Remi

Mađarević, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

REMI

Završni rad

Tomislav Mađarević

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 18.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Tomislav Mađarević
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4089, 28.07.2017.
OIB Pristupnika:	90122122593
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Remi
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	[Rezervirano: T. Mađarević] Napraviti android aplikaciju koja će omogućiti jednom korisniku igranje kartaške igre Remi. Korisnik dobiva 15 karti iz dva 'promiješana' špila uključujući određeni broj joker karti (prema postavkama). Omogućiti automatsko sortiranje karti, a isto tako i da korisnik može svoje karte premjestiti. Korisnik odbacuje jednu kartu i povlači iz špila novu, sve dok ne stekne uvjete za pobjedu.
Prijedlog ocjene završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 1 razina
Datum prijedloga ocjene od strane mentora:	18.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.09.2022.

Ime i prezime studenta:	Tomislav Mađarević
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4089, 28.07.2017.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Remi**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora ,

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

Igra remi nedvojbeno je kralj svih tradicionalnih igara . To je jedna od najpopularnijih kartaških igara svih vremena i ima mnogo različitih varijanti diljem svijeta. Iako igra ima mnogo verzija, osnove igre ostaju iste. Igra zahtijeva od svih igrača da poslože karte, podijeljene na početku igre, u potrebne kombinacije kako bi dali valjanu kombinaciju. Igrači moraju stalno osmišljavati strategiju kako bi odabrali i odbacili prave karte. S pojavom tehnologije, igra je sada dostupna za igranje online. Za remi se vjeruje da ima više podrijetala iz različitih geografskih područja, a prema najraširenijoj citiranoj teoriji, remi potječe iz Španjolske. Igra se kasnije proširila objema Amerikama nakon što su Španjolci započeli masovne migracije na područje južne Amerike u 19. stoljeću. Remi se originalno zvao eng.“conquian“ za što se vjeruje da je originalna verzija kartaške igre.[1]

Pri izradi aplikacije potrebna su znanja i vještine stečene na FERIT-u ,a naročito kolegiji:

- „Programiranje I“
- „Programiranje II“
- „Objektno orijentirano programiranje“
- „Razvoj programske podrške objektno orijentiranim načelima“
- „Algoritmi i strukture programiranja“
- „Osnove razvoja web i mobilnih aplikacija“

Struktura završnog rada je napravljena tako da nakon uvoda slijedi opis tehnologija korištenih u pri izradi aplikacije te nakon toga će biti opisana sama izrada aplikacije i njen izgled.

1.1. Zadatak završnog rada

Zadatak završnog rada je napraviti android aplikaciju koja će omogućiti jednom korisniku igranje kartaške igre Remi. Korisnik dobiva 15 karti iz dva 'promiješana' špila uključujući određeni broj džoker karti (prema postavkama). Omogućiti automatsko sortiranje karti, a isto tako i da korisnik može svoje karte premjestiti. Korisnik odbacuje jednu kartu i povlači iz špila novu, sve dok ne stekne uvjete za pobjedu.

2. TEORIJSKA PODLOGA

U ovom poglavlju bit će opisano integrirano razvojno okruženje(eng. Integrated Development Environment - IDE) , programski jezik i opisni jezik korišteni pri izradi zadane aplikacije i pravila igre remi sa svrhom lakšeg razumijevanja i sagledavanja ovog rada.

2.1. ANDROID STUDIO

Uz druga popularna integrirana razvojna okruženja ili skraćeno IDE (engl. „Integrated development environment“) kao što su: Visual Studio, Visual Studio code, IntelliJ IDEA, Eclipse , PCharm , Android Studio zasigurno drži svoje mjesto na listi popularnosti.

Android Studio je IDE kojim se programeri služe pri izradi raznih Android aplikacija, a temeljen je na IntelliJ IDEA za JVM (engl. „Java virtual machine“), koji omogućuje pokretanje Java klasa i datoteka na bilo kojem uređaju i operacijskom sustavu uz opciju korištenja programerskih alata kao predlošci koda, emulator, Gradle , Github integraciju uz pomoć koje programeri mogu odvojeno slati i primati kodove svojih kolega te zajedno raditi na složenim projektima .

2.1.1. ANDROID ARHITEKTURA

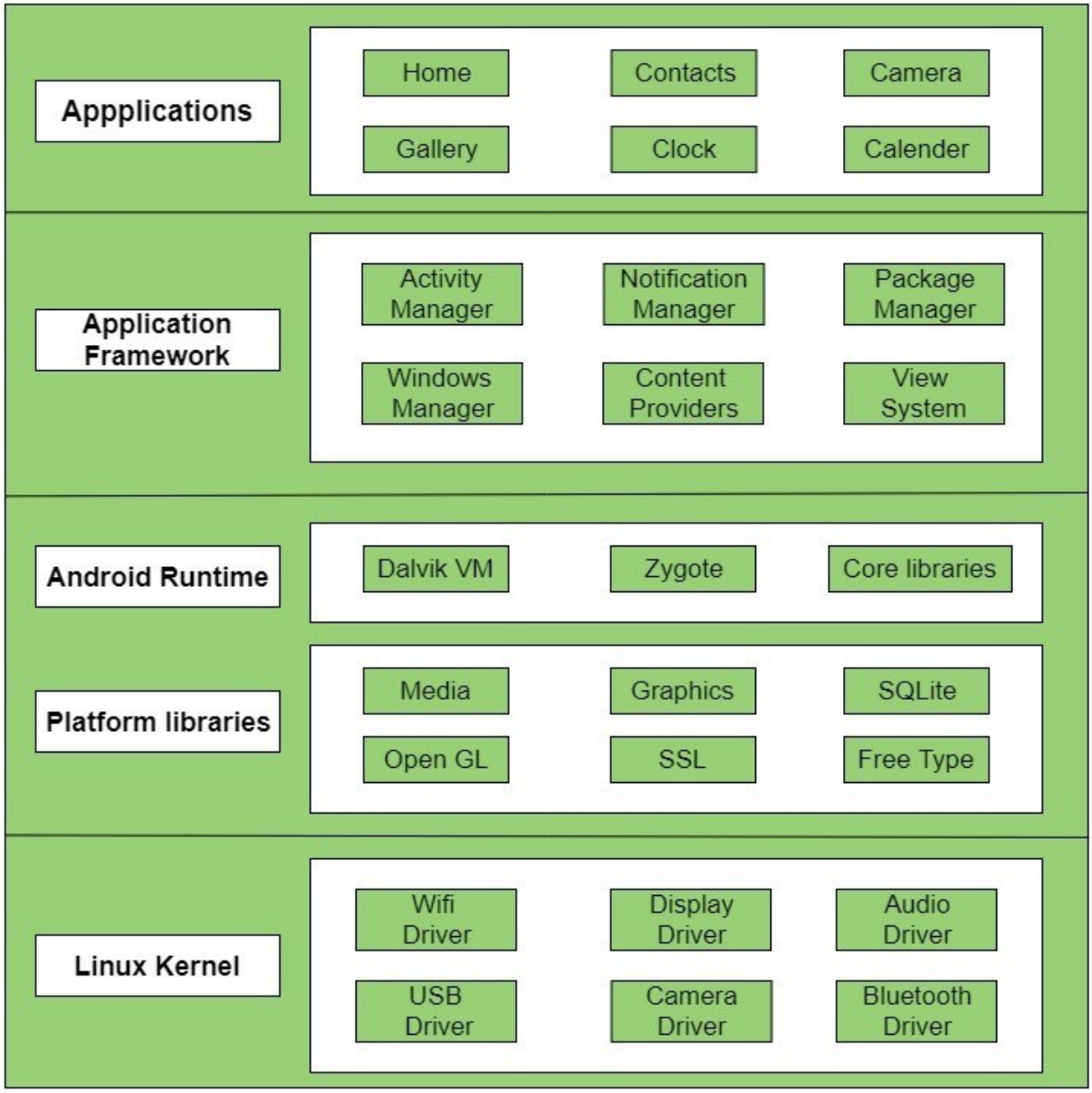
Android arhitektura sadrži različit broj komponenti za podršku svim potrebama Android uređaja. Android softver sadrži open-source Linux Kernel koji ima zbirku brojnih C/C++ biblioteka koje su izložene kroz usluge frameworka aplikacije.[3]

Među svim komponentama Linux kernel pruža glavnu funkcionalnost funkcija operativnog sustava za pametne telefone, a Dalvik Virtual Machine (DVM) pruža platformu za pokretanje android aplikacije.[3]

Glavne komponente android arhitekture su sljedeće[3]:

- **Aplikacijski sloj**- tvornički predinstalirane aplikacije kao što su kamera, poruke, galerija , kontakti itd. nalaze se u ovom sloju i aplikacije preuzete iz trgovine play kao viber, whatsapp i dr. bit će instalirane samo na ovom sloju. Ovaj sloj je zapravo najviši sloj android arhitekture.

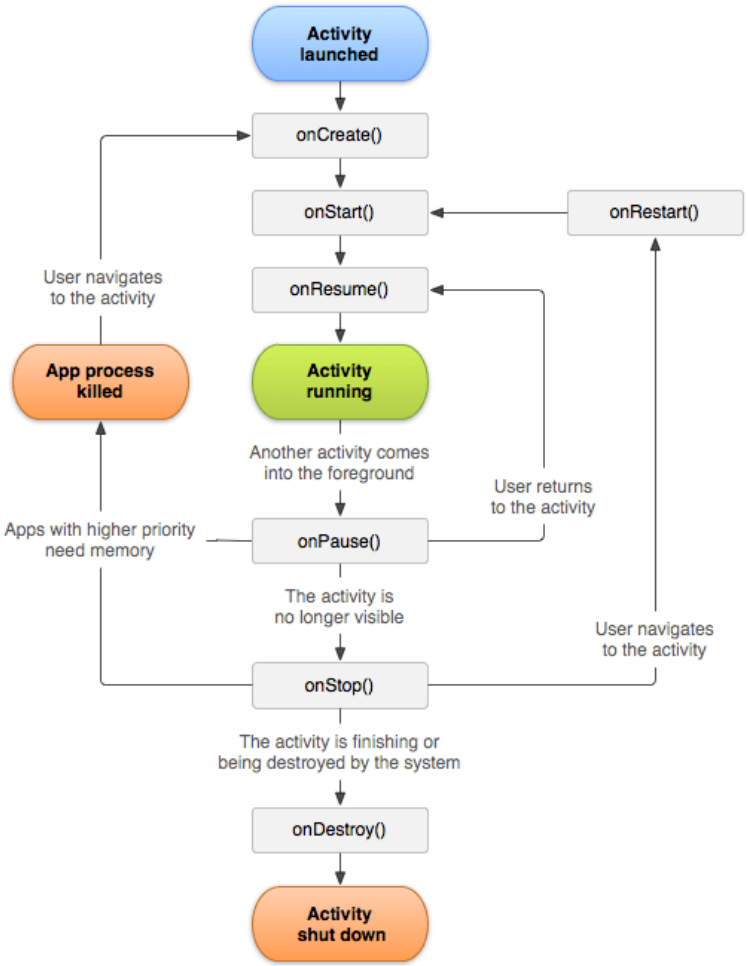
- **Aplikacijski okvir (framework)** – softverska knjižnica koja pomoću apstrakcije omogućuje pristupanje hardveru u određenom okruženju. Za komunikaciju aplikacija s hardverom veoma su bitne njegove klase.
- **Android runtime** – struktuiran je od osnovnih biblioteka i Dalvik Virtualni Stroj (eng. Dalvik Virtual Machine – DVK). Postavlja temelj aplikacijskom okviru (eng. Framework) i pokreće aplikaciju uz pomoć osnovnih knjižnica (engl. Libraries).
- **Android datoteke** - uključuju različite osnovne biblioteke C/C++ i biblioteke temeljene na Javi kao što su mediji, grafika, površinski upravitelj, OpenGL itd. za pružanje podrške za Android razvoj.
- **Linux jezgra** (engl. **Kernel**) – Upravlja svim raspoloživim upravljačkim programima (engl. „drivers“) koji su zaduženi za: kameru , zaslon , audio , memoriju, bluetooth itd. On je zapravo srce android arhitekture jer osigurava sloj apstrakcije između hardvera i ostalih komponenti android arhitekture.



