Android aplikacija Conwayeve igre života

Matijanić, Borna

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKLUTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski stručni studij Elektrotehnike, smjer Informatika

ANDROID APLIKACIJA CONWAYEVE IGRE ŽIVOTA

Završni rad

Borna Matijanić

Osijek, 2018.

SADRŽAJ

1. UVOD

Završni rad sastoji se od dvaju dijelova. U teorijskom se dijelu rada polazi od osnovnih biografskih crta autora igre koja je predložak rada. Razrađujući njezin kontekst pruža se uvid u strukturu, temeljna obilježja i konkretnu primjenu. Ta razrada se vrši na temelju podataka koje iznosi sam Conway, a na koje se referira i Martin Gardner u časopisu *Scientific American*.[1] Osim toga, teorijski dio rada donosi i rezime najvažnijih spoznaja o koncepciji *Igre života,* njezinim pravilima i uporabi, a popraćen je i pripadajućim ilustracijama koje sadrže odgovarajuća potrebna pojašnjenja. Drugi dio rada koncipiran je kao praktična primjena teorijski razložene tematike, a sadrži prikaz procesa izrade android aplikacije na temelju Conwayeve *Igre života*. U tim se poglavljima također nalaze konkretni primjeri izvornog koda te njihov tematski okvir i analiza. Time se nastoji realizirati mogućnost praktične primjene aplikacije sa svrhom što detaljnijeg uvida u strukturu *Igre života*. Na koncu se, na temelju navedenih primjera i analize teorijskih zapažanja, stječe zaključak o tim dvama pogledima na *Igru života* te se iskazuje značaj njezine uporabe.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je objasniti koncept i pravila igre života, njezin cilj i svrhu. Nakon objašnjenog cilja igre, idući korak je izrada android aplikacije u Java programskom jeziku. Svrha aplikacije je uvid u igru života, odnosno upravljanje simulacijom razvoja staničnih automata.

[1] Martin Gardner (1914. – 2010.) bio je američki matematičar te pisac popularne znanosti i rekreacijske matematike. Spomenuti članak o Conwayevoj *Igri života* objavio je u broju 223 znanstvenoga časopisa *Scientific American*, 1970. godine.

2. CONWAYEVA IGRA ŽIVOTA

U prethodnom poglavlju iznesen je zadatak završnog rada, a u poglavljima koja slijede prikazat će se koncept igre, s obzirom na njezina temeljna obilježja, funkciju i uporabu.

2.1. O autoru *Igre* života

John Horton Conway britanski je matematičar rođen 26. prosinca 1937. godine. Velik doprinos dao je u mnogim granama rekreacijske matematike, a osobito je poznat po izumu simulacije staničnih automata nazvanome *Igrom života*. Trenutno djeluje kao *professor emeritus*[2] matematike na Sveučilištu Princeton u New Jerseyu.

Rođen je u Liverpoolu, kao sin Cyrila Hortona Conwaya i Agnes Boyce. Zanimanje za matematiku pokazao je još u najranijoj dobi, a do jedanaeste godine života izrazio je ambiciju da postane matematičarom. Nakon završetka srednjoškolskoga obrazovanja, John Conway odlazi na studij matematike u Cambridge. S obzirom na činjenicu da je u dotadašnjem tijeku školovanja bio poznat kao nevjerojatno introvertiran adolescent, svoj je odlazak na Cambridge prepoznao kao mogućnost transformacije u novu osobu.

Prvostupničku diplomu stekao je 1959. godine te je započeo s provedbom istraživanja u teoriji brojeva pod mentorstvom Harolda Davenporta.[3] Riješivši otvoreni problem koji je Davenport postavio, pišući brojeve kao sumu petih potencija, počeo se zanimati za beskonačne nizove. Čini se da je njegovo zanimanje za igre započelo tijekom godina studiranja na Cambridgeu gdje je, provodeći sate igranja igara u zajedničkoj prostoriji, postao strastveni igrač tavle (engl. *backgammon*). Doktorirao je 1964. godine te je imenovan predavačem matematike na Sveučilištu u Cambridgeu.

Nakon napuštanja Cambridgea 1987. godine, prihvatio je imenovanje na poziciju *John von Neumann profesor u primijenjenoj i računalnoj matematici* na Sveučilištu Princeton na kojemu djeluje i danas.

---

[2] *Professor emeritus* počasno je zvanje koje se dodjeljuje zaslužnim redovitim profesorima sveučilišta u mirovini koji su se posebno istaknuli svojim znanstvenim ili umjetničkim radom, imaju posebne zasluge za razvoj i napredak sveučilišta te su ostvarili međunarodnu reputaciju na temelju međunarodno priznate nastavne, znanstvene ili umjetničke izvrsnosti.

[3] Harold Davenport (1907. – 1969.) engleski je matematičar poznat po svome opsežnom djelovanju u matematičkoj teoriji brojeva.

2.2. Igra života – temeljna obilježja

Igra života predstavlja simulaciju staničnih automata. Ideju za izradom igre Conway je dobio istražujući problematiku kojom se bavio John von Neumann, a koji je pokušavao pronaći Turingov stroj, odnosno matematički model izračuna kojim je definiran apstraktni stroj. Upravo zbog analogije s usponima i padovima te izmjena društva živućih organizama, ona pripada rastućoj klasi nazvanoj „igre simulacije“, odnosno igrama koje prikazuju procese nalik onima iz stvarnoga života. [1] Zanimljivo je i važno napomenuti da je prvu inačicu igre Conway osmislio na ploči za igranje dama (engl. *checkerboard*), pa Gardner prema [1] napominje da je za njezino igranje bila potrebna ploča odgovarajuće veličine te mnogo uzoraka elemenata (organizama) u dvjema bojama, za što su vrlo dobro služile male dame ili čipovi za poker. Istočnjačka „go ploča“ mogla je biti korištena ako su se mogli pronaći plošni predmeti dovoljno mali da mogu stati unutar ćelija („go kamenčići“ nisu se mogli rabiti jer nisu plošni). Osim toga, mogla se rabiti i olovka uz papir s kvadratićima, ali je bilo mnogo jednostavnije, osobito za početnike, koristiti predmete i ploču.

