Mobilna aplikacija Turingov stroj

Marić, Matija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MOBILNA APLIKACIJA TURINGOV STROJ

Diplomski rad

Matija Marić

Osijek, 2016.

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	
Studij, smjer:	
Mat. br. studenta, godina upisa:	
Mentor:	
Sumentor:	
Predsjednik Povjerenstva:	
Član Povjerenstva:	
Naslov diplomskog rada:	
Primarna znanstvena grana rada:	
Sekundarna znanstvena grana (ili polje) rada:	
Zadatak diplomskog rada:	
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: Postignuti rezultati u odnosu na složenost zadatka: Jasnoća pismenog izražavanja: Razina samostalnosti:

Potpis sumentora:	Potpis mentora:

Dostaviti:

1. Studentska služba

U Osijeku, godine

Potpis predsjednika Odbora:

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

IZJAVA O ORIGINALNOSTI RADA

Osijek,

Ime i prezime studenta:	
Studij :	
Mat. br. studenta, godina upisa:	

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

izrađen pod vodstvom mentora

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

1.1. Zadatak diplomskog rada

Izraditi mobilnu aplikaciju koja simulira rad Turingovog stroja te grafički pokazuje postupak simuliranja.

1.2. Organizacija rada

Ovaj rad objasnit će teorijske podloge za razumijevanje rada Turingovog stroja. Osim toga opisati će i ostale važne teorijske koncepte potrebne za razvoj simulatora Turingovog stroja na pametnom mobilnom uređaju. Svaki teorijski koncept obradit će se kroz nekoliko poglavlja gdje će se pobliže upoznati povijest, filozofija i najvažnije karakteristike koncepta. Tekst će biti popraćen i slikama radi lakšeg razumjevanja tematike.

Rad je sastavljen od sedam poglavlja. Drugo poglavlje opisuje Java tehnologiju koja se u praktičnom dijelu rada koristi kao primarni programski jezik i podloga za sve ostale komponente. Treće poglavlje sadrži osnovne principe rada Android aplikacije, njezine glavne komponente i opis izvođenja na mobilnom uređaju. U četvrtom poglavlju dan je pregled JavaScript programskog jezika i D3 *frameworka* koji su bili potrebni za grafičke simulacije u praktičnom dijelu rada. Peto poglavlje daje kratak pregled povijesti Turingovog stroja te je opisan način njegova rada. Šesto poglavlje opisuje arhitekturu mobilne aplikacije, njezino korištenje i moguće rezultate korištenja. Sedmo poglavlje sadrži zaključak u kojemu je opisan razvoj aplikacije, motivacija njene izrade i pregled svih korištenih tehnologija. Nakon zaključka dan je popis literature koja je korištena pri izradi ovog rada. Na samom kraju nalazi se kratak sažetak na hrvatskom i engleskom jeziku te životopis.

2. JAVA

Razvoj Java programskog jezika započeli su James Gosling, Mike Sheridan i Patrick Naughton u lipnju 1991. godine. Java je izvorno dizajnirana za interaktivne televizije, ali je taj koncept bio prenapredan za digitalne kabelske televizije tog vremena. Jezik, to jest projekt pod čijim nazivom je Java razvijana, početno je imao naziv *Oak,* prema drvetu hrasta koje se nalazilo izvan Goslingovog ureda. Poslije toga, projekt razvoja Java tehnologije preimenovan je u *Green*, te je naposljetku preimenovan u *Java*. Gosling je Javu dizajnirao s C/C++ sintaksom, jer su tadašnji programeri njome bili dobro upoznati.[1] Prva javna implementacija bila je *Java 1.0* 1995. godine. Koncept koji je postavljen kao cilj bio je „*Write Once, Run Anywhere"* (engl. „*piši jednom, pokreni bilo gdje"*) s besplatnim izvođenjima na popularnim platformama. Bila je prilično sigurna i njena sigurnost mogla se podesiti, pružajući mogućnost ograničenja pristupa mreži i datotekama. Glavni web preglednici ugradili su je nedugo nakon toga u svoje standardne konfiguracije pomoću sigurne *applet* konfiguracije. Nove verzije za velike i male platforme (J2EE i J2ME) razvijene su nakon dolaska nove verzije Javae, *Java 2.*

U 1997. godini Sun Microsystems (tvrtka u kojoj je i razvijen Java programski jezik, danas pod nazivom Oracle) pristupio je *ISO/IEC JTC1* standardnom tijelu, a kasnije i *ECMAi International,* s ciljem standardiziranja Javae no nakon kratkog vremena povukao se iz tog procesa. Java je tako tehnologija čiji standard kontrolira i postavlja Java zajednica. Većina Sunovih implementacija Jave je besplatno, a prihodi se ostvaruju uz pomoć specijaliziranih proizvoda kao što je *Java Enterprise System*. Najpoznatiji Java paketi su *Software Development Kit* (SDK) i *Runtime Environment* (JRE) koji je podskup SDKa. Glavna razlika je u tome što kod JREa prevoditelj nije prisutan. [2]

2.1. Filozofija

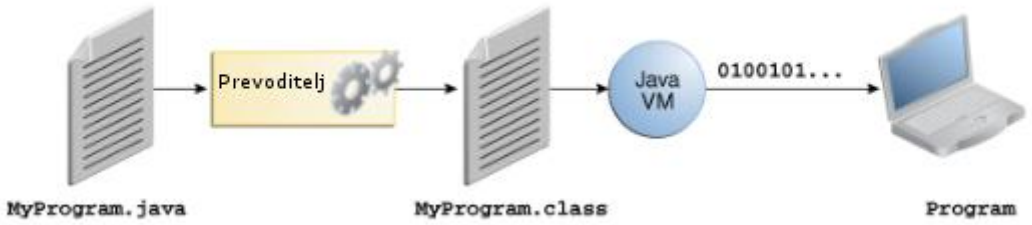
Pri razvoju Java programskog jezika postojalo je pet glavnih ciljeva i mogućnosti koje je jezik svakako trebao sadržavati, a to su:

- korištenje objektno orijentirane metodologije
- mogućnost pokretanja istog programa na više operacijskih sustava
- ugrađena podrška za korištenje računalnih mreža
- mogućnost sigurnog izvršavanja kôda s udaljenih izvora
- jednostavno korištenje kroz odabir dobrih dijelova drugih objektno orijentiranih programskih jezika.[2]

2.2. Java programski jezik

U Java programskom jeziku cijeli izvorni kôd pisan je tekstualnim datotekama koje završavaju *.java* nastavkom (engl. *extension*). Izvorne datoteke su tada uz pomoć Java prevoditelja prevedene (engl. *compiled*) u datoteke s *.class* nastavkom. One ne sadrže kôd koji je razumljiv procesoru računala nego *bytecode*, odnosno strojni jezik Java virtualnog stroja tj. Java VM (engl. *Java Virtual Machine, JVM*). Java alat za pokretanje tada pokreće aplikaciju s instancom JMVa. Postupak izvođenja Java kôda prikazan je na slici 2.1..

Kako je JVM dostupan na više različitih operacijskih sustava, ista *.class* datoteka može se pokrenuti na Windows, Solaris, Linux ili Mac operacijskim sustavima. Neki od virtualnih strojeva, kao što je *Java SE HotSpot*, omogućuju dodatne korake kako bi poboljšali izvođenje aplikacije. Ti koraci obuhvaćaju različite zadatke poput pronalaska dijelova kôda koji negativno utječu na brzinu izvršavanja aplikacije (engl. *bottlenecks*) i ponovno prevođenje u izvorni kôd dijelova često korištenog kôda.[3]



Sl. 2.1. *Prikaz izvođenja Java kôda.[3]*

2.3. Java platforma

Platforma je hardversko ili softversko okruženje u kojem se neki program izvršava. Veliki broj platformi može se opisati kao kombinacija operacijskog sustava i temeljnog sklopovlja. Java platforma se razlikuje od većine drugih platformi u tome što je to platforma koja se sastoji samo od softverskog sloja koji se izvodi povrh drugih hardverski zasnovanih platformi.

Java paltforma sastoji se od dva dijela:

- Java virtualnog stroja (JVM)
- Java aplikacijsko programsko sučelje ili API (engl. *Application Programming Interface*)

JVM je osnova Java platforme i detaljnije je objašnjen u prethodnom poglavlju. API je velika zbirka gotovih softverskih komponenti koja pruža mnoge korisne mogućnosti. Grupiran je u biblioteke srodnih razreda (engl. *classes*) i sučelja (engl. *interfaces*) koje se nazivaju paketi (engl.

packages). Na slici 2.2. prikazana je struktura Java platforme te je vidljivo kako API i JVM razdvajaju program od temeljnog hardvera.