Slika 2.1. Arhitektura Android Operacijskog Sustava

2.1.2. ACTIVITY

Klasa aktivnosti ključna je komponenta Android aplikacije, a način na koji se aktivnosti pokreću i spajaju temeljni je dio aplikacijskog modela platforme. Activity je klasa u čijoj instanci Android pokreće kod pozivanjem specifičnih metoda povratnog poziva koje odgovaraju određenim fazama njegovog životnog ciklusa dok se u nekim drugim programima aplikacije pokreću metodom main() što čini nekakvu već programsku paradigmu.[11]



Slika 2.2. Prikaz „zivotnog ciklusa“ jednog activitya[12]

2.1.3. RecyclerView

Odgovor na problem učinkovitog prikazivanja velikih skupova podataka dao je RecyclerView. Programer predaje podatke i definira izgled svake stavke, dok RecyclerView po potrebi dinamički kreira elemente neke android aplikacije. Kao što naziv implicira, RecyclerView reciklira te pojedinačne elemente. Kada se element ili tzv. „widget“ pomakne sa zaslona, RecyclerView ne uništava njegov prikaz ,umjesto toga, on ponovno koristi prikaz za nove stavke koje su se pomicale na zaslonu. Ponovna upotreba elemenata značajno pridonosi poboljšanju izvedbe, unapređujući „runtime“ aplikacije i smanjujući potrošnju energije.[8]

RecyclerView je nastao kao proširenje za ListView komponentu i on delegira posao LayoutManageru pa je rezultat toga mogućnost pomicanja „itema“ koji čine sadržaj tog RecyclerView-a vertikalno i horizontalno. RecyclerView zahtjeva korištenje ViewHolder oblikovanog obrasca koji omogućuje manje poziva findViewById metodi nad View objektima što

dodatno ubrzava rad. Za razliku od ListViewa, RecyclerView omogućuje bolju interakciju s elementima iz liste, animaciju pojedinih elemenata, prilagodbu razmaka između elemenata i slično. Njegovo postavljanje je složenije u odnosu na ListView. [9]

2.1.4. FRAGMENTI

Fragment predstavlja višekratni dio korisničkog sučelja aplikacije. Fragment sam definira i kontrolira vlastiti izgled, posjeduje vlastiti životni ciklus i može rukovati vlastitim ulaznim događajima. Fragmenti su uvijek „domaćini“ neke druge aktivnosti ili drugog fragmenta jer ne mogu živjeti sami za sebe. Hijerarhija prikaza fragmenta postaje dio hijerarhije pogleda glavnog računala ili se na nju pripaja.[10]

2.2. KOTLIN

Od svih dostupnih i popularnih opcija programskog jezika, Kotlin je jedan od najmlađih. Ipak, u posljednjih nekoliko godina njegova je popularnost značajno porasla. Nakon što ga je Google proglasio službenim jezikom razvoja Androida, sve više kompanija ga je počelo razmatrati za svoje projekte. Danas biti Kotlin programer znači biti konkurentan stručnjak na tržištu rada.[4]

Kotlin je open-source programski jezik čija povijest započinje 2016. godine. Jezik je kreiran od strane JetBrains-a, kompanije koja radi na tome da Kotlin postane glavni programski jezik kako za android tako i za iOS.[4]

Programski jezik Kotlin radi na Java Virtual Machine (JVM), što ga čini izravnim konkurentom poznatijoj i zrelijoj Javi s više od 20 godina povijesti. Oba se jezika mogu koristiti u istim sferama, uključujući razvoj poslužitelja, klijenta, weba i Androida. Kombinacija moćnih značajki s čistim kodom privlači programere iz različitih industrija da svoju pozornost usmjere na Kotlin.[4]

2019. godine dogodila se prekretnica u razvoju Kotlina. Google ga je proglasio preferiranim programskim jezikom za razvoj Android aplikacija, što mu je podiglo status u očima mnogih.[4]

Kotlin je programski jezik opće namjene, što znači da se može primijeniti u različitim sferama. Obično se Kotlin koristi za višeplatformski mobilni, Android, JavaScript i razvoj na strani poslužitelja.[4]

2.2.1. USPOREDBA KOTLIN I JAVA PROGRAMSKIH JEZIKA

Mnogo je godina Java bila jedini programski jezik za Android i razvoj na strani poslužitelja. Međutim, usponom Kotlina promijenio se njegov dominantni položaj. U to vrijeme počelo je suprotstavljanje Kotlina i Jave koje do danas nije riješeno.[4]

Za razliku od jave kotlin ima[5]:

- Lambda izraze i ugrađene funkcije
- Funkcije proširenja
- Null sigurnost
- Smart Cast
- String templates
- Svojstva
- Primarne konstruktore
- Delegaciju prve klase
- Zaključianje tipa za tipove varijabli i svojstava
- Singletone
- Varijancu deklaracije i projekciju tipa
- Izraze raspona
- Operator overloading
- Companion objekte
- Data klase
- Odvojena sučelja za read-only i promjenjive zbirke
- Korutine
- Casting

Općenito, Kotlin je sigurniji zahvaljujući null sigurnosti. Fleksibilniji je i sažetiji te omogućuje razvoj složenih rješenja s manje redaka koda. Ovi aspekti smanjuju mogućnost pojavljivanja bugova i grešaka u procesu razvoja softvera.

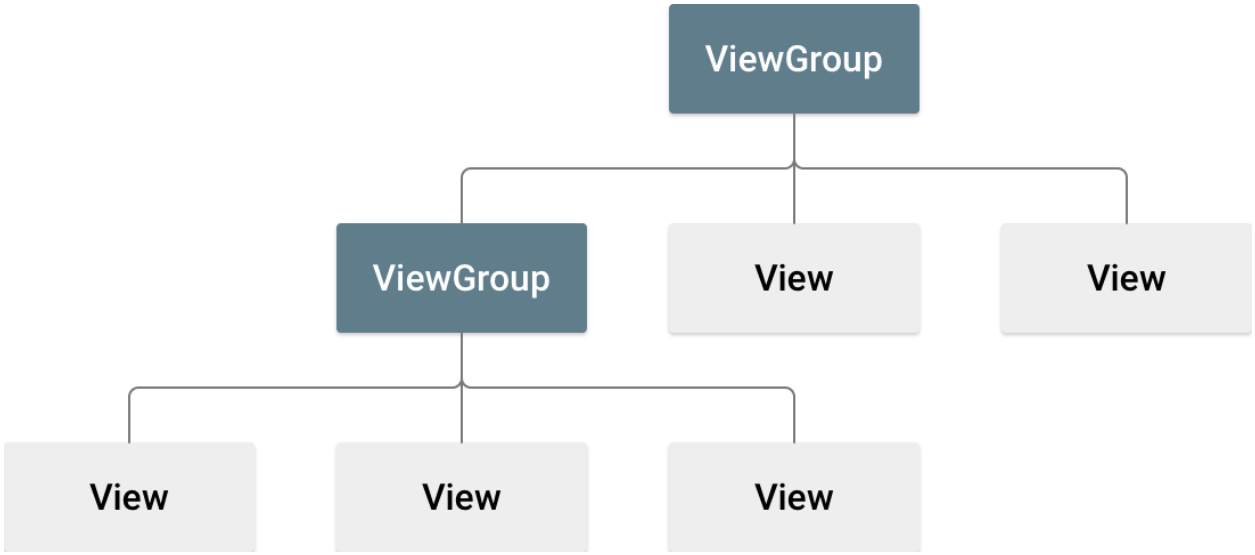
2.3. XML

Extensible Markup Language(XML) uz: HTML, KML, MathML, SGML, XHTML pripada skupini opisnih (engl. „Markup“) jezika. XML ima svrhu definiranja skupa pravila uz pomoć kojih je omogućeno formatiranje dokumenata u format koji je čitljiv ljudima i strojevima. Jednostavnost, općenitost i upotrebljivost na internetu čine jedne od osnovnih ciljeva dizajna XML-a. Uz dokumente dizajn XML-a široko se koristi za predstavljanje proizvoljnih struktura podataka kao npr. u web uslugama.[6]

- XML je označni jezik poput HTML-a
- XML je dizajniran za pohranu i prijenos podataka

- XML je dizajniran da bude samoopisni

XML opisni jezik se koristi unutar layout foldera android aplikacije. Layout definira strukturu korisničkog sučelja aplikacije. Svi elementi unutar layout-a su izgrađeni koristeći hijerarhiju View i ViewGroup objekata. View obično preslikava element koji korisnik može vidjeti i imati interakciju.Svaki View će imati jedinstveni ID atribut da se može dohvatiti pomoću findViewById(Int) funkcije unutar bilo koje kotlin klase. Dok je ViewGroup nevidljivi spremnik koji definira strukturu izgleda za View i druge ViewGroup objekte, kao što je prikazano na slici 2.3.[7]



Slika 2.3. Ilustracija hijerarhije View-a koji definira izgled korisničkog sučelja

Objekti View obično se nazivaju "widgeti" i mogu biti jedna od mnogih podklasa, kao što su Button ili TextView. Objekti ViewGroup obično se nazivaju "Layouti" i mogu biti jedan od mnogih tipova koji pružaju drugačiju strukturu izgleda, kao što je LinearLayout ili ConstraintLayout. U ovom radu korišten je ConstraintLayout.

2.4. PRAVILA IGRE REMI

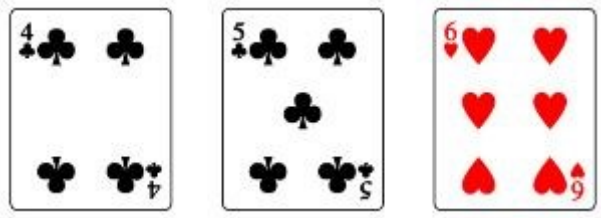
U ovoj aplikaciji korisnik igra sam sa sobom remi pa su u odnosu na ostale varijacije remija sa 2,3,4+ igrača isto tako i ova nebitno malo razlikuje, no cilj igre i dalje ostaje isti, a to je da korisnik mora isprazniti vlastiti špil karata sa odgovarajućim zadovoljavajućim kombinacijama da bi pobijedio.

