Mobilna aplikacija za rješavanje logičke zagonetke Sudoku

Dražetić, Danijel

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski sveučilišni studij računarstva

MOBILNA APLIKACIJA ZA RJEŠAVANJE LOGIČKE ZAGONETKE SUDOKU

Završni rad

Danijel Dražetić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 18.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Danijel Dražetić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4051, 26.09.2019.
OIB studenta:	49031832249
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna aplikacija za rješavanje logičke zagonetke Sudoku
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	18.09.2020.
Datum potvrde ocjene Odbora:	23.09.2020.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 23.09.2020.

Ime i prezime studenta:	Danijel Dražetić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4051, 26.09.2019.
Turnitin podudaranje [%]:	21

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za rješavanje logičke zagonetke Sudoku**

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

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

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

Potpis studenta:

SADRŽAJ

1. UVOD

Tema ovog završnog rada je izrada mobilne aplikacije za rješavanje logičke zagonetke sudoku. Cilj mobilne aplikacije je pružanje korisniku raznovrsne sudoku zagonetke koje korisnik pokušava riješiti i to je uspješno ostvareno. Korisnik rješava sudoku zagonetku tako što upisuje brojeve u prazno polje pazeći pritom na kriterije sudoku zagonetke. Broj koji korisnik unosi u prazno polje ovisi o broju ponavljanja toga broja u tom retku, stupcu i polju 3x3. Platforma za koju je aplikacija napravljena su mobilni android uređaji. U 2. poglavlju se nalazi povijest i pravila Sudoku igre te primjeri sa slikama kako bi se trebalo igrati. U 3. poglavlju detaljnije su opisani alati koji su korišteni pri izradi ove mobilne aplikacije. Sav kod ove aplikacije napisan je u softverskom okruženju *Android studio IDE*, dok je korištena *Java* kao programski jezik te *XML (eng. Extensible Markup Language)* kao proširivi jezik za označavanje podataka. U 5. poglavlju je opisan postupak razvoja aplikacije Sudoku koji se dijeli na dva dijela. Prvi dio je algoritamski gdje se pokazuje algoritam za izradu točno riješenih sudoku tablica te algoritam za provjeru točno riješene tablice. Drugi dio je vizualni izgled aplikacije kakva je predstavljena krajnjem korisniku.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je objasniti pravila i načine igranja zagonetke Sudoku te izraditi mobilnu aplikaciju za rješavanje logičke zagonetke sudoku u razvojnom okruženju Android studio.

2. SUDOKU IGRA

2.1. Pregled matematičkih rezultata u logičkoj igri sudoku

Riješena sudoku zagonetka je posebna vrsta latinskog kvadrata sa dodatnim svojstvom da se svaki broj smije samo jednom ponoviti u 3x3 polju, odnosno svako 3x3 polje treba sadržavati sve brojeve u rasponu od 1 do 9. Latinski kvadrat je $n \times n$ tablica s n različitih simbola tako da se svaki simbol pojavljuje točno jednom u svakom retku i svakom stupcu. Dokazano je da formula logike prvog reda vrijedi za sudoku zagonetku samo ako vrijedi i za latinske kvadrate. Logika prvog reda ili logika predikata je zbirka formalnih sustava koji se koriste u matematici i računarstvu. [1]

Znanstvenici G. McGuire, B. Tugemann i G. Civario su dokazali 2012. godine da je potrebno minimalno imati 17 početnih brojeva u sudoku da bi dobili jedinstveno rješenje. Na konferenciji u Bostonu 2013. godine objavljeno je kako je dokaz ispravan te predstavlja važan iskorak u polju Sudoku matematike. [2]

Maksimalan broj početnih brojeva sudoku zagonetke da bi smo dobili jedinstveno rješenje jest 40. Do sada su otkrivene samo dvije takve varijacije zagonetke sa 40 početnih brojeva. Postoji oko 6 500 000 000 različitih sudoku zagonetki sa 36 početnih brojeva te oko 2600 različitih sudoku zagonetki sa 39 početnih brojeva koji daju jedinstveno rješenje. [3]

Postoji 6 670 903 752 021 072 936 960 ili $6{,}67 \times 10^{21}$ različitih rješenja sudoku zagonetke. Ovaj broj je objavio *QSCGZ* 2003.godine. 2005. godine Felgenhauer i Jarvis su analizom permutacija potvrdili broj različitih rješenja sudoku zagonetke koji je objavio *QSCGZ*. [4]

Opći problem rješavanja Sudoku zagonetke veličine $n^2 \times n^2$ sa poljima $n \times n$ je NP-težak. NP-potpuni problemi imaju svojstvo da im se rješenja mogu brzo provjeriti, s tim da trenutne metode pronalaženja problema nisu skalabilne. Računalni programi poput povratnog praćenja mogu učinkovito riješiti većinu 9 x 9 zagonetki, ali kada se broj n poveća događa se kombinatorna eksplozija. Kombinatorna eksplozija je brzi rast složenosti problema jer na kombinatoriku utječu ulazni podaci, ograničenja i granice problema. [5]

2.2. O autoru sudoku igre

Igru Sudoku je izmislio Amerikanac Howard Garns pod imenom *NumberPlace* 1979. godine. Garns je bio umirovljeni arhitekt i samostalni sastavljač zagonetki iz Connersville-a, Indiana. Slavu je stekao tek nakon što je preminuo jer je Sudoku kao što ju danas znamo postala popularna tek nakon njegove smrti. Rođen je 2.5.1905. u Indiani (SAD) gdje je i umro 6.10.1989. godine zbog raka. 1922. godine je diplomirao na Tehničkoj gimnaziji u Indanapolisu. Poslije toga upisuje Sveučilište u Illinoisu gdje je diplomirao arhitekturu 1926. godine. Radio je za očevu tvrtku do Drugog svjetskog rata. Kada je rat počeo postao je kapetan u Američkom vojnom korpusu inženjera *(US Army Corps of Engineers).* Nakon rata se je pridružio tvrtki *Daggett architecture* gdje je radio do svog umirovljenja. [6]

2.3. Povijest sudoku igre

Igra Sudoku se prvi puta pojavljuje 1892. godine. U početku je bila drukčija nego što je sada. U prošlosti Sudoku se je sastojao samo od 4 polja. Dok se nije pojavio Howard Garns i izmijenio pravila i dimenzije na 9 polja. Garns je igru Sudoku pod imenom *NumberPlace* objavio u američkom magazinu *Dell Magazines*. Prvi puta je ova igra bila objavljena u Japanu od strane *Nikoli* 1984. godine u novinama *Monthly Nikolist* pod nazivom *„Sūji wa dokushin ni kagiru“* što znači „brojevi moraju biti jednoznamenkasti“. Nakon Garnsove smrti Japanci mijenjaju ime igre u Sudoku što znači jedinstven broj na japanskom jeziku. U posljednje vrijeme posebno je popularna, te se održavaju turniri, državna i svjetska prvenstva. Postoje i lakše verzije za djecu, a to su 2x2 i 3x3 dimenzija. [7]

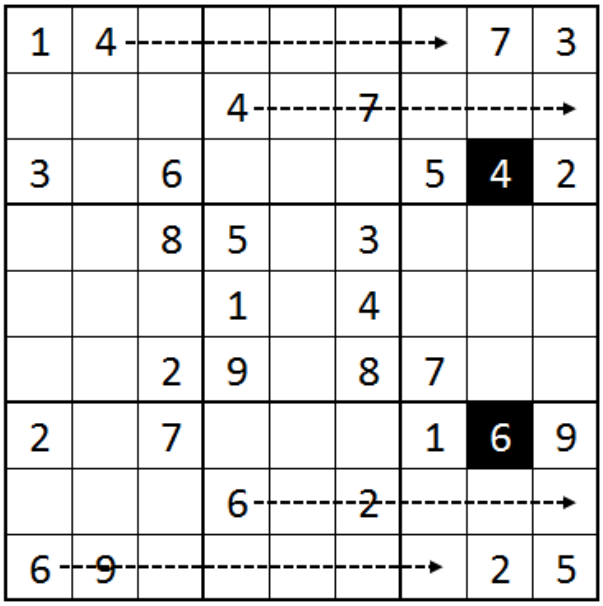
2.4. Pravila sudoku igre

Sudoku je vrsta matematičke zagonetke čije je rješavanje temeljeno na logici. Sastoji se od jednog velikog kvadratnog polja. To polje je podijeljeno na 9 manjih kvadrata dimenzija 3x3. Ukupno se sastoji od 81 polja u kojima imamo neke već predodređene brojeve a ostale brojeve sami upisujemo. U polja upisujemo brojeve od 1 do 9 s time da se svaki broj smije pojaviti točno 9 puta. Problematika je u tome što se jedan broj smije pojaviti samo jednom u svakom retku, stupcu i kvadratu 3x3. Sudoku može imati više rješenja pa se treba paziti da odmah na početku ne dođe do greške. Pogađanje i nagađanje većinom vode na krivi put.