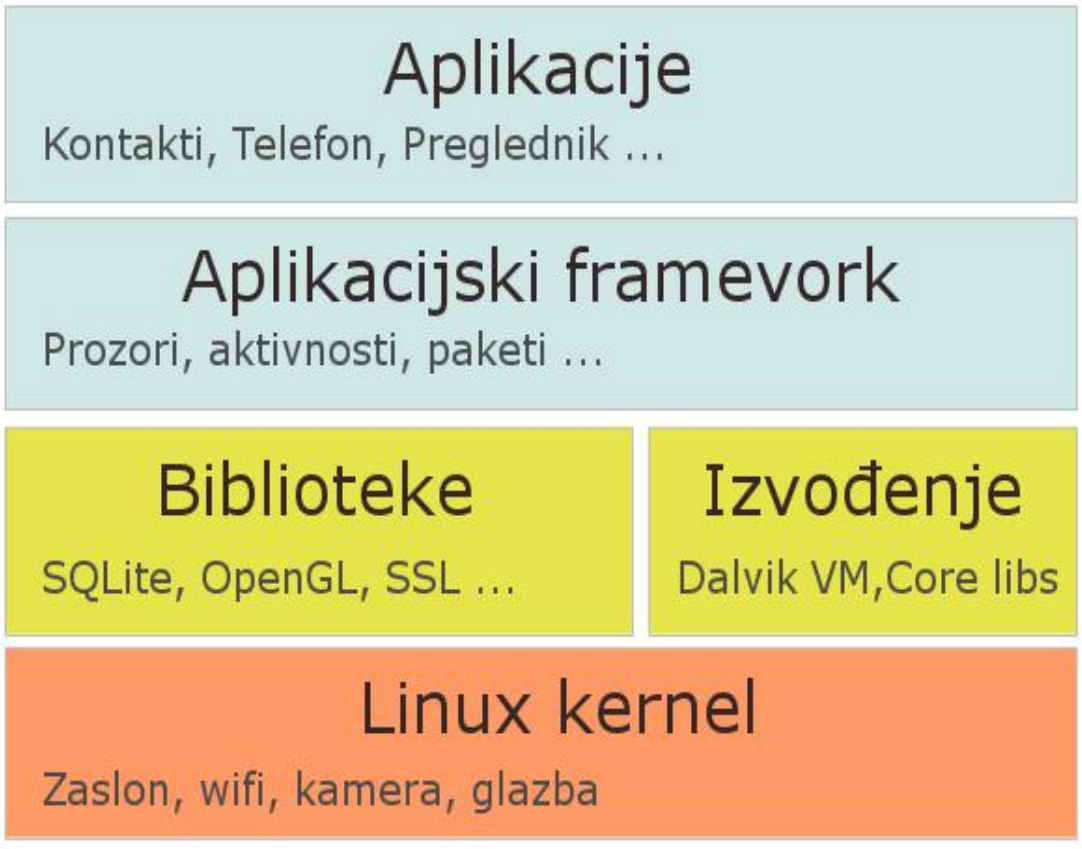
Sl. 2.2. *Prikaz strukture Java platforme. [3]*

Kao okruženje koje je hardverski neovisno, Java platforma može biti sporija od izvornog (engl. *native*) kôda. Međutim kako tehnologije prevoditelja i virtualnih strojeva napreduju, performanse Java platforme sve su bliže performansama izvornog kôda bez uskraćivanja neovisnost (engl. *portability*).[3]

3. ANDROID

Android je operacijski sustav baziran na Linux kernelu. Projekt koji je odgovoran za razvoj Android sustava naziva se „*Android open Source Project*“ ili AOSP i vođen je od strane Googlea. Android operacijski sustav može se podijeliti na četiri dijela kao što je prikazano na slici 3.1.. Pri razvoju Android aplikacija uglavnom se koriste gornja dva sloja. Pojedini dijelovi mogu se opisati na sljedeći način:

- Aplikacije – AOSP sadrži nekoliko početnih aplikacija: preglednik, kamera, galerija, glazba, telefon i mnoge druge.
- Aplikacijski *framework* – „*Application Programming Interaface*“ ili API koji omogućuje interakciju na visokoj razini između Android sustava i android aplikacija.
- Biblioteke (engl. *libraries*) i izvođenje (engl. *runtime*) – Biblioteke sadrže mnoge funkcije koje omogućuju dodatne mogućnosti Android operacijskog sustava kao što su spremanje podataka, pregledavanje web stranica u pregledniku, grafički prikazi i slično. Android *Runtime* i ostale Java biblioteke brinu se o pokretanju Android aplikacija.
- Linux kernel – Komunikacijski sloj između softvera i hardvera.[4]



Sl. 3.1. *Prikaz građe Android operacijskog sustava. [4]*

3.1. Povijest Androida

Android je osnovan u Palo Altou u Kaliforniji 2003. godine od strane Andy Rubina, Richa Minera, Nicka Searsa i Chrisa Whitea kako bi razvili pametnije mobilne uređaje koji su svijesni lokacije i potreba vlasnika uređaja. Prve namjere tvrtke bile su razviti operacijski sustav za digitalne kamere. Nakon shvaćanja da je tržište digitalnih kamera premalo, poduzeće je usmjerilo svoja nastojanja u razvijanje operacijskog sustava za pametne mobilne uređaje koji bi postao konkurencija tadašnjem Symbian i Microsoft Windows Mobile operacijskim sustavima.

Unatoč dostignućima osnivača i ranih zaposlenika, Android Inc. djeluje u tajnosti otkrivajući samo ono što je rađeno na softveru za mobilne uređaje. Iste godine Rubinu ponestaje novca za daljnji rad poduzeća. Blizak Rubinov prijatelj (Steve Perlman) tada mu nudi 10000$ gotovine u koverti i odbija udio u tvrtki.

U srpnju 2005. Google kupuje Android Inc. za najmanje 50 milijuna $, a ključni zaposlenici uključujući Rubina, Minera i Whitea nakon kupnje ostaju u tvrtci. U to vrijeme nije se puno znalo o Android Incu., ali mnogi su pretpostavljali da ovim potezom Google planira ulazak na tržište mobilnih uređaja. To se kasnije pokazalo istinitim jer je tim predvođen Rubinom razvijao platformu za mobilne uređaje pokretanu Linux kernelom. Google je tržištu Android platformu predstavio kao fleksibilan i nadogradiv sustav te je uz pomoć partnerstva s nekoliko hardverskih i softverskih tvrtki, tržištu i proizvođačima pokazao da je otvoren za suradnje na svim razinama.

Rani prototip s kodnim imenom „*Sooner*“ bio je sličan BlackBerry telefonu bez zaslona osjetljivog na dodir i sa fizičkom QWERTY tipkovnicom. Poslije je uređaju ugrađen odgovarajući zaslon kako bi se natjecao s drugim objavljenim pametnim uređajima u to vrijeme, LG Pradaom iz 2006. i Apple Iphoneom iz 2007. godine.

Dana 5. Studenog 2007. godine osnovan je Open Handset Alliance. To je konzorcij tehnoloških tvrtki uključujući Google, proizvođače uređaja kao što su HTC, Sony i Samsung, telekomunikacijske tvrtke poput T-Mobile i proizvođače čipova Qualcomm i Texas Instruments itd. Cilj konzorcija je razvijanje standarda za mobilne uređaje. Taj dan, Android je predstavljen kao prvi proizvod konzorcija, platforma za mobilne uređaja zasnovana na Linux kernelu. Prvi dostupni komercijalni uređaj bio je 22.10.2008. godine HTC Dream.[5]

3.2. Izvođenje izvornog kôda na Android uređaju

Java izvorni kôd uz pomoć Java prevoditelja prevodi se u Java *.class* datoteke. Android SDK sadrži alat naziva *dx*, koji pretvara Java *.class* datoteke u datoteke s *.dex* (engl. *Dalvik Executable*) nastavkom. Sve Java datoteke s *.class* nastavkom smještene su u *.dex* datoteku. Tijekom tog procesa pretvaranja, vrši se optimiziranje podataka unutar *.dex* datoteke. Na primjer, ako se ista varijabla nalazi u više različitih *.class* datoteka, *.dex* datoteka sadrži samo jednu referencu te varijable. *.dex* datoteke zbog toga su prema veličini puno manje u odnosu na odgovarajuće *.class* datoteke. Ove datoteke i ostali resursi Android projekta poput slika i XML datoteka, pakiraju se u datoteku s *.apk* (engl. *Android* Pacakge) nastavkom. Alat koji vrši stvaranje *.apk* datoteke naziva se *aapt* (engl. *Android Asset packaging Tool*). Rezultirajuća *.apk* datoteka sadrži sve potrebne informacije za pokretanje Android aplikacije te ju je tada moguće pokrenuti na Android uređaju.

Od verzije Androida 5.0, *„Android RunTime“* ili ART korišten je za pokretanje svih Android aplikacija. ART koristi *„Ahead Of Time compliation“*, to jest prevođenje ispred vremena izvođenja. Tijekom instalacije aplikacije na Android uređaj, aplikacijski kôd prevodi se u strojni kôd. Takav način rada rezultira u 30% više kôda za prevođenje, ali omogućuje brže izvođenje pri pokretanju. Osim bržeg izvođenja, ovim načinom rada štedi se baterija, jer se prevođenje vrši samo jednom i to u vremenu kada se aplikacija prvi puta pokrene. *Dex2oat* alat uzima prije nastalu *.dex* datoteku i mijenja i prevodi ju u *ELF* (engl. *Executable and Linkable Format*) datoteku. *ELF* datoteka sadrži *.dex* kôd, prevedeni izvorni kôd i meta podatke. Sadržavanje *.dex* kôda omogućuje da postojeći alati i dalje ispravno rade. Sakupljanje smeća to jest „*garbage collection*“ u ARTu optimizirano je tako da su vremena u kojima aplikacija ne daje odziv (engl. *freez*) uvelike smanjena.[4]

3.3. Dijelovi Android aplikacije

Android aplikacija sastoji se od sljedećih komponenti:

- Aplikacija (engl. *Application*) – Android aplikacija može imati jednu „*Application*“ klasu koja se instancira prije bilo koje druge Android komponente. To je zadnja komponenta koja je zaustavljena tijekom zaustavljanja aplikacije. Ako nije drugačije definirano, Android sam stvara početnu *„Application“* klasu.
- Aktivnost (engl. *Activity*) – aktivnost je vizualna prezentacija Android aplikacije. Android aplikacija može imati jednu ili više aktivnosti. Aktivnosti koriste poglede (engl. *views*) i fragmente (engl. *fragments*) kako bi stvorili korisničko sučelje putem kojeg je moguća interakcija s korisnikom.