Temeljna ideja igre jest započeti s jednostavnom konfiguracijom elemenata (organizama) - jednoga po ćeliji, a zatim promatrati kako se oni mijenjaju dok se primjenjuju Conwayevi „genetički zakoni“ za rođenja, smrti i opstanke. [2] Conway je, nakon dugoga razdoblja eksperimentiranja, vrlo pažljivo odabrao relevantna pravila, a ona podrazumijevaju činjenicu da je potrebno je ispuniti tri cilja:

1. Ne bi trebao postojati nikakav početni uzorak za koji postoji jednostavan dokaz da populacija može rasti bez ograničenja.

2. Trebali bi postojati početni uzorci koji se izgledno mogu umnažati bez ograničenja.

3. Trebali bi postojati jednostavni početni uzorci koji se tijekom znatnoga razdoblja umnažaju i mijenjaju kako bi do završetka stigli na tri moguća načina: nestajući u potpunosti (zbog prenapučenosti ili pretjerane usamljenosti), naseljavajući se u stabilnu konfiguraciju koja ostaje nepromjenjiva i nakon toga, ili ulazeći u fazu oscilacije u kojoj se ponavlja beskonačan krug dvaju ili više stanja.

Drugim riječima, pravila moraju biti takva da učine ponašanje populacije nepredvidivim.

Igra života djeluje tako da, nakon što se postavi početno stanje igre, ona ne zahtijeva daljnje naredbe, već simulira ponašanje stanica putem već spomenutih definiranih pravila. Igra se odvija na dvodimenzionalnoj koordinatnoj mreži (engl. g*rid*). Nakon što se postavi početno stanje igre,

promatra se kako skupina ćelija evoluira u složeniju strukturu međusobno zavisnih ćelija koje imaju svojstva.

Svaka ćelija može imati jedno od dva moguća stanja - živuće ili mrtvo. Također, svaka ćelija ovisi o svojih 8 susjeda. Simulacija igre izvodi se u koracima te se u svakom koraku primjenjuju 4 pravila na svaku ćeliju:

1. Svaka živuća ćelija s manje od 2 susjeda umire (od usamljenosti).
2. Svaka živuća ćelija s 2 ili 3 susjeda preživljava.
3. Svaka živuća ćelija s više od 3 živuća susjeda umire (od prepopulacije).
4. Svaka mrtva ćelija s točno 3 susjeda postaje živuća (od reprodukcije).

Pritom je važno napomenuti da se sva rođenja i smrti odvijaju *simultano*. Zajedno čine jednu generaciju ili, kako bi se to još moglo nazvati, „pokret“ u čitavoj „povijesti života“ početne konfiguracije. [1] Conway pritom predlaže sljedeći proces za činjenje koraka:

1. Započeti s uzorkom koji se sastoji od crnih elemenata.

2. Odrediti položaj svih elemenata koji će umrijeti te ih identificirati postavljanjem bijelog elementa na vrh svakoga od njih.

3. Odrediti položaje svih praznih ćelija u kojima će se odviti rođenja, a obilježiti ih postavljanjem crnih elemenata na vrh svake od njih.

4. Nakon što je uzorak provjeren dva puta, kako bi se otklonila mogućnost eventualno načinjenih pogrešaka, ukloniti sve mrtve elemente (u parovima) te zamijeniti novorođene organizme crnim elementima.

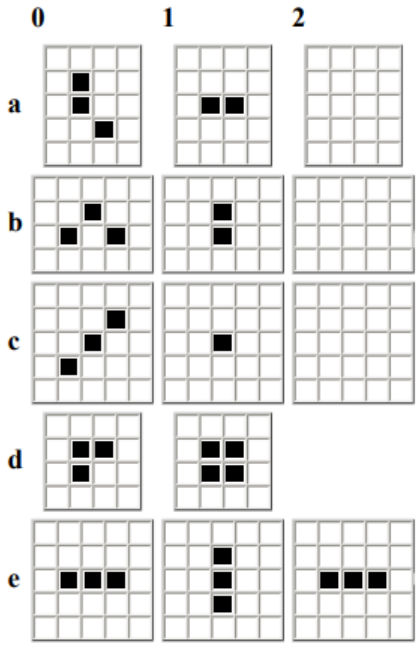
Ono što se također može uočiti jest to da se populacija konstantno kreće neuobičajeno te da pritom prolazi kroz neočekivane promjene. U nekim slučajevima društvo u konačnici izumire (svi elementi iščezavaju), premda se to ne bi trebalo dogoditi sve dok se ne izmjeni veliki broj generacija. Većina polaznih uzoraka ili dosiže stabilne oblike (koje Conway naziva *statičnim organizmima*, engl. *still lifes*) koje se ne mogu mijenjati, ili uzorke koji osciliraju zauvijek. Uzorci bez početne simetrije pokazuju tendenciju postati simetričnima. Jednom kada se to dogodi, simetrija se ne može izgubiti, premda se može povećati.