Da bi korisnik mogao izbaciti jednu kombinaciju, zadana kombinacija mora zadovoljavati jedan od dva uvjeta:

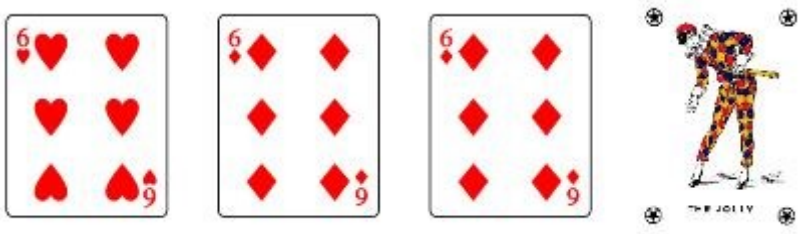
1. Karte koje čine kombinaciju kada se uzlazno ili silazno poredaju po njihovoj brojčanoj vrijednosti (2=2,3=3, … ,J=11 , Q=12, K=13, A=14 ili 1) čine niz gdje se svaka brojčana vrijednost sljedeće karte razlikuje od prethodne za jedan uz uvjet da su sve karte zadane kombinacije u istom znaku, npr. u srcetu i da je minimalni broj karata koje čine zadanu kombinaciju 3.
2. Karte koje čine kombinaciju su istih brojčanih vrijednosti , ali različitih znakova i broj karata koje čine kombinaciju nije manji od 3 ili veći od 4.

Joker karte mogu sačinjavati kombinacije navedene pod 1) i 2) i one mijenjaju bilo koju kartu potrebnu da bi kombinacija bila zadovoljavajuća.

Primjer **nezadovoljavajućih** kombinacija:



Slika 2.3. Kombinacija ne zadovoljava uvjet pod 1) jer nisu sve karte u istom znaku

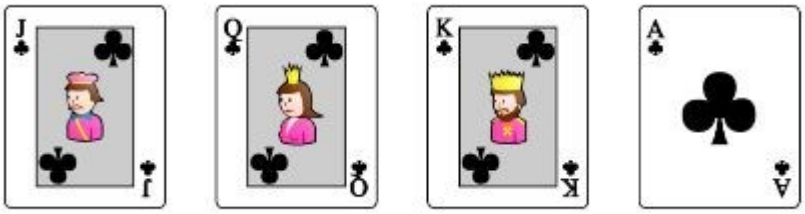


Slika 2.4. Kombinacija ne zadovoljava uvjet pod 2) jer su 2 karte istog znaka

Primjer **zadovoljavajućih** kombinacija:

Slika 2.5. kombinacija zadovoljava uvjet pod 2)

Slika 2.6. kombinacija zadovoljava uvjet pod 1) ,ovdje AS ima vrijednost 1



Slika 2.7. kombinacija zadovoljava uvjet pod 1), ovdje AS ima vrijednost 14

3. IZRADA ANDROID APLIKACIJE ZA KARTAŠKU IGRU REMI

Za izradu aplikacije korišten je android studio, u ovom odjeljku će biti prikazane klase i objašnjeno njihovo međudjelovanje unutar java foldera te prikazani „Layout-i“ unutar „res“ foldera sa pripadajućim XML-om.

3.1. UVOD U REMI

Svaka aplikacija unutar android studija sastoji se od **manifest, java** , „**res**“ foldera i „**gradle script**-a“.

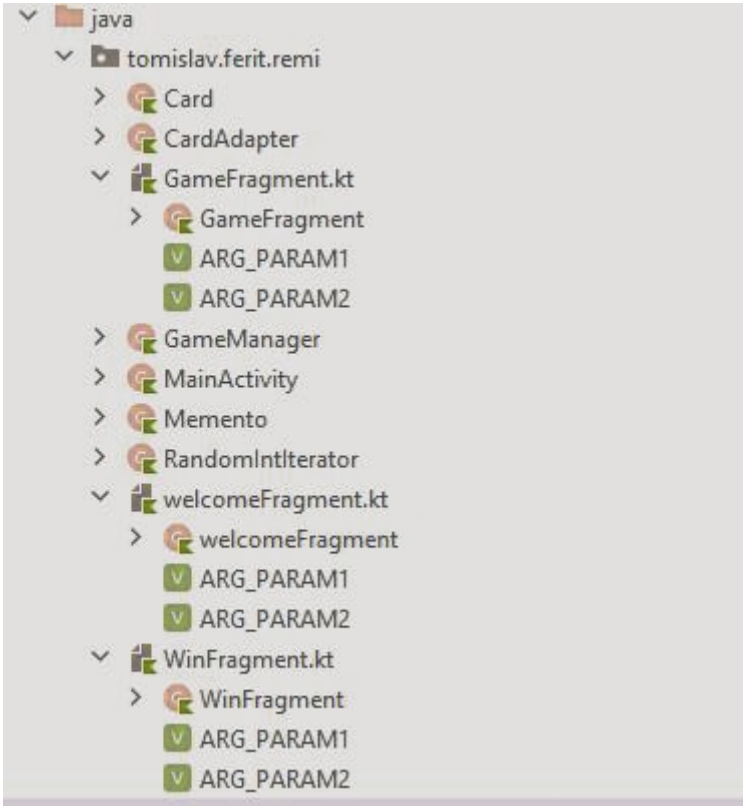
- **manifest** - opisuje bitne informacije o aplikaciji alatima za izradu Androida, operativnom sustavu Android i Google Playu.

- **java**- Mapa Java sadrži sve datoteke java i Kotlin izvornog koda (.java) koje stvaramo tijekom razvoja aplikacije, uključujući druge testne datoteke. Ako kreiramo bilo koji novi projekt koristeći Kotlin, prema zadanim postavkama datoteka klase MainActivity. kt automatski će se stvoriti pod nazivom paketa "com."
- **res** - koristi se za pohranu vrijednosti za resurse koji se koriste u mnogim Android projektima za uključivanje značajki boja, stilova, dimenzija itd.
- **gradle script**- Android Studio koristi Gradle, napredni skup alata za izradu, za automatizaciju i upravljanje procesom izrade, dok omogućuje definiranje fleksibilnih prilagođenih konfiguracija izrade. Svaka build konfiguracija može definirati vlastiti skup koda i resursa, dok ponovno koristi dijelove koji su zajednički svim verzijama vaše aplikacije.

3.2. APLIKACIJA REMI

Unutar java foldera aplikacije nalaze se klase:

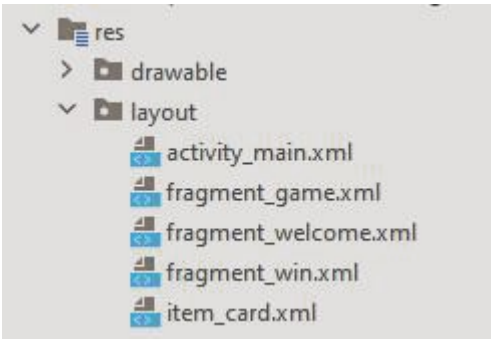
- MainActivity
- welcomeFragment
- GameFragment
- WinFragment
- Card
- CardAdapter
- GameManager
- Memento
- RandomIterator ,prikazane na slici 3.1.



Slika 3.1. Prikaz kotlin klasa unutar java foldera

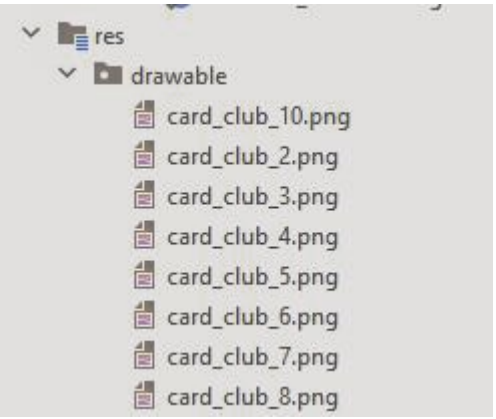
Unutar „res/layout“ foldera nalaze se XML datoteke:

- activity_main
- fragment_welcome
- fragment_game
- fragment_win
- item_card , prikazane na slici 3.2.



Slika 3.2. Prikaz XML datoteka unutar layout foldera

Unutar res/drawable foldera se nalaze „.png“ datoteke, odnosno slike karata koje će biti korištene u aplikaciji.



Slika 3.3. Prikaz drawable foldera

Unutar values foldera se nalaze folderi strings.xml i colors.xml unutar kojih su spremljeni podaci kasnije korišteni u našoj aplikaciji.

strings.xml

Edit translations for all locales in the translations editor.

```
<resources>
    <string name="app_name">REMI</string>
    <!-- TODO: Remove or change this placeholder text -->
    <string name="hello_blank_fragment">Hello blank fragment</string>
    <string name="RUMMY">RUMMY</string>
    <string name="start">START</string>
    <string name="card">Card</string>
    <string name="swap">SWAP</string>
    <string name="hello_guys">Hello guys</string>
    <string name="number_of_joker_cards">Number Of Joker Cards:</string>
    <string name="replace">Replace</string>
    <string name="check">Check</string>
    <string name="undo">UNDO</string>
    <string name="mark_unmark">MARK/UNMARK</string>
    <string name="sort">SORT</string>
    <string name="you_won">YOU WON</string>
    <string name="play_again">PLAY AGAIN</string>
    <string-array name="Desired_Number_Of_Joker_Cards">
        <item>0</item>
        <item>1</item>
        <item>2</item>
        <item>3</item>
        <item>4</item>
        <item>5</item>
        <item>6</item>
        <item>7</item>
        <item>8</item>
        <item>9</item>
        <item>10</item>

    </string-array>
```

Slika 3.4. prikaz string.xml datoteke

Main activity je prvo što se prikazuje na ekranu android aplikacije i u aplikaciji Remi on izgleda kao na slici 3.5.:

Slika 3.5. prikaz MainActivity-a

Funkcija onCreate služi za inicijaliziranje MainActivitya i ona se poziva kada activity starta.

Unutar onCreate funckije većinu inicijalizacije treba implementirati (npr. pozivanje setContentView za napuhavanje korisničkog sučelja activity-a , korištenje findViewById funkcije za programsku interakciju s widgetima itd.)

U ovoj aplikaciji, funkcija setContentView napuhava layout pod imenom activity_main koji izgleda kao na slici 3.6.

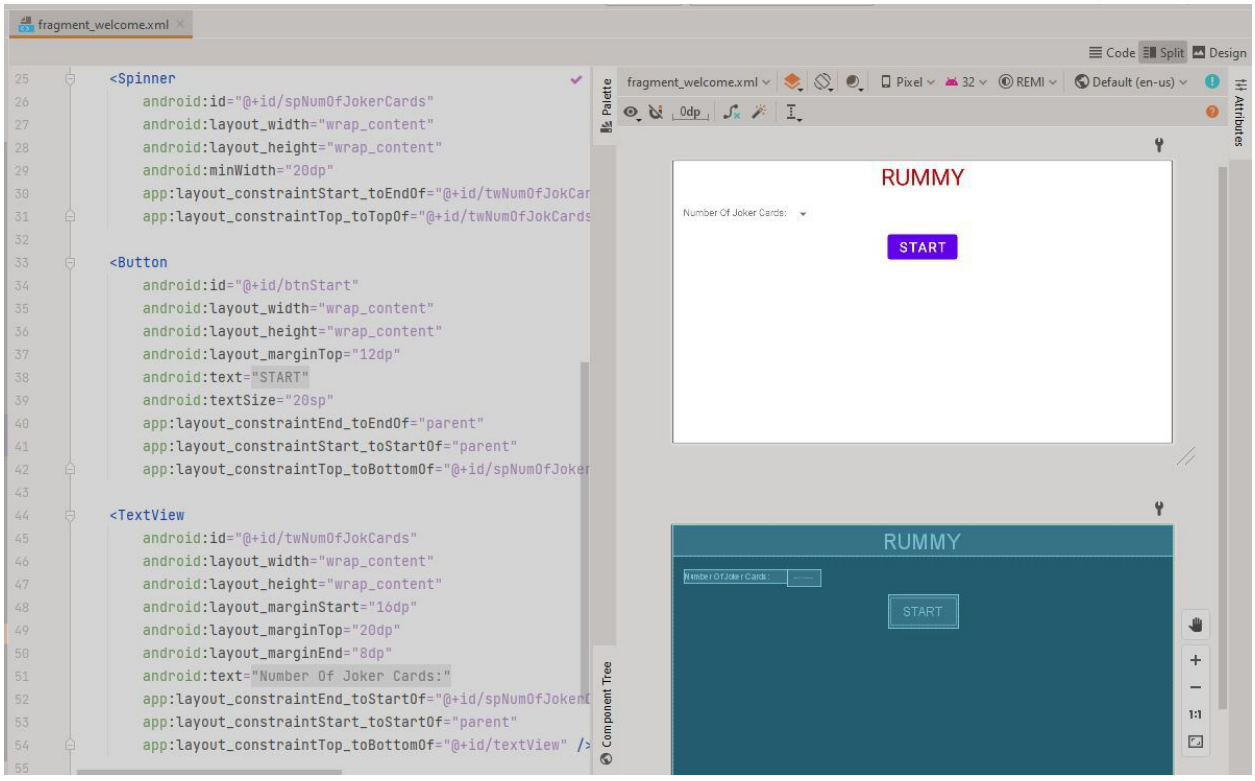
supportFragmentManager metoda je klase FragmentManager koja je odgovorna za izvođenje radnji nad fragmentima aplikacije, kao što je njihovo dodavanje, uklanjanje ili zamjena te njihovo dodavanje na stog. U ovoj aplikaciji metoda vrši prikaz welcomeFragment() fragmenta unutar FrameLayout elementa activity_main.xml foldera prikazanog na slici 3.6. te dodavanje tog fragmenta na stog što će omogućiti kasnije vraćanje na taj isti fragment aplikacije.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <FrameLayout
        android:id="@+id/flMainActivity"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 3.6. prikaz activity_main layouta