- Odašiljač/Prijamnik (engl. *broadcast receiver*) – služi za slušanje sistemskih poruka i namjera (engl. *intents*). Prijamnik biva obaviješten sa strane Android sustava ako dođe do pojavljivanja određenog događaja. Na primjer moguće je osluškivati promijene stanja uređaja poput dolaznih poziva.
- Servis (engl. *Service*) – izvršava zadatke bez pružanja korisničkog sučelja te može komunicirati direktno s drugim Android komponentama. Na primjer odašiljač/prijamnik može putem obavještajnog Android *frameworka* obavijestiti korisnika o nekoj promijeni.
- Poslužitelj sadržaja (engl. *Content provider*) – poslužitelj sadržaja definira strukturirano sučelje za podatke aplikacije. Poslužitelj se može koristiti za pristupanje podatcima unutar aplikacije, ali se također može koristiti za dijeljenje podataka s drugim aplikacijama. Android sadrži SQLite bazu podataka koja se često koristi u sprezi s poslužiteljem sadržaja. SQLite baza podataka sprema podatke kojima poslije može pristupiti poslužitelj.
- Kontekst (engl. *Context*) – instance klase „*android.content.Context*“ omogućuje povezivanje prema Android sustavu i prema uređaju na kojem je aplikacija pokrenuta. Kontekst omogućuje pristupanje sustavskim i aplikacijskim resursima i servisima. Uz pomoć konteksta može se na primjer provjeriti veličina zaslona uređaja.[4]

4. JAVASCRIPT

JavaScript je višeplatformski, dinamički, objektno orijentirani skriptni jezik koji sadrži operatore, standardne ugrađene objekte i metode. Sintaksa je temeljena na Java i C programskim jezicima, pa tako JavaScript i navedeni jezici imaju puno sličnosti. Jedna od ključnih razlika je ta što JavaScript nema razrede (klase), umjesto toga JavaScript sadrži objektne prototipe (engl. *prototypes*). Sljedeća bitna razlika je da kod JavaScripta sve je objekt, pa su tako i funkcije objekti, dajući im tako mogućnost držanje kôda i prosljeđivanja kao svakog drugog objekta.

JavaScript sadrži standardne biblioteke objekata poput *„Array“, „Date“* i *„Math“* i osnovni skup elemenata kao što su operatori, kontrolne strukture i izjave. Osnovni JavaScript može se proširiti i upotrebljavati na razne načine uz pomoć dodavanja dodatnih objekata. Neki od njih su:

- Klijentski (engl. *Client*-side) Javascript koji proširuje osnovni JavaScript na način da dodaje objekte za upravljanje preglednikom i njegovim DOMom (engl. *Document Object Model*), to jest strukturom neke web stranice. Na primjer klijentske nadogradnje omogućuju aplikaciji dodavanje HTML elemenata i reagiranje na događaje izazvane sa strane korisnika poput klika mišem, unos podataka u formu ili kretanje kroz web stranicu
- Poslužiteljski (engl. *Server-side*) JavaScript proširuje osnovni JavaScript na način da dodaje objekte potrebne za pokretanje JavaScripta na poslužitelju. Poslužiteljske nadogradnje omogućuju aplikaciji komunikaciju s bazom podataka, osiguravaju kontinuitet informacija od jednog do drugog pozivanja aplikacije ili omogućuju manipulaciju datoteka na poslužitelju.[6]

Za razliku od većine programskih jezika JavaScript nema koncept ulaza ili izlaza. Dizajniran je tako da radi kao skriptni jezik u okruženju domaćina (engl. *host environment*) te je zadatak domaćina da osigura mehanizme za komunikaciju s vanjskim svijetom. Najčešće domaće okruženje je preglednik, ali JavaScript prevoditelji mogu se naći na puno drugih mjesta uključujući Adobe Acrobat, Adobe Photoshop, SVG slike, u poslužiteljskom okruženju poput Node.js ili NoSQL baze podataka kao projekt otvorenog kôda Apache CouchDB, u ugrađenim računalima (engl. *embedded computers*), u kompletnim desktop okruženjima kao što je GNOME, jedan od najpopularnijih grafičkih korisničkih sučelja ili GUIa (engl. *Graphics User Interface*) Linux operacijskog sustava i tako dalje.[7]

4.1. Povijest JavaScript

JavaScript je osmislio Brendan Eich 1995. godine dok je bio inženjer u Netscapeu te je prvi puta javno objavljen 1996. godine. Na početku se JavaScript trebao zvati *LiveScript* ali je preimenovan zbog marketinških odluka koje su pokušale iskoristiti popularnost Sun Microsystemovog Java programskog jezika. Unatoč sličnosti u nazivlju, jezici imaju vrlo malo zajedničkog te je to razlog konfuzije od tada.

Nekoliko mjeseci poslije Microsoft je objavio JScript s Internet Explorerom 3. JScript je bio uglavnom kompatibilan JavaScriptu. Nekoliko mjeseci poslije toga Netscap je dao zahtjev Ecma International-u, Europskoj organizaciji za postavljanje standarda, za standardiziranje JavaScript-a što je rezultiralo prvim izdanjem ECMAScript standarda iste godine. Standard je 1999. godine dobio značajne promjene s izdanjem ECMAScripta 3 te se od tada nisu dogodile veće promjene. Četvrto izdanje ECMAScript standarda napušteno je zbog političkih neslaganja u vezi kompleksnosti jezika. Mnogi dijelovi 4. izdanja koristili su se za stvaranje osnove 5. izdanja ECMAScript standarda objavljenog u prosincu 2009. i za osnove 6. izdanja objavljenog u lipnju 2015. godine.[7]

4.2. JavaScript i Java

JavaScript i Java su u nekim stvarima slični, dok su u drugima potpuno različiti. JavaScript programski jezik nalikuje Javai, ali ne sadrži Java statičke tipove i jaku provjeru tipa (engl. *strong type checking*). JavaScript slijedi većinu Java sintaksi, konvencija imenovanja i osnovne konstrukcije pri kontroli tokova.

Za razliku od Java prevoditeljskog sustava razreda izgrađenih od deklaracija, JavaScript podržava izvođački sustav zasnovan na malom broju tipova podataka koji predstavljaju brojčane, *bool* i *string* vrijednosti. JavaScript ima prototipno zasnovani (engl. *prototype based*) objektni model umjesto uobičajenog razredno zasnovanog objektnog modela. Prototipni zasnovani model omogućuje dinamično nasljeđivanje, što znači da ono što se nasljeđuje može biti različito za pojedine objekte. JavaScript također podržava funkcije bez nekakvih posebnih deklarativnih zahtjeva. Funkcije mogu biti svojstva (engl. *properties*) objekata izvršavajući se kao slobodne metode.

JavaScript je vrlo neformalan jezik u usporedbi s Javom. U JavaScriptu nije potrebno deklarirati sve varijable, klase i metode. Nije potrebno brinuti jesu li metode javne (engl. *public*), privatne (engl. *private*) ili zaštićene (engl. *protected*), te nije potrebno implementirati sučelja (engl. *interfaces*).

Java je razredno zasnovan programski jezik, dizajniran za brzo izvođenje s tipnom sigurnošću (engl. *type safety*). Tipna sigurnost znači da nije moguće skalirati Java cijeli broj (engl. *integer*) u referencu objekta ili pristupiti privatnoj memoriji pri oštećenju Java *bytecodea*. Java razredno zasnovani model znači da se programi isključivo sastoje od razreda i njihovih metoda. Nasljeđivanje i ostale karakteristike zahtijevaju usku vezu u razrednoj hijerarhiji. Navedeni zahtjevi i karakteristike čine Java programiranje kompleksnijim od JavaScript programiranja.[6]

4.3. D3

D3, D^3 ili d3.js je JavaScript biblioteka za stvaranje podatkovnih vizualizacija. Skraćenica D3 stoji za *Data Driven Documents* (hrv. dokumenti vođeni podacima). Podatke daje korisnik D3a, uglavnom programer, dok se dokumenti odnose na dokumente pri razvoju web stranica, a to su svi oni koji se mogu prikazati u web pregledniku poput HTML i SVG dokumenata. D3 je zadužen za vođenje u smislu povezivanja podatka i dokumenta. Glavni autor razvoja D3a uz neke druge pojedince je Mike Bostock. Projekt je otvorenog koda i potpuno besplatno dostupan na GitHub-u. D3 je objavljen pod BSD licencom pa ga je moguće u nekomercijalne i komercijalne svrhe koristiti, mijenjati i prilagođavati.

U osnovi D3 je softver koji olakšava stvaranje i manipulaciju web dokumenata s podacima, na način da:

- učitava podatke u memoriju preglednika
- veziva podatake s elementima u dokumentu stvarajući nove elemente po potrebi
- transformira prepoznate elemente dodavanjem odgovarajućih vizualnih svojstava
- upravlja stanjima elemenata kao odgovor na korisnički unos

Korak transformacije je najvažniji jer uz pomoć njega nastaje prikazivanje vizualizacija. D3 osigurava strukturu za primjenu transformacija, ali pravila prikazivanja definira sam programer. Programer odlučuje o vizualnom dizajnu, pa tako na primjer može odlučivati o visini osi, svijetlini krugova, kategoriji prema kojoj će biti prikazani podatci, koje boje će se koristiti i puno više. Programer daje koncept i stvara pravila, a D3 ih izvodi.

2005. godine Jeffrey Heer, Stuart Card i James Landay objavili su prefuse, alat za korištenje vizualizacija podataka na web stranicama. prefuse je pisan u Java programskom jeziku te je zahtijevao Java nadogradnje (engl. *plug-in*) kako bi se mogao izvoditi u web pregledniku. prefuse je prvi alat koji je omogućio jednostavnije prikazivanje vizualizacija na web stranicama. Do tada su sve vizualizacije bile individualni projekti. Dvije godine kasnije Jeffrey Heer objavio je Flare, alat sličan prefuseu koji je pisan u ActionScriptu, kako bi se vizualizacije mogle vidjeti

uz pomoć Adobe Flash Playera. Kao i kod prefusea, potrebno je bilo koristi nadogradnju preglednika. Flare je bio veliki napredak za prikazivanje vizualizacija u web stranicama. Razvojem web tehnologija postalo je jasno kako se vizualizacije podataka mogu razviti uz pomoć izvornih tehnologija preglednika bez upotrebe dodatnih nadogradnji. 2009. godine Jeffrey Heer preselio se u Stanford gdje je savjetovao studenta diplomskog studija Michaela Bostocka. U 2011. godini Mike Bostock, Vadim Ogievetsky i Jeffrey Heer službeno su objavili D3, najnoviji alat za razvoj web vizualizacija. D3 je za razliku od drugih alata koristio samo izvorne tehnologije i direktno upravljao nad web dokumentom.[8]

5. TURINGOV STROJ

5.1. Kratka biografija Alana Turinga

Alan Turing rođen je 23. lipnja 1912. godine u Londonu. U svom seminarskom radu iz 1936. godine dokazao je da ne može postojati univerzalni algoritamski postupak za određivanje istine u matematici te kako će matematika uvijek sadržavati neizvjesne mogućnosti. Rad je osim toga uveo Turingov stroj. Njegovi radovi na tu temu široko su priznati kao temelj istraživanja na području umjetne inteligencije.

