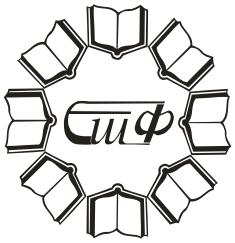
ELEKTROTEHNIČKI FAKULTET
UNIVERZITET U ISTOČNOM SARAJEVU



Android aplikacija „Space battle“

dokumentacija

Odsjek: Računarstvo i informatika

Predmet: Paraleni procesorski sistemi

Studenti:

Dejana Klačar - 1747

Nikola Kostović - 1582

Mentor: *Nikola Davidović*

Sadržaj

Uvod

Ovaj seminarski je nastao kao dokumentacija za projekat „Space battle“ koji je prijavljen za predmet „Paraleni procesorski sistemi“ na Elektrotehničkom fakultetu u Istočnom Sarajevu. Osim samog opisa rada aplikacije, unutar seminarskog rada se osvrćemo na sam pojam paralelizma i same upotrebe paralelizma unutar aplikacija.

Dokument opisuje kako je moguće iskorsti višenitno programiranje u igricama. U okviru kompletnog projekta nije iskorišten nijedan gotov alat od koga se može krenuti u dalju realizaciju logike za neku od igrica.

Višenitno programiranje zadnjih godina postaje sve više prisutno, jer aplikacije koje se danas razvijaju postaju više ovisne od raznih servisa koji se obično nalaze na udaljenim računarima. S druge strane, postoje aplikacije koje u svom sastavu imaju funkcije koje izvršavaju vrlo duge operacije, pri čemu je korisnički interfejs uglavnom nepomičan(zamrznut) te danas predstavlja staromodan i prevaziđen način dizajniranja aplikacija.

Svaka aplikacija posjeduje samo jednu nit i ona se zove UI ili glavna nit. Sve operacije u aplikaciji se izvršavaju sekvencijalno: jedna po jedna odnosno, svaka operacija mora čekati sve prethodne operacije da bi se sama izvršila. To predstavlja ozbiljan problem pa postoji mogućnost izvršavanja više paralelnih operacija odjednom.

Ovaj rad se sastoji od tri cjeline. U prvoj cjelini je opisano radno okruženje u kojem je aplikacija rađena. Druga cjelina opisuje višenitno programiranje u programskim jezicima Java i C#. Treća cjelina opisuje samu aplikaciju.

1 Operativni sistem i okruženje

Android je operativni sisem za mobilne uređaje baziran na modifikovanoj verziji Linux kernela uz korištenje drugih projekata otvorenog koda a takođe je komercijalno podržan od strane Google kompanije. Jedna od najbitnijih osobina ovog operativnog sistema jeste da u potpunosti besplatan i otvorenog koda tako da bilo koja kompanija(ili pojedinac) može da prilagodi ovaj system svojim potrebama. Ovaj operativni system je primarno namijenjen za movilne uređaje kao što su smartfoni i tableti. Android je prvobitno najavljen 2007. godine a prvi uređaj sa Android operativnim sistemom je izašao na tržište u septembru 2008. godine. To je bio uređaj kompanije HTC koji je pod nazivom HTC Dream. Iako je imao nedostatke kao što su nekompatibilnost sa određenim aplikacijama I nedostatak opcija koje su konkurenti imali na svojim uređajima, ovaj telefon je smtran inovacijom zbog novina koje je uveo – potpuna integracija Google servisa i sistem notifikacija.

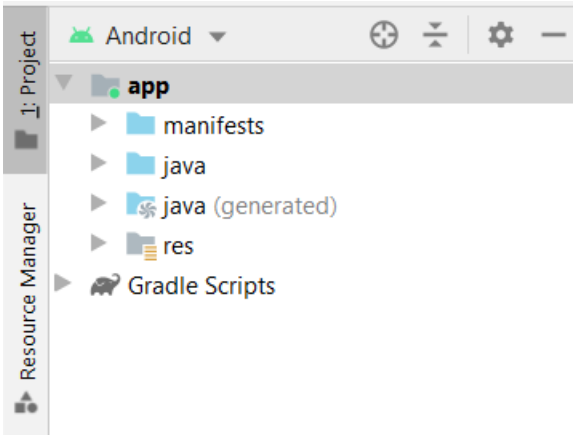
Za rad aplikacije koristili smo AndroidStudio. AndroidStudio je razvojno okruženje zasnovano na IntelliJ IDEA softveru. Namjenjen prevenstveno za razvoj android mobilnih aplikacija, Android Studio je optimalno okruženje koje pruža sve mogućnosti trenutno dostupne u ovoj grani programiranja. Korisnički interfejst je kreiran tako da su najkorišćenije funkcionalnosti uvijek dostupne na vidljivim mjestima, a takođe I pruža mogućnost reorganizacije paleta alata onako kako korisniku odgovara. Pored ove dvije karakteristike, AndroidStudio pruža i više nego dovoljno opcija za neomatani razvoj Android aplikacije, uključujući podešavanja različitih perspektiva za prikaz strukture projekta, integraciju sa nekoliko različitih kontrola koda(version control ili source control) i mnoge druge.