Unutar fragment_welcome layouta nalazi se Spinner element pomoću kojeg korisnik klikom može izabrati jedan od ponuđenih itema tog spinnera. U ovoj aplikaciji spinner omogućuje korisniku da izabere željeni broj Joker karti (od 0 do 10) koji će biti ubačeni u igrin špil od 104 karte.

```xml
<Spinner
    android:id="@+id/spNumOfJokerCards"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="20dp"
    app:layout_constraintStart_toEndOf="@+id/twNumOfJokCar
    app:layout_constraintTop_toTopOf="@+id/twNumOfJokCards

<Button
    android:id="@+id/btnStart"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="12dp"
    android:text="START"
    android:textSize="20sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/spNumOfJoker

<TextView
    android:id="@+id/twNumOfJokCards"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="20dp"
    android:layout_marginEnd="8dp"
    android:text="Number Of Joker Cards:"
    app:layout_constraintEnd_toStartOf="@+id/spNumOfJoker
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView" />
```



Slika 3.7. fragment_welcome layout

welcomeFragment.kt

```
65      override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
66          super.onViewCreated(view, savedInstanceState)
67
68          val spNumOfJokerCards: Spinner = view.findViewById(R.id.spNumOfJokerCards)
69          val btnStart=view.findViewById<Button>(R.id.btnStart)
70      // Create an ArrayAdapter using the string array and a default spinner layout
71          val adapter =  ArrayAdapter.createFromResource(
72              view.context,
73              R.array.Desired_Number_Of_Joker_Cards,
74              android.R.layout.simple_spinner_item
75          )
76          adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
77          spNumOfJokerCards.adapter = adapter
```

Slika 3.8. welcomeFragment() klasa dio 1.

onViewCreated metoda garantira nam da je View kreiran pa tako izbjegavamo moguće probleme pri korištenju findViewById funkcija koje koristimo da bismo pristupili i djelovali na određene widgete određenog layouta kao što su buttoni, spinneri, textView-ovi itd.

Na slici 3.8. prikazano je kako pomoću adaptera podešavamo funkcionalnosti spinera „spNumOfJokerCards“. (podešavamo mjesto prikaza, iteme koji će se prikazivati u spineru i vrstu spinnera.)

```
spNumOfJokerCards.onItemSelectedListener= object : AdapterView.OnItemSelectedListener{

    override fun onItemSelected(parent: AdapterView<*>, view: View?, position: Int, id: Long) {
        Toast.makeText(parent.context,
            text: "Your desired number of Joker cards is: ${parent.getItemAtPosition(position).toString()}",
            Toast.LENGTH_SHORT).show()
        val input=parent.getItemAtPosition(position).toString().toInt()

        setFragmentResult( requestKey: "requestKey", bundleOf( ...pairs: "NumberOfJokerCards" to input))
    }

    override fun onNothingSelected(p0: AdapterView<*>?) {
        TODO( reason: "Not yet implemented")
    }
}

btnStart.setOnClickListener { it: View!

    parentFragmentManager.beginTransaction().apply { this: FragmentTransaction
        replace(R.id.flMainActivity,GameFragment())
        addToBackStack( name: null)
        commit()
    }
}
```

Slika 3.9. welcomeFragment() klasa dio 2.

spNumOfJokerCards koristi apstraktnu metodu onItemSelectedListener zbog čega stvara anonimni objekt (jedna od funkcionalnosti kotlina koja omogućava korištenje apstraktnih metoda bez stvaranja klase koja nasljeđuje sučelje te metode) koji nasljeđuje sučelje AdapterView.OnItemSelectedListener te override-a apstraktnu metodu onItemSelected pomoću koje odabrani item spinnera (željeni broj Joker karti u špilu igre) šaljemo kao varijablu nazvanu „input“ bilo kojem fragmentu uz pomoć funkcije setFragmentResult koja ima svoj jedinstveni ključ (eng. Request Key) i bundle unutar kojeg šalje podatak koji će se kasnije dohvaćati.

btnStart je button widget welcomeFragment()-a i klikom na njega zamijenjuje se trenutni fragment u kojem se dugme nalazi (zato se i koristi parentFragmentManager umjesto supportFragmentManager funkcije) sa novim GameFragment()-om te se welcomeFragment() dodaje na stog.)

Prije prikaza sljedećeg fragmenta (GameFragment()-a) objasnit ćemo klase objekata koji su korišteni unutar tog fragmenta.

```
data class Card (
    var image: Int,
    var numberValue: Int,
    var suit: String,
    var isChecked: Boolean
){ }
```

Slika 3.10. prikaz Data class Card

Data class je jednostavna klasa podataka koja kako samo ime govori služi za držanje podataka i/ili stanja i ona sadrži standardnu funkcionalnost. U ovoj aplikaciji data class Card sadrži 4 podatka unutar svog primarnog konstruktora prikazana na slici 3.10., a to su:

1. **image : Int** – to je slika jedne od 53 moguće karte poker špila sa jokerom. Te slike će biti dohvaćane iz drawable datoteke.
2. **numberValue: Int** -to je brojčana vrijednost pojedine karte
3. **suit: String** -to je znak ili boja pojedine karte, imamo ih 4

4. **isChecked: Boolean** – iznad svake karte stoji widget zvani checkBox koji korisnik može klikom „označiti“ ili „odznačiti“, na slici 3.11. prikazano je izgled layouta koji se odnosi na ovu klasu Card.

```
<ImageView
    android:id="@+id/iwCardImage"
    android:layout_width="69dp"
    android:layout_height="106dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/cbCard"
    android:contentDescription="Card" />

<CheckBox
    android:id="@+id/cbCard"
    android:layout_width="48dp"
    android:layout_height="48dp"
    android:layout_marginTop="8dp"
    android:checked="false"
    app:layout_constraintEnd_toEndOf="@+id/iwCardImage"
    app:layout_constraintStart_toStartOf="@+id/iwCardImage"
    app:layout_constraintTop_toTopOf="parent" />
```

Slika 3.11. prikaz XML koda unutar layout foldera item_card

Da bi bilo moguće igračeve karte poredat horizontalno te ih premještat, odbacivat , razgledavat koristit će se recyclerView.

Da bi recyclerView bio korišten potrebno je implementirati adapter klasu koja nasljeđuje RecyclerView.Adapter te implementirati 3 metode zvane : onCreateViewHolder, onBindViewHolder i getItemCount.

Adapter koji ova aplikacija koristi zove se CardAdapter i njegov kod će biti prikazan i objašnjen u sljedećim slikama.

```
class CardAdapter(
     var Cards: MutableList<Card>
):RecyclerView.Adapter<CardAdapter.CardViewHolder>()
{
    inner class CardViewHolder(itemView: View) :RecyclerView.ViewHolder(itemView)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CardViewHolder {
        val itemView=LayoutInflater.from(parent.context).inflate(R.layout.item_card,parent, attachToRoot: false)
        return CardViewHolder(itemView)
    }

    override fun onBindViewHolder(holder: CardViewHolder, position: Int) {
        val checkBox =holder.itemView.findViewById<CheckBox>(R.id.cbCard)
        val cardImage = holder.itemView.findViewById<ImageView>(R.id.iwCardImage).also { it: ImageView!
            it.setImageResource(Cards[holder.adapterPosition].image)
        }
        checkBox.setOnCheckedChangeListener(object : CompoundButton.OnCheckedChangeListener{
            override fun onCheckedChanged(compoundButton: CompoundButton?, bool: Boolean) {
                Cards[holder.adapterPosition].isChecked = checkBox.isChecked
            }
        })

        checkBox.isChecked=Cards[holder.adapterPosition].isChecked
        if(checkBox.isChecked)
            Toast.makeText(holder.itemView.rootView.context,
                text: "You checked card[ ${holder.adapterPosition} ]",
                Toast.LENGTH_SHORT).show()
        }

    override fun getItemCount(): Int {
        return Cards.size
    }
```

Slika 3.12. CardAdapter

Kao što se vidi na slici 3.12. CardAdapter sadrži samo jedan atribut zvani Cards koji će činiti promjenjivo polje objekata tipa Card.

Ugniježđena klasa označena kao „inner“ može pristupiti članovima svoje vanjske klase. Unutarnje klase nose referencu na objekt vanjske klase. U slučaju CardAdaptera unutarnja klasa je CardViewHolder i ona sadrži itemView koji u ovom slučaju je zapravo layout folder item_card prikazan na slici 3.11.

Metoda onCreateViewHolder se poziva kada recyclerView treba novi viewHolder (u ovom slučaju novu kartu).

Novi viewHolder će prikazivati iteme adaptera koristeći metodu onBindViewHolder.

U ovoj aplikaciji unutar onBindViewHolder referenciramo checkBox i imageView widgete koji su unutar spomenute XML datoteke item_card na slici 3.11. u varijable checkBox i

cardImage (slika 3.12.) s time da će se svaki resurs slike CardViewHoldera postaviti redom kako su karte poredane u polju Cards , a checkBox-u biti omogućeno označavanje pritiskom.

checkBox koristi setOnCheckedChangeListener koji za argument sadrži anonimni objekt apstraktne klase CompoundButton.OnCheckedChangeListener koji pomoću metode onCheckedChange omogućuje da pri svakom pritisku checkBox-a određene karte recyclerView zapamti to stanje i ne briše ga pri recikliranju.

getItemCount metoda vraća broj karata unutar adaptera tj. Veličinu polja atributa Cards.

Sljedeća klasa koja će biti prikazana je GameManager() i ona se koristi unutar GameFragmenta(), u njoj su zapravo kodirane sve funkcionalnosti igre remi, radi čistoće koda, koje će biti objašnjene u daljnjem tekstu.