U ranom djetinjstvu Turing pokazuje znakove visoke inteligencije koje neki od njegovih učitelja prepoznavaju, ali ne pridodaju im dovoljno poštovanje i važnost. Kada je Turing pristupio poznatoj i neovisnoj Sherborne školi u dobi od 13 godina postao je posebno zainteresiran za matematiku i znanost. Nakon Sherbornea, Turing je upisan na Kraljevsko Sveučilište u Cambridge-u, Engleskoj, te tamo studira od 1931. do 1934. godine. Kao rezultat njegovog rada u kojem je dokazao centralni granični teorem, nakon završetka studija Turing je odabran za asistenta na Sveučilištu. U 1936. godini Turing je objavio rad pod nazivom „*On Computable Numbers, with an Application to the Entscheidungsproblem*“ (hrv. Računanje brojeva s primjenom na *Entscheidungsproblem*) u kojem je predstavio ideju o univerzalnom stroju (poslije nazvanom Turingov stroj) sposobnim za računanje svega što je moguće računati. Središnji koncept modernih računala zasnovan je na Turingovom radu. Tijekom sljedeće dvije godine Turing je studirao matematiku i kriptologiju na institutu za napredno istraživanje u Princetonu u New Jersey-u. Nakon primitka doktorata od strane Princton Sveučilišta 1938. godine, vratio se u Cambridge. Tada je prihvatio posao od vlade i „*Cypher School“*, britanske organizacije za razbijanje kodova.

Tijekom Drugog svjetskog rata, Turing je bio vodeći sudionik pri razbijanju kodova tijekom rata, posebno njemačkih. Radio je u Bletchley Parku u kojem se nalazila ratna stanica gdje je napravio 5 glavnih napredaka na području kriptoanalize uključujući specificiranje uređaja pod nazivom „bombe“. To je bio elektromehanički uređaj korišten kao pomoć pri dešifriranju njemačkih kriptiranih signala pomoću „*Enigme*“. Turingov doprinos pri dešifriranju koda nije stao na tome te je napisao još dva rada o matematičkim pristupima razbijanja koda koji su postali važni dodaci „*Code and Cypher School*“ škole.

Turing se sredinom 1940-ih preselio u London te tamo počeo raditi za „*National Physical Laboratory*“. Među njegovim najznačajnijim doprinosima tijekom rada u Londonu bilo je vođenje dizajna ACEa ili „*Automatic Computing Engine*“, pri kojem je razvio nacrt za računala s mogućnosti pohrane programa. Iako potpuna verzija ACEa nikada nije izgrađena, koncept je

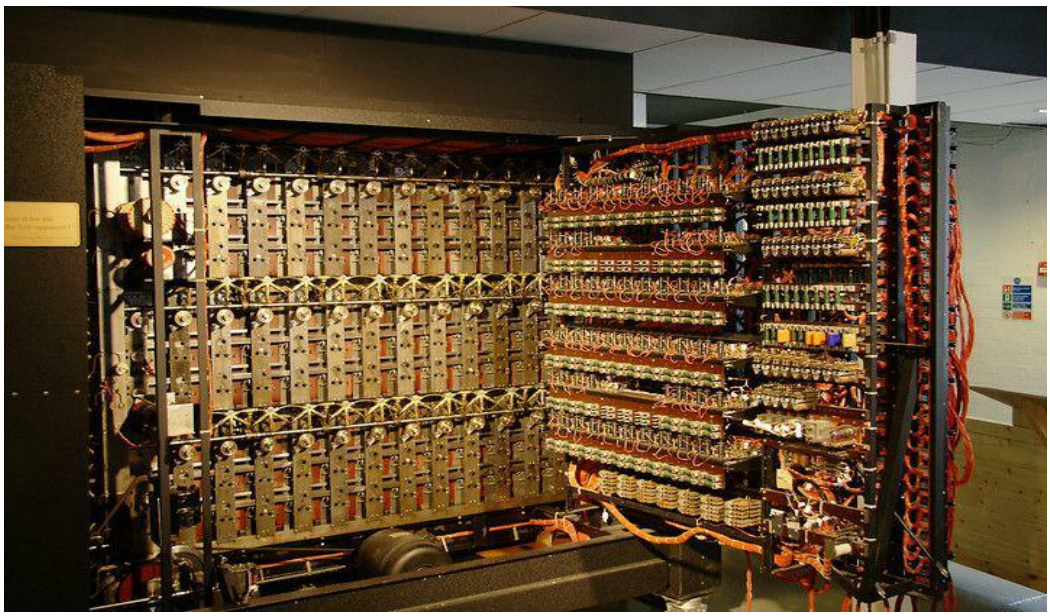
korišten kao model mnogih tehničkih kooperacija diljem svijeta tijekom nekoliko sljedećih godina. Utjecaj je imao na engleski *„Electric DEUCE“ i* američki *„Bendic G-15“* koji su od strane mnogih prozvani prvim svjetskim osobnim računalima među ostalim modelima.

Turing je držao visoke položaje i funkcije na matematičkom te poslije na računalnom laboratoriju na Sveučilištu u Manchsester-u u kasnim 1940-ima. Prvi puta se bavi pitanjem umjetne inteligencije u svom radu „*Computing machinery and intelligence*“ 1950. godine gdje je tada predložio pokus poznat kao Turingov test. Turingov test bio je nastojanje stvaranja standarda za inteligentni dizajn tehničke industrije. Tijekom posljednjih desetljeća test je značajno utjecao na rasprave o umjetnoj inteligenciji.[9]

5.2. Turingov stroj

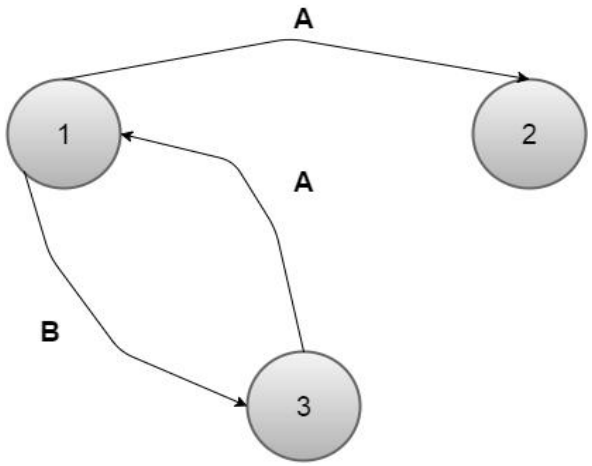
Dok je Babbageov stroj praktična stvar, Turingov stroj je na neki način stroj uma. Turingov stroj nije zamišljen kako bi računao tablice brojeva, nego za rješavanje problema u logici te da ispita granice računanja i ljudskih misli. Neovisno o svojoj jednostavnosti, Turingov stroj može simulirati bilo koji računalni algoritam bez obzira koliko on kompliciran bio.

Ukratko, Turingov stroj je konačni (engl. *finite*) automat s sposobnosti čitanja i pisanja podataka na vrpcu. Na slici 5.1. prikazan je jedan takav Turingov stroj. Konačni automat sastoji se od niza konačnih stanja. Kada se simbol postavlja na ulaz stroja, na primjer jedan znak iz neke abecede, on mijenja stanje na način da sljedeće stanje ovisi o trenutnom stanju i simbolu koji je postavljen na ulazu.[10]



Sl. 5.1. *Prikaz Turingovog stroja. [11]*

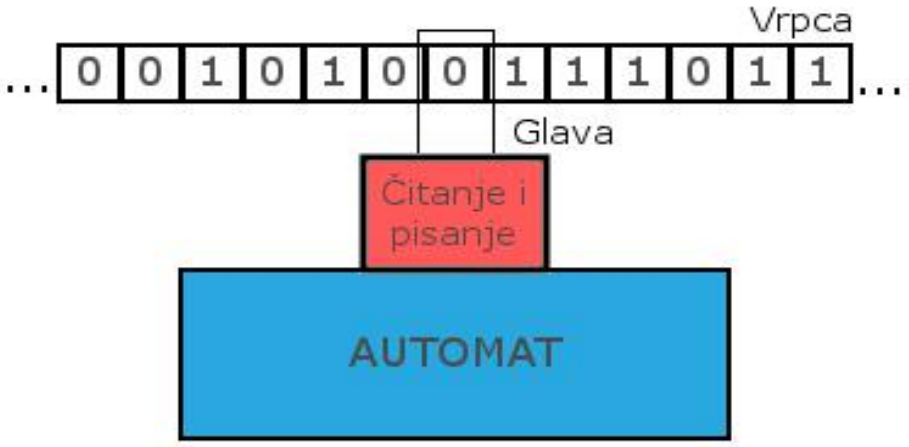
Za lakše razumijevanje, stroj s konačnim brojem stanja na slici 5.2. predstavljen je uz pomoć krugova gdje svaki krug predstavlja jedno stanje te strjelice koje pokazuju koje stanje slijedi za svaki simbol na ulazu stroja. Prema primjeru sa slike 5.2. stroj s konačnim brojem stanja ima tri stanja. Ako je uređaj u stanju 1 tada ga simbol na ulazu A pomiče u stanje 2 te ga simbol B pomiče u stanje 3.