Nadalje, Conway je pretpostavio da nijedan uzorak ne može rasti neograničeno. Drugim riječima, bilo koja konfiguracija konačnoga broja elemenata ne može rasti iznad konačne gornje granice broja elemenata na polju. Ta je pretpostavka vjerojatno najdublje i najvažnije pitanje koje postavlja

igra. Zbog toga je Conway tada ponudio nagradu u iznosu od pedeset američkih dolara prvoj osobi koja može dokazati ili opovrgnuti pretpostavku do završetka godine. [3] Jedan način za pobijanje pretpostavke bio bi otkriti uzorke koji nastavljaju dodavati elemente u polje: „pištolj“ (engl. *gun*) – konfiguraciju koja opetovano ispucava objekte u kretnji, kao što je tzv. „klizač“ (engl. *glider*) ili „parni vlak“ (engl. *puffer train*) – konfiguraciju koja se pomiče, ali ostaje iza „tragova dima“. [1] S vremenom je ta pretpostavka opovrgnuta jer su razvojem računala simulacije postale jednostavnije, početni uzorci veći. [3]

Prema tome, potrebno je promotriti što se događa s različitim jednostavnim uzorcima. Jednostavni organizmi od jednog ili para elemenata (raspoređenih gdje god) očito će nestati u prvom koraku.

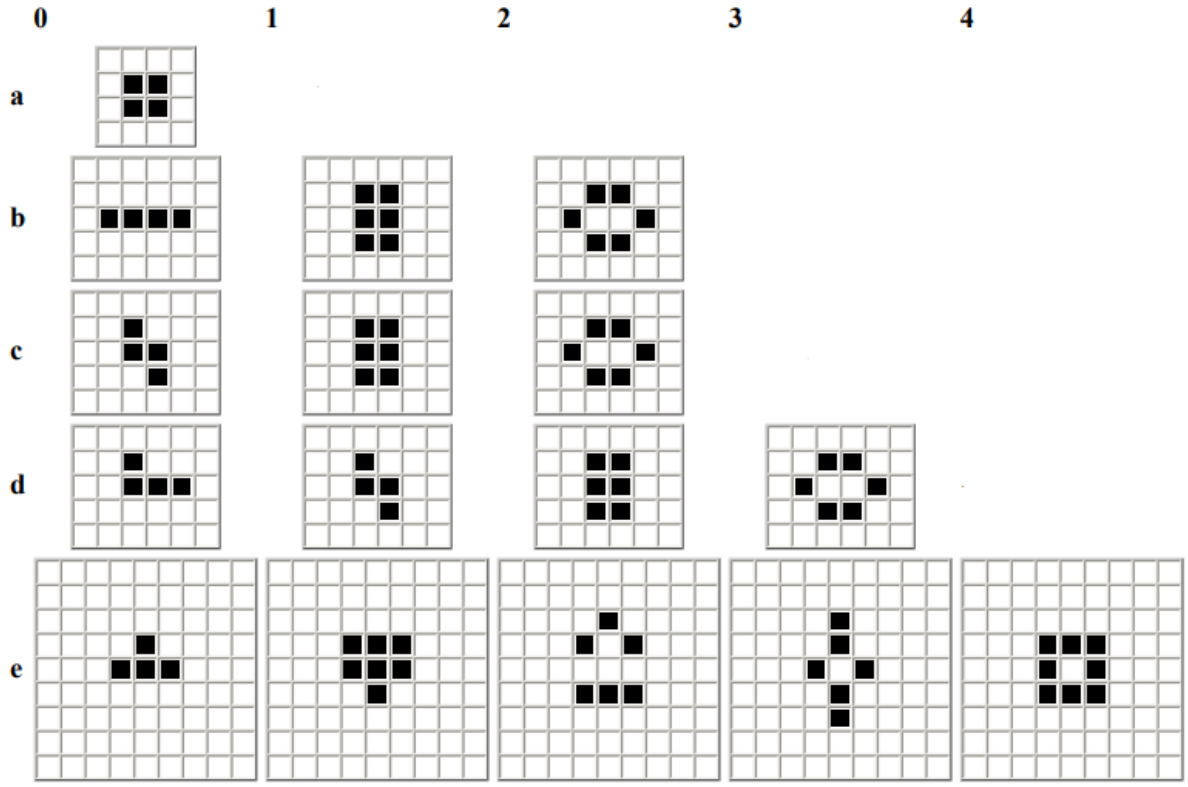
Polazni uzorak od tri elementa također smjesta umire, osim ako najmanje jedan element ima dva susjeda.



Sl. 2.1. *Pet trojki koje ne nestaju u prvom koraku*

Ilustracija iznad prikazuje da pet trojki ne nestaju u prvom koraku (njihova je orijentacija irelevantna). Prve tri (*a, b, c*) nestaju u drugom koraku. Kod oblika *c* nije važno ništa osim toga da jedan dijagonalni lanac elemenata, koliko god dug, gubi svoje rubne elemente sve dok lanac u potpunosti ne nestane. Brzinu kojom se šahovski kralj pomiče u bilo kojem smjeru Conway naziva „brzina svjetlosti.“ [4] Prema tome, dijagonalni lanac propada na svakom kraju brzinom svjetlosti.

Uzorak *d* postaje stabilan „blok“ (dva po dva kvadrata) u sljedećem koraku. Uzorak *e* najjednostavnije predstavlja ono što se naziva „bistabil“ (engl. *flip-flops*). Izmjenjuje se između vodoravnih i okomitih nizova po tri. Conway ga naziva „žmigavac“ (engl. *blinker*).



Sl. 2.2. *Povijest života pet tetrominoa*

Prethodna ilustracija prikazuje povijest života pet tetrominoa (oblik sastavljen od četiri ortogonalno spojena elementa).