Slika 3.13. GameManager

GameManager klasa sadrži atribute:

- **numberOfJokerCards** – broj Joker karti koji će se dodati u igrin špil (gameDeckOfCards)

- **context** – kontekst u kojem se game manager nalazi(u ovom slučaju će biti unutar GameFragment-a), potreban radi izbacivanja poruka zvanih Toast.
- **playerDeckOfCards**- igračev špil karata koji će mu biti nasumično dodijeljen iz igrinog špila karata(gameDeckOfCards)
- **gameDeckOfCards**- igrin špil karata koji se sastoji od 104 poker karte plus određeni broj Joker karata ovisno o korisnikovom odabiru.
- **savedStates**- promjenjiva lista tipa Memento koja sadrži redom odbačene kombinacije karata koje zadovoljavaju jedan od dva uvjeta da bi bile izbačene. Ovaj atribut će biti korišten ako korisnik poželi vratiti svoje odbačene kombinacije radi boljeg kmbiniranja s ostalim kartama.

, i metode:

- **createGameDeckOfCards()**- metoda koja vraća promjenjivu listu objekata tipa Card, tj. u igrin špil karata ubacuje dva seta poker karata od 52 karte sa odgovarajućim brojem Joker karata po korisničkim postavkama.
- **createPlayerDeckOfCards()-**metoda koja vraća promjenjivu listu objekata tipa Card, tj. ona iz igrinog špila karata (atribut gameDeckOfCards) nasumično povlači 15 karata i daje ih igraču.
- **discardCheckedCards(MutableList<Card>)**- metoda koja kao argument prima igračev špil karata i provjerava jel određene kombinacije karata zadovoljavaju jedan od dva uvjeta da bi bile izbacivane.(Kombinacijom se smatraju sve karte koje čiji checkboxovi su označeni)
- **isAce(Card)**- metoda koja vraća true ako karta predana u argumentu je as , u suprotnom vraća false. Ova metoda se koristi unutar metode isThreeOrMoreInARow kada se provjerava niz karata gdje je as karta najmanje brojčane vrijednosti.
- **isJoker(Card)**- metoda koja vraća true ako karta predana u argumentu je joker , u suprotnom vraća false.
- **isThreeOrFourOfSameNumAndDifSuit(MutableList<Card>)-** metoda koja vraća true ako igračeva kombinacija zadovoljava uvjet 2) naveden u odjeljku **2.4.PRAVILA IGRE REMI,** u suprotnom vraća false.
- **isThreeOrMoreInARow(MutableList<Card>)-** metoda koja vraća true ako igračeva kombinacija zadovoljava uvjet 1) naveden u odjeljku **2.4.PRAVILA IGRE REMI,** u suprotnom vraća false.

- **removeJokers(MutableList<Card>)-** metoda koja vraća listu objekata tipa Card samo bez Joker karata.
- **replace(MutableList<Card>)-** metoda koja izbacuje odabranu kartu iz igračevog špila i nasumično uzima drugu iz špila igre (gameDeckOfCards).
- **restoreFromMemento()-** metoda koja vraća zadnju izbačenu kombinaciju karata ,koja je zadovoljila jedan od 2 uvjeta za izbacivanje, nazad u igračev špil karata.
- **save (List<Card>)-** metoda koja sprema u atribut savedStates listu objekata tipa Card.
- **sortAllChecked(MutableList<Card>)-** metoda koja sortira sve karte označenih checkboxova po uzlaznoj veličini brojčane vrijednosti po njihovim respektabilnim bojama/znakovima.
- **swichTwoCheckedCards(MutableList<Card>)-** metoda koja omogućuje igraću da zamijeni mjesta dvaju označenih karata.
- **swapAndUncheck(MutableList<Card>,Int,Int)-** metoda koju koristi metoda swichTwoCheckedCards, navedena iznad, koja za argumente prima promjenjivu listu objekata tipa Card i dva argumenta tipa Int koji predstavljaju dvije željene pozicije koje bi igrač zamijenio.

U sljedećem dijelu rada biti će prikazani kodovi pojedinih metoda klase GameManager.

```
private fun createGameDeckOfCards() : MutableList<Card> {
    val gameDeckOfCards = mutableListOf<Card>(
        Card(R.drawable.card_club_2, numberValue: 2, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_3, numberValue: 3, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_4, numberValue: 4, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_5, numberValue: 5, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_6, numberValue: 6, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_7, numberValue: 7, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_8, numberValue: 8, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_9, numberValue: 9, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_10, numberValue: 10, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_j, numberValue: 11, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_q, numberValue: 12, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_k, numberValue: 13, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_club_a, numberValue: 14, suit: "CLUB", isChecked: false),
        Card(R.drawable.card_diamond_2, numberValue: 2, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_3, numberValue: 3, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_4, numberValue: 4, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_5, numberValue: 5, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_6, numberValue: 6, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_7, numberValue: 7, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_8, numberValue: 8, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_9, numberValue: 9, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_10, numberValue: 10, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_j, numberValue: 11, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_q, numberValue: 12, suit: "DIAMOND", isChecked: false),
        Card(R.drawable.card_diamond_k, numberValue: 13, suit: "DIAMOND", isChecked: false),
```

Slika 3.14. createGameDeckOfCards dio 1.

```
private fun createGameDeckOfCards() : MutableList<Card> {
    val gameDeckOfCards = mutableListOf<Card>(...)

    for(i in 0 until this.numberOfJokerCards){
        gameDeckOfCards.add(Card(R.drawable.joker, numberValue: 100, suit: "JOKER", isChecked: false))
    }

    return gameDeckOfCards

}
```

Slika 3.15. createGameDeckOfCards dio 2.

Korištenje objekta randomIterator nam omogućava da nasumične karte ne budu ponovo isto podijeljenje pri ponovom pokretanju aplikacije jer će seed unutar generatora nasumičnih brojeva biti drugačiji, slika 3.16. :

```
private fun createPlayerDeckOfCards(): MutableList<Card>{
    val playerDeckCards= mutableListOf<Card>()

    var seed = Random.nextInt()
    val randomIterator = RandomIntIterator( range: 0 until 9999)
    if(randomIterator.hasNext()){
         seed= randomIterator.next()
    }
    else{
        //renew if needed
        randomIterator.randomize()
    }
    for(i in 0..14){

        val randomPosition = Random(seed).nextInt( from: 0, until: this.gameDeckOfCards.size-1)
        playerDeckCards.add(this.gameDeckOfCards[randomPosition])
        this.gameDeckOfCards.removeAt(randomPosition)

    }
     return playerDeckCards
}
```

Slika 3.16. createPlayerDeckOfCards

```
fun switchTwoCheckedCards(oldPlayerDeckOfCards : MutableList<Card>): MutableList<Card>{

    val positions = arrayListOf<Int>()
    for(position in 0 until oldPlayerDeckOfCards.size){

        if(oldPlayerDeckOfCards[position].isChecked)
            positions.add(position)
    }

    for(i in 0 until positions.size)

        if(positions.size != 2) {
            Toast.makeText(
                context,
                text: "You checked ${positions.size} checkboxes instead of 2!",
                Toast.LENGTH_LONG
            ).show()

            return oldPlayerDeckOfCards
        }

        val newPlayerDeckOfCards= swapAndUncheck(oldPlayerDeckOfCards, positions[0], positions[1])
        return newPlayerDeckOfCards
}
```

Slika 3.17. swichTwoCheckedCards

```
private fun swapAndUncheck(
    listToSwap: MutableList<Card>,
    position1: Int,
    position2: Int
): MutableList<Card> {

    val tmp = listToSwap[position1]
    listToSwap[position1] = listToSwap[position2]
    listToSwap[position2] = tmp

    listToSwap[position1].isChecked=false
    listToSwap[position2].isChecked=false


    return listToSwap
}
```

Slika 3.18. swapAndUncheck

Metoda replace kao argument prima igračev špil karata te gleda koji checkbox igračeve karte je označen (mora biti samo jedan) te zatim iz tog špila izbacuje tu kartu i nasumično dodjeljuje drugu, slika 3.19.:

```
fun replace(PlayerDeckOfCards : MutableList<Card>): MutableList<Card>{

    val positions = arrayListOf<Int>()
    for(position in 0 until PlayerDeckOfCards.size){

        if(PlayerDeckOfCards[position].isChecked)
            positions.add(position)

    }
    if(positions.size != 1) {
        Toast.makeText(
            context,
            text: "You checked ${positions.size} checkboxes instead of 1!",
            Toast.LENGTH_LONG
        ).show()
        return PlayerDeckOfCards  //return the same as inputted

    }
    val randomPositionInGameDeck = Random.nextInt( from: 0, until: this.gameDeckOfCards.size-1)
    val newCard=this.gameDeckOfCards[randomPositionInGameDeck]
    gameDeckOfCards.removeAt(randomPositionInGameDeck)
    PlayerDeckOfCards.removeAt(positions[0])
    PlayerDeckOfCards.add(positions[0],newCard)
    return PlayerDeckOfCards
}
```

Slika 3.19. replace

```
    private fun save(discardedCards : List<Card>) {
        savedStates.add(Memento(discardedCards))
    }
    fun restoreFromMemento(): List<Card> {
        if(savedStates.isEmpty())
            throw (Exception("SavedStates are empty!"))
        return savedStates.removeLast().getDiscardedCards()

    }
```

Slika 3.20. save i restoreFromMemento

Metoda discardCheckedCards odbacuje kombinacije označenih karata koje zadovoljavaju uvjete i sprema ih u atribut savedStates pomoću metode save, slika 3.21.:

```
fun discardCheckedCards(playerCards :MutableList<Card>) : MutableList<Card>{

    val positions = arrayListOf<Int>()
    for(position in 0 until playerCards.size){
        if(playerCards[position].isChecked)
            positions.add(position)
    }
    val playerCardsToBeChecked = mutableListOf<Card>()
    for(position in 0 until positions.size){
        playerCardsToBeChecked.add(playerCards[positions[position]])
    }
    if(isThreeOrMoreInARow(playerCardsToBeChecked)){
        save(playerCardsToBeChecked)
        for(position in positions.size -1 downTo(0) ){
            playerCards.removeAt(positions[position])
        }
        Toast.makeText(this.context,
            text: "You sucssesfully removed your cards",
            Toast.LENGTH_LONG).show()
    }
    else if(isThreeOrFourOfSameNumAndDifSuit(playerCardsToBeChecked)){
        save(playerCardsToBeChecked)
        for(position in positions.size -1 downTo(0) ){
            playerCards.removeAt(positions[position])
        }
        Toast.makeText(this.context,
            text: "You sucssesfully removed your cards",
            Toast.LENGTH_LONG).show()
    }
    return playerCards
}
```

Slika 3.21. discardCheckedCards

```
private fun isThreeOrMoreInARow(cardsToBeCheckedForDiscarding : MutableList<Card>) : Boolean{

    val sortedCardsToBeCheckedForDiscarding = cardsToBeCheckedForDiscarding.sortedBy { it.numberValue }
    var possibleAceFlag= false
    var jokerCounter=0
    for(position in 0 until sortedCardsToBeCheckedForDiscarding.size){
        if(isJoker(sortedCardsToBeCheckedForDiscarding[position])) { //Ako je Joker karta
            jokerCounter++
        }
    }
    if(sortedCardsToBeCheckedForDiscarding.size<3){
        return false
    }
    for(position in 0 .. sortedCardsToBeCheckedForDiscarding.size -2){
        val card= sortedCardsToBeCheckedForDiscarding[position]
        val numberValue =card.numberValue
        val suit =card.suit
        if(card.numberValue==2){ possibleAceFlag=true}

        val nextCard=sortedCardsToBeCheckedForDiscarding[position+1]
        val nextNumberValue=nextCard.numberValue
        val nextSuit =nextCard.suit

        if(isJoker(card) || isJoker(nextCard)){
            break
        }
        else if(suit != nextSuit) {
            return false
        }
```