Sl. 5.2. *Prikaz promijene stanja Turingovog stroja.*

Turingov stroj je automat s konačnim stanjem koji ima neograničenu zalihu papirne trake na koju može pisati i s koje može čitati uz pomoć glave (engl. *head*). Jednostavan primjer prikazan je na slici 5.3.. Postoje mnoge formulacije Turingovog stroja, ali u suštini stroj čita simbol s trake te ga tada koristi kao ulaz za konačni automat. Automat uz pomoć simbola na ulazu i trenutnog stanja čini tri stvari:

- Ispisuje nešto na vrpcu
- Pomiče traku lijevo ili desno za jednu ćeliju
- Postavlja se na novo stanje



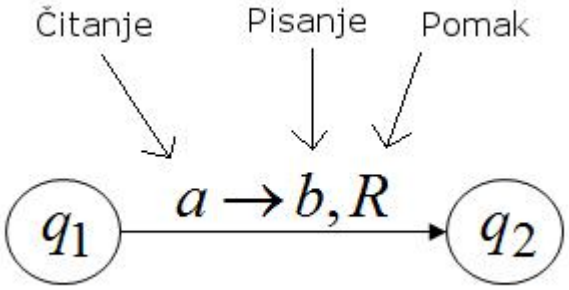
Sl. 5.3. *Prikaz dijelova Turigovog stroja.*

Turingov stroj može izvesti i posebnu akciju, a to je zaustavljanje ili *halt* te je to ponašanje vrlo važno. Na primjer Turingov stroj može prepoznati niz simbola napisanih na vrpci i zaustavlja se u posebnom stanju koje se zove konačno stanje (engl. *final state*). Bitno je to što postoje nizovi simbola koje Turingov stroj može prepoznati, a s druge strane obični konačni automat ne može prepoznati. Konačni automat može prepoznati niz koji sadrži tri simbola „A“ nakon kojih slijedi niz od tri simbola „B“ (AAABBB). Turingov stroj je moćniji u usporedbi s konačnim automatom jer zna brojati. Kako bi se otkrilo je li broj simbola „B“ i simbola „A“ jednak, potrebno je izvršiti nešto poput brojanja ili usporedbe. Ako uzmemo u obzir da može biti proizvoljno puno „A“ simbola prije nego dođe do prikaza simbola „B“, može se dogoditi da je potreban neograničen prostor za brojanje ili usporedbu. Konačni automat nema neograničen spremnik dok s druge strane Turingov stroj ima tu mogućnost uz pomoć svoje vrpce.

Potrebno je primijetiti da vrpca korištena kod Turingovog stroja nije beskonačna, nego samo neograničena što čini bitnu razliku. Turingov stroj je napravljen tako da ima vrlo dugačku, ali konačnu vrpcu i ako stroj ikada ostane bez vrpce moguće je dodati samo nove vrpce.[10]

5.3. Primjer rada Turingovog stroja

Na slici 5.4. prikazan je način grafičkog određivanja promjene stanja konačnog automata (engl. *state diagrams*), čija će se praktična izvedba u obliku mobilne aplikacija koja automatski izračunava i grafički prikazuje dijagram stanja detaljnije objasniti u praktičnom dijelu ovog rada.



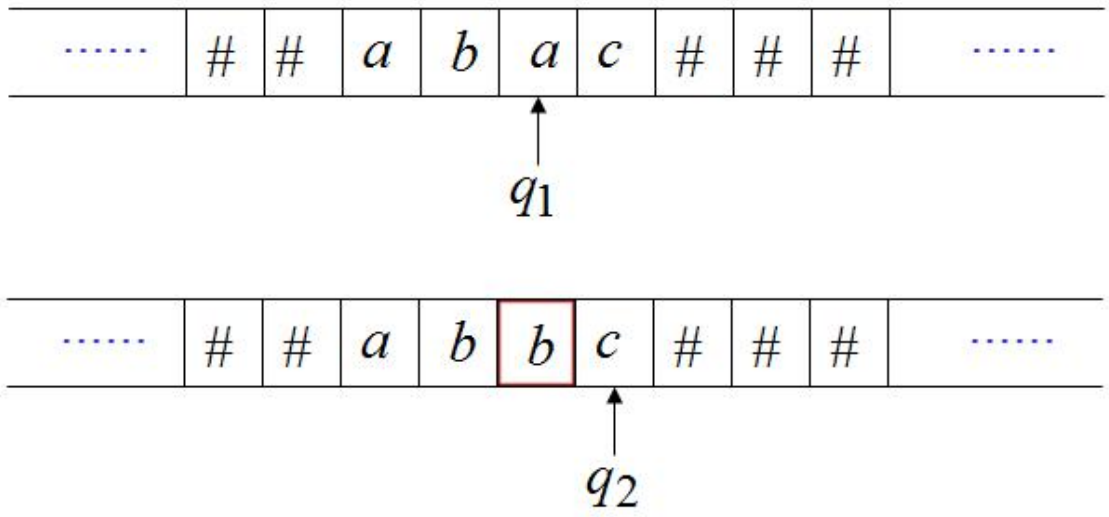
Sl. 5.4. *Grafički prikaz promjene stanja Turingovog stroja.*

q1 i q2 označavaju dva stanja. Kako bi Turingov stroj prešao iz stanja q1 u stanje q2, Turingov stroj obavlja sljedeće operacije:

- Turingov stroj, to jest glava za čitanje i pisanje, čita vrijednost u ćeliji vrpce te ona prema slici 5.4. mora biti jednaka “a”
- Tada se na mjestu gdje je zapisan simbol “a” upisuje nova vrijednost “b”

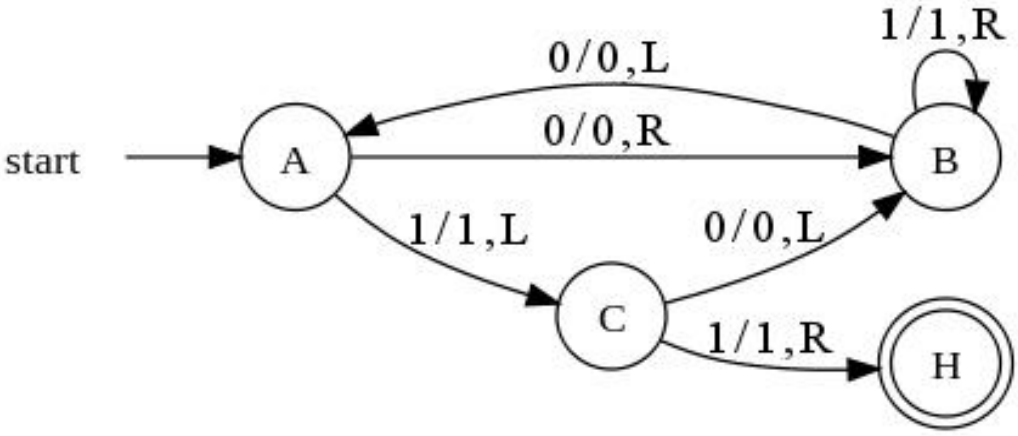
- Nakon toga se vrpca Turingovog stroja prema slici 5.4. pomiče za jednu ćeliju vrpce u desno (engl. „R“-*Right*).

Iz perspektive Turingovog stroja cjelokupna tranzicija iz stanja q1 u stanje q2 prikazana je na slici 5.5..



Sl. 5.5. *Jednostavan prikaz rada Turingovog stroja*

Na slici 5.6. prikazan je složeniji sustav dijagrama stanja. Dijagram osim početnog stanja „A“ ima i konačno stanje „H“ te samo ako sustav dođe do konačnog stanja „H“ može se zaključiti da sustav radi ispravno. Dijagram na slici 5.6. jednako se obrađuje kao jednostavniji primjer na slici 5.4..



Sl. 5.6. *Prikaz složenijeg dijagrama stanja.[12]*

Tok rješenja dijagrama stanja sa slike 5.6. prikazan je u tablici.

Tab. 5.1. *Tablica s rješenjima za dijagram stanja sa slike 5.6..[12]*

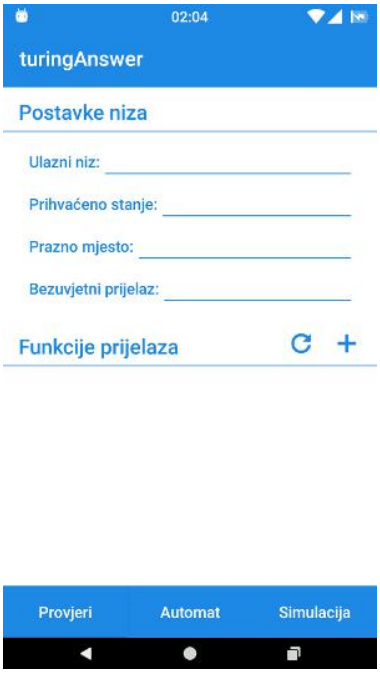
Slijed	Stanje	Glava i vrpca Turingovog stroja												
		0	0	0	0	0	0	**0**	0	0	0	0	0	0
1	A	0	0	0	0	0	0	**0**	0	0	0	0	0	0
2	B	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0
3	A	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
4	C	0	0	0	0	**1**	**1**	**0**	0	0	0	0	0	0
5	B	0	0	0	**1**	**1**	**1**	**0**	0	0	0	0	0	0
6	A	0	0	**1**	**1**	**1**	**1**	**0**	0	0	0	0	0	0
7	B	0	0	0	**1**	**1**	**1**	**1**	**1**	0	0	0	0	0
8	B	0	0	0	0	**1**	**1**	**1**	**1**	**1**	0	0	0	0
9	B	0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	0	0	0
10	B	0	0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	0	0
11	B	0	0	0	0	0	0	**0**	**1**	**1**	**1**	**1**	**1**	0
12	A	0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0
13	C	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0
14	H	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0

6. ARHITEKTURA APLIKACIJE

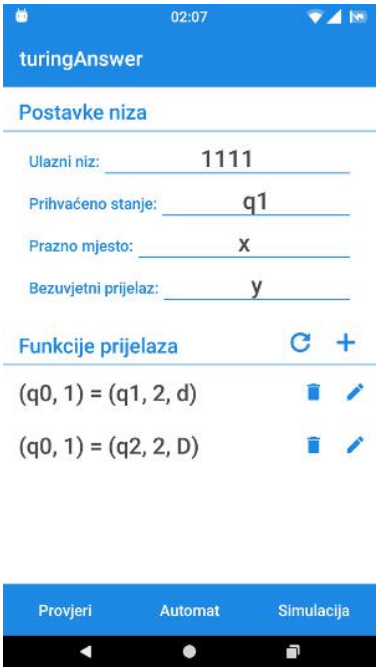
Aplikacija se sastoji od tri aktivnosti. Svaka od te tri aktivnosti podjeljena je na dva dijela, odnosno dvije klase. Prva klasa služi za postavljanje izgleda aktivnosti i njenu interakciju s korisnikom. Ona nasljeđuje drugu klasu koja je zadužena za logiku same aktivnosti. Takve klase nazivaju se baznim klasama. U njima se nalaze naredbe za manipulaciju nitima, poljima i ostalim varijablama.