Glave prednosti AndroidStudio okruženja u odnosu na druge platforme za razvoj Android aplikacija su:

- Ugrađeno android emulator – virtuealni uređaj koji omogućava brzo i lako testiranje napravljenih aplikacija. Podržava mnoštvo različitih verzija Androida što omogućava programeru prilagođavanje aplikacije što većem broju različitih android uređaja.
- Inteligentni editor koda koji omogućava lako kodiranje sa predikcijom često korištenih metoda
- *Gradle* sistem za pripremu kompilacija *apk* fajla kao I automatsa instalacija na uređaj. *Gradle* omogućava jednostavni menadžment paketa i biblioteka što pojednostavljuje proces kompajlovanja projekta
- Vizuelni editor i XML fajl.
- Ugrađena podrška za Google Cloud platformu što omogućava korištenje različitih servisa Firebase sistema

- Profajler koji omogućava pregled uticaja aplikacije na performanse uređaja kao i pregled resursa koje aplikacija korisi.

Za razvoj aplikacije u Android Studio okruženju neophodno je poznavati par fundamentalnih stvari. Prilikom kreiaranja novog projekta, okruženje generiše stablo projekta koje čine folderi projekta. Stablo projekta prikazano je na slici.



U folderu *manifest* se nalazi jedan fajl – Android Manifest koji je zapravo XML dokument u kome se definišu opšte postavke aplikacije kao što su ime, servisi aplikacije kao i broj Aktivitija koje će aplikacija imati. Aktiviti predstavlja logičku cjelinu koja se sastoji od izgleda i pozadinskog koda.

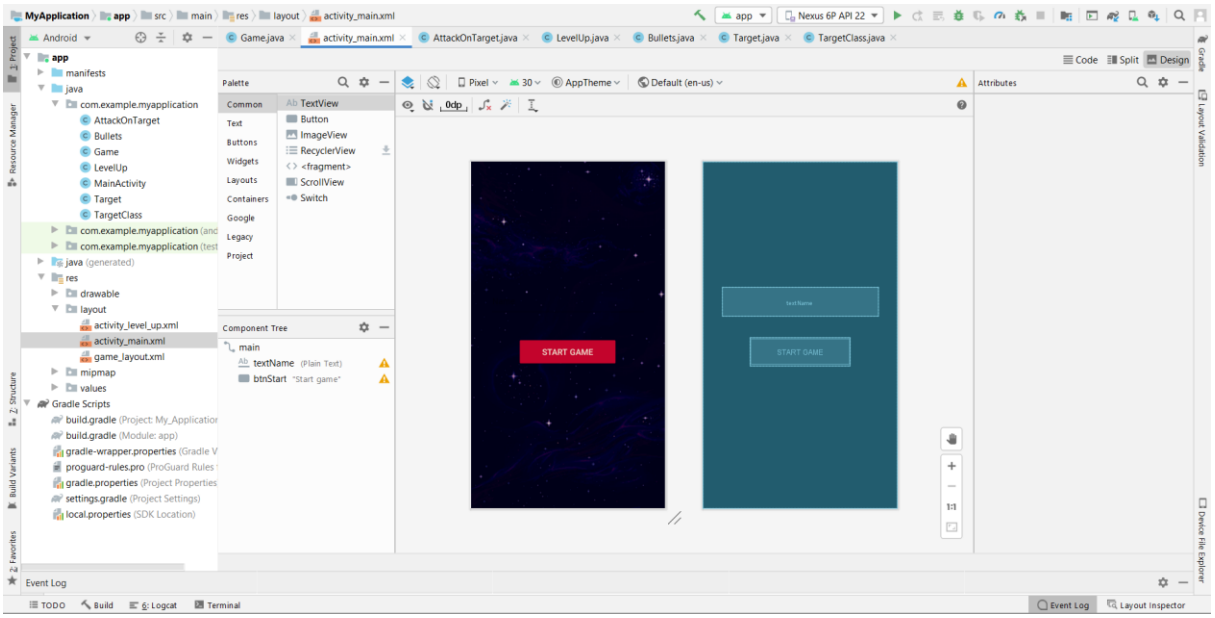
Folder *java* sadrži sav programski kod za aplikaciju i svi Kotlin i Java fajlovi su smješteni u ovom folderu. U slučaju kompleksnije aplikacije, poželjno je da se fajlovi smještaju u pakete koji onda predstavljaju podfoldere foldera *java.*

Res folder sadrži XML dokumente koje predstavljaju različite vizuelne komponente aplikacije najćešće izgled Aktivitija I Fragmenata.

