() {
    private val SYSTEM_DIALOG_REASON_KEY = "reason"
    private val SYSTEM_DIALOG_REASON_RECENT_APPS = "recentapps"
    private val SYSTEM_DIALOG_REASON_HOME_KEY = "homekey"

    override fun onReceive(context:Context, intent:Intent) {
        val action = intent.action
        if (action == Intent.ACTION_CLOSE_SYSTEM_DIALOGS)
        {
            val reason = intent.getStringExtra(SYSTEM_DIALOG_REASON_KEY)
            if (reason != null)
            {
                Log.e(TAG, "action:$action,reason:$reason")
                if (mListener != null)
                {
                    if (reason == SYSTEM_DIALOG_REASON_HOME_KEY)
                    {
                        mListener!!.onHomePressed()
                    }
```

Kod 64. Dio koda koji prati jeli korisnik pritisnuo home button

Ako je korisnik duže držao home button, tada se šalje signal „recentapps“, a ova klasa omogućuje korisniku implementiranje metode onHomeLongPressed u ovom slučaju.

```
interface OnHomePressedListener {
    fun onHomePressed()
    fun onHomeLongPressed()
}
```

Kod 65. Sučelje čije metode korisnik može implementirati na pritiske home button-a

7.3. DbHelper

DbHelper je klasa koja služi kao pomoć pri komunikaciji s lokalnom SQLite bazom podataka. Putem nje se kreiraju tablice zas ve modele, ažuriraju sve promjene i dohvaćaju svi potrebni podaci iz baze. Sve metode su statičke, a prilikom prvog pokretanja kreiraju se sve tablice. DbHelper klasu moguće je uvijek dohvatiti i konteksta jer je Context klasa proširena članom baze podataka. Radi lakšeg rada s lokalnom bazom korištena je AnkoDb biblioteka.

Zbog bolje čitljivosti i pisanja koda, nazivi svih tablica spremljeni su u konstante.

```
const val TABLE_RESULT = "result"
const val TABLE_OPERATIONS = "operations"
const val TABLE_PET_ITEMS = "pet_items"
const val TABLE_GOLD = "gold"
const val TABLE_HAPPINESS = "happiness"
```

Kod 66. Nazivi svih tablica

Kreiranje tablica i inicijalizacija podataka izvršava se unutar onCreate metode.

```
override fun onCreate(db: SQLiteDatabase) {
    val dateTimeMillis = Calendar.getInstance().timeInMillis
    // Results
    db.run {
        createTable(TABLE_RESULT, true,
                "id" to INTEGER + PRIMARY_KEY + UNIQUE,
                "date" to TEXT,
                "score" to INTEGER,
                "numAns" to INTEGER,
                "gameType" to TEXT
        )
…
// Default gold value to 0
execSQL("INSERT INTO $TABLE_GOLD (id, value) VALUES(0, 0)")
…
```

Kod 67. Kreiranje tablice rezultata i postavljanje predefiniranekoličine novca

Kod tablice za spremanje podataka o količini novca potreban nam je uvijek samo jedan redak kojeg ćemo ažurirati pa na nju postavljamo posebnu opciju prilikom kreiranja.

```
// Gold
createTable(TABLE_GOLD, true,
        "id" to INTEGER + PRIMARY_KEY + SqlTypeModifier.create("CHECK (id = 0)"),
        "value" to INTEGER + DEFAULT("0")
)
```

Kod 68. Kreiranje tablice za spremanje količine novca

Prilikom dodavanja novca poziva se funkcija addGold unutar koje se provjerava hoće li količina novca pasti ispod 0 u slučaju kupovine neke stvari. To naravno neće biti dozvoljeno pa će funkcija vratiti false, a ako postoji dovoljno novca, funkcija će vratiti true.

```
fun addGold(value: Int, ctx: Context): Boolean {
    if (instance == null) {
        instance = DbHelper(ctx.applicationContext)
    }

    val currGold = instance!!.use {
        select(TABLE_GOLD, "value").whereArgs("id == 0").exec { parseSingle(IntParser) }
    }

    if (currGold + value < 0)
        return false

    instance!!.use {
        execSQL("UPDATE " + TABLE_GOLD + " SET value = value + " + value.toString())
    }
    return true
}
```

Kod 69. Funkcija za dodavanje novca u bazu

Dodavanje sreće vrši se na sličan način. Jedina razlika je što se u funkciji za dodavanje sreće provjerava jeli ona veća od 100 prilikom dodavanja. Ako zbroj trenutne sreće i broja za koji se ona povećava veći od 100, sreća će biti postavljena na maksimalnu vrijednost(100).

Od ostalih funkcija tu su još funkcije za dohvaćanje sreće, svih stvari koje se mogu kupiti ljubimcu zajedno s njihovim trenutnim stanjem, ažuriranje istih, dodavanje operacija, rezultata i dohvaćanje postotka po operaciji.