6.1. Postavke niza i funkcije prijelaza

Postavke niza i funkcije prijelaza postavljaju se na početnom zaslonu, odnosno u prvoj aktivnosti koja se otvori pokretanjem aplikacije. Aktivnost je vizualno podijeljena na tri dijela što se može primjetiti na slikama 6.1. i 6.2.. Prvi dio odnosi se na postavke niza, drugi dio na funkcije prijelaza, a u trećem dijelu nalaze se tri gumba kojima se pokreće provjera zadanog zadatka, simulacija korak po korak ili iscrtavanje samog automata. Vrlo je važno podesiti sve parametre pravilno kako bi Turingov stroj mogao ispravno raditi.



Sl. 6.1. *Početni zaslon*



Sl. 6.2. *Popunjeni svi parametri*

U postavkama niza postoje četiri parametra koja se moraju ispuniti. Prvi parametar je ulazni niz. On postavlja traku stroja na zadane vrijednosti. Te vrijednosti se kasnije provjeravaju preko funkcija prijelaza i ostalih postavki niza kako bi se zaključilo da li je taj niz znakova prihvatljiv ili nije. Kako bi se znalo da li je niz prihvatljiv ili nije potrebno je zadati i konačno ili

prihvaćeno stanje automata po kojemu Turingov stroj radi. Konačno stanje je drugi parametar postavki niza. Treći parametar označava simbol praznog mjesta na traci. Ponekad provjera unesenog niza na traci zahtjeva brisanje nekog simbola trake pa se umjesto tog simbola onda upisuje simbol praznog mjesta. Drugi primjer je odlazak glave za čitanje simbola izvan granica niza tako da je potrebno zadati neki simbol koji se očekuje u tom slučaju. Zadnji parametar postavki je simbol bezuvjetnog prijelaza. Automat stroja može sadržavati mogućnost prijelaza iz jednog stanja u drugo bez da je pročitan neki poseban simbol već se taj prijelaz događa spontano ničim izazvan. Iz tog razloga simbol bezuvjetnog prijelaza mora biti postavljen kako bi stroj prepoznao ovu mogućnost automata po kojemu radi.

Funkcionalnost i izgled automata određuju funkcije prijelaza. One su zapravo skup pravila koje određuju mogućnost prijelaza iz jednog stanja u drugo te tako i mogućnost dolaska u konačno stanje. Funkcija prijelaza sastoji se od pet parametara koji moraju biti postavljeni. Parametri su sljedeći (s lijeva na desno): trenutno stanje, pročitani simbol, sljedeće stanje, simbol pisanja i pomak glave čitanja. Funkcija se može primjeniti samo ako su zadovoljena prva dva parametra, dakle ako je automat u odgovarajućem stanju i ako je na traci odgovarajući simbol. Ako se funkcija primjeni tada se koriste ostala tri parametra. Automat se prebacuje u sljedeće stanje, na traku se umjesto pročitanog simbola zapisuje novi simbol te se glava za čitanje trake pomiče lijevo ili desno od sadašnje pozicije.

Zadavanje postavki niza i funkcija prijelaza vrši se preko *Dialoga*. Svaka postavka ima neka ograničenja pri unosu vrijednost. Ograničenja kod postavki niza su sljedeća:

- ulazni niz nema nikakva ograničenja i dozvoljava unos svih znakova
- prihvaćeno stanje dozvoljava unos samo dva znaka, oba znaka moraju biti brojevi
- prazno mjesto dozvoljava unos samo jednog znaka, znak može biti bilo što
- bezuvjetni prijelaz dozvoljava unos samo jednog znaka, znak može biti bilo što

Ograničenja kod funkcija prijelaza su sljedeća:

- trenutno stanje dozvoljava dva znaka, znakovi moraju biti brojevi, barem jedna funkcija mora imati trenutno stanje 0 jer automat kreće s radom iz stanja q0
- pročitani simbol dozvoljava jedan znak, znak može biti bilo što
- sljedeće stanje dozvoljava dva znaka, znakovi moraju biti brojevi
- simbol pisanja dozvoljava jedan znak, znak može biti bilo što
- pomak glave čitanja dozvoljava jedan znak, znak može biti L, R, D, l, r ili d

Funkcije prijelaza stvaraju se pritiskom na gumb u obliku plusa. S lijeve strane tog gumba nalazi se gumb za brisanje svih funkcija. Osim brisanja svih funkcija, one se mogu brisati i pojedinačno.

Ako se uoči da je nešto krivo uneseno ili ako postoji neki drugi razlog izmjene nekog parametra, na svakoj funkciji prijelaza pored gumba za brisanje postoji i gumb za izmjenu u obliku olovke.

6.1.1. Provjera ulaznog niz

Provjera ulaznog niza pokreće se pritiskom na gumb *Provjeri*. Ono što se tu provjerava je prihvaćenost ulaznog niza, odnosno da li se unešenim ulaznim nizom može doći do konačnog stanja automata prema zadanim funkcijama prijelaza. Kako bi se provjera pokrenula, svi parametri postavki niza moraju biti popunjeni te mora postojati minimalno jedna funkcija prijelaza. Ako korisnik pokuša pokrenuti provjeru bez prethodno ispunjenih zahtjeva, prikazati će se *Toast* poruka s upozorenjem i provjera se neće izvršiti.

Prva operacija koja se izvrši pritiskom gumba *Provjeri* je postavljanje varijabli u pripadnoj baznoj klasi. Varijable su popunjene informacijama iz postavki niza i prijelaznih funkcija. Kada se varijable postave, pokreće se zasebna nit na kojoj se rade provjere. Na zasebnoj niti pokreće se *while* petlja koja se ne prestaje izvršavati dok se ne ostvari konačno stanje ili dok se ne zaključi da automat ne može doći u konačno stanje. Na samom početku *while* petlje nalazi se *if* blok koji provjerava u kojem stanju se automat trenutno nalazi. Ako je stanje konačno, kreirana nit se uništava i šalje se poruka korisniku kako je niz prihvaćen. U slučaju da stanje automata nije konačno, kreće se u potragu funkcije prijelaza koja bi se mogla iskoristit. Potraga se izvršava tako da se prođe kroz sve funkcije prijelaza i one pogodne se spremi u posebno polje. Pogodne funkcije su one koje trenutno stanje imaju jednako stanju automata te im je pročitani simbol odgovarajući ili imaju znak za bezuvijetni prijelaz. Ovisno o broju pogodnih funkcija odlučuje se o daljnjim postupcima. Ako je samo jedna funkcija odgovarajuća, ona se primjenjuje na automat i *while* petlja kreće ispočetka. Ukoliko su dvije ili više funkcija odgovarajuće, tada se radi novi objekt u koji se sprema trenutno stanje svih varijabli. Taj objekt sprema se u posebno polje istovrsnih objekata. Kada je spremanje trenutnih vrijednosti varijabli završeno, uzima se jedna od pogodnih funkcija prijelaza, primjeni se na automat i *while* petlja kreće ispočetka te se cijeli proces odvija kao i inače. Ovime se odabire jedan od više mogućih puteva koji su trenutno dostupni. Ako se u konačnici taj put pokaže kao neuspješan, uzima se objekt s spremljenim varijablama i trenutne vrijednosti varijabli vraćaju se na staro. Sada je moguće odabrati drugačiji put kroz automat u odnosu na onaj koji je odabran u prvom slučaju. Na ovaj način moguća je provjera svih odgovarajućih funkcija prijelaza i odabir one koja je najprikladnija. Kod pretraživanja odgovarajućih funkcija prijelaza moguće je i da niti jedna nije prikladna za trenutno stanje automata. To ukazuje na jedan od dva moguća zaključka. Prvi je da

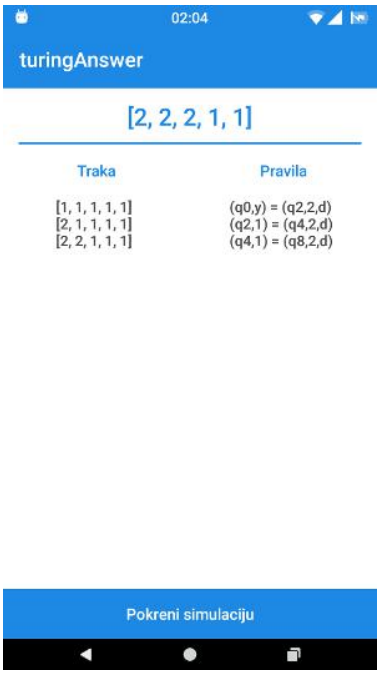
je odabrana kriva funkcija prijelaza od više odgovarajućih prilikom zadnjeg odabira. Drugi zaključak bi bio taj da automat ne može doći u konačno stanje zadanim ulaznim nizom i zadanim funkcijama prijelaza. Koji je od dva zaključka točan ovisi o polju u koje se spremaju objekti s trenutnim vrijednostima varijabli. Ako polje nije prazno, uzima se zadnji unešeni objekt, vraća se vrijednost varijablama iz tog objekta i *while* petlja kreće ispočetka. U slučaju da je polje prazno, onda automat ne može doći u konačno stanje, uništava se kreirana nit i korisniku se šalje poruka kako niz nije prihvaćen.