Kvadrat [a] je, kao što se može vidjeti, nepromjenjiva figura. Tetrominoi *b* i *c* u sljedećem koraku dosižu stabilnu figuru koja se naziva „košnica“ (engl. *beehive*). Košnice su često izrađeni uzorci. Tetromino *d* postaje košnicom u trećem koraku. Najzanimljiviji je tetromino *e* koji se nakon devet koraka pretvara u četiri izolirana žmigavca, bistabile koji se nazivaju „semafori“ (engl. *traffic lights*). To je također česta konfiguracija. Prema tome, ilustracija zapravo prikazuje dvanaest najčešćih formi mrtve prirode, odnosno formi koje ne nestaju niti se šire, nego nakon određenog broja koraka zadržavaju svoju krajnju stabilnu ili bistabilnu formu.

Jedno od najznačajnijih Conwayevih otkrića jest upravo klizač od pet elemenata. Nakon dva koraka, on se polagano pomiče te se dijagonalno preslikava. Geometri to nazivaju „refleksija

klizanja“ (engl. *glide reflection*) – odakle potječe i sam naziv konfiguracije. Nakon još dva koraka klizač se ispravlja te se pomiče za jednu ćeliju dijagonalno dolje desno, u odnosu na svoj početni položaj.

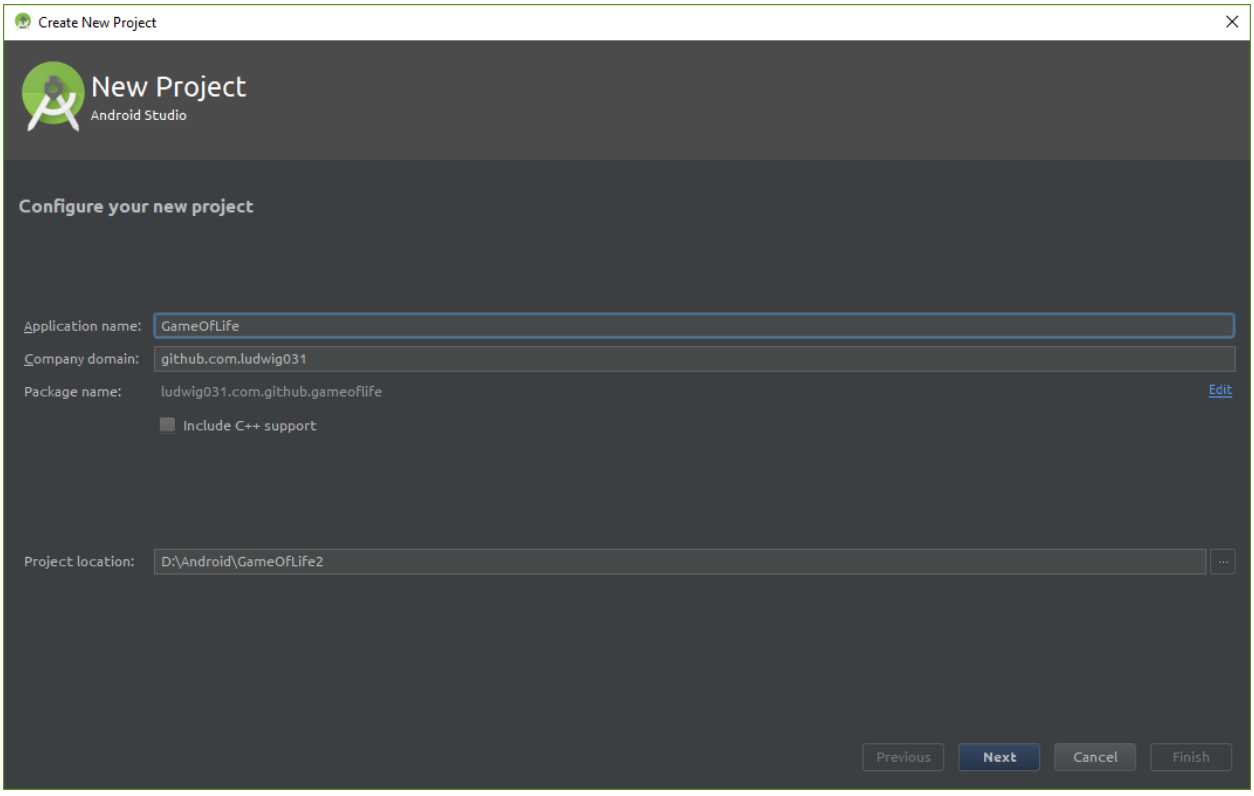
Već je ranije spomenuto da se brzina kretanja šahovskoga kralja naziva brzinom svjetlosti. Zanimljivo je napomenuti da se Conway za taj naziv odlučio jer podrazumijeva najveću moguću brzinu koju bilo koja vrsta pokreta može dosegnuti na ploči. Nijedan se uzorak ne može replicirati dovoljno brzo da bi se pomicao takvom brzinom. Conway je dokazao da je maksimalna brzina dijagonalno četvrtina brzine svjetlosti. Budući da se klizač replicira u jednakoj orijentaciji nakon četiriju koraka i premješta se za jednu ćeliju dijagonalno, može se reći da klizi po polju brzinom od četvrtine brzine svjetlosti.

3. ANDROID APLIKACIJA „IGRA ŽIVOTA“

Cilj izrade aplikacije *Igra života* jest simulacija igre života na uređajima s operativnim sustavom Android i prikaz razvoja predefiniranog početnog stanja. Pokazat će se proces postavljanja i izrade projekta uz objašnjenja pojedinih dijelova koda pisanih u Java programskom jeziku i XML opisnom odnosno označnom jeziku.