```
execSQL("INSERT INTO $TABLE_PET_ITEMS (name, price, permanent, activated, " +
        "bought, picture, bindedElementId, happiness) VALUES('Drink', 5, 0, 0, 0, " +
        R.drawable.drink.toString() + ", 0, 10)")
```

Kod 70. Primjer dodavanja stvari za ljubimca u bazu

Kod dodavanja retka u tablicu operatora za svaki operator koji se nalazio u jednadžbi dodaje se vrijednost 1 ili -1 ako je zadatak bio točan, odnosno netočan, a inače će biti upisana 0. Da bi se od toga izračunao prosjek, potrebno je izbaciti nule jer u obzir ulaze rezultati samo onih operatora koju su se pojavljivali u jednadžbama. Pošto prosjek može biti negativan, mora se rezultatu dodati još vrijednost 1 i sve podijeliti s 0.02 da bi se dobio postotak rješivosti.

```
fun getPercentageByOperation(ctx: Context): List<Int> {
    if (instance == null) {
        instance = DbHelper(ctx.applicationContext)
    }

    var percentageList = listOf(0)

    instance!!.use {
        percentageList = select(TABLE_OPERATIONS,
                "CAST((AVG(NULLIF(plus, 0)) + 1)/0.02 AS INTEGER) AS plus",
                "CAST((AVG(NULLIF(minus, 0)) + 1)/0.02 AS INTEGER) AS minus",
                "CAST((AVG(NULLIF(divide, 0)) + 1)/0.02 AS INTEGER) AS divide",
                "CAST((AVG(NULLIF(multiple, 0)) + 1)/0.02 AS INTEGER) AS multiple")
                .exec { parseSingle(parserOperatorsPercentage) }
    }
    return percentageList
}
```

Kod 71. Dohvaćanje prosjeka rješivosti po operatoru

Na kraju se kao rezultat vraća lista s postocima. Da bi se izraz pretvorio u listu potrebno ga je parsirati. Za parsiranje se mogu koristiti predefinirani parseri ili oni ručno kreirani. U ovom slučaju koristi se ručno napravljen parser za parsiranje rezultata o postocima u listu.

```
private val parserOperatorsPercentage = rowParser { plus: Long?, minus: Long?, divide: Long?,
                                        multiple: Long? ->
    listOf(plus.toFiftyIfNull(), minus.toFiftyIfNull(), divide.toFiftyIfNull(),
            multiple.toFiftyIfNull())
}
```

Kod 72. Parser za pretvaranje prosjeka po operatoru u listu

8. Korištene biblioteke

8.1. Anko SQLite

Anko SQLite je kolekcija parsera i alata za lakše upravljanje i komuniciranje SQLite bazom podataka kreirana od strane JetBrains-a. Anko značajno olakšava pristup podacima iz baze, smanjuje mogućnost pojavljivanja grešaka i povećava čitljivost i lakoću održavanja koda.

Da bi se koristila ova biblioteka, potrebno ju je dodati u gradle.build datoteku. Prilikom korištenja SQLiteOpenHelper klase, obično se poziva metoda getReadableDatabase() ili getWritableDatabase() (rezultat je zapravo isti u proizvodnom kodu), ali onda je potrebno pozvati metodu close() na trenutno otvorenoj bazi. Osim toga, mora se negdje pohraniti pomoćna klasa. Također, prilikom korištenja paralelizma potrebno je voditi brigu o svim procesima. Sve je to prilično teško i zato razvojni programeri Androida nisu zainteresirani za standardni SQLite API i preferiraju upotrebu prilično zahtjevnih omotača kao što su ORM-ovi. Anko SQLite zamjenjuje klasični SQLite API i rješava gore navedene probleme. Umjesto korištenja try-catch blokova, baza se putem Anko SQLite biblioteke može pozvati iz bilo koje aktivnosti na vrlo jednostavan način.

```
database.use {
    // `this` is a SQLiteDatabase instance
}
```

Kod 73. Pozivanje instance SQLite baze

Anko biblioteka omogućuje jednostavne asinkrone pozive te se brine za bilo kakve moguće iznimke.

```
class SomeActivity : Activity() {
    private fun loadAsync() {
        async(UI) {
            val result = bg {
                database.use { ... }
            }
            loadComplete(result)
        }
    }
}
```

Kod 74. Asinkroni poziv instance baze

Anko omogućuje jednostavno kreiranje ili brisanje tablica unutar baze. U SQLiteu postoje pet glavnih vrsta: NULL, INTEGER, REAL, TEXT i BLOB. No, svaki stupac može imati neke modifikatore poput PRIMARY KEY ili UNIQUE. Ti se modifikatori pomoću Anko biblioteke mogu jednostavno dodati na vrstu bilo koje kolone.

```
database.use {
    createTable("Customer", true,
            "id" to INTEGER + PRIMARY_KEY + UNIQUE,
            "name" to TEXT,
            "photo" to BLOB)
}
```

Kod 75. Kreiranje tablice

Ubacivanje podataka u tablicu se klasično izvršava pomoću ContentValues klase. Anko omogućuje direktno ubacivanje podataka na najjednostavniji način.