Folder *GradleScript* sadrži nekoliko fajlova od kojih se najznačajniji *build.* Gradle fajlovi u kojima se definišu biblioteke koje aplikacija koristi, kao I trenutna verzija I naziv paketa aplikacije.

Kao što je već ranije rečeno, izgled android aplikacije se definiše u XML fajlu a takodje je moguće koristit *drag&drop* dizajner dok se XML kod automatski generiše u pozadini.

Na slici je prikazan izgled okruženja:



Sa lijeve strane se nalazi paleta UI elementa dok se sa desne strane nalazi konkretan prozor aplikacije. Naravno, ukoliko za tim ima potrebe. UI elemente je moguće generisati I programski, kroz kod.

Osnovni prozor Android aplikacije se zove Activity. Glavni prozor ili MainActivity dolazi automatski generisan prilikom pravljenja projekta. Svaki aktiviti se sastoji od Kotlin ili Java klase kao I pratećim XML fajlom koji definiše izgled tog aktivitija. Poželjno je da aplikacija ima što manje aktivitija budući da su oni zahtjevni u pogledu resursa telefona. Zbog toga se u Androidu mogu koristiti posebne modularne cjeline koje se zovu fragmenti. Fragment predstavlja logičku cjelinu koja se nalik aktivitiju, sastoji od logičkog dijela sa kodom I grafičkog UI dijela. Svaki fragment mora biti ubačen u neki aktiviti da bi bio prikazan na odgovarajući način. Prednost korištenja fragmenata je manja zahtjevnost aplikacije ukoliko se koriste fragmenti a ne aktivitiji a takođe je moguće iste fragmente iznova koristiti u različitim aktivitijima što doprinosi modularnosti koda.

2 Višenitno programiranje u jezicima C# i Java

Osnovna nit koja se provlači kroz rad se može izreći sledećim rečenicama: Opšti programski jezici kao što su ovde navedeni Java i C# imaju istu popularnost i zastupljenost na tržištu i kod programera. Iz toga nameće se stav da su njihove mogućnosti iste. Sintaksa je veoma slična, analogne klase postoje u oba jezika. Java i C# imaju zato iste višenitne mogućnosti.

Niti (engl. threads), odnosno laki procesi (engl. lightweight proceses), predstavljaju bazične celine za izvršavanje koda pod savremenim operativnim sistemima. Niti su programska celina koja treba da obavi jedan posao. Niti (jedna ili više) pripadaju jednom klasičnom odnosno teškom (engl. heavyweight) procesu. U klasičnom kontekstu jedan težak proces ima svoj programski brojač i druge procesorske registre, memorijske sekcije poput koda, podataka i steka, i ulazno-izlazne resurse.

Niti kao laki procesi i delovi jednog istog procesa imaju svoje unikatne resurse i zajedničke resurse sa ostalim nitima istog procesa. Od unikatnih resursa imaju poseban identifikator niti (engl. thread ID), posebnu vrednost programskog brojača, vrednosti drugih registara procesora kao i poseban stek. Mnogi moderni softverski paketi su višenitni: programi za prikazivanje Web stranica, programi za obradu teksta itd.

Korišćenje višenitnog koncepta ima sledeće prednosti:

1. **Manje vrijeme odziva**
   Višenitna tehnika omogućava interaktivnim aplikacijama da nastave rad, čak i kada je deo programa blokiran ili izvršava neku dugotrajnu specifičnu operaciju. Na Primjer, višenitni Web čitač može da nastavi interakciju s korisnikom u jednoj niti, dok druga nit simultano učitava neku veliku sliku sa Interneta, treća sledeću sliku itd.
2. **Ekonomičnost**
   Ogleda se u deljenju prostora i resursa kao i uštedi vremena koje takođe drastično utiče na performanse. Niti dele memoriju i sve ostale resurse koji pripadaju istom procesu.
3. **Iskorišćenje višeprocesorske arhitekture**
   Bilo koje niti mogu se istovremeno izvršavati, svaka na različitom procesoru.

Podrška za korisničke niti realizuje se preko biblioteke za rad s korisničkim nitima. Ova biblioteka obezbeđuje podršku za stvaranje niti, raspored izvršavanja niti i upravljanje nitima, ali bez uticaja jezgra. Najznačajnije vrste korisničkih niti su: POSIX Pthreads, Mac C-threads i Solaris UI-threads.