Kao što je prethodno utvrđeno, provjera ulaznog niza može završiti prihvaćanjem ili ne prihvaćanjem tog niza. U praksi postoji i treći rezultat provjere, a to je da se do rješenja ne može doći. Ovaj rezultat dolazi zbog mogućenosti beskonačne petlje. Ako je cjelokupni zadatak krivo postavljen ili ako se dogodi graška pri unosu nekog parametra lako je moguće da se dogode uvijeti koji prouzrokuju beskonačnu petlju. Ona se manifestira primjenom dvije ili više funkcija prijelaza kružno bez prestanka. Ako se ovakva pojava ne otkrije i ne zaustavi na vrijeme može doći do prekida rada cijele aplikacije. Gledano sa strane operacijskog sustava, dugotrajnim izvođenjem petlje prekomjerno se koriste resursi mobilnog uređaja poput RAM memorije i procesora te zbog toga operacijski sustav može donijeti odluku o prisilnom prekidanju rada aplikacije. Kako bi se spriječio prisilan prekid rada aplikacije u njoj je implementiran poseban algoritam koji ima zadaću prepoznati moguću beskonačnu petlju, prekinuti provjeru ulaznog niza i obavijestiti korisnika primjerenom porukom. Algoritam se zasniva na mjerenju vremena izvođenja cjelokupne provjere ulaznog niza, odnosno *while* petlje koja je opisana ranije u tekstu. Ako je vrijeme izvođenja petlje duže od dozvoljenog vremena, tada se zaustavlja rad petlje. Dozvoljeno vrijeme izvršavanja nije statično nego ono ovisi o broju funkcija prijelaza i vremenima izvođenja svake pojedine funkcije. Izraz preko kojega se dobije dozvoljeno vrijeme izvođenja glasi ovako: najdulje vrijeme izvođenja jedne funkcije prijelaza * ukupan broj funkcija prijelaza * 2. Mjerenje vremena izvođenja svake pojedine funkcije prijelaza vrši se prilikom same provjere ulaznog niza primjenom pojedine funkcije. To znači da maksimalno dozvoljeno vrijeme izvršavanja nije poznato prije početka provjere ulaznog niza i nije konstantno tijekom cijele provjere. Svakom prijelaznom funkcijom koja traje duže od prethodnih funkcija maksimalno vrijeme izvođenja se korigira i povećava kako bi se prilagodilo najdužem pojedinačnom vremenu. Najduže vrijeme izvršavanja pojedine funkcije dogodi se kada je moguće primjeniti više funkcija prijelaza te se stvara kopija svih varijabli potrebnih za provjeru ulaznog niza te se tek nakon stvaranje kopije izvršava jedna od pogodnih funkcija prijelaza. Ovakav način određivanja maksimalnog vremena izvršavanja je prilagodljiv na bilo koju brzinu procesora tako da bih se za isti zadatak na različitim uređajima trebalo dobiti odgovarajuće

maksimalno vrijeme specifično za uređaj na kojem se izvodi. Prema izrazu za dobivanje maksimalnog vremena izvođenja moglo bih se zaključiti kako se svaka funkcija prijelaza može izvršiti samo 2 puta što se čini kao mali broj ponavljanja. No uzimajući u obzir kako neki koraci u provjeri ulaznog niza (oni koji sadržavaju stvaranje kopija varijabli) traju puno duže od drugih koraka, a uz to su i puno rijeđi, broj mogućih ponavljanja funkcija prijelaza je zapravo veći od 2.

6.2. Korak po korak simulacija

Korak po korak simulacija slijedno prikazuje promjene na traci Turingovog stroja prilikom provjere ulaznog niza. Svaka promjena stanja automata prikazana je s razmakom od sekunde uz ispis poduzetih koraka. Svaki korak sadrži trenutno stanje trake i funkciju prijelaza koja je primjenjena u tom slučaju. Koraci su ispisani ispod trake stroja i ispisuju se u stupcima, lijevi stupac zadužen je za stanja trake, a desni stupac za ispis korištenih pravila. Raspored trake i stupaca moguće je vidjeti na slici 6.3..



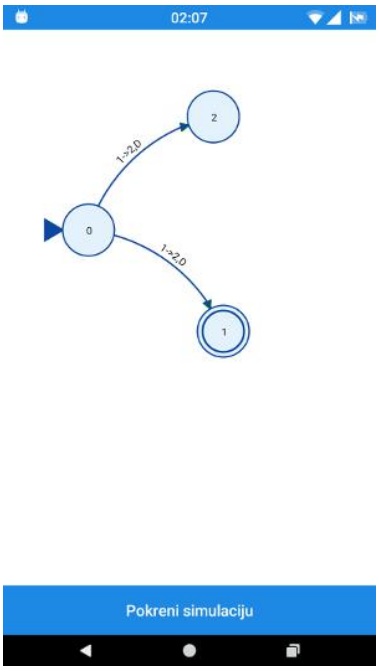
Sl. 6.3. *Simulacija korak po korak*

Odabir ove simulacije izvršava se na početnom zaslonu gumbom *Simulacija* pri dnu ekrana. Ovaj gumb moguće je stisnuti samo ako je provjera ulaznog niza prije toga napravljena. Ukoliko se nakon provjere mijenjala neka postavka niza ili funkcija prijelaza ova simulacija se ni tada neće moći izvršiti već je potrebno ponovno pokrenuti provjeru ulaznog niza i tek onda će biti omogućen pristup ovoj simulaciji. Razlog tomu je način na koji simulacija radi. Tijekom izvođenja ove simulacije nema ponovnog izvršavanja provjere ulaznog niza, već se zapravo

prikazuje rezultat provjere koja je prethodno napravljena. Tijekom provjere bilježi se redoslijed izvršavanja prijelaznih funkcija koji se sprema u posebno polje, a pritiskom gumba *Simulacija* to polje šalje se aktivnosti koja je zadužena za simulaciju. Na dnu zaslona simulacije nalazi se gumb *Pokreni simulaciju* koji radi ono što mu i sam naziv kaže, pokreće simulaciju.

Kao i kod provjere ulaznog niza, prije nego što se krene sa izvršavanjem, prvo se postavljaju varijable pripadne bazne klase. Najbitnija varijabla koja se u ovom koraku postavlja je upravo ona koja se prenosi otvaranjem ove aktivnosti, a to je polje s redoslijedom izvršavanja funkcija prijelaza. Kada je postavljanje gotovo pokreće se nova nit na kojoj se vrši simulacija. Ta nit blokira se svake sekunde kako bih se koraci izvršavanja ispisivali na zaslon ranije opisanim načinom. Na samom početku niti nalazi se *for* petlja koja prolazi kroz polje upotrebljenih funkcija prijelaza u provjeri. Nakon svake dohvaćene funkcije pripremaju se ispisi o stanju trake i dohvaćene funkcije trenutnog koraka. Nakon pripreme ispisa oni se i objavljuju, na automat se primjeni dohvaćena funkcija, nit se blokira na jednu sekundu i zatim sve ide ispočetka. Nit se zaustavlja i uništava kada se sve funkcije prijelaza obrade, a time završava i simulacija

6.3. Vizualizacija automata



Sl. 6.4. *Iscrtavanje automata*

Vizualizacija automata predstavlja grafički prikaz automata koji pruža mogućnost izvođenja simulacije na isto tako grafički način. Slika 6.4. prikazuje automat iz aplikacije koji je sastavljen od svjetloplavih krugova koji su spojeni s tamnoplavim strelicama. Iznad strelica je

ispisana prijelazna funkcija u skraćenom obliku. Ta funkcija omogućuje prijelaz između dva stanja koja strelica povezuje. Krug koji ima unutar sebe još jednu kružnicu je stanje koje se naziva prihvaćeno ili konačno stanje. Isto tako krug koji ima uza sebe mali trokut posebnog je naziva, a to je početno stanje. Početno stanje je uvijek nulto stanje te je iz tog razloga jako bitno da barem jedna od zadanih prijelaznih funkcija ima za trenutno stanje nulu. Broj svakog stanja upisano je unutar svih krugova.

Kako bih se pristupilo vizualnoj prezentaciji automata potrebno je pritisnuti gumb s nazivom *Automat* na početnom zaslonu aplikacije. Dostupnost ovog gumba, odnosno mogućnost njegovog pritiska, ovisi o istom zahtjevu kao i kod korak po korak simulacije. Provjera ulaznog niza mora biti izvršena i nakon nje ne smije se mjenjati niti jedan od parametara postavki niza i prijelaznih funkcija. Razlog tomu je vrlo sličan način izvođenja ove simulacije kao i korak po korak simulacije. Najvažnije varijable koje prima za ovo nadležna aktivnost su:

- polje svih funkcija prijelaza
- polje upotrebljenih funkcija prijelaza koje se dobije nakon odrađene provjere
- polje sa svim provjeravanim stanjima prilikom provjere niza (neovisno da li se preko njih dođe do krajnjeg rješenja)

Polje svih funkcija prijelaza potrebno je kako bih se mogao nacrtati cijeli automat, dok druga dva polja su potrebna za simulaciju.

Automat se crta pomoću skripte (internet stranice) napisane u JavaScriptu u kojoj je korištena D3 biblioteka. Ta skripta izvršava se u *WebView* elementu nadležne aktivnosti. Kako bi skripta nacrtala automat, ona mora dobiti podatke u JSON formatu. Iz tog razloga, prije nego se pripremi *WebView,* potrebno je iz polja svih prijelaznih funkcija generirati podatke o stanjima automata i njihovoj povezanosti. Kada su podatci generirani prvo se pokreće skripta koja u sebi ne sadrži nikakve podatke o automatu pa iz tog razloga za početak neće ništa biti iscrtano na ekranu. Kada je učitavanje gotovo, preko *WebViewa* poziva se jedna JavaScript funkcija kojoj se predaju generirani podatci i zatim se stranica osvježava. Cijeli ovaj postupak odvija se jako brzo stoga korisnik neće primjetiti da se prvo učitala samo stranica bez podataka i da je tek naknadno obavljeno postavljanje podataka i crtanje grafa. Nakon osvježavanja automat će biti nacrtan.