2.5. Načini rješavanja sudoku igre

Jedini način rješavanja sudoku-a je metoda eliminacije, a tu se koristi svojstvom da se jedan broj smije pojaviti samo jednom u pojedinom stupcu, retku i polju 3x3.
Slijede primjeri sa metodama eliminacije:

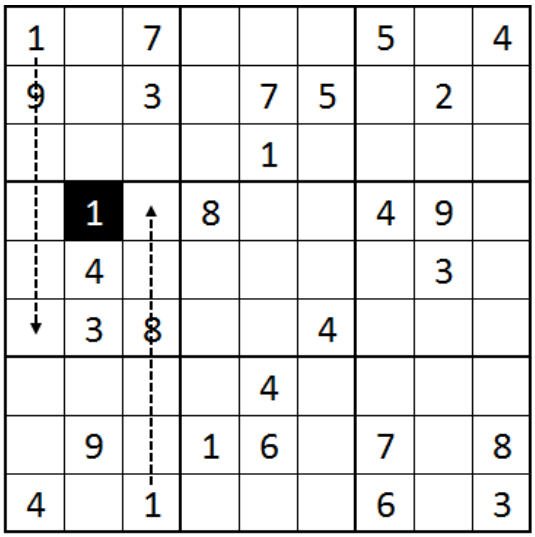
a) ELIMINACIJA – po redovima



Sl. 2.1 *eliminacija po redovima*

Pitamo se gdje u gornjem desnom polju 3x3 možemo postaviti broj 4. Na slici iznad možemo lako uočiti eliminacijom po redovima da se broj 4 može jedino upisati u treći redak. Isto tako, gledajući kvadratić veličine 3x3 koji se nalazi dolje desno, vidimo da se broj 6 u tom kvadratu može nalaziti samo u gornjem redu.

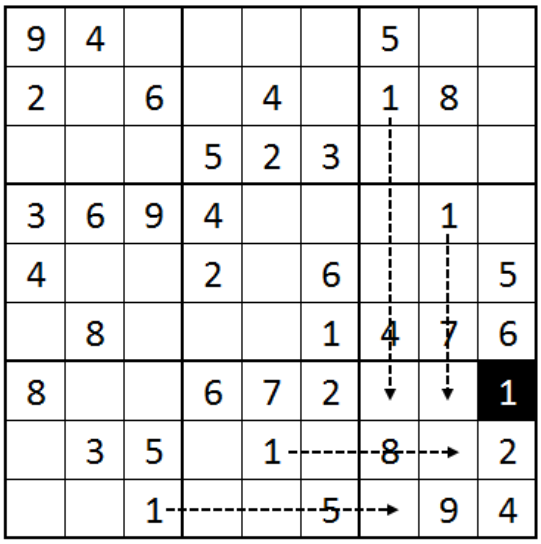
b) ELIMINACIJA - po stupcima



***Sl. 2.2**. eliminacija po stupcima*

Na gornjoj slici možemo prateći kretanje broja 1 kroz prvi i treći stupac uočiti da za broj 1 u srednjem stupcu ima samo jedno moguće mjesto.

c) ELIMINACIJA – po redovima i stupcima



***Sl. 2.3**. eliminacija po redovima i stupcima*

Vidimo na gornjoj slici da u kvadratiću 3x3 koji se nalazi dolje desno kombiniranom eliminacijom sa sigurnošću možemo upisati broj 1 u obilježeno polje.

d) ELIMINACIJA – zauzetost polja

4			8		9	1		
		7					9	
9	5			2				7
1				9				3
3	9	2	4		7	8		
6				3				9
7	2	4 9 ?		8			6	
	1	4 9 ?				2		
	6	3	1		2			4

***Sl. 2.4**. eliminacija zauzetost polja*

Na slici iznad prvo eliminacijom po stupcima i redovima uočavamo da brojeve 4 i 9 možemo upisati samo u polja obilježena upitnikom. Iako nije poznat točan položaj 4 i 9 znamo da su ta dva polja zauzeta. Nastavljamo sa eliminacijom broja 6 i uočavamo polje u koje ide broj 6.

e) ELIMINACIJA – po malim kvadratima

			1		4			
		1				9		
	9		7		3		6	
8	?	7				1		6
3		4				5		9
	5		4		2		3	
						6		
			8		6			

***Sl. 2.5** eliminacija po malim kvadratima*

Na slici iznad vidimo da u polju 3x3 sa znakom „?“ nedostaju brojevi 1, 2, 5, 6, 9. Eliminacijom po redu na tom mjestu ne mogu biti 1 i 6. Eliminacijom po stupcu ne mogu biti 5 i 9. Ostaje samo jedna mogućnost, a to je broj 2. [8]

3. KORIŠTENI ALATI I RAZVOJNA OKRUŽENJA

3.1. Android

Android je otvoreni operacijski sustav koji je najviše namijenjen za mobilne uređaje kao što su pametni telefoni i tablet računala. Otvoreni operacijski sustavi su sustavi otvorenog programskog koda. Osim činjenice da su besplatni, važni je naglasiti da ih svaki korisnik može dorađivati. Pored mobilnih uređaja android se još koristi i u pametnim televizorima pod nazivom *Android TV*, u automobilima pod nazivom *Android Auto* te u pametnim satovima pod nazivom *Wear OS*. Android je temeljen na jezgri *Linux*. Jezgra *Linux* je srž operacijskog sustava i najniži apstrakcijski sloj koji je izveden programski. Služi kao most između programa koji se izvode i svega u njihovoj okolini. Android platforma prilagođena je za uporabu na uređajima s većim zaslonima, poput pametnih telefona koji rabe 2D grafičku knjižnicu ili 3D grafičku knjižnicu temeljenu na *OpenGL ES 2.0* specifikacijama. Za pohranu podataka upotrebljava se *SQLite* relacijska baza podataka, napisana u programskom jeziku C. Android je najprodavaniji mobilni operacijski sustav za pametne telefone od 2011. i za tablete od 2013. godine. [9]

Android je isprva razvila tvrtka Android Inc. Tvrtku su osnovali Andy Rubin, Rich Miner, Nick Sears i Chris White u listopadu 2003. godine. Prva namjera tvrtke je bila razviti operacijski sustav za digitalne kamere, no to je tržište bilo premalo pa su promijenili cilj i okrenuli se operacijskom sustavu za mobilne uređaje kao izravnu konkurenciju Symbian-u i Windows mobile-u. 2005. godine Google kupuje tvrtku Android Inc. 5. studenoga 2007. godine osnovan je *Open Handset Alliance (OHA)* s ciljem stvaranja javnog standarda za mobilne uređaje te je odmah istog dana bilo prvo javno predstavljanje Androida kao operacijskog sustava baziranog na Linux jezgri. Prvi mobilni uređaj sa android sustavom je bio *T-Mobile G1* ili pod drugim nazivom *HTC Dream*. Dosadašnje inačice android sustava redom po datumu predstavljanja su prikazane na tablici ispod. [10]

Inačica sustava	Datum predstavljanja	Inačica sustava	Datum predstavljanja
1. Beta verzija	5.11.2007.	10. KitKat 4.4	3.9.2013.
2. 1.0 / 1.1 verzija	23.9.2008.	11. Lollipop 5.0	25.6.2014.
3. Cupcake 1.5	30.4.2009.	12. Marshmallow 6.0	5.10.2015.
4. Donut 1.6	15.9.2009.	13. Nougat 7.0	22.8.2016.
5. Eclair 2.0 / 2.1	26.10.2009.	14. Oreo 8.0	21.8.2017.
7. Honeycomb 3.0	22.2.2011.	15. Pie 9.0	6.8.2018.
8. Ice Cream Sand 4.0	19.10.2011.	16. Q 10.0	11.3.2019.