Niti jezgra direktno podržava operativni sistem. Konkretno, jezgro izvršava operacije stvaranja niti, raspoređivanja izvršavanja niti i upravljanja nitima u prostoru jezgra. Niti jezgra se po pravilu sporije prave, a upravljanje njima unosi veće vremensko premašenje u odnosu na upravljanje korisničkim nitima, ali generalno gledano jezgro ozbiljnije i efikasnije upravlja svojim nitima nego biblioteka za rad s korisničkim nitima.

Mnogi sistemi podržavaju obe vrste niti, a zavisno od toga kako se korisničke niti mapiraju/preslikavaju u niti jezgra postoje tri glavne koncepcije odnosno tri višenitna (engl. multithreading) modela:

- Model više u jednu (engl. **many-to-one**). U ovom modelu, više korisničkih niti mapira se u jednu nit jezgra.
- Model jedna u jednu (engl. **one-to-one**). U ovom modelu, koji je karakterističan za operativne sisteme Windows NT familije poput 2000, XP i Viste, svaka korisnička nit mapira se u jednu nit jezgra. Ovaj model obezbeđuje mnogo bolje konkurentno izvršavanje niti, dozvoljajući da druge niti nastave aktivnosti u slučaju kada jedna nit obavi blokirajući sistemski poziv. Takođe se omogućava da se više niti jezgra izvršavaju paralelno na višeprocesorskoj arhitekturi.

Model više u više (engl. **many-to-many**). U ovom modelu, više korisničkih niti mapira se u manji ili isti broj niti jezgra pri čemu mapiranje zavisi od operativnog sistema a naročito od broja procesora. Ovo je najkompleksniji i najkvalitetniji model.

Konkurentno programiranje predstavlja pisanje takvih programa koji se sastoje od više kooperativnih procesa i niti koje se izvršavaju simultano ili paralelno, pri tom koristeći zajedničke resurse računarskog sistema. Za pisanje konkurentnih programa mogu poslužiti mnogobrojna razvojna okruženja, među kojima su i programski jezik Java i C#, na koje se odnosi ovaj rad.

3 Rad sa nitima u Javi

U Javi se nit može napraviti odnosno definisati na dva načina:

1. Implementacijom interfejsa Runnable i

2. Proširenjem klase Thread.

Prvi način je u opštem slučaju mnogo prikladniji nego izvođenje potklase iz Thread klase zato što tako može da se izvede klasa iz neke klase koja nije klasa Thread, a da ona i dalje predstavlja nit. Kako Java dozvoljava samo jednu baznu klasu, ako neka klasa se stvori proširenjem Thread klase, ona neće imati mogućnost nasleđivanja funkcionalnosti iz neke druge klase.

Jedino što je potrebno je da se implementira metoda run() u kojoj se nalazi kod koji će stvarno da izvršava niti.

```
public class Primjer_1 implements Runnable {
public Primjer_1() {
// konstruktor
}
public void run() {
}
public static void main(String args[]) {
Primjer_1 jedna_nit = new Primjer_1();

Thread prva_nit;
prva_nit = new Thread(jedna_nit);
prva_nit.start(); //startovanje prva niti
druga_nit.start(); //startovanje druge niti
}
```

Ipak run() metodu ne pozivate direktno već prvo se poziva metoda start() koja će da pripremi sve što je potrebno za pokretanje niti.

Ekvivalentan Primjer kada se proširuje klasa Thread izgleda ovako:

```
public class Primjer_2 extends Thread {
public Primjer_2(String str) {
// konstruktor super(str);
}
public void run() {
}
public static void main(String args[]) {
Primjer_2 prva_nit = new Prijmer_2("Prva nit");
Thread druga_nit = new Primjer_2("Druga nit");
prva_nit.start(); //startovanje prve niti
druga_nit.start(); //startovanje druge niti
}
```

Ovde je iskorišćena mogućnost da se u konstruktoru niti pozove konstruktor natklase super() sa imenom niti kao argumentom.

3.1 Stanje niti u Javi

Niti se sastoje iz niza koraka koji slede jedan za drugim. U svakom trenutku nit se nalazi na nekom stanju. Stanje nam govori šta se događa sa niti, šta nit radi ili je u mogućnosti da uradi. Postoji više klasifikacija stanja, a ovde će biti napravljena analogija sa stanjima procesa kod operativnih sistema.

