Mobilna aplikacija "Potapanje brodova"

Novak, Marijan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

MOBILNA APLIKACIJA „POTAPANJE BRODOVA“

Završni rad

Marijan Novak

Osijek, 2016.

Sadržaj

1. UVOD

Ovaj rad pratit će izradu mobilne izvedbe svima poznate društvene igre potapanja brodova. Ona treba biti zasnovana na otvorenom mobilnom operacijskom sustavu Android tvrtke Google Inc. Aplikacija mora omogućiti korisnicima da mogu igrati na jednom uređaju ili da povežu dva uređaja putem bežične mreže poput *Bluetootha* ili *WiFi*-ja. Kada korisnici igraju na jednom uređaju, tj. lokalnu igru, prvo će jedan postaviti svoje brodove, proslijediti drugom da učini isto, pa će igra početi. Pojavit će se tablični prikaz postavke igračevih brodova i na dodir osjetljiv prikaz na kojemu pogađamo protivničke brodove. Nakon odigranog reda, aplikacija će otvoriti prijelazni prikaz koji će zaštiti korisnikovu postavu igre od pogleda protivnika i omogućiti mu da pritiskom na tipku nastavi igru. Tijek igre pratit će i opisivati obavijesti u korisničkom sučelju. U aplikaciju će se moći upisati imena dvaju korisnika, pratiti rezultate igre poput pobjednika i vremena igre te ih pohraniti u datoteku u obliku *MySQL* baze podataka. Kada korisnici igraju svaki na svojem uređaju, povezat će se preko *Bluetootha* i igra će teći istim tijekom kao i kod lokalne igre s izuzetkom sada nepotrebnog prijelaznog prikaza. Aplikacija će također sadržavati aktivnost gdje će se moći uključiti i isključiti zvuk, pročitati rezultate igre te osnovne informacije o autoru. Pismeni dio rada će opisati značajke i arhitekturu Android platforme i ukratko je objasniti. Prikazat će se osnove rada u razvojnom okruženju Android Studio i programskom jeziku Java na kojem je zasnovan. Način izvođenja dijelova aplikacije bit će objašnjeni preko priloženog koda, dijagrama tokova i sličnog. Kako bi se ova aplikacija izradila potrebne su teorijske podloge iz objektno orijentiranog programiranja, programskog jezika Java i Android operacijskog sustava. Njih je relativno jednostavno steći iz razne besplatne literature i *online* tečajeva.

U idućim poglavljima detaljnije će se opisati tijek izrade mobilne aplikacije, rezultati i korišteni alati. Drugo poglavlje postavit će teorijske podloge potrebne za rad opisujući sam Android operacijski sustav, Android Studio razvojno okruženje i programski jezik Java. Objasnit će logiku same igre potapanja brodova. Slijedeće poglavlje opisat će ideju njene implementacije kao mobilne aplikacije. Četvrto će dati konkretna rješenja za postizanje ciljeva postavljenih unutar ideje te opisati njihovu funkciju, način rada i tijek izvođenja. Navest će se efikasnost koda i eventualne greške te prijedlozi za njihovo ispravljanje. Na kraju, zaključit će se o uspješnosti izvedbe rada i problemima s kojima smo se susreli.

1.1. Zadatak rada

Zadatak rada jest izraditi mobilnu aplikaciju za Android operacijski sustav koja će implementirati popularnu igru „Potapanje brodova“ s mogućnosti igre u dvoje na jednom ili dva bežično povezana uređaja. Također, potrebno je omogućiti praćenje rezultata i statistike igre, te njihovu pohranu u datoteku. Pismeni dio rada pratit će tijek izrade aplikacije i opisati značajke i arhitekturu samog Android operacijskog sustava.

2. TEORIJSKE PODLOGE I KORIŠTENI ALATI

U ovom poglavlju bit će postavljene teorijske podloge potrebne za rad na zadatku. Opisat će se Android operacijski sustav, njegove značajke i arhitektura. Programsko okruženje bit će objašnjeno i prikazano slikama. Objasnit će se osnove Java programskog jezika, njegove karakteristike i upotreba pri izradi Android aplikacija.

2.1. Android operacijski sustav

Android je otvoreni operacijski sustav za mobilne uređaje izdan od strane tvrtke Google Inc [1]. Vodi ga Open Handset Alliance, grupa čiji je cilj ubrzati inovacije na području mobilnih operacijskih sustava i ponuditi korisnicima što bolje iskustvo i prilagodljivost pri upotrebi [4]. Ta grupa izdala je skup alata za razvoj Android aplikacija pod imenom Android SDK. Taj skup alata baziran je na objektno orijentiranom programskom jeziku Java. Samu ideju Androida pokrenuli su Andy Rubin, Rich Miner, Nick Sears i Chris White 2003. godine fokusirajući se na razvoj mobilne platforme potpuno prilagodljive zahtjevima korisnika. Zbog te prilagodljivosti sustav je lako prenijeti na razne uređaje slične pametnim telefonima. S tehničke strane Android možemo gledati kao Linux operativni sustav razvijen za ARM procesorsku arhitekturu koji se sastoji od modificirane Linux jezgre i skupa biblioteka za podršku. Iako je Linux distribucija, po nekim stvarima odstupa od ostalih, pa ne može pokretati druge standardne Linux programe. Njegova otvorenost olakšava programski razvoj što rezultira mnogim dostupnim aplikacijama. Sama mnogobrojnost programskih rješenja privlači i zadržava korisnike. Android je trenutno najrasprostranjeniji mobilni operacijski sustav. Koristi se na pametnim telefonima, tabletima, pametnim satovima i mnoštvu sličnih uređaja. Od izdavanja 2007. godine njegova upotreba eksponencijalno raste, pa je u drugom kvartalu 2015. godine pokretao čak 82.8% mobilnih telefona i tableta dostupnih na tržištu, naspram iOS-ovih 13.9% i Windows Phoneovih 2.6% [8]. Kroz godine Google je izdavao više verzija Androida od kojih je svaka donijela neku novost i poboljšanje u performansama, sučelju i podršci. Trenutno je najnovija stabilna Android 5.0.x Lollipop izdana 25. lipnja 2015. godine. Slijedeća verzija je Android 6.0.x Marshmallow koja još nije dostupna većem broju korisnika [4].

Slika 2.1. Ilustracija rasta upotrebe Androida [4]

2.1.1. Značajke