Tb. 3.1. Inačice android sustava

Sl. 3.2 Prikaz sučelja android 1.1

Sl. 3.3 Prikaz sučelja android 10

Na slikama 3.2 i 3.3 vidimo usporedbu sučelja android 1.1 i androida 10. Ove dvije slike nam prikazuju koliko je čovječanstvo napredovalo u zadnjih 12 godina. Odmah se uočava razlika u dizajnu. Noviji mobiteli su veći i kvalitetnije izrađeni. Sučelje je puno fluidnije te ima mnoštvo više funkcionalnosti u odnosu na prošle modele.

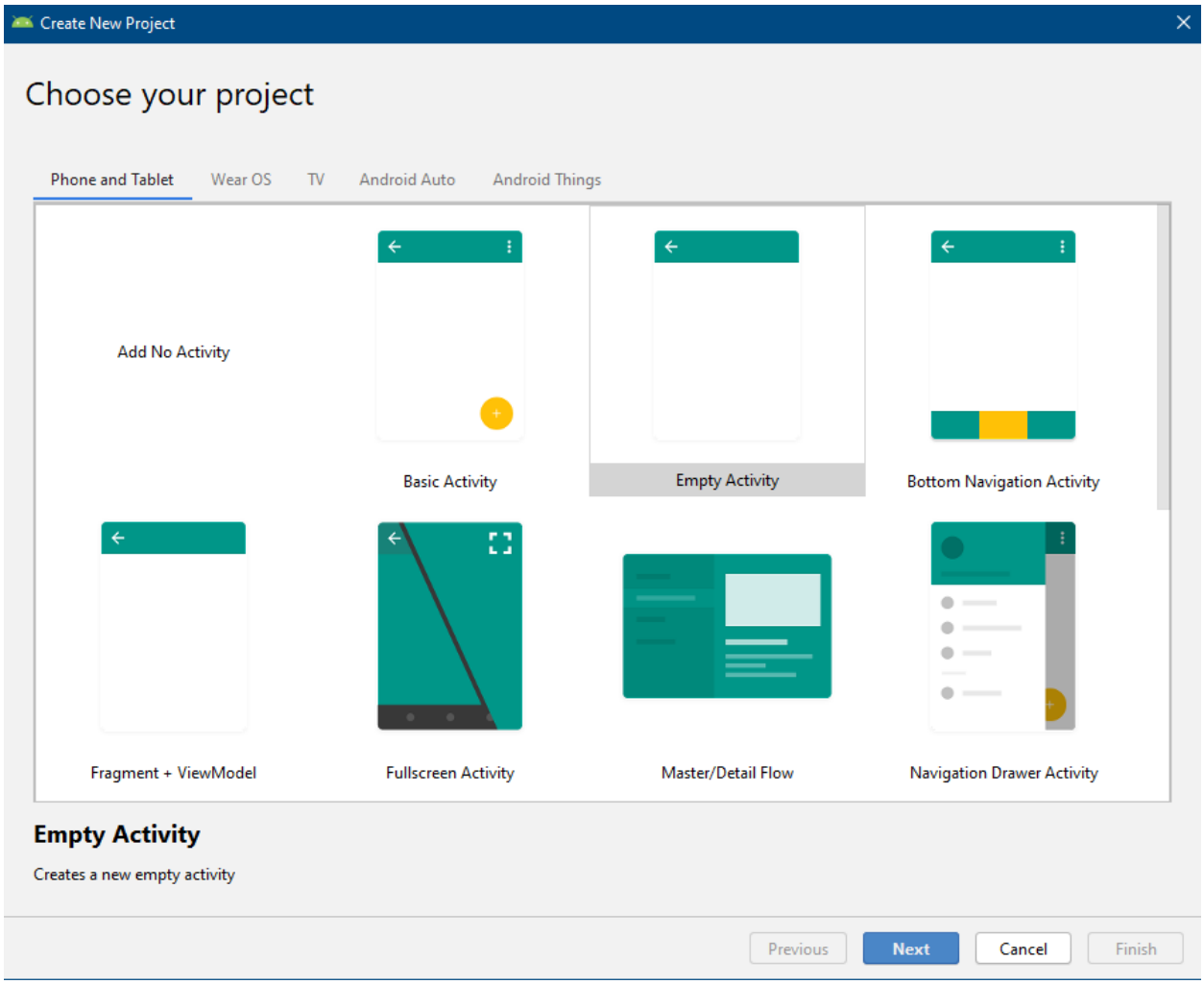
3.2. Android studio

Android Studio je službeno razvojno okruženje (*IDE – integrated development environment*) za razvoj Android aplikacija. Temelji se na integriranom razvojnom okruženju *Intelij IDEA*. *Intelij IDEA* je integrirano razvojno okruženje napisano u programskom jeziku Javi za razvoj računalnog softvera. Android studio je dostupan za skinuti na MacOS, Linux te Windows operacijskim sustavima. Neke od mnogobrojnih funkcionalnosti su: Gradle sustav za izgradnju, virtualni Android emulator za pokretanje i debugiranje aplikacija u Android studiju, mogućnosti integracije i potpisivanja aplikacija u *ProGuardu*, pokretanje aplikacija bez stvaranja novog APK-a te podršku za programske jezike C++, Javu, Kotlin, Go. [11] Da bi se mogao pokrenuti Android Studio računalo mora imati minimalno 4 GB RAM memorije dok je 8 GB preporučeno. Potrebno je imati najmanje 2 GB praznog prostora na tvrdom disku dok je preporučeno imati dvostruko više. Minimalna rezolucija koja podržava razvojno okruženje je 1280 x 800. Za operacijske sustave kod Windowsa potrebno je minimalno imati *Windows 7*, kod MacOS-a je potrebno imati 10.10 *Yosemite* ili više, dok kod Linuxa mora biti *GNOME ili KDE*.

Prva verzija Android Studija je bila 1.0 koja je predstavljena 2014. godine dok je trenutna verzija 4.0 predstavljena u svibnju ove godine.