Slika 3.22. isThreeOrMoreInARow dio 1.

```
        else if( nextNumberValue-numberValue != 1){
            if(possibleAceFlag==true){
                if(isAce(nextCard)) {
                    continue
                }
            }
            if(jokerCounter>0){
                jokerCounter--
                continue
            }
            return false
        }
    }
    return true
}
```

Slika 3.23. isThreeOrMoreInARow dio 2.

```
private fun isThreeOrFourOfSameNumAndDifSuit(cardsToBeCheckedForDiscarding : MutableList<Card>) : Boolean {

    if(cardsToBeCheckedForDiscarding.size<3 || cardsToBeCheckedForDiscarding.size>4){
        return false
    }
    val copyOfCardsToBeCheckedForDiscarding= mutableListOf<Card>()
    for(position in 0 until cardsToBeCheckedForDiscarding.size){
        copyOfCardsToBeCheckedForDiscarding.add(cardsToBeCheckedForDiscarding[position])
    }
    val cardsWithoutJokers = removeJokers(copyOfCardsToBeCheckedForDiscarding)
    var counter = 0
    for(position1 in 0 until cardsWithoutJokers.size){
         counter = 0
        val card1=cardsWithoutJokers[position1]
        for(position2 in 0 until cardsWithoutJokers.size){
            val card2= cardsWithoutJokers[position2]

            if(card1.numberValue != card2.numberValue){
                return false
            }
            if(card1.suit == card2.suit){
                counter++
                Log.d( tag: "counter2", msg: "counter[$position2] $counter")
                if(counter>1){
                    return false
                }
            }
        }
    }
    return true
```

Slika 3.24. isThreeOrFourOfSameNumAndDifSuit

```
fun sortAllChecked(playerCards: MutableList<Card>): MutableList<Card>{

    val sortedPlayerCards = mutableListOf<Card>()

    val copyOfPlayerCards= mutableListOf<Card>()
    for(position in 0 until playerCards.size){
        copyOfPlayerCards.add(playerCards[position])
    }

    val clubCards = mutableListOf<Card>()
    val heartCards = mutableListOf<Card>()
    val spadeCards = mutableListOf<Card>()
    val diamondCards = mutableListOf<Card>()
    val jokers = mutableListOf<Card>()

    val positions = arrayListOf<Int>()
```

Slika 3.25. sortAllChecked dio 1.

```
for(position in 0 until playerCards.size){
    if(playerCards[position].isChecked) {
        if (playerCards[position].suit == "CLUB"){
            clubCards.add(playerCards[position])
            positions.add(position)
        }
        else if (playerCards[position].suit == "HEART"){
            heartCards.add(playerCards[position])
            positions.add(position)
        }
        else if (playerCards[position].suit == "SPADE"){
            spadeCards.add(playerCards[position])
            positions.add(position)
        }
        else if (playerCards[position].suit == "DIAMOND"){
            diamondCards.add(playerCards[position])
            positions.add(position)
        }
        else if(isJoker(playerCards[position])){
            jokers.add(playerCards[position])
            positions.add(position)
        }
    }
}
```

Slika 3.26. sortAllChecked dio 2.

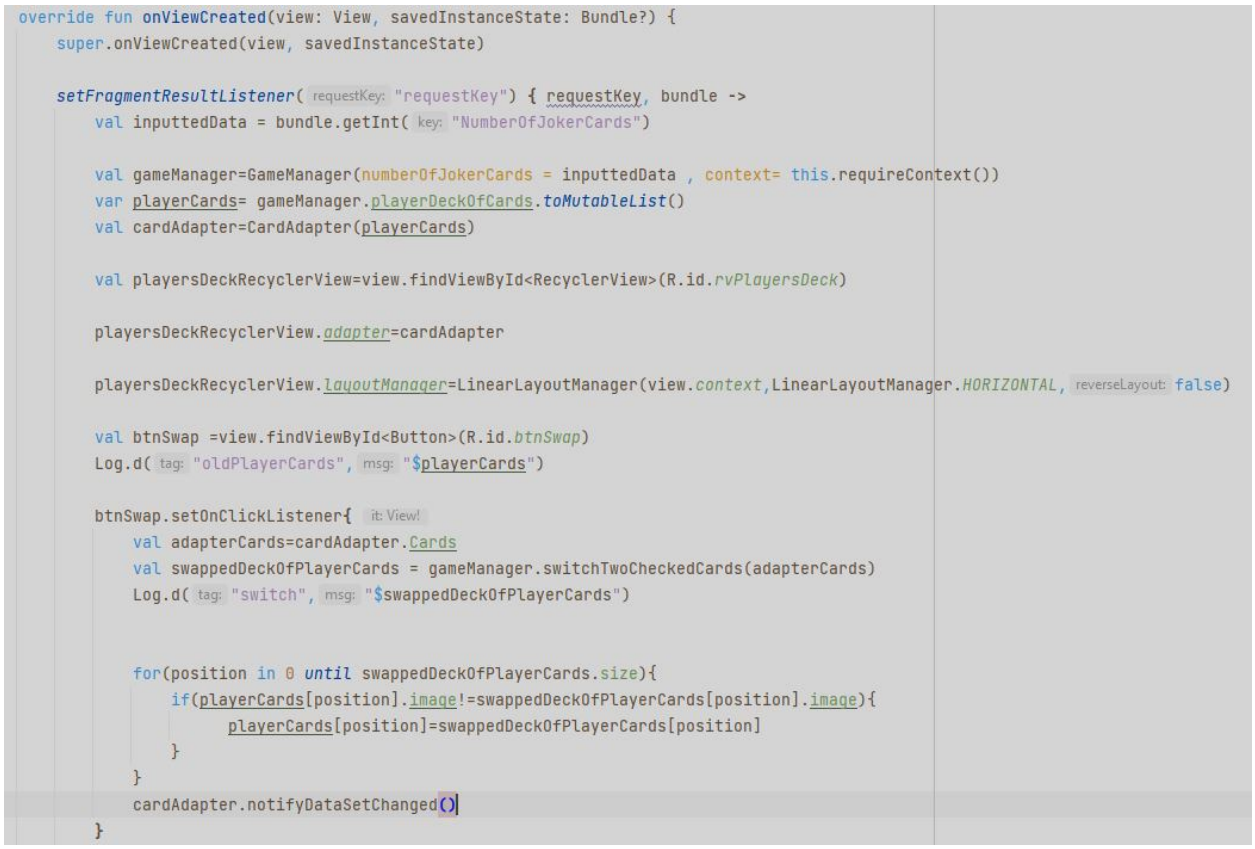
```
for(position in positions.size-1 downTo(0)){
    copyOfPlayerCards.removeAt(positions[position])
}

if(clubCards.isNotEmpty()){
    clubCards.sortBy { it.numberValue }
    for(position in 0 until clubCards.size){
        sortedPlayerCards.add(clubCards[position])
    }
}
if(heartCards.isNotEmpty()){
    heartCards.sortBy { it.numberValue }
    for(position in 0 until heartCards.size){
        sortedPlayerCards.add(heartCards[position])
    }
}
if(spadeCards.isNotEmpty()){
    spadeCards.sortBy { it.numberValue }
    for(position in 0 until spadeCards.size){
        sortedPlayerCards.add(spadeCards[position])
    }
}
if(diamondCards.isNotEmpty()){
    diamondCards.sortBy { it.numberValue }
    for(position in 0 until diamondCards.size){
        sortedPlayerCards.add(diamondCards[position])
    }
}
```

Slika 3.27. sortAllChecked dio 3.

```
    if(jokers.isNotEmpty()){
        for(position in 0 until jokers.size){
            sortedPlayerCards.add(jokers[position])
        }
    }

    if(copyOfPlayerCards.isNotEmpty()){
        for(position in 0 until copyOfPlayerCards.size){
            sortedPlayerCards.add(copyOfPlayerCards[position])
        }
    }

    if(sortedPlayerCards.isEmpty())
        return playerCards

    return sortedPlayerCards

}
```

Slika 3.28. sortAllChecked dio 4.

Nakon što korisnik odabere željeni broj Joker karata i stisne button start prelazi u welcomeFragmenta u GameFragment čiji kod će biti prikazan u sljedećem dijelu.



Slika 3.29. dio koda unutar fragmenta GameFragment dio 1.

Funkcija setFragmentResultListener dohvaća bundle poslan iz welcomeFragment-a, tj. dohvaća broj željenih Joker karti unutar špila igre koju je korisnik sam odabrao.

Objekt gameManager je inicijaliziran tako da je za prvi argument konstruktora postavljen broj Joker karti, a za drugi kontekst unutar kojeg je inicijaliziran, u ovom slučaju je to fragment GameManager.

Na slici 3.29. vidimo primjer referenciranja widgeta button swap(unutar fragment_game layouta) i postavljanje click listenera koji prilikom „klika“ na to dugme zamjenjuje dvije karte igračevog špila koristeći metodu objekta gameManager zvanu switchTwoCheckedCards i ako su zadovoljeni svi uvjeti dolazi do zamjene dviju karata igračevog špila pri čemu svaku promjenu liste koju koristi cardAdapter moramo obavijestiti sami adapter da je došlo do promjene u suprotnom ona neće biti vidljiva unutar recyclerViewa.

Na slici 3.30. prikazani su načini na koji su ugrađene funkcionalnosti igre u dugmad Replace, Check i Undo.

```
val btnReplace = view.findViewById<Button>(R.id.btnReplace)

btnReplace.setOnClickListener{  it: View!
    val adapterCards=cardAdapter.Cards
    playerCards=gameManager.replace(adapterCards)
    cardAdapter.notifyDataSetChanged()
}

val btnCheck =view.findViewById<Button>(R.id.btnCheck)

btnCheck.setOnClickListener {  it: View!
    val adapterCards = cardAdapter.Cards
     playerCards=gameManager.discardCheckedCards(adapterCards)
    Log.d( tag: "Check", msg: "${playerCards}")
    cardAdapter.notifyDataSetChanged()
    if(playerCards.isEmpty()){
        parentFragmentManager.beginTransaction().apply {  this: Frag
            replace(R.id.flMainActivity, WinFragment())
            addToBackStack( name: null)
            commit()
        }
    }
}

val btnUndo = view.findViewById<Button>(R.id.btnUndo)
btnUndo.setOnClickListener {  it: View!
    if(gameManager.savedStates.isNotEmpty()){
        playerCards.addAll(gameManager.restoreFromMemento())
        cardAdapter.notifyDataSetChanged()
    }
}
```

Slika 3.30. dio koda unutar fragmenta GameFragment dio 2.