```
database.use {
    insert("User",
            "id" to 42,
            "name" to "John",
            "email" to "user@domain.org"
    )}
```

Kod 76. Ubacivanje retka u tablicu

Za dohvaćanje podataka Anko koristi princip građenja upita sa svojim metodama.

Tablica 1. Popis Anko metoda za dohvaćanje podataka

Metoda	**Opis**
column(String)	Dodavanje kolone u upit
distinct(Boolean)	Izdvajanje upita
whereArgs(String)	Specifikacija „where“ uvjeta
whereArgs(String, args) :star:	Specifikacija „where“ uvjeta s argumentima
whereSimple(String, args)	Specifikacija „where“ uvjeta s nepoznanicama
orderBy(String, [ASC/DESC])	Poredak po koloni
groupBy(String)	Grupiranje po koloni
limit(count: Int)	Ograničenje broja redaka
limit(offset: Int, count: Int)	Ograničenje broja redaka nakon prvih n redaka
having(String)	Specifikacija „having“ izraza
having(String, args)	Specifikacija „having“ izraza s argumentima

Prilikom ovakvog načina dohvaćanja podataka, znatno se smanjuje mogućnost greške, a većina okruženja za razvoj predlaže naredbe pa to znatno ubrzava pisanje koda.

```
db.select("User", "name")
      .whereArgs("(_id > {userId}) and (name = {userName})",
            "userName" to "John",
            "userId" to 42)
```

Kod 77. Primjer dohvaćanja korisnika iz baze

8.2. MVEL

MVEL je biblioteka koja se koristi za evaluaciju izraza napisanih pomoću Java sintakse. Za razliku od Jave, MVEL je dinamički pisan, što znači da definicija tipova nije potrebna. U ovoj aplikaciji MVEL se koristio isključivo za evaluiranje matematičkih jednadžbi koje su, zbog fleksibilnosti, bile generirane u obliku stringa te će s toga biti objašnjen samo dio koji se koristio u te svrhe.

```
val evaluated = MVEL.evalToString(equationStr)
if (evaluated.toDouble().toInt() == number.toInt()) {
   rightAns = number
   return true
```

Kod 78. Primjer evaluacije izraza

U ovom slučaju se metodi evalToString predaje jednadžba u obliku stringa kao argument koja također vraća reprezentaciju broja u stringu. Kasnije se taj string parsira u double pa zatim u integer (da se dobije cijeli broj) te se uspoređuje s ispravnim rješenjem.

9. Korišteni alati

Za pisanje koda aplikacije korišten je Android Studio koji od verzije 3.0 službeno podržava programski jezik Kotlin te ima potpunu podršku za njega. Automatsko generiranje koda, dovršavanje naredbi i provjera sintaksnih pogrešaka značajno olakšavaju i ubrzavaju razvoj u Kotlinu.

Za provjeru podataka unutar baze korišten je DB browser for SQLite. To je jednostavan alat otvorenog koda unutar kojeg se mogu pregledavati, brisati i dodavati podaci SQLite baza podataka. Za izvoz baza podataka s uređaja korištena je File Explorer opcija Android Studia.

Za ručno podešavanje i stiliziranje fontova korišten je FontForge, također alat otvorenog koda.

10. Zaključak

U ovom radu detaljno su objašnjeni principi razvoja aplikacije za učenje matematike. Iako je ovaj projekt u početku bio zamišljen kao jednostavan generator matematičkih zadataka, uz malo modifikacija od njega je napravljena vrlo zanimljiva igra koja može privući mnoštvo osnovnoškolaca i potaknuti ih na učenje matematike. Aplikacija ima dobru podlogu za daljnji razvoj i proširenje namjene za stariju publiku. Uz dodavanje novih tipova zadataka, intuitivno generiranje jednadžbi i promoviranje, aplikacija može postići globalan utjecaj.

Cilj aplikacije je bio da ostane lokalna i samoodrživa, odnosno da ne ovisi o bilo kakvom serveru. Programski kod aplikacije pisan je tako da bude lako održiv i podložan za daljnji razvoj. U radu je također vrlo detaljno opisan programski jezik Kotlin te su istaknute njegove prednosti nad Javom koje su prikazane isječcima koda.

_____

Tomislav Brabec

11. Literatura

[1] Wikipedia: Java (software platform)
https://en.wikipedia.org/wiki/Java_(software_platform) (22.5.2018.)

[2] Kotlinlang: Comparison to Java Programming Language
https://kotlinlang.org/docs/reference/comparison-to-java.html (27.5.2018.)

[3] Wikipedia: Android Studio
https://en.wikipedia.org/wiki/Android_Studio (10.6.2018.)

[4] Github: Anko SQLite
https://github.com/Kotlin/anko/wiki/Anko-SQLite (15.6.2018.)

[5] Github: MVEL
https://github.com/imona/tutorial/wiki/MVEL-Guide (16.6.2018.)