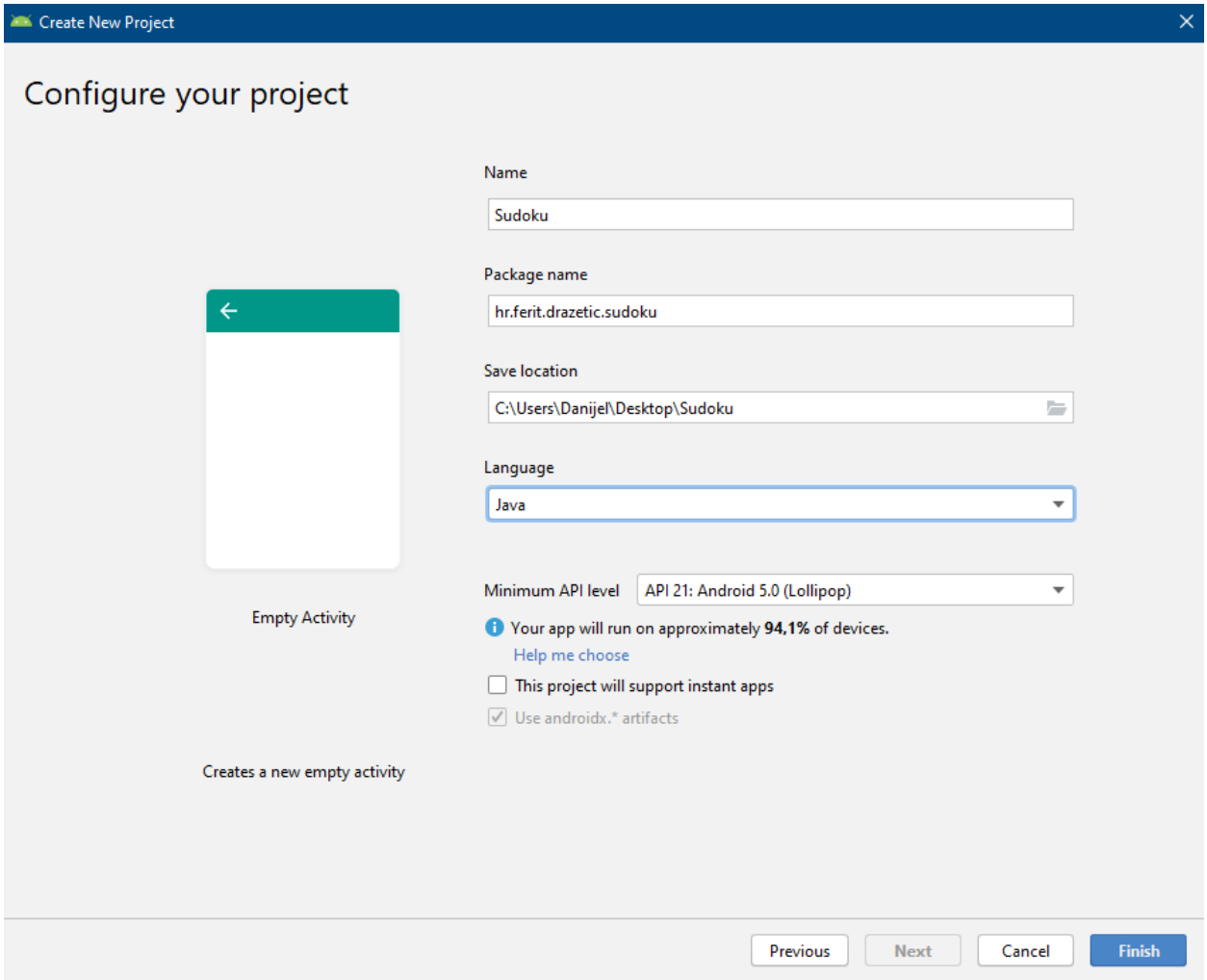
3.2.1. Kreiranje projekta u Android Studiju

Kod kreiranja novog projekta u Android Studiju korisnik dobije izbornik sa slike 3.4. Izbornik nudi izbor platforme za koju se želi razviti aplikacija. Moguće platforme su mobilni uređaji, pametni satovi, pametni televizori i *Android Auto* za ugradnju u pametne automobile. Ukoliko korisnik želi razvijati za mobilne uređaje izabire prvu opciju. Izbornik još korisniku nudi da izabere oblik aktivnosti *(eng. Activitya)*. Aktivnost predstavlja jedan zaslon aplikacije. Ponuđeni su razni predlošci, a za potrebe stvaranja aplikacije Sudoku izabrao sam praznu aktivnost. Kada je korisnik odlučio što želi i označio klikom miša, ide dalje klikom na gumb *Next*.

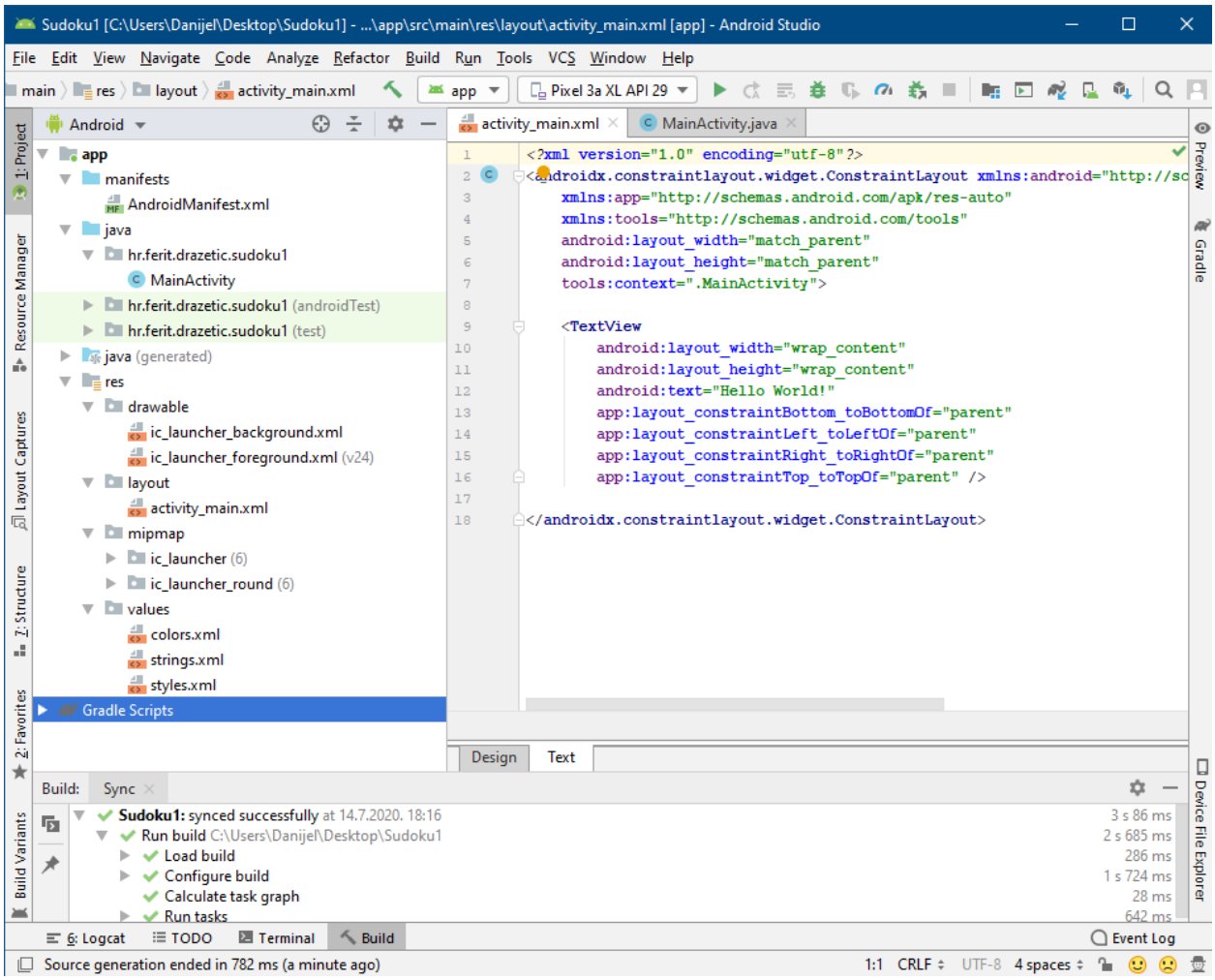


Sl. 3.4. Izrada novog projekta

Nadalje kako je prikazano na slici 3.5 korisnik mora unijeti par nužnih informacija kako bi se kreirala aplikacija. Recimo potrebno je napisati naziv aplikacije. Naziv aplikacije ne mora biti jedinstven. Jedinstven mora biti *Package name*. Zato što je kod u projektima Jave organiziran u pakete pa paketi moraju biti različiti kako bi se mogli razlikovati projekti tj. aplikacije. Iduće što biramo jest lokacija gdje želimo spremiti našu aplikaciju. Poslije toga izabiremo programski jezik u kojem želimo pisati aplikaciju. Još se izabire minimalni API nivo koji se preporuča da se izabire najmanji mogući da može obuhvatiti što više mobilnih uređaja na kojima se može pokrenuti izrađena aplikacija. U ovom slučaju to je API 21 koji pokriva otprilike 94.1% mobilnih uređaja u svijetu. Kada smo sve prošli klikom na gumb *Finish* Android studio kreće u izradu novog projekta.