Za crtanje automata zadužena je opcija *force* iz D3 biblioteke. Njoj je potrebno predati podatke o čvorovima (stanjima automata) i linkovima (veze između stanja) u JSON formatu, dimenzije prostora za crtanje grafa, udaljenost između čvorova (dužina linka) i iznos sile kojom će se čvorovi međusobno odbijati kako se ne bih preklapali. Rezultat uporabe *forcea* je dijagram koji je bez obzira na broj čvorova i linkova uvijek pravilno i čitljivo iscrtan. Najveća prednost ovoga alata je ta što se pri crtanju grafa ne mora voditi računa o položaju čvorova na ekranu,

odnosno ne moraju se računati koordinate čvorova i ne mora se paziti na njihov raspored kako se pri crtanju veza među njima te veze ne bi preklapale. Osim krugova koji predstavljaju stanja automata i crta koje povezuju ta stanja, na grafu se još nalaze strijelice koje pokazuju iz kojega se čvora izlazi a u koji ulazi, unutar svakog čvora nalazi se broj koji označava o kojem je stanju riječ i iznad svakog linka piše prijelazna funkcija u skraćenom obliku. Linkovi u ovom slučaju nisu ravne linije već su dijelovi elipse, dakle blago su zakrivljeni. Ako se prelazi iz čvora koji je sa lijeve strane u čvor koji je sa desne strane tada je upotrebljen gornji dio elipse. Tekst na linku tada se nalazi sa gornje strane. U drugom slučaju, kada se iz čvora sa desne strane prelazi u čvor sa lijeve strane, tada se koristi donji dio elipse i tada je tekst sa donje strane. *Force* također nudi opciju manipulacije čvorovima grafa u stvarnom vremenu. Nakon što je graf nacrtan, moguće je pritiskom na čvor povući ga u bilo kojem smjeru i tako promijeniti izgled grafa. Pošto se ovaj graf, odnosno cijela internet stranica, prikazuje na mobilnom uređaju, manipulcija grafom je isključena. Razlog tomu je drugačija navigacija na mobilnim uređajima u odnosu na računalo, a u nekim slučajevima manipulacija grafom ne radi kako treba ili uopće ne radi.

U nadležnoj aktivnosti, ispod *WebView* elementa, nalazi se i gumb za pokretanje simulacije. Početak same simulacije obilježava postavljanje varijabli zadužene bazne klase. Zatim se pokreće posebna nit na kojoj će se simulacija i provoditi. Na početku izvođenja niti stvara se *for* petlja koja prolazi kroz polje svih provjeravanih stanja u provjeri ulaznog niza. Pri svakom dohvaćenom stanju provjerava se pripada li stanje nekoj od funkcija u polju koje sadrži korištene funkcije prijelaza (funkcije prijelaza koje vode do konačnog riješenja). Zatim se poziva JavaScript funkcija učitane internet stranice preko *WebViewa* kojoj predajemo vrijednost stanja i boju. Vrijednost stanja jednaka je identifikatoru kruga u automatu koji predstavlja to stanje. Boja se šalje kako bih se odabrani krug mogao obojati i time prezentirati trenutno označeno stanje. Označeno stanje je žute boje ako stanje pripada nekoj od funkcija u polju korištenih funkcija prijelaza, a ako ne pripada onda je roze boje. Kada se stanje oboja u željenu boju, nit se zaustavlja na jednu sekundu kako bi se mogla vidjeti tranzicija među stanjima. U protivnom simulacija bi se prebrzo izvela te korisnik ne bih ništa vidio. Stanja koja se oboje u žuto do kraja simulacije ostaju žuta te se na taj način prikazuje put do konačnog stanja. Stanja koja su obojana u rozo nisu dio puta do konačnog stanja i ona se u sljedećem koraku vraćaju u plavu boju. Ovaj proces se ponavlja dok nisu dohvaćena sva stanja. Na kraju simulacije u ovisnosti o tome da li je niz prihvaćen ili ne, stanje na kojemu automat stane biti će obojeno u crveno ako ono nije konačno, odnosno u zeleno ako je konačno.

7. ZAKLJUČAK

Ova aplikacija nastala je kao produkt kontinuiranog rada i unapređenja kroz dva kolegija tijekom studiranja. Inicijalno je napravljena kao praktični rad u sklopu kolegija pod nazivom Automati i formalni jezici koji se morao napraviti kako bi se položile laboratorijske vježbe. Tadašnja verzija imala je samo provjeru ulaznog niza uz neka ograničenja. Ograničenja su se odnosila na to da je aplikacija riješavala zadatke koji su se strogo svodili na deterministične automate i ostvarenje bezuvjetnog prijelaza nije bilo moguće. Dizajn aplikacije je bio dosta nepregledan, crno bijeli. Za potrebe kolegija Vizualizacija podataka aplikacija je doživjela radikalne promjene, kako funkcionalne tako i kozmetičke. Baza rada provjere ulaznog niza ostala je ista s dodatkom realizacije bezuvjetnog prijelaza. Od novih funkcija dodane su simulacija korak po korak i crtanje automata. Simulacija na automatu nije bila moguća, a izvođenje simulacije korak po korak uvelike se razlikovalo od današnje izvedbe. Prihvaćanjem ove aplikacije za diplomski rad, napravljenje su najveće promjene u radu aplikacije do tada. Cijela logika provjere ulaznog niza je ponovno osmišljena kako bi sada radila i na nedeterminističkim automatima, simulacija korak po korak je puno efikasnije napisana kao i crtanje automata kojemu je sada dodana i simulacija.

Osim prvotnog motiva za izradu ove aplikacije, daljnji motiv za unapređenje bio je u tome što je ova aplikacija jedinstvena na tržištu. Trenutno postoje neke aplikacije koje nude rješavanje Turingovih zadataka, ali niti jedna od njih ne omogućuje simuliranje i crtanje automata.

Dosta je tehnologija bilo potrebno kako bi ova aplikacija imala mogućnosti kakve ima sada. Programski jezici koji su korišteni su sljedeći: Java, XML, JSON, HTML, CSS, JavaScript i biblioteka D3. Od razvojnih okruženja koristili su se Eclipse (rane verzije aplikacije), Android Studio, Sublime i internet preglednik Chrome.

LITERATURA

[1] Wikipedia, Java (programming language), (https://en.wikipedia.org/wiki/Java_(programming_language)), pristup ostvaren 17.6.2016

[2] Free Java Guide & Tutorials, History of Java programming language, (http://www.freejavaguide.com/history.html), pristup ostvaren 16.06.2016

[3] Oracle, The Java Technology, (https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html), pristup ostvaren 15.06.2016

[4] Vogella GmbH, Android development with Android Studio – Tutorial, (http://www.vogella.com/tutorials/Android/article.html#introduction-into-development-of-android-applications), pristup ostvaren 19.06.2016

[5] Wikipedia, Android (operating system), (https://en.wikipedia.org/wiki/Android_(operating_system)), pristup ostvaren 20.06.2016

[6] MDN, JavaScript Guide, (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction), pristup ostvaren 22.06.2016

[7] MDN, A re-introduction to JavaScript (JS tutorial), (https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript), pristup ostvaren 22.06.2016

[8] Murray S., Interactive Data Visualization for the Web, O'Reilly Media, Inc., Sebastopol, 2013

[9] Biography.com, Alan Turing Biography, (http://www.biography.com/people/alan-turing-9512017), pristup ostvaren 24.06.2016

[10] I Programmer, James. M, What is a Turing Machine?, (http://www.i-programmer.info/babbages-bag/23-turing-machines.html), pristup ostvaren 24.06.2016

[11] Bug.hr, Jukić T., Godina Alana Turinga, (http://www.bug.hr/vijesti/godina-alana-turinga/113165.aspx), pristup ostvaren 24.06.2016

[12] Wikipedia, Turing machinge, (https://en.wikipedia.org/wiki/Turing_machine), pristup ostvaren 25.06.2016

SAŽETAK

Tema ovog diplomskog rada je mobilna aplikacija Turingov stroj. Glavni zadatak aplikacije je rješavanje Turingovih zadataka, crtanje automata i izvođenje simulacija rješavanja. Aplikacija je napravljena za Android operacijski sustav.

Ovaj rad sastoji se od dva dijela. Prvi dio opisuje korištene tehnologije i daje objašnjenje na pitanje što je Turingov stroj i kako se rješavaju Turingovi zadatci. Drugi dio daje odgovor na pitanje kako aplikacija radi.

Aplikacija je postupno razvijana kroz dva kolegija i diplomski rad. U nju je utrošeno puno vremena i kao rezultat toga je aplikacija koja je jedinstvena na tržištu po pitanju svojih mogućnosti.

Ključne riječi: Turingov stroj, Android aplikacija, računanje i simulacija, jedinstvena

ABSTRACT

Mobile application Turing machine

The topic of this paper is mobile application Turing machine. Main purpose of this application is solving Turing tasks, drawing automata and simulating solving techniques to user. Application is made for Android operating system.

This paper is divided into two parts. First part is all about describing used technologies and giving some answers about what is Turing machine and how to solve Turing tasks. The second part describes how mobile application is made.

Application is developed through two collage courses and this thesis. A lot of time is spent in this application and the result is unique application on market in terms of functionality.

Key words: Turing machine, Android application, solving and simulating, unique

ŽIVOTOPIS

Matija Marić rođen je 15.01.1992. u Bjelovaru. Osnovnu školu završio u Bizovcu s odličnim uspjehom.

2007. godine upisuje Elektrotehničku i prometnu školu Osijek u Osijeku, smjer elektrotehničar. 2011. godine završava srednju školu s odličnim uspjehom.

Iste godine upisao je preddiplomski studij elektrotehnike na Elektrotehničkom fakultetu Osijek. Nakon prve godine mijenja smijer studiranja i upisuje se u drugu godinu preddiplomskog studija računarstva. U predviđenom vremenu položio je sve ispite na preddiplomskom studiju.

Nakon preddiplomskog studija upisuje diplomski studij, smjer procesno računarstvo. Prvu godinu završava u roku te upisuje drugu godinu.

U međuvremenu odrađuje neobaveznu praksu u trajanju od dva mjeseca u Virovitici. Nakon prakse nastavlja sa radom od kuće za Plavu tvornicu, tvrtkom u kojoj je praksu i odradio.

Matija Marić