3.1. Postavljanje projekta u Android studiju

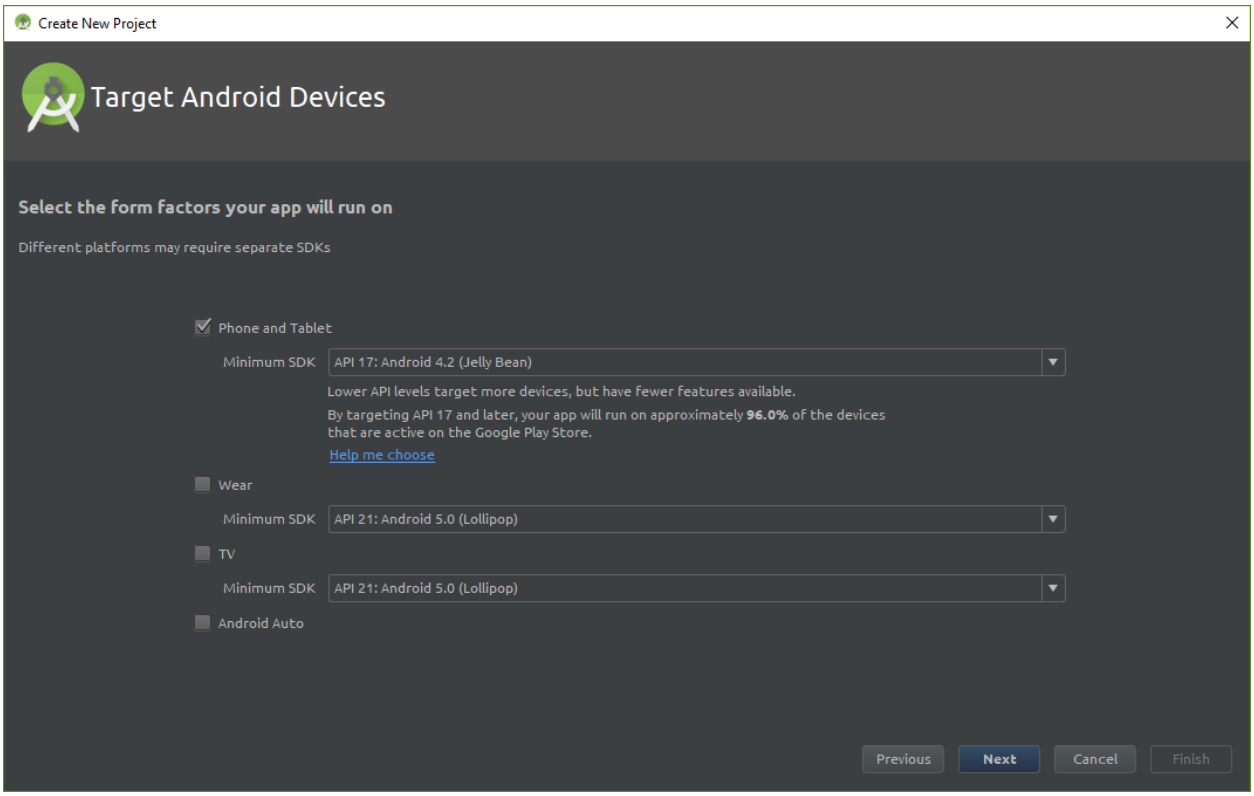
Android studio jest službeni IDE za razvijanje aplikacija namijenjenih operativnom sustavu Android.[4] IDE podrazumijeva integriranu skupinu alata potrebnih za razvoj, odnosno pisanje, testiranje i verzioniranje koda. Kod postavljanja novog projekta, nužno je definirati njegovo ime te domenu tvrtke za što je postavljen link do vlastitog GitHub profila[5] na kojemu je praćeno verzioniranje koda, kao i razvoj projekta.



Sl. 3.1. *Prvi korak postavljanja projekta*

[4] IDE jest oznaka za *integrated development environment*, odnosno integrirano sučelje za razvoj.

[5] https://github.com/ludwig031



Sl. 3.2. *Drugi korak postavljanja projekta*

U idućem je koraku nužno definirati vrstu uređaja za koje je aplikacija namijenjena te minimalnu verziju Androida na kojoj aplikacija može biti pokrenuta. Verzija androida ključna je zbog dopuštene razine API-a[6]. Nakon postavljanja projekta dobiva se osnovni set direktorija i datoteka u okviru kojih se izrađuje aplikacija.

3.2. Razvoj projekta

3.2.1. Android manifest

Svaka aplikacija, odnosno projekt, mora imati *AndroidManifest.xml* datoteku (s točno tim imenom) u ishodišnom direktoriju izvora projekta. Ta datoteka opisuje bitne informacije o aplikaciji i razvojnim alatima te android operacijskom sustavu. [5]

[6] API (*application programming interface*) skup je funkcija i procedura koje dopuštaju izradu aplikacija koje pristupaju mogućnostima ili podatcima operacijskog sustava.