Sl. 3.5. nastavak izrade novog projekta



Sl. 3.6 izgled kreirane nove aplikacije

Na slici iznad vidimo kreiranu novu aplikaciju za platformu mobilnih uređaja. Aplikacija sadrži:

- ***AndroidManifest.xml*** – osnovna datoteka projekta koja deklarira osnovne informacije da se pokrene aplikacija (ime paketa, verziju, aktivnosti, namjere).
- ***java*** – sadrži Java klase organizirane po paketima.
- ***res*** – sadrži resurse projekta (slike i XML datoteke koje opisuju izgled sučelja)
    - ***drawable*** – sadrži slike koje aplikacija koristi. Također sadrži ikonu aplikacije.
    - ***layout*** – sadrži XML definicije pogleda
    - ***values*** – sadrži XML datoteke (npr. boje, stringovi, stilovi)

3.3. Java

Java je objektno orijentirani programski jezik koji je inspiriran jezikom *C*. Java u usporedbi s *C* jezikom zahvaljujući hermetički zatvorenom okolišu u kojem svaki program operira pruža bolji stupanj sigurnosti i pouzdanosti. Radi toga je popularna kod razvoja programa za pametne telefone. *Java* je osnovni jezik za programiranje sustava *Android* koji je u vlasništvu tvrtke *Google*. Sam podatak da u svijetu ima preko 10 milijuna korisnika programskog jezika *Jave* nam govori da se radi o jednom od najkorištenijih programskih jezika. *Javu* su razvili James Gosling i Patrick Naughton u tvrtki *Sun Microsystems*. Razvoj je trajao 4 godine kao dio projekta *Green* s početkom 1991. godine te je objavljen u studenom 1995. godine. Prva verzija je pod nazivom *JKD1.0* objavljena 23.1.1996. dok je posljednja pod nazivom *Java SE 14* objavljena 17.3.2020.
Osnivatelji dok su stvarali Javu postavili su pet primarnih ciljeva, a to su:

- Mora biti jednostavan, objektno orijentiran i sličan ostalim jezicima.
- Mora biti čvrst i siguran.
- Mora biti prenosiv i neutralan za arhitekturu.
- Mora se izvršavati s visokim performansama.
- Mora se moći interpretirati.

Dizajnirali su sintaksu po uzoru na C/C++ jezik zato što je C jezik bio svim programerima od prije poznat. Sav kod se piše u klasama, a svaki podatak je objekt. Java ne podržava višestruko nasljeđivanje klasa i preopterećivanje operatora dok C++ podržava. [12]

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // ispisuje string u konzolu
    }
}
```

Sl. 3.7 primjer koda u jeziku Java

Na slici 3.7 vidimo jednostavan primjer programskog jezika Java koji je ujedno i najpoznatiji primjer širom svijeta. U primjeru se nalazi javna klasa pod nazivom *HelloWorldApp* te je njena jedina funkcionalnost da ispiše tekst *„Hello World“* u konzolu.

Na slici 3.8. vidimo ključne ili rezervirane riječi u Java jeziku. Rezervirane riječi su riječi koje se ne mogu koristiti kao objekti ili nazivi varijabli u Java jeziku jer se već koriste u sintaksi Java programskog jezika. Koriste se za unutarnje postupke ili predstavljaju neke unaprijed definirane radnje. Ako se upotrijebi bilo koja od navedenih riječi kao identifikator dobit će se greška i neće se moći izvršiti radnja. Iako se *goto* i *const* više ne koriste u Java jeziku, one se i dalje ne mogu koristiti kao ključne riječi.

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Sl. 3.8. ključne riječi u Javi

Varijable u Javi spremaju samo jednu određenu vrstu podataka i nijednu drugu. Svaki krivi unos će rezultirati tako da kompajler izbacuje sintaksnu grešku. U programskom jeziku *Java* postoje dvije kategorije tipova podataka koje dijelima na primitivne ili jednostavne i neprimitivne ili složene tipove. Primitivne tipove podataka dijelimo na 4 vrste, a to su cjelobrojni (*byte, short, int, long*), brojevi s pomičnim zarezom (*float* i *double*), znakovni (*char*) te logički (*boolean*). Dok se neprimitivni dijele na nizove, klase i sučelja.

Cjelobrojni tipovi podataka koriste se za prikaz cijelih brojeva koji mogu biti pozitivni i negativni. Najmanji cjelobrojni tip podataka je *byte* koji se koristi za čitanje iz datoteka i komuniciranje putem mreže. Najslabije se koristi tip *short* dok sa druge strane je tip *int* najčešće korišten. Uglavnom se koristi za brojače petlji i indekse nizova. Tip *long* koristimo za prikaze dugačkih cijelih brojeva koji prelaze granice tipa *int*.

Brojevi s pomičnim zarezom ili realni brojevi se koriste za sva izračunavanja kod kojih treba povećati preciznost tako da se prikaže i decimalni dio. Tip *float* osigurava jednostruku preciznost koja je pogodna za 32 bitna računala dok tip *double* osigurava dvostruku preciznost koja je pogodna za 64 bitna računala. Tip *double* je precizan kad je riječ o vrlo velikim ili vrlo malim vrijednostima dok tip *float* nije dovoljno precizan. Tip *char* sadržava jedan znak iz *Unicode* skupine znakova, a svaki taj znak zauzima 16 bitova. Tip *boolean* može sadržavati samo dvije vrijednosti *true* ili *false* odnosno 1 ili 0.

Na slici 3.9 vidimo prikaz primitivnih tipova podataka sa informacijama o njima kao što su osnovna vrijednost koja je početna kada deklariramo varijablu, veličina koju zauzimaju u memoriji te minimalna i maksimalna vrijednost koju mogu doseći. [13]

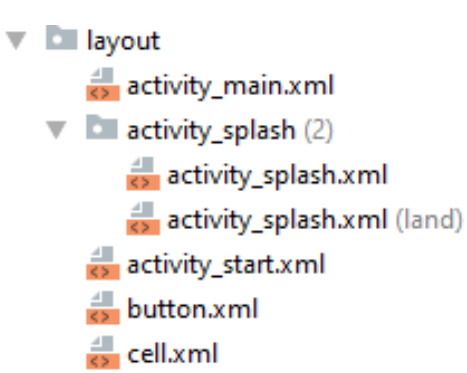
označava da slijedi hexdecimalni kod određenog slova

	tip	sadrži	osnovna vrijednost	veličina	min. vrijednost	maks. vrijednost
logički	**boolean**	true ili false	false	1 bit	N.A	N.A.
	char	Unicode character	\u0000	16 bits	\u0000	\uFFFF
cjelo-brojni	**byte**	signed integer	0	8 bits	-128	127
	short	signed integer	0	16 bits	-32768	32767
	int	signed integer	0	32 bits	-2147,483,648	2147,483,647
	long	signed integer	0	64 bits	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
broj s pomičnim zarezom	**float**	IEEE 754 floating point	0.0	32 bits	-3.40292347E+38	3.40292347E+38
	double	IEEE 754 floating point	0.0	64 bits	-1.7976931E+308	1.7976831E+308

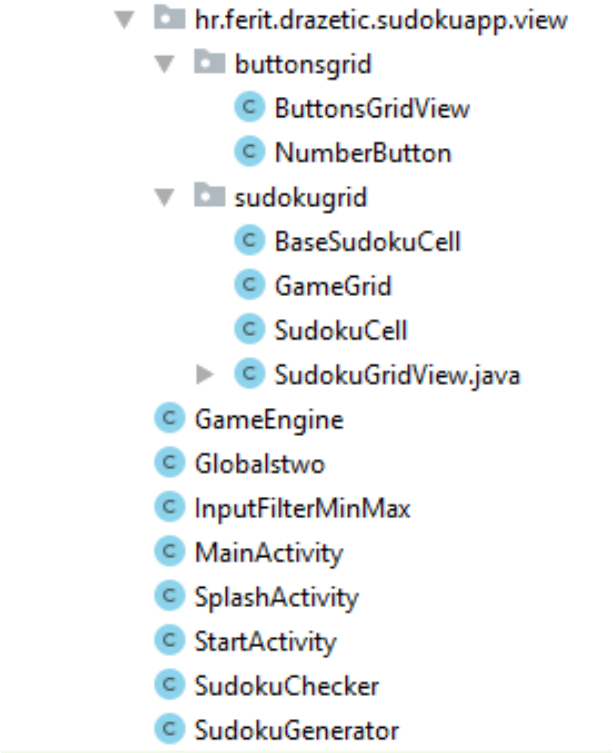
Sl. 3.9 Primitivni tipovi podataka u programskom jeziku Javi

4. MOBILNA APLIKACIJA

Mobilna aplikacija Sudoku je pisana u softverskom okruženu *Android Studio*, dok je sav kod napisan u programskom jeziku *Java* te u proširivom jeziku za označavanje podataka *XML (eng. Extensible Markup* Language). XML-om zapisujemo dokumente i podatke u tekstualnom formatu. XML datoteke se mogu vidjeti na slici 4.1. Kod aplikacije je raspodijeljen u 14 klasa koje se mogu vidjeti na Slici 4.2. Dvije od tih 14 klasa su *activity-ji* odnosno aktivnosti pod nazivom *StartActivity* i *MainActivity*, dok je jedna od klasa *Splash* aktivnost pod nazivom *SpashActivity*. Klase aktivnosti su ključne komponente Androidove aplikacije, a način pokretanja i sastavljanja aktivnosti osnovni je dio modela aplikacijske platforme. Aktivnost je prikaz jednog zaslona aplikacije. *Splash* aktivnost se pojavljuje pri pokretanju aplikacije i obično nam govori pojedinosti o aplikaciji koja se pokreće i prikazuje logo aplikacije. Dok se prikazuje *Splash* aktivnost u pozadini se učitava aplikacija i tu se poništava kašnjenje između pokretanja aplikacije i trenutka kada se pojavi prvi zaslon aplikacije odnosno aktivnost.



Sl. 4.1. XML datoteke



Sl. 4.2. Klase u aplikaciji

4.1. Stvaranje i provjera točnosti tablice sudoku

Dvije najvažnije klase unutar aplikacije koje su ujedno i „mozak“ cijele aplikacije su *SudokuGenerator* i *SudokuChecker*. Obje klase su singletoni. Singleton se koristi za kreiranje jedne instance klase koja se koristi u čitavoj aplikaciji. Dijelove sudoku generatora možemo vidjeti na slikama 4.3.- 4.4. Sudoku generator je zadužen da prilikom pokretanja aplikacije izradi različitu nasumično popunjenu točno riješenu sudoku tablicu te nasumično izabire i očisti određen broj polja koje naknadno korisnik popunjava. Na slici 4.3. vidimo javnu metodu *generateGrid()* kojom se vrši nasumično unošenje brojeva od 1 do 9 u polje. Metoda radi tako što izabire nasumičan broj i ako broj zadovoljava sve kriterije koji se provjeravaju funkcijom *checkConflick*() broj se sprema u to polje i nastavlja se dalje do idućeg polja gdje se ponovno vrši ista radnja. U slučaju da broj ne zadovolji kriterije, bira se novi nasumični broj dok ih ne zadovolji.