Nit može biti u 5 stanja ako koristimo konačni automat sa istim brojem stanja, a ona su sledeća:

- Start – trenutak formiranja niti, nova nit.
- Ready – nit ima sve potrebne resurse ali čeka na procesor.
- Run – instrukcije niti se izvršavaju.
- Wait – nit čeka neki događaj, miruje, blokirana je, suspendovana je.
- Stop – svršeno stanje, kraj postojanja niti.

Prevođenje niti iz jednog stanja u drugo se naziva *state transition* odnosno tranzicija stanja. Broj strelica i njihova usmerenost nam ukazuju na način prevođenja.

U Javi se u stanje start dolazi kad se izvrši naredba:

```
Thread druga_nit = new Primjer_2(“Druga nit”);
```

Prelaz u stanje ready se radi pozivom metode start() za neki nit-objekat:

```
prva_nit.start(); //startovanje prve niti
```

Prelaz u stanje *run* ne izvodi Java virtuelna mašina već domaćinski (host) operativni sistem, po nekoj od šema za dodelu procesorskog vremena. Kada se nit izvršava (running) onda se u stvari izvršava kod koji se nalazi u run() metodi nit-objekta. Posle isteka izvršavanja rada na procesoru, nit se vraća u stanje ready (runnable, not running).

U stanje *wait* nit prelazi na više načina:

- Pozivanjem sleep() metode.
- Pozivanjem suspend() metode.
- Pozivanjem wait() metode.
- Pozivanjem blokirajuće U/I operacije.

Metoda *sleep()* zaustavlja rad tekuće ili imenovane niti zadatim brojm milisekundi. Za to vreme će se neka druga niti ili više njih izvršavati:

prva_nit.sleep(5000); //mirovanje prve niti od 5 sekundi

Metoda *suspend()* blokira odnosno suspenduje izvršavanje tekuće ili imenovane niti, sve dok se ona ne odblokira na jedini mogući način – metodom *resume().* Po nekima obe metode su zasterele i više se ne koriste zato i nisu iskorištene u realizaciji projekta.

```
prva_nit.suspend(); //mirovanje prve niti
```

Ovde ide kod koji se izvrsava dok je prva nit suspendovana

```
prva_nit.resume(); //prestanak mirovanja
```

Pozivanjem operacija koja se odnose na ulazno/izlazne uređaje često dovodi do blokade niti. Takva operacija se neće nastaviti, sve dok se ne oslobodi zauzeti U/I uređaj. Povratak u predhodno stanje se događa kada operacija koja je blokirala U/I uređaj bude završena.

Blokirana nit se može deblokirati jedino odgovarajućom inverznom operacijom od operacije koja ju je blokirala (sleep – done sleeping, suspend – resume, wait – notify, block on I/O – I/O complete). Ako se pokuša deblokirati nit sa neodgovarajućom inverznom operacijom, javiće se IllegalThreadStateException izuzetak.

U stanje stop nit dolazi na jedan od četiri načina:

- Metoda je došla do svog prirodnog kraja. Izbegava se tako što se postavlja beskonačna ili uslovna petlja:

```
public void run() {
while(signal) { //ovde ide kod za izvrsavanje }
}
```

- Desio se neuhvaćeni izuzetak. Npr. to se događa sa pozivom suspend() metode za nit koja nije u aktivnom stanju, već je u start ili stop stanju. Izbegava se tako što se celo telo run metode stavlja u kombinaciju try/catch blokova:

```
public void run() {
try {
while(signal) { //ovde ide kod za izvrsavanje }
}
catch(Exception e) { //kod za obradu izuzetka }
}
```

  Kada se nadređena nit ili program u okviru koga je nit kreirana i pokrenuta završi. Izbegava se pravljenjem korisničke niti, koja nastavlja da postoji i posle smrti nadređene niti odnosno nadređenog procesa. Nit koja nije korisnička naziva se pozadinska (daemon) nit. Pozadinska nit se određuje pre poziva metode start() sa argumentom true za metodu setDaemon:

```
Thread druga_nit = new Primjer_2("Druga nit");
druga_nit.setDaemon(true); //Odredi da je druga nit tipa pozadinska nit
druga_nit.start(); //startovanje druge niti
```