Između ostalog, ta datoteka mora sadržavati:

- Ime paketa, koji je najčešće i *namespace* koda. Androidovi alati koriste tu informaciju za određivanje lokacije entiteta pri izgradnji projekta.
- Komponente aplikacije, što podrazumijeva sve aktivnosti, servise i vanjske poveznice.
- Ovlasti koje su aplikaciji potrebne za pristup zaštićenim dijelovima sustava ili drugih aplikacija.
- Značajke software-a i hardware-a koje aplikacija zahtijeva, što utječe na uređaje koji mogu instalirati aplikaciju.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ludwig031.gameoflife" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity
            android:screenOrientation="portrait"
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Sl. 3.3. *AndroidManifest.xml*

3.2.1. Main activity

Klasa *Activity* odnosno *aktivnost* jest neophodna komponenta android aplikacije i način na koji su aktivnosti pokretane i povezane čini osnovni dio aplikacijskog modela. [6] Svaka aktivnost nasljeđuje metode klase *Activity*. Igra života sastoji se od jedne aktivnosti u kojoj se simulacijom upravlja i u kojoj se ona odvija. Simulacijom se upravlja pomoću 4 gumba:

- **Start/Stop** pokreće odnosno zaustavlja odvijanje simulacije.

```
/* com/ludwig031/gameoflife/MainActivityFragment.java */
final Button startButton = (Button) view.findViewById(R.id.start_button);
startButton.setText(simulationFragment.isStart() ? "STOP" : "START");
startButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (simulationFragment.isStart()) {
            startButton.setText("Start");
            simulationFragment.stop();
        } else {
            startButton.setText("Stop");
            simulationFragment.start();
        }
    }
});

/* com/ludwig031/gameoflife/SimulationFragment.java */
public void start() {
    flag = true;

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            while (flag) {
                cells = engine.nextGen(cells);

                Fragment fragment = SimulationFragment.this.getTargetFragment();
                if (fragment instanceof CellsListener) {
                    ((CellsListener)fragment).update(cells);
                }

                try {
                    Thread.sleep(500);
                } catch (InterruptedException exp) {
                    break;
                }
            }
        }
    });

    thread.start();
    this.thread = thread;
}

/* com/ludwig031/gameoflife/SimulationFragment.java */
public void stop() {
    flag = false;
    Thread t = this.thread;
    if (t != null) {
        t.interrupt();
        this.thread = null;
    }
}
```

Sl. 3.4. *Implementacija „Start/Stop“ gumba*

- **Korak** pokreće simulaciju koja se odvija samo jednu *generaciju* i automatski zaustavlja.

```
/* com/ludwig031/gameoflife/MainActivityFragment.java */
final Button stepButton = (Button) view.findViewById(R.id.step_button);
stepButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (simulationFragment.isStart()) {
            Toast.makeText(MainActivityFragment.this.getContext(), "Prvo zaustavi
proces", Toast.LENGTH_SHORT).show();
        } else {
            simulationFragment.step();
        }
    }
});


/* com/ludwig031/gameoflife/SimulationFragment.java */
public void step() {
    cells = engine.nextGen(cells);

    Fragment fragment = SimulationFragment.this.getTargetFragment();
    if (fragment instanceof CellsListener) {
        ((CellsListener)fragment).update(cells);
    }
}
```

Sl. 3.5. *Implementacija „Korak“ gumba*

- **Nasumce** postavlja 150 ćelija u živuće stanje.

```
/* com/ludwig031/gameoflife/MainActivityFragment.java */
final Button randomButton = (Button) view.findViewById(R.id.random_button);
randomButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (simulationFragment.isStart()) {
            Toast.makeText(MainActivityFragment.this.getContext(), "Prvo zaustavi
proces", Toast.LENGTH_SHORT).show();
        } else {
            simulationFragment.random();
        }
    }
});


/* com/ludwig031/gameoflife/SimulationFragment.java */
public void random() {
    Random rnd = new Random();

    cells = new Cells(20, 20);
    for (int i = 0; i < 150; i++) {
        int row = rnd.nextInt(20);
        int col = rnd.nextInt(20);
        cells.getCell(row, col).setAlive(true);
    }
    Fragment fragment = SimulationFragment.this.getTargetFragment();
    if (fragment instanceof CellsListener) {
        ((CellsListener) fragment).update(cells);
    }
}
```

Sl. 3.6. *Implementacija „Nasumce“ gumba*

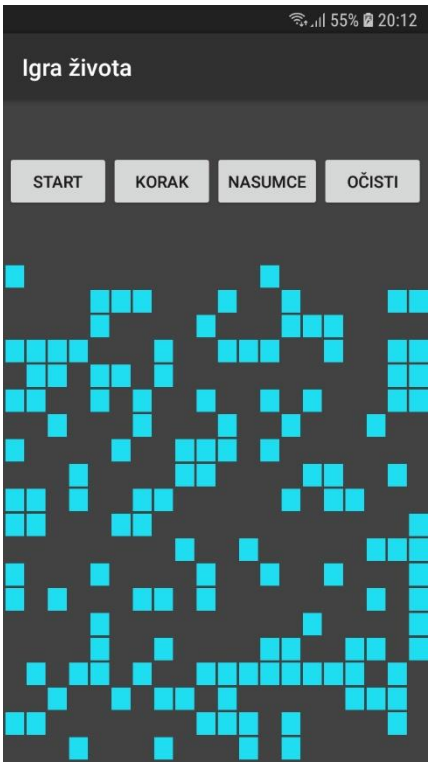
- **Očisti** postavlja sve živuće ćelije u neživo stanje.

```
/* com/ludwig031/gameoflife/MainActivityFragment.java */
final Button clearButton = (Button) view.findViewById(R.id.clear_button);
clearButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (simulationFragment.isStart()) {
            Toast.makeText(MainActivityFragment.this.getContext(), "Prvo zaustavi
proces", Toast.LENGTH_SHORT).show();
        } else {
            simulationFragment.clear();
        }
    }
});

/* com/ludwig031/gameoflife/SimulationFragment.java */
public void clear() {
    cells = new Cells(20, 20);
    Fragment fragment = SimulationFragment.this.getTargetFragment();
    if (fragment instanceof CellsListener) {
        ((CellsListener) fragment).update(cells);
    }
}
```

Sl. 3.7. *Implementacija „Očisti“ gumba*

Ispod četiri upravljačka gumba nalazi se mreža od 20*20 ćelija unutar koje se sama simulacija odvija i prikazuje. Pri prvom otvaranju aplikacije ćelije su u mreži postavljene nasumce i to po principu 150 živih ćelija u odnosu na 250 neživih. Pri svakom idućem otvaranju prikazuje se posljednje spremljeno stanje iz radne memorije uređaja.