```
public class SudokuGenerator extends Globalstwo {

    private static SudokuGenerator instance;
    private ArrayList<ArrayList<Integer>> Available = new
ArrayList<ArrayList<Integer>>();
    private Random rand = new Random();
    public SudokuGenerator(){}

    public static SudokuGenerator getInstance(){
           //...
    }
    public int[][] generateGrid(){
        int[][] Sudoku = new int[9][9];
        int currentPos = 0;
        while( currentPos < 81 ){
            if( currentPos == 0 ){
                clearGrid(Sudoku);
            }
            if( Available.get(currentPos).size() != 0 ){
                int i = rand.nextInt(Available.get(currentPos).size());
                int number = Available.get(currentPos).get(i);
                if( !checkConflict(Sudoku, currentPos , number)){
                    int xPos = currentPos % 9;
                    int yPos = currentPos / 9;
                    Sudoku[xPos][yPos] = number;
                    Available.get(currentPos).remove(i);
                    currentPos++;
                }else{
                    Available.get(currentPos).remove(i);
                }
            }else{
                for( int i = 1 ; i <= 9 ; i++ ){
                    Available.get(currentPos).add(i);
                }
                currentPos--;
            }
        }
        return Sudoku;
    }
```

Sl. 4.3. metoda za stvaranje tablice

Na slici 4.4 se nalazi metoda *removeElements*() čija je funkcionalnost da očisti točno određeni broj polja nasumičnim izborom. Broj polja koji će biti obrisan određuje korisnik aplikacije koji upisuje broj u prvom zaslonu aplikacije. Taj broj se prenosi preko singleton klase *Globalstwo*() koja ima metode za dohvaćanje i postavljanje broja praznih polja. Na drugom zaslonu aplikacije će korisnik popunjavati prazna polja i time rješavati sudoku zagonetku. Metoda *clearGrid*() prazni sva polja Sudoku-a te popunjava polje *Available* brojevima od 1 do 9. Iz polja *Available* se nasumično popunjavaju polja brojevima.

```
public int[][] removeElements( int[][] Sudoku ){

        int i = 0;
        Globalstwo g=Globalstwo.getInstance();

        while( i <g.getData() ){
            int x = rand.nextInt(9);
            int y = rand.nextInt(9);
            if( Sudoku[x][y] != 0 ){
                Sudoku[x][y] = 0;
                i++;
            }
        }
        return Sudoku;
    }
    private void clearGrid(int [][] Sudoku){
        Available.clear();
        for( int y =  0; y < 9 ; y++ ){
            for( int x = 0 ; x < 9 ; x++ ){
                Sudoku[x][y] = -1;
            }
        }
        for( int x = 0 ; x < 81 ; x++ ){
            Available.add(new ArrayList<Integer>());
            for( int i = 1 ; i <= 9 ; i++){
                Available.get(x).add(i);
            }
        }
    }
```

Sl. 4.4. metode za brisanje elemenata

Na slici 4.5 se nalazi metoda *checkConflict*() koja se poziva u metodi *generateGrid*(). Metoda *checkConflict*() prima kao argumente trenutno popunjenu tablicu do te pozicije, broj koji se provjerava i poziciju u koju se treba upisati. Metoda provjerava pomoću tri pomoćne metode sve kriterije koji trebaju biti zadovoljeni da se broj upiše u trenutnu poziciju. Prve dvije metode su *checkHorizontalConflict*() i *checkVerticalConflict*() koje provjeravaju da se isti broj ne nalazi u trenutnom retku odnosno stupcu tablice. Metoda *checkRegionConflict*() je kompleksnija od prve dvije i ona provjerava da se već ne nalazi isti broj u tom 3x3 polju.

```
    private boolean checkConflict( int[][] Sudoku , int currentPos , final
int number){
        int xPos = currentPos % 9;
        int yPos = currentPos / 9;
        if( checkHorizontalConflict(Sudoku, xPos, yPos, number) ||
checkVerticalConflict(Sudoku, xPos, yPos, number) ||
checkRegionConflict(Sudoku, xPos, yPos, number) ){
            return true;
        }
        return false;
    }
    private boolean checkHorizontalConflict( final int[][] Sudoku , final int
xPos , final int yPos , final int number ){
        for( int x = xPos - 1; x >= 0 ; x-- ){
            if( number == Sudoku[x][yPos]){
                return true;
            }
        }
        return false;
    }



    private boolean checkVerticalConflict( final int[][] Sudoku , final int
xPos , final int yPos , final int number ){
        for( int y = yPos - 1; y >= 0 ; y-- ){
            if( number == Sudoku[xPos][y] ){
                return true;
            }
        }
        return false;
    }

    private boolean checkRegionConflict( final int[][] Sudoku , final int
xPos , final int yPos , final int number ){
        int xRegion = xPos / 3;
        int yRegion = yPos / 3;
        for( int x = xRegion * 3 ; x < xRegion * 3 + 3 ; x++ ){
            for( int y = yRegion * 3 ; y < yRegion * 3 + 3 ; y++ ){
                if( ( x != xPos || y != yPos ) && number == Sudoku[x][y] ){
                    return true;
                }
            }
        }
        return false;
    }
}
```

Sl. 4.5. kriteriji za dodavanje broja

Posao klase *SudokuChecker*, čiji se dijelovi koda nalaze na slikama 4.6.- 4.8., je taj da svaki puta kad korisnik unese broj u tablicu provjerava točnu riješenost sudoku-a. Provjera se vrši pomoću metode *checkSudoku()* koja u sebi sadrži 3 metode. Ako tablica zadovoljava ove 3 metode, metoda *checkSudoku*() vraća istinu (*eng. True*) i to znači da je točno riješen sudoku.