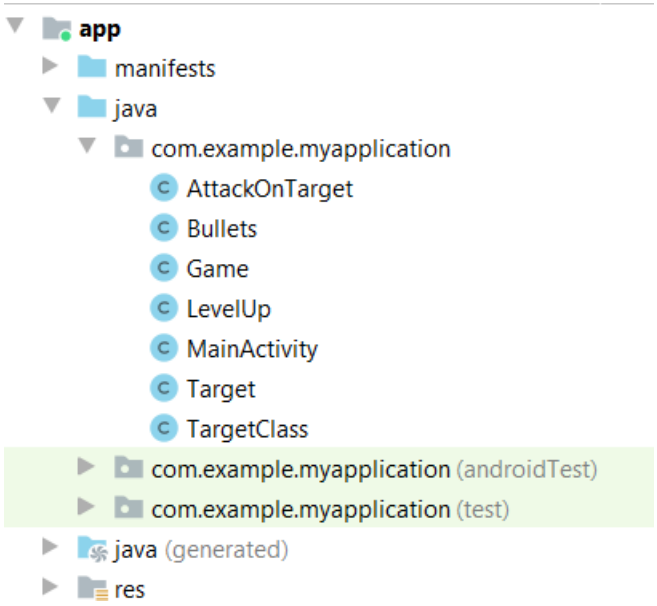
  Pozivanjem metode stop(). Po nekima ova metoda se ne koristi jer dovodi do nestabilnog ponašanja programa:

```
druga_nit.stop(); //kraj postajanja niti
```

  Stanje niti se ispituje preko metoda isAlive() koji vraća logičku vrednost true ako je nit aktivna (bilo da je u stanju ready, run ili wait). Ako je nit u stanju start ili stop onda se vraća vrednost false:

```
if(druga_nit.isAlive())
System.out.println("Druga nit je aktivna!");
Else
System.out.println("Druga nit nije aktivna!");
```

4 Opis procesa, klasa i stanja



Ova aplikacija počinje izvršavanjem klase MainActivity gdje korisnik unosi ime i započinje igru.

Nakon toga počinje da se izvršava aktivnost Game.

Ovde se kreiraju instance sledećih klasa: Bullets, Target, TargetClass, AttackOnTarget.

Bullets - klasa koja implementira Runnable interfejs za krerianje Thread-a. Unutar nje metoda run služi za kretanje metaka koji su se kreriali u Game aktivnosti.

```
@Override
public void run() {
    while(runThread)
    {
        for (int i=0;i<bullets.size();i++) {
            if(bullets.get(i).getY()>100)
                bullets.get(i).setY(bullets.get(i).getY() - 20);
            else{
                bullets.remove(i);
            }
        }
        try {
            Thread.sleep( millis: 40);
        } catch (InterruptedException e) {
            System.out.println("interrupted");
        }
    }
}
```

TargetClass - klasa koja naslijeđuje klasu ImageView za predstavljanje metu i smjer u kom se meta kreće.

```
public class TargetClass extends ImageView {
    boolean moveRight=false;
    public TargetClass(Context context) {
        super(context);
        setImageResource(R.drawable.target);
    }

    public boolean isMoveRight() { return moveRight; }

    public void setMoveRight(boolean moveRight) { this.moveRight = moveRight; }
}
```

Target - klasa koja implementira Runnable interfejs za krerianje Thread-a. Unutar nje metoda run služi za kretanje meta koje su se kreriale u konstruktoru Target klase.

```
    public void run() {
        while(runThread)
        {
            if(targets.size()>0)
            {
                if(targets.get(0).getX()<=0)
                {
                    for(int i=0;i<targets.size();i++) {
                        targets.get(i).setMoveRight(true);
                    }
                }
                else if(targets.get(targets.size()-1).getX()>=(this.screen.getWidth()-100))
                {
                    for(int i=0;i<targets.size();i++) {
                        targets.get(i).setMoveRight(false);
                    }
                }
                for(int i=0;i<targets.size();i++)
                {
                    if(targets.get(i).isMoveRight())
                    {
                        targets.get(i).setX(targets.get(i).getX() + 20);
                    }
                    else
                    {
                        targets.get(i).setX(targets.get(i).getX() - 20);
                    }
                }
            }
```

AttackOnTarget - klasa koja sadrži metodu za detektovanje sudara koja se poziva u odvojenom Thread-u.

```
//metoda za detektovanje susreta
public void AttackDetect(){

        for(int j=0;j<bullets.size();j++)
        {
            for(int i=0;i<targets.size();i++)
            {
                ImageView targeti=targets.get(i);
                ImageView bulletj=bullets.get(j);

                if((bulletj.getX()>=targeti.getX() && bulletj.getX()<=(targeti.getX()+targeti.getWidth()))
                        && (bulletj.getY()<=(targeti.getY()+targeti.getHeight()) && bulletj.getY()>=targeti.getY()))
                {
                    screen.removeView(targets.get(i));
                    screen.removeView(bulletj);
                    bulletj.setY(0);
                    this.targets.remove(i);
                    points++;
                    this.score.setText(points+"");
                }
                else if(bulletj.getY()<150)
                {
                    screen.removeView(bulletj);

                }
            }
        }
```