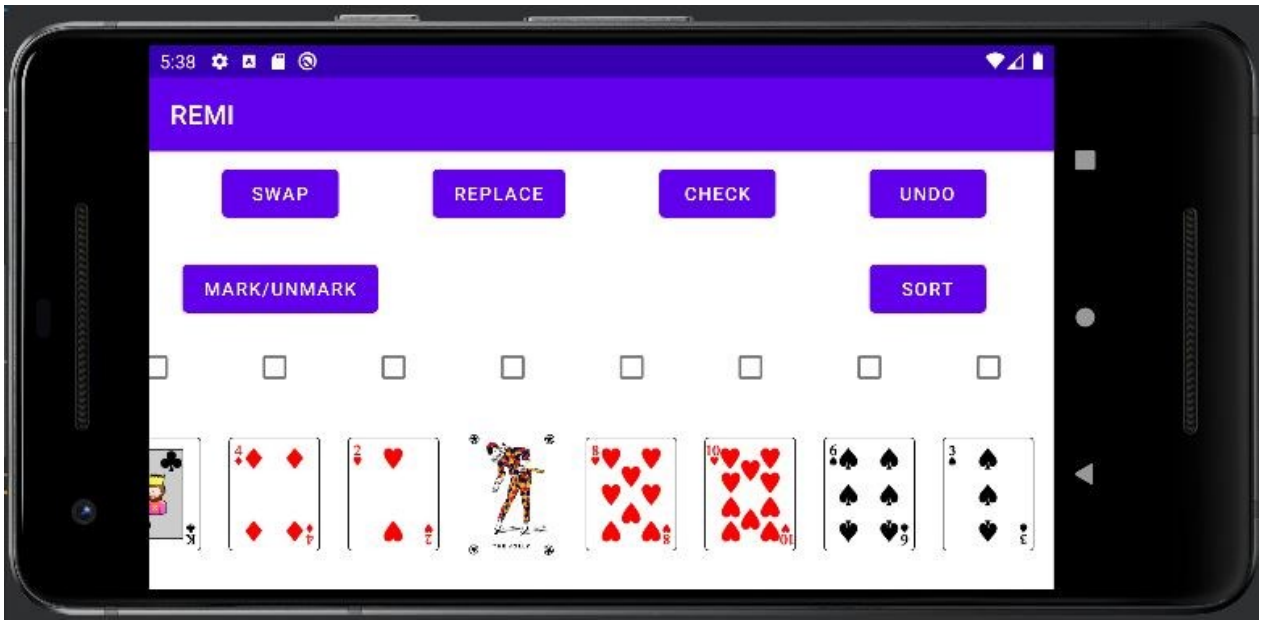
Ostale funkcionalnosti gameManagera su na isti način napravljene.

4. IZGLED ANDROID APLIKACIJE REMI

Prilikom ulaska u aplikaciju korisnik pomoću spinnera može izabrati ponuđeni broj Joker karti.

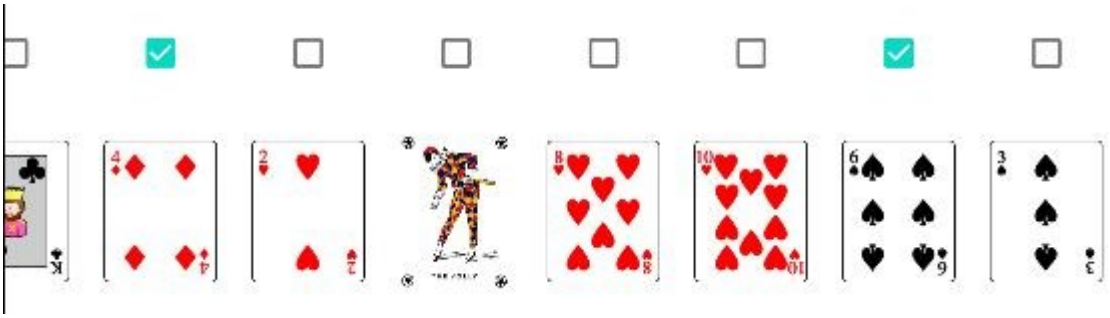
Slika 4.1. Prikaz ulaska u igru.

Nakon pritiska dugma „Start“ prikazuje se grafičko sučelje na slici 4.2.:

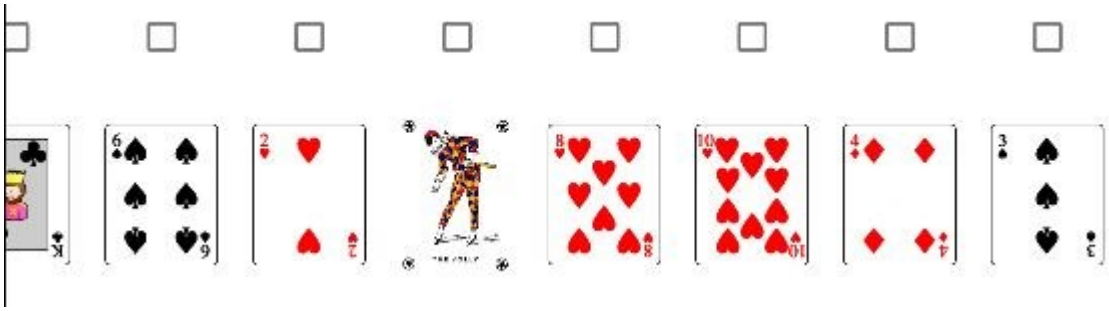


Slika4.2. Prikaz početka igre

Korištenjem dugmadi “SWAP“ možemo zamijeniti dva položaja bilo kojih karata u igračevom špilu kao što je prikazano na slikama 4.3. i 4.4.:



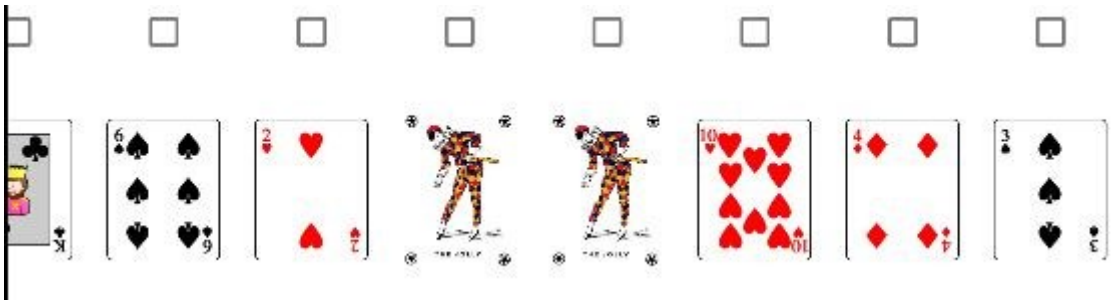
Slika 4.3. Prikaz dvaju označenih karata kojima korisnik želi zamijeniti mjesta



Slika 4.4. Poredak karata nakon pritiska dugmadi „SWAP“

Korištenjem dugmadi „REPLACE“ korisnik odbacuje jednu neželjenu kartu iz špila i dobija novu iz špila igre kao što je prikazano na slikama 4.5. i 4.6.:

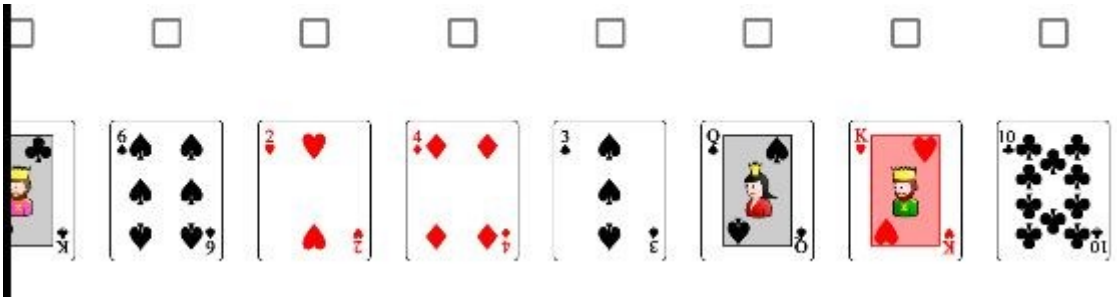
Slika 4.5. Karta 8 srce prije izbacivanja



Slika 4.6. Nova izvučena Joker karta na prijašnjem mjestu izbačene 8 srce

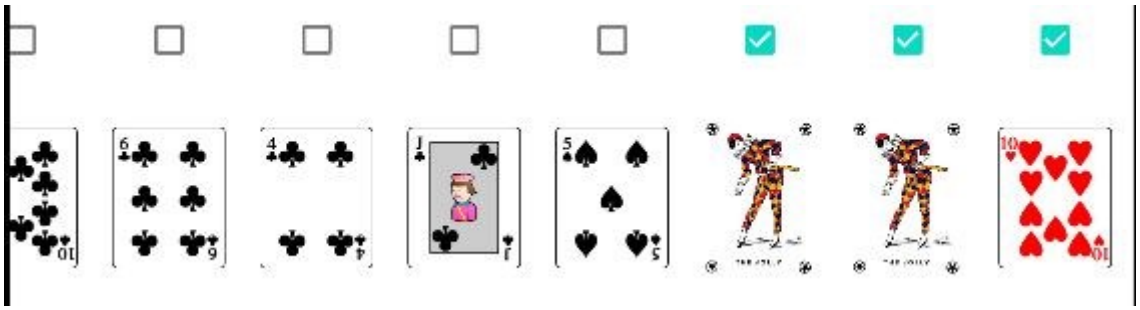
Ako korisnik želi izbaciti kombinaciju mora označiti checkboxove karata koje bi izbacio i ako kombinacija tih karata bude zadovoljavajuća po jednom od 2 uvjeta karte će biti maknute iz igračevog špila kao što je prikazano na slikama 4.7. i 4.8.:

Slika 4.7. Igračeva kombinacija prije (Joker, Joker, 10 srce) prije pritiska dugmadi „CHECK“



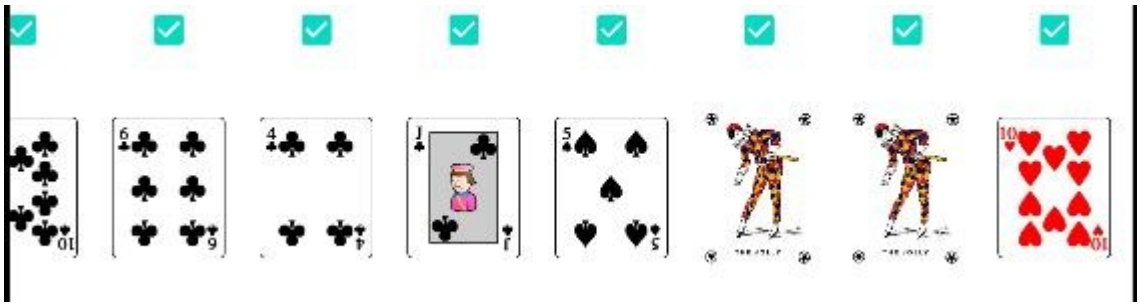
Slika 4.8. Igračev špil nakon izbacivanja prethodne kombinacije

Pritiskom na dugmad „UNDO“ igraću će se vratiti zadnja zadovoljavajuća kombinacija karata koju je izbacio, dugmad „UNDO“ ima mogućnost vraćanja redom od zadnje do prve zadovoljavajuće kombinacije koju je igrač izbacio. U ovom slučaju igraču će se pritiskom na „UNDO“ vratiti kombinacija karata Joker, Joker i 10srce na zadnje mjesto u špilu kao što je prikazano na slici 4.9.:

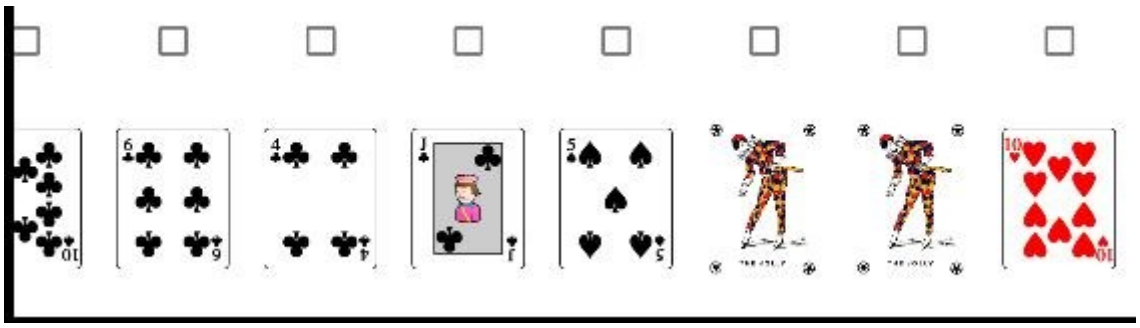


Slika 4.9. Zadnja vraćena kombinacija nakon pritiska dugma „Undo“

Pritiskom na dugmad „MARK/UNMARK“ ako svi checkBoxovi nisu označeni, postat će i obratno kao na slikama 4.10. i 4.11.:

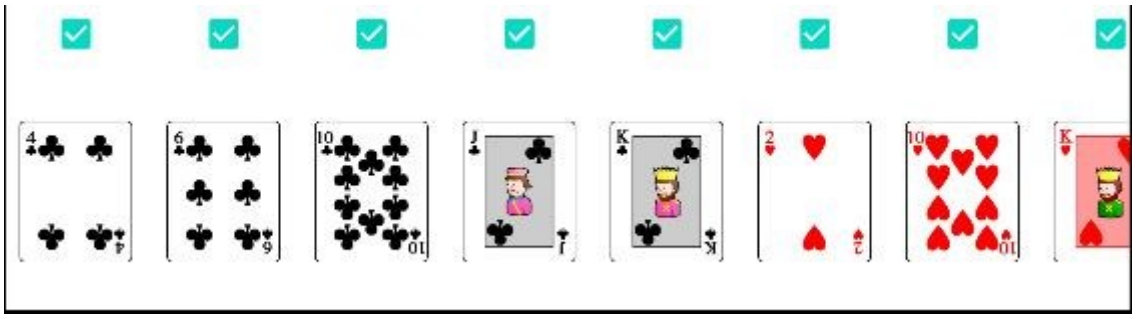


Slika 4.10. prvi pritisak na dugme „MARK/UNMARK“

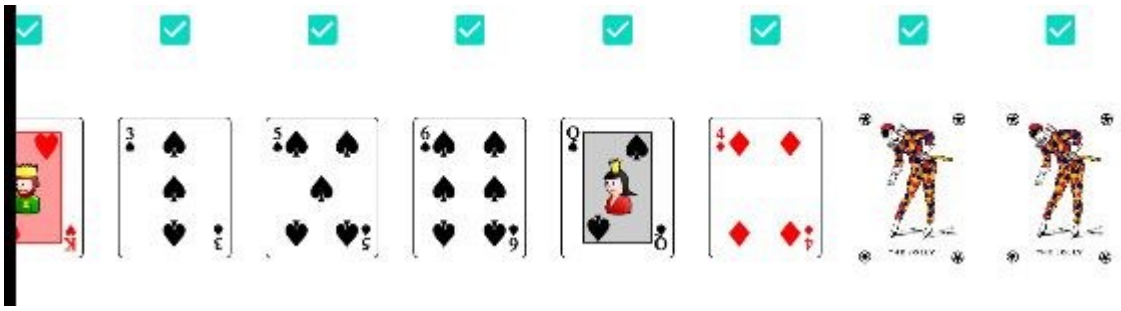


Slika 4.11. drugi pritisak na dugme „MARK/UNMARK“

Pritiskom na dugme „SORT“ sortiramo sve označene karte po uzlaznim veličinama njihovih brojčanih vrijednosti unutar njihovih znakova tj. boja. U ovom primjeru označit ćemo kao na slici 4.10. sve igračeve karte potom pritisnuti „SORT“ te će rezultat biti kao na slici 4.12. i 4.13.:

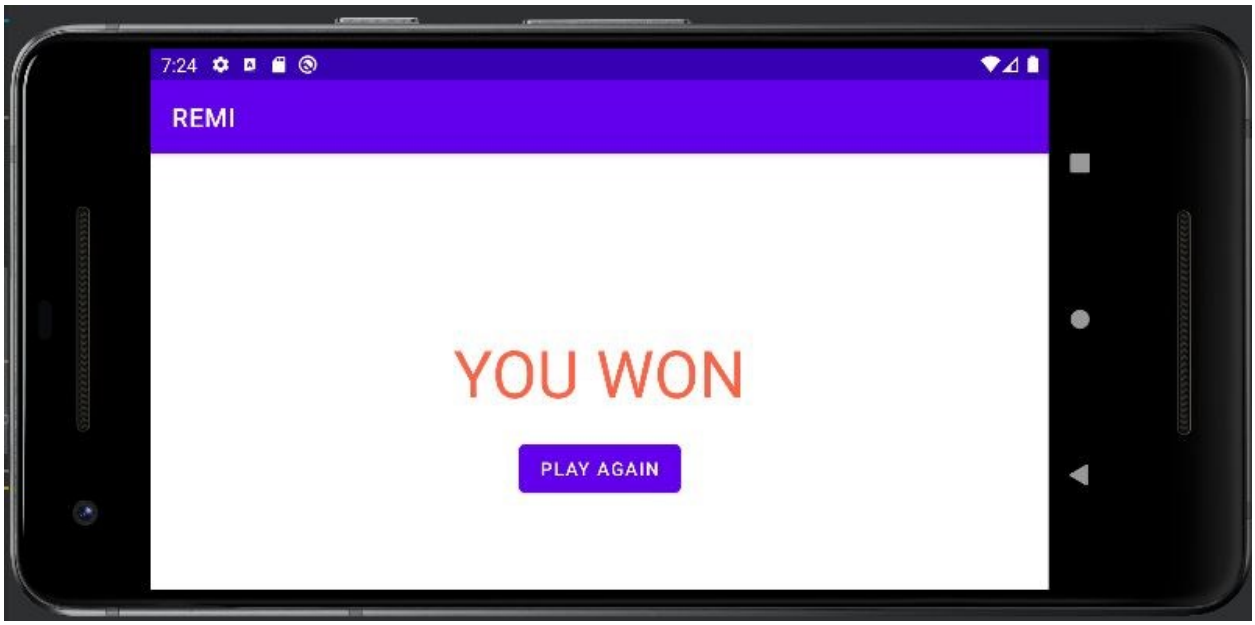


Slika 4.12. prikaz sortiranih karata igračevog špila dio 1.



Slika 4.13. prikaz sortiranih karata igračevog špila dio 2.

Nakon što igrač izbaci sve karte iz vlastitog špila, prelazi u novi WinFragment koji mu daje obavijest da je pobijedio i daje mu mogućnost ponovne igre kao na slici 4.14.:



Slika 4.14. prikaz UI nakon izbacivanja svih karata

Pritiskom na dugme „PLAY AGAIN“ igrač se vrača na početni ekran kao na slici 4.1.

5. ZAKLJUČAK

Nove mobilne aplikacije niču iz dana u dan u sve većem broju što čini potrebu za programskim inženjerima u tom, ali i u općenitom području eksponencijalno većom.

Android studio je jedan od najpopularnijih integriranih programskih okruženja što ga čini idealnim za korištenje pri izradi android aplikacija.

Kotlin je relativno novi programski jezik koji je kompanija Google počela preferirati 2019. za android development što ga uz bolju sintaksu, null-sigurnost i ostale značajke iz dana u dan čini sve popularnijim naspram stare Jave.

U projektu su postignuti svi osnovni projektni zadaci i funkcionalnosti uz dodatak funkcionalnosti „UNDO“ korištenjem obrasca memento koji omogućava igraču da se vrati do prve izbačene kombinacije krenuvši redom od zadnje i „MARK/UNMARK“ koji omogućava označavanje ili odznačavanje svih igračevih karata.

Za prikaz špila igračevih karata korištena je android značajka recyclerView koji je nastao kao potreba za proširenjem ListViewa jer smanjuje poziv funkcije findViewById što omogućuje manju upotrebu računalnih resursa velikim količinama podataka i omogućuje horizontalni prikaz podataka dok ListView omogućuje samo vertikalni. Uz recyclerView korišteni su i fragmenti koji omogućuju modularan dizajn aplikacije.

Svakako postoje dodatne mogućnosti koje bi se mogle implementirati u ovaj rad kao uvođenje više AI igrača ili omogućiti online igranje itd.

LITERATURA

[1] THE BRIDGE, "A brief history of rummy: How the game evolved“, dostupno na: https://thebridge.in/card-games/history-rummy-game-20510#:~:text=The%20Origin%20and%20History%20of%20Rummy&text=According%20to%20the%20most%20widely,variant%20of%20this%20skill%20game.[pristupljeno 27.lipnja 2021.]

[2] Android studio [online] , dostupno na: https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio [pristupljeno 27.lipnja 2021.]

[3] Android architecture[online], dostupno na: https://www.geeksforgeeks.org/android-architecture/ [pristupljeno 28.lipnja 2021.]

[4] Kotlin , dostupno na: https://learn.g2.com/what-is-kotlin [pristupljeno 28. lipnja 2021.]

[5] Kotlin i Java , dostupno na: https://kotlinlang.org/docs/comparison-to-java.html [pristupljeno 29.lipnja 2021.]

[6] XML [ONLINE] , dostupno na: https://www.geeksforgeeks.org/xml-basics/?ref=gcse [pristupljeno 29.lipnja 2021.]

[7] Layouts [online] , dostupno na : https://developer.android.com/guide/topics/ui/declaring-layout [pristupljeno 29.lipnja 2021.]

[8] RecyclerView [ONLINE], dostupno na: https://developer.android.com/guide/topics/ui/layout/recyclerview [pristupljeno 29.lipnja 2021.]

[9] Predložak za laboratorijsku vježbu LV8 kolegija "Osnove razvoja web i mobilnih aplikacija"

[10] Fragmenti[online], dostupno na: https://developer.android.com/guide/fragments [pristupljeno 29.srpnja 2022.]

[11] Activity [online], dostupno na : https://developer.android.com/guide/components/activities/intro-activities [pristupljeno 1. kolovoza 2022.]

[12] Activity lifecycle, dostupno na: https://w3cschoool.com/activity-lifecycle [pristupljeno 1. kolovoza 2022.]

SAŽETAK

U ovom završnom radu je napravljena android aplikacija kartaške igre remi. Korišteno razvojno okruženje je Android studio. Aplikacija je programirana koristeći programski jezik Kotlin i opisni jezik XML. Cilj aplikacije je omogućiti jednom korisniku igranje kartaške igre remi. Korisniku tj. igraču je omogućeno zamijeniti poredak, automatsko sortiranje, odbacivanje vlastite karte i uzimanje jedne iz špila igre, poništavanje izbacivanja zadnje kombinacije karata do prve kombinacije karata redom , automatsko označavanje ili odznačavanje svih karata ovisno o njihovom trenutnom statusu označenosti. Ključne klase i funkcije aplikacije su pojašnjene i njihov isječak koda je prikazan.

KLJUČNE RIJEČI: android , aplikacija, karte , kotlin , remi .

SUMMARY

In this paper, an android application for rummy card game was created. The IDE used is Android studio. The application is programmed using the Kotlin programming language and the XML markup language. The goal of the application is to enable one user to play the rummy card game. The User, i.e. the Player, is enabled to change order,to use automatic sorting, to discard own cards and taking one from the game deck, undoing the discarding of the last combinations of cards up to the first combination of cards in a row, automatically marking or unmarking all cards depending on their current checkbox status. The key classes and functions of the application are explained and their code snippet is shown.

KEY WORDS: android, application, cards, kotlin, rummy.