```
public class SudokuChecker {

    private static SudokuChecker instance;
    private SudokuChecker(){}

    public static SudokuChecker getInstance(){
        if(instance==null){
            instance=new SudokuChecker();
        }
        return  instance;
    }

   public boolean checkSudoku( int[][] Sudoku){
        return (checkHorizontal(Sudoku) && checkVertical(Sudoku) &&
checkRegions(Sudoku));
    }
```

Sl. 4.6. klasa SudokuChecker()

Metode *checkHorizontal*() i *checkVertical*() se nalaze na slici 4.7. Ove metode provjeravaju jesu li sva prazna polja popunjena i da se svaki pojedini broj pojavljuje samo jednom u svakom retku odnosno stupcu. U slučaju suprotnog odmah vraćaju netočno (*eng. False*) što znači da tablica ne zadovoljava kriterije.

```
   private boolean checkHorizontal(int[][] Sudoku) {
        for( int y = 0 ; y < 9 ; y++ ){
            for( int xPos = 0 ; xPos < 9 ; xPos++ ){
                if( Sudoku[xPos][y] == 0 ){
                    return false;
                }
                for( int x = xPos + 1 ; x < 9 ; x++ ){
                  if( Sudoku[xPos][y] == Sudoku[x][y] || Sudoku[x][y] == 0 ){
                        return false;
                    }
                }
            }
        }
        return true;
    }

    private boolean checkVertical(int[][] Sudoku) {
        for( int x = 0 ; x < 9 ; x++ ){
            for( int yPos = 0 ; yPos < 9 ; yPos++ ){
                if( Sudoku[x][yPos] == 0 ){
                    return false;
                }
                for( int y = yPos + 1 ; y < 9 ; y++ ){
                  if( Sudoku[x][yPos] == Sudoku[x][y] || Sudoku[x][y] == 0 ){
                        return false;
                    }
                }
            }
        }
        return true;
    }
```

Sl. 4.7. metode za provjeru redaka i stupaca

Metode *checkRegions*() i *checkRegion*() rade zajedno i provjeravaju ponavljanje broja u poljima 3x3 kojih ima 9 u cijeloj tablici. U slučaju da se nalaze dva ista broja u istom polju 3x3 metoda odmah vraća netočno, u suprotnome vraća istinu. Metode se nalaze na slici 4.8.

```
    private boolean checkRegions(int[][] Sudoku) {

        for( int xRegion = 0; xRegion < 3; xRegion++ ){

            for( int yRegion = 0; yRegion < 3 ; yRegion++ ){

                if( !checkRegion(Sudoku, xRegion, yRegion) ){
                    return false;
                }
            }
        }
        return true;
    }

    private boolean checkRegion(int[][] Sudoku , int xRegion , int yRegion){

        for( int xPos = xRegion * 3; xPos < xRegion * 3 + 3 ; xPos++ ){

            for( int yPos = yRegion * 3 ; yPos < yRegion * 3 + 3 ; yPos++ ){

                for( int x = xPos ; x < xRegion * 3 + 3 ; x++ ){

                    for( int y = yPos ; y < yRegion * 3 + 3 ; y++ ){

                        if( (( x != xPos || y != yPos) && Sudoku[xPos][yPos]
== Sudoku[x][y] ) || Sudoku[x][y] == 0 ){
                            return false;
                        }
                    }
                }
            }
        }
        return true;
    }
}
```

Sl. 4.8. metode za provjeru polja 3x3

4.2.Grafički izgled aplikacije

Aplikacija se sastoji od 2 zaslona te jednog *splash* zaslona koji se prikazuje samo pri pokretanju aplikacije kojeg možemo vidjeti na slici 4.9. Na prvom zaslonu aplikacije *StartActivity* (slika 4.10.) nalazi se *Edittext* u koji se unosi željeni broj praznih polja u Sudoku. Što je veći broj praznih polja to je teže za riješiti Sudoku. Na zaslonu se prikazuje broj praznih polja u odnosu na težinu rješivosti radi lakše orijentacije. Svaka tablica je generirana tako da je sigurno najmanje jedan način da se riješi, a negdje postoji i više načina. Upisivanje broja je ograničeno klasom *InputFilterMinMax* koja korisniku dozvoljava unos brojeva u rasponu između 1 i 80.

Sl. 4.9. izgled splash zaslona

Sl. 4.10. izgled početnog zaslona

Pritiskom na dugme *Započnite sudoku* korisnika prebacuje na drugi zaslon pod nazivom *MainActivity* (slika 4.11.) u kojem se generira tablica sa određenim brojem praznih polja. Na drugom zaslonu se nalazi tablica te 9 dugmadi koji sadrže brojeve od 1 do 9 te dugme *DEL* koje služi za brisanje broja iz odabranog polja. Kada se popuni cijela tablica i pomoću klase *SudokuChecker* se odredi da je točno riješena tablica, svi brojevi se zaključavaju i mijenja se boja cijele tablice u sivo. Ukoliko se je popunila cijela tablica, a tablica nije promijenila boju znači da je korisnik negdje pogriješio i mora naći grešku. Izgled riješene tablice se nalazi na slici 4.12. Pritiskom tipke na mobilnom uređaju za unazad se vraćamo na prvi zaslon gdje ponovno određujemo broj praznih polja.

Sl. 4.11. izgled glavnog zaslona

Sl. 4.12. izgled točno riješene tablice

Prilikom klika na željeno polje u Sudoku tablici, izabrano polje mijenja boju u *magenta* te broj u izabranom polju možemo uređivati. To smo uspjeli dobiti primjenom elementa *setOnItemClickListener*() (slika 4.13) koji nam služi za klikanje po zaslonu i nalazi se u klasi *SudokuGridView*. Bojanje polja radi tako što se pamti posljednja pozicija polja koja je izabrana u varijablu *previousPosition*. Idućim izborom polja se prethodno polje vraća u svoju prvotnu boju koja je bila spremljena u varijablu *ink*.

```
showGrid=(SudokuGridView) findViewById(R.id.sudokuGridView) ;
final int[] previousPosition = {-1};
final int[] ink = {0};

setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
        int x = position % 9;
        int y = position / 9;
        GameEngine.getInstance().setSelectedPosition(x,y);

     if(previousPosition[0]!=-1)
        {
         showGrid.getChildAt(previousPosition[0]).setBackgroundColor(ink[0]);
        }
        ColorDrawable cr=(ColorDrawable)
showGrid.getChildAt(position).getBackground();
        ink[0] =cr.getColor();
        showGrid.getChildAt(position).setBackgroundColor(Color.MAGENTA);
        previousPosition[0] =position;
    }
});
```

Sl. 4.13. setOnItemClickListener u klasi SudokuGridView

Klasa *GameGrid* (slike 4.14. i 4.15.) je klasa koja se brine za izgled i prikaz tablice aplikacije. U metodi *setGrid*() koja se nalazi na slici 4.14, postavljamo brojeve u tablicu koju ćemo prikazivati na zaslonu aplikacije te bojaju se polja s obzirom na položaj i sadržaj unutar polja. Ako je polje prazno onda se polje boja svjetlijim tonom boje kao što je prikazano na slici 4.11. Metodom *setNotModifiable*() postavljamo sva polja koja nisu prazna da se ne mogu uređivati i brisati, odnosno brojevi koji su prikazani na zaslonu aplikacije pri pokretanju korisnik ne može obrisati jer su oni zadani.