Sl. 3.8. *Prikaz početnog stanja pri ulasku u aplikaciju*

3.2.2. Ćelija

Ćelija, kao objekt, definirana je klasom *Cell* te sadrži metode koje opisuju trenutno stanje ćelije, mijenjaju to stanje, dohvaćaju i postavljaju broj aktivnih susjednih ćelija. Također je postavljeno zadano stanje ćelije na neživo i broj susjednih ćelija na 0.

```
/* com/ludwig031/gameoflife/Cell.java */
public class Cell {
    public boolean isAlive() {
        return alive;
    }
    public void setAlive(boolean alive) {
        this.alive = alive;
    }
    public int getAliveNeighbourCount() {
        return aliveNeighbourCount;
    }
    public void setAliveNeighbourCount(int aliveNeighbourCount) {
        this.aliveNeighbourCount = aliveNeighbourCount;
    }

    private boolean alive = false;
    private int aliveNeighbourCount = 0;
}
```

Sl. 3.9. *Definicija klase Cell*

Svaki objekt klase *Cell* povezan je s istovjetnim objektima koji ga okružuju. Oni utječu na to hoće li u idućoj generaciji ćelija promijeniti stanje ili ostaje istog stanja kao u prethodnoj generaciji. Kako bi mogli odrediti sve to, potrebne su nam informacije o poziciji same ćelije unutar mreže te nekakav brojač koji broji broj živih susjednih ćelija. Te dodatne informacije definirali smo u klasi *Cells* koja u određenim metodama implementira klasu *Cell*.

```
/* com/ludwig031/gameoflife/Cells.java */
public class Cells {
    public Cells(int row, int col) {
        this.row = row;
        this.col = col;

        for (int i = 0; i < this.row * this.col; i++) {
            cells.add(new Cell());
        }
    }

    public int Incrementer(int _r, int _c) {
        if (getCell(_r, _c).isAlive()) {
            return 1;
        } else {
            return 0;
        }
    }

    public Cell getCell(int row, int col) {
        int index = (row * this.col) + col;
        return cells.get(index);
    }

    public int getRow() {
        return this.row;
    }

    public int getCol() {
        return this.col;
    }

    public List<Cell> getList() {
        return cells;
    }

    private final List<Cell> cells = new ArrayList<>();
    private final int row;
    private final int col;
}
```

Sl. 3.10. *Definicija klase Cells*

Postavljanje ćelija u mrežu ovisno o njihovom statusu i poziciji odrađuje *CellAdapter* klasa. Ovisno o stanju ćelije, klasa joj dodjeljuje pozadinsku boju **#1FDEF1** ako je ćelija živa, odnosno **#424242** ako nije.

```
/* com/ludwig031/gameoflife/CellAdapter.java */
public class CellAdapter extends ArrayAdapter<Cell> {
    private Context mContext;
    private List<Cell> cells;

    public CellAdapter(Context c, List<Cell> cells) {
        super(c, R.layout.cell, cells);
        this.cells = cells;
        mContext = c;
    }

    public View getView(int position, View convertView, @NonNull ViewGroup parent) {
        TextView textView;
        if (convertView == null) {
            LayoutInflater inflater = LayoutInflater.from(mContext);
            textView = (TextView)inflater.inflate(R.layout.cell, null);
        } else {
            textView = (TextView) convertView;
        }

        Cell cell = cells.get(position);

        if (cell.isAlive()) {
            textView.setBackgroundColor(0xFF1FDEF1);
        } else {
            textView.setBackgroundColor(0xFF424242);
        }

        return textView;
    }
}
```

Sl. 3.11. *Definicija klase CellAdapter*

3.2.3. Osnovni pokretač

Za određivanje koje žive i nežive ćelije ostaju kakve jesu, a koje mijenjaju stanje u svakom evolucijskom koraku odnosno generaciji, brine se klasa *BaseEngine*. Za svaku ćeliju iteriramo po susjednim ćelijama i zbrajamo živuće ćelije. Na osnovu tog brojača primjenjujemo glavna pravila po kojima se određuje status ćelije u idućoj generaciji.

```java
/* com/ludwig031/gameoflife/ BaseEngine.java */
public class BaseEngine implements Engine {
    @Override
    public Cells nextGen(Cells cells) {
        final int row = cells.getRow();
        final int col = cells.getCol();

        for (int r = 0; r < row; r++) {
            for (int c = 0; c < col; c++) {
                Cell cell = cells.getCell(r, c);

                int aliveNeighbourSize = 0;

                int _r = r - 1;
                _r = _r >= 0 ? _r : row - 1;

                int _c = c - 1;
                _c = _c >= 0 ? _c : col - 1;
                aliveNeighbourSize += cells.Incrementer(_r, _c);

                _c = c;
                aliveNeighbourSize += cells.Incrementer(_r, _c);

                _c = c + 1;
                _c = _c < col ? _c : 0;
                aliveNeighbourSize += cells.Incrementer(_r, _c);


                _r = r;

                _c = c - 1;
                _c = _c >= 0 ? _c : col - 1;
                aliveNeighbourSize += cells.Incrementer(_r, _c);


                _c = c + 1;
                _c = _c < col ? _c : 0;
                aliveNeighbourSize += cells.Incrementer(_r, _c);


                _r = r + 1;
                _r = _r < row ? _r : 0;

                _c = c - 1;
                _c = _c >= 0 ? _c : col - 1;
                aliveNeighbourSize += cells.Incrementer(_r, _c);


                _c = c;
                aliveNeighbourSize += cells.Incrementer(_r, _c);


                _c = c + 1;
                _c = _c < col ? _c : 0;
                aliveNeighbourSize += cells.Incrementer(_r, _c);


                cell.setAliveNeighbourCount(aliveNeighbourSize);
            }
```