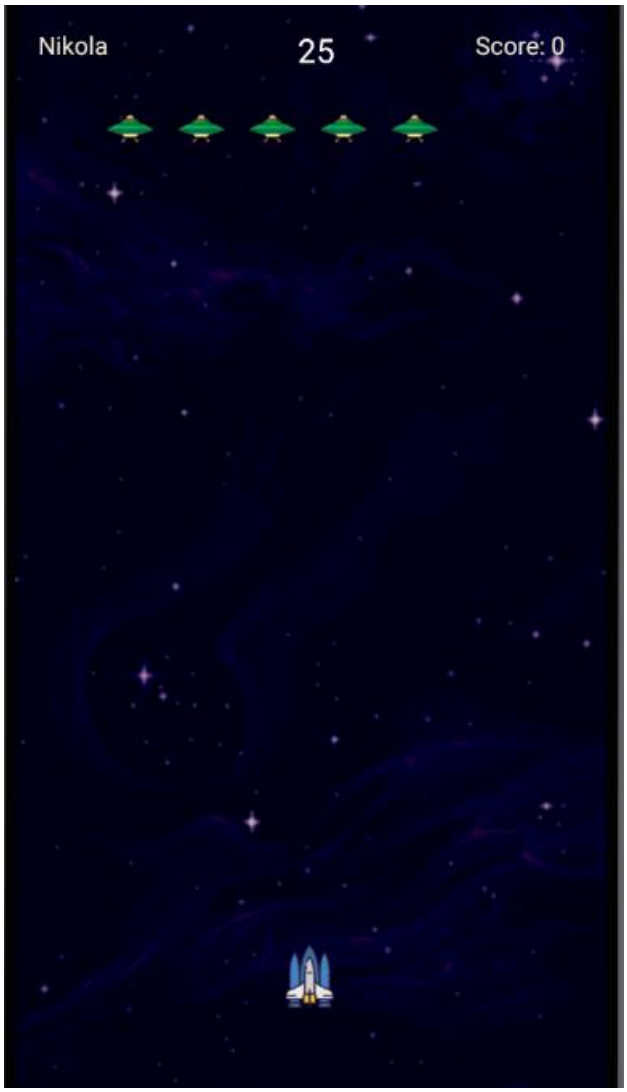
U klasi Game krerira se Thread za kreiranje metaka.

```
//thread za dodavanje metaka
createBulletThread=new Thread((Runnable) () → {
        while(runThread)
        {
            runOnUiThread(() → {
                    if(add) {
                        //Krerianje metak i dodvanje u niz
                        ImageView bullet = new ImageView( context: Game.this);
                        bullet.setX(attackerImage.getX() + attackerImage.getWidth()/2);
                        bullet.setY(attackerImage.getY());
                        bullet.setImageResource(R.drawable.bullet);
                        bullets.add(bullet);
                        screen.addView(bullet);
                    }
            });
            try {
                Thread.sleep( millis: 500);
            } catch (InterruptedException e) {
                System.out.println("interrupted");
            }
        }
});
```

5 Tok igre

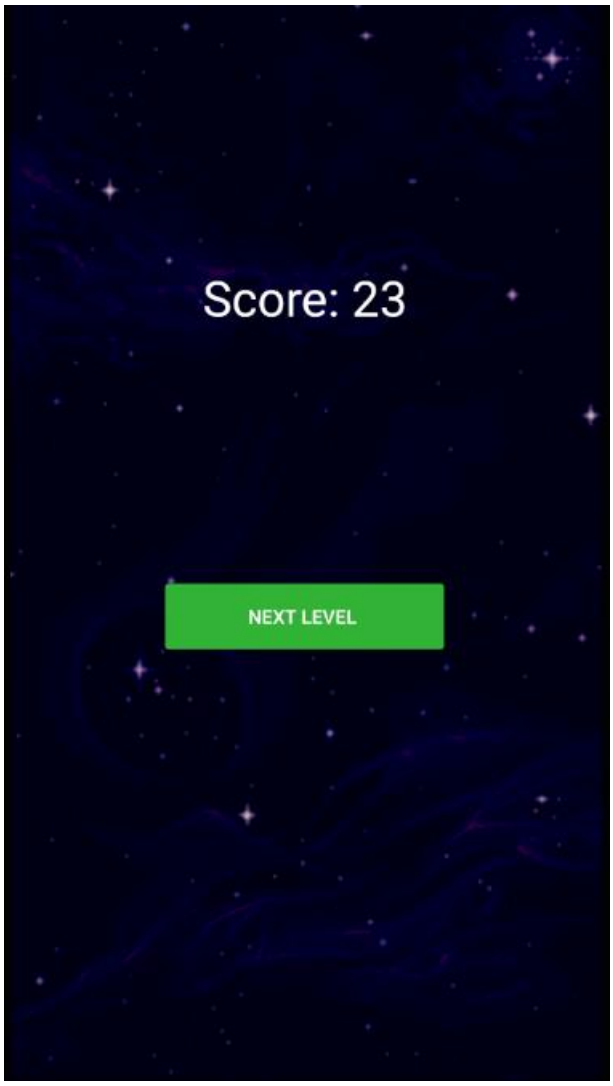
Ova aplikacija počinje izvršavanjem klase MainActivity gdje korisnik unosi ime i započinje igru.