```
public class GameGrid {

    private SudokuCell[][] Sudoku = new SudokuCell[9][9];
    private Context context;

    public GameGrid( Context context ){
        this.context = context;

        for( int x = 0 ; x < 9 ; x++ ){
            for( int y = 0 ; y < 9 ; y++){

                Sudoku[x][y] = new SudokuCell(context);
            }
        }
    }

public void setGrid( int[][] grid ){

    for( int x = 0 ; x < 9 ; x++ ){
        for( int y = 0 ; y < 9 ; y++){

            Sudoku[x][y].setInitValue(grid[x][y]);

            if( grid[x][y] != 0 ){
                Sudoku[x][y].setNotModifiable();
            }
            if( grid[x][y] == 0 ){
                Sudoku[x][y].setBackgroundColor(Color.rgb(200,245,245));
            }
            if( grid[x][y] != 0 ){
                Sudoku[x][y].setBackgroundColor(Color.rgb(150,220,220));
            }

            if((x==4 || x==5 || x==3)){
                if (y==0 || y==1 || y==2 || y==6 || y==7 || y==8){
                    if( grid[x][y] != 0 ){

Sudoku[x][y].setBackgroundColor(Color.rgb(242,173,55));}
                    if( grid[x][y] == 0 ){

Sudoku[x][y].setBackgroundColor(Color.rgb(243,221,184));
                    }
                }
            }
            else if ((y==4 || y==5 || y==3)){
                if (x==0 || x==1 || x==2 || x==6 || x==7 || x==8){
                    if( grid[x][y] != 0 ){

Sudoku[x][y].setBackgroundColor(Color.rgb(242,173,55));}
                    if( grid[x][y] == 0 ){

Sudoku[x][y].setBackgroundColor(Color.rgb(243,221,184));
                    }
                }
            }


        }
    }
}
```

Sl. 4.14. metoda za postavljanje tablice

Na slici 4.15 se nalazi ostatak klase *GameGrid* koja sadrži metode za dohvaćanje i postavljanje elemenata u tablici te funkciju *checkGame*() koja koristeći funkciju *checkSudoku*() iz klase *SudokuChecker* provjerava točnost riješene tablice. U slučaju ako je točno riješena tablica poziva se metoda *setGridSolved*() koja cijelu tablicu zaključava i mijenja boju u sivo. Te se još izbacuje *toast* poruka sa natpisom da je riješen sudoku.

```
public void setGridSolved( int[][] grid ){
        for( int x = 0 ; x < 9 ; x++ ){
            for( int y = 0 ; y < 9 ; y++){
                if( grid[x][y] != 0 ){
                    Sudoku[x][y].setNotModifiable();
                    Sudoku[x][y].setBackgroundColor(Color.rgb(135,135,135));
                }
            }
        }
    }
    public SudokuCell[][] getGrid(){
        return Sudoku;
    }
    public SudokuCell getItem(int x , int y ){
        return Sudoku[x][y];
    }
    public SudokuCell getItem( int position ){
        int x = position % 9;
        int y = position / 9;
        return Sudoku[x][y];
    }
    public void setItem( int x , int y , int number ){
        Sudoku[x][y].setValue(number);
    }
    public void checkGame(){
        int [][] sudGrid = new int[9][9];
        for( int x = 0 ; x < 9 ; x++ ){
            for( int y = 0 ; y < 9 ; y++ ){
                sudGrid[x][y] = getItem(x,y).getValue();
            }
        }
        if( SudokuChecker.getInstance().checkSudoku(sudGrid)){

            setGridSolved(sudGrid);
            MainActivity.getmInstanceActivity().solvedSetText();
            Toast.makeText(context, "ČESTITAMO! Vratite se unazad da bi ste
ponovno zaigrali.", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Sl. 4.15. ostatak GameGrid klase

5. ZAKLJUČAK

Tema ovog završnog rada je razvoj mobilne aplikacije za rješavanje logičke zagonetke sudoku koja je pisana u softverskom okruženju Android studio programskim jezikom Java te *XML* jezikom kao proširivi jezik za označavanje podataka. U početku se moglo upoznati sa povijesti sudoku igre te pravilima iste. Nadalje, u završnom radu su opisani svi alati koji su korišteni za izradu mobilne aplikacije te razvoj mobilne aplikacije.

Glavna prepreka je bila napraviti generator koji stvara sudoku tablice. Uspješno je napravljeno pomoću funkcije koja je nasumično dodavala brojeve u tablicu pritom pazeći na određene kriterije koji su po propozicijama sudoku zagonetke.

Na kraju može se zaključiti da je izrađena aplikacija jednostavna za korištenje i dostupna je svima koji posjeduju android uređaj. Za razvoj ove aplikacije mogli smo koristiti i neko drugo razvojno okruženje poput *Fluttera* s kojim bi dobili slične rezultate. Mjesta za razvoj aplikacije još postoji kao što je naprimjer baza podataka koja prikuplja rezultate te brojač vremena koji odbrojava vrijeme koje je proteklo rješavajući određenu sudoku zagonetku.

LITERATURA

[1] Berthier, Denis, „The Hidden Logic of Sudoku“ (2007.g), LULU PR , str 76-86.

[2] G. McGuire, B. Tugemann, G. Civario. "There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem" https://arxiv.org/abs/1201.0749

[3] High clue tamagotchis (zadnji pristup 18.9.2020)

http://forum.enjoysudoku.com/high-clue-tamagotchis-t30020-135.html

[4] Felgenhauer, Bertram; Jarvis, Frazer (2005.g), Enumerating possible Sudoku grids

[5] *"NP complete – Sudoku" (PDF). Imai.is.su-tokyo.ac.jp.* (zadnji pristup 18.9.2020)

[6] Howard Garns, Wikipedia (zadnji pristup 10.7.2020.)

https://en.wikipedia.org/wiki/Howard_Garns

[7] Sudoku, Wikipedia (zadnji pristup 10.7.2020.)

https://en.wikipedia.org/wiki/Sudoku#History

[8] Sudoku- kako ga riješiti, eMUŠKARAC (zadnji pristup 15.7.2020)

https://www.emuskarac.com/razonoda/sudoku-kako-ga-rijesiti/

[9] Android (operacijski sustav), Wikipedia (zadnji pristup 23.7.2020)

https://hr.wikipedia.org/wiki/Android_(operacijski_sustav)

[10] Dosadašnje inačice sustava Android, Wikipedia (zadnji pristup 11.7.2020)

https://hr.wikipedia.org/wiki/Dosada%C5%A1nje_ina%C4%8Dice_sustava_Androida

[11] Android studio, Wikipedia (zadnji pristup 14.7.2020)

https://en.wikipedia.org/wiki/Android_Studio

[12] Java - povijest, Wikipedia (zadnji pristup 10.8.2020)

https://en.wikipedia.org/wiki/Java_(programming_language)#History

[13] Uvod u programiranje FESB, *varijable i primitivni tipovi*, (zadnji pristup 10.8.2020)

http://laris.fesb.hr/java/varijable.htm

SAŽETAK

Ovaj završni rad se bavi razvojem rješavanja sudoku zagonetke koja je razvijena za android platformu. U radu je opisana povijest sudoku zagonetke te osnova pravila koja su potrebna da se zagonetka može riješiti. U teorijskom dijelu su opisani alati koji su korišteni pri razvoju aplikacije, a to su softversko okruženje Android studio, android kao platforma i java kao korišteni programski jezik. U razvojnom dijelu su prikazani i objašnjeni korak po korak algoritmi koji su potrebni za stvaranje i provjeru točnosti sudoku zagonetke pritom pazeći na zadane kriterije. Na kraju se nalazi grafički izgled aplikacije koji se predstavlja korisniku.

Ključne riječi:

Algoritam, Android studio, Java, Sudoku

ABSTRACT

This project deals with the development of a sudoku puzzle solution developed for the android platform. Project describes the history of the sudoku puzzle and the basics of the rules needed to solve the puzzle. The theoretical part describes the tools used in the development of the application, which are the software environment Android studio, android as a platform and java as the programming language. The development section presents and explains step by step the algorithms needed to create and check the accuracy of sudoku puzzles while observing the given criteria. Finally, there is a graphical layout of the application that is presented to the user.

Keywords:

Algorithm, Android studio, Java, Sudoku

ŽIVOTOPIS

Danijel Dražetić je rođen 10. 09. 1998. godine u mjestu Požega, Hrvatska. Imena njegovih roditelja su Petar i Marica. Pohađao je Osnovnu školu fra Kaje Adžića u Brodskom Drenovcu odnosno od 5. do 8. razreda u Pleternici. Već u osnovnoj školi dobiva interes za matematikom i informatikom stoga upisuje Matematičku gimnaziju u Požegi. Svake godine redovno se natjecao na županijskim natjecanjima iz matematike i informatike i sudjelovao na HONI-ju (Hrvatsko otvoreno natjecanje u informatici). Nakon srednje škole upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija, smjer računarstvo. Posjeduje osnovno znanje programskih jezika C, C++, C#, osnovno znanje opisnog jezika HTML te razvoj android aplikacija u programskom jeziku Java.

Danijel Dražetić