Sl. 3.12. *Definicija klase BaseEngine*

4. ZAKLJUČAK

U proučavanju teorije igre i tijekom procesa izrade aplikacije za igru *Igra života*, može se prije svega zaključiti da je njezina glavna uloga simulacija rada staničnih automata, što znači da ona pripada igrama koje prikazuju procese slične onima iz stvarnoga života. Može se uočiti da je temeljna ideja igre započeti s jednostavnom konfiguracijom elemenata, a zatim promatrati kako se one razvijaju dok se primjenjuju Conwayevi zakoni za rođenja, smrti i opstanke. Primjenom tih zakona vidimo da su oni takvi da čine ponašanje populacije nepredvidivim. Pokazano je kako svaka ćelija može imati jedno od dva moguća stanja – živuće ili mrtvo. Također, svaka ćelija ovisi o svojih osam susjeda. Simulacija igre izvodi se u koracima, a pritom se u svakom koraku simultano primjenjuju četiri pravila reprodukcije, na svaku ćeliju.

Nadalje, cilj izrade same aplikacije je omogućiti simulaciju *Igre života* na uređajima s operativnim sustavom Android, odnosno prikaz razvoja zadanoga početnog stanja kroz generacije. Na taj se način nastojalo upoznati čitatelja s algoritmima Conwayeve *Igre života* te izraditi mobilnu aplikaciju za Android platformu u Java programskom jeziku.

U konačnici, može se zaključiti da je završetkom same izrade aplikacije uistinu realizirana mogućnost njezine praktične primjene te je izvršen svrsishodan uvid u njezinu strukturu. Koristeći aplikaciju uviđamo da su ostvarene zakonitosti simulacije rada staničnih automata, posebno one najvažnije koje se odnose na nepredvidivost razvoja populacije kroz generacije.

5. LITERATURA

5.1 Internetski izvori

[1] Gardner, Martin, 1970. *Mathematical games* http://www.ibiblio.org/lifepatterns/october1970.html/ (pristupljeno 15.6.2017.)

[2] Inventing Game of Life – Numberphile, Youtube https://youtu.be/R9Plq-D1gEk/ (pristupljeno 16.6.2018.)

[3] Does John Conway hate his Game of Life?, Youtube https://youtu.be/E8kUJL04ELA/ (pristupljeno 16.6.2018.)

[4] Conway's Game of Life, Wikipedia https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life (pristupljeno 15.8.2017.)

[5] Android developers guide https://developer.android.com/ (pristupljeno 15.8.2018.)

[6] Android Fundamentals, Udacity https://www.udacity.com/course/new-android-fundamentals--ud851 (pristupljeno 16.8.2018.)

SAŽETAK

Tema rada je izrada i prikaz procesa izrade android aplikacije na temelju *Igre života* Johna Conwaya. U teorijskom su dijelu rada izneseni podatci iz biografije autora igre Johna Conwaya, a zatim je pružen uvidu strukturu igre s obzirom na njezina temeljna obilježja, funkciju i uporabu. U tim je poglavljima dan uvod u temeljnu tematiku rada – android aplikaciju, koja i čini središnji dio rada. Na koncu je, na temelju navedenih primjera i analize teorijskih zapažanja, iznesen zaključak o ideji i o korisnosti realizirane inačice *Igre života*.

Ključne riječi: android aplikacija, *Igra života*, John Conway

ABSTRACT

Android application of John Conway's 'Game of life'

Thesis is devoted to developing and displaying the Android application development process based on John Conway's *Game of life.* Theoretical part of the paper presents data from the biography of John Conway – author of the game and then provided insight into the structure of the game with regard to its fundamental features, function and use. In these chapters, there is an introduction to the fundamental theme of the work - the android app, which is the central part of the work. In the end, based on the aforementioned examples and the analysis of theoretical observations, a conclusion was made about the idea and the usefulness of the realized version of the Game of Life.

Keywords: android application, *Game of life*, John Conway

ŽIVOTOPIS

Borna Matijanić je rođen 22. lipnja 1992. godine u Našicama. Osnovnu školu Matije Petra Katančića pohađao je u Valpovu, a srednju školu završio u Osijeku u Elektrotehničkoj i prometnoj školi Osijek, stekavši zvanje Tehničara za računalstvo. Od osnovne škole u doticaju je s programiranjem, stoga 2014. godine upisuje stručni studij Informatike na Elektrotehničkom fakultetu Osijek, koji danas nosi ime Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. Istovremeno, uz srednjoškolsko i visokoškolsko obrazovanje, bavio se različitim poslovima vezanima uz programiranje, telekomunikacije i fotografiju. Osim toga, raspolaže znanjem engleskoga jezika te posjeduje iskustvo u programiranju u jezicima C++, Java, PHP, Ruby, SQL. Aktivno bavljenje razvojem programskih rješenja planira nastaviti i svom u budućem radu.

Borna Matijanić

_____