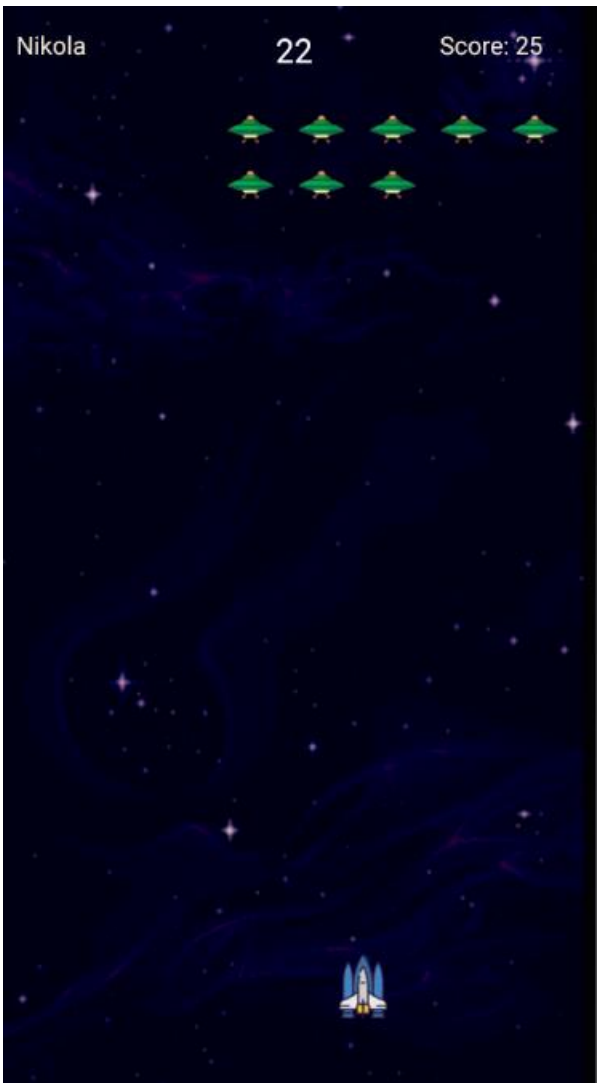
Nakon toga počinje da se izvršava Game aktivnost gdje počinje prvi nivo igre. U vrhu ekrana se nalaze mete koje se kreću lijevo-desno i koje je potrebno uništiti da bi se prešao trenutni nivo.



Pomjeranjem „napadača“ kreiraju se meci koji se ispaljuju sa njegove trenutne pozicije. Ako igrač uništi sve mete prije isteka vremena, prelazi na sledeći nivo. Na svakom sledećem nivou se dupla broj meta. Ako korisnik ne stigne uništiti mete prije isteka vremena, igra se završava i korisnik može ponovo da započne igru od prvog nivoa.

Bodovi se računaju tako što se na kraju svakog nivoa sabere preostalo vrijeme i broj unistenih meta.





Zaključak

Na osnovu aplikacije „Space battle“ prikazana je mogućnost multithreading-a. Konkretno u ovom slučaju multithreading je korišćen za istovremeno i nezavisno kretanje meta i metaka u igrici, kao i detektovanje njihovih sudara. Na jednoj niti bi ova funkcionalnost bila nemoguća zbog same arhitekture procesora računara koji u jednom trenutku može da obavlja samo jednu funkciju na jednoj niti. Sama aplikacija pokazuje veliku moć paralelizma u svijetu programiranja jer istovremeno računar može da izvršava više procesa. Možemo primjetiti porast igara za mobilne uređaje u zadnjih nekoliko godina, te zbog te činjenice, i činjenice da svi novi uredjaji imaju ekran na dodir, možemo reći da je potražnja za igrama na takvim uređajima dominantna naspram klasičnih puteva igranja poput starih arkadnih uredjaja ili modernih konzola. To naravno govori da budućnost igara za mobilne uređaje nije upitna, te da ulaganje u razvoj istih je isplativ i veoma profitabilan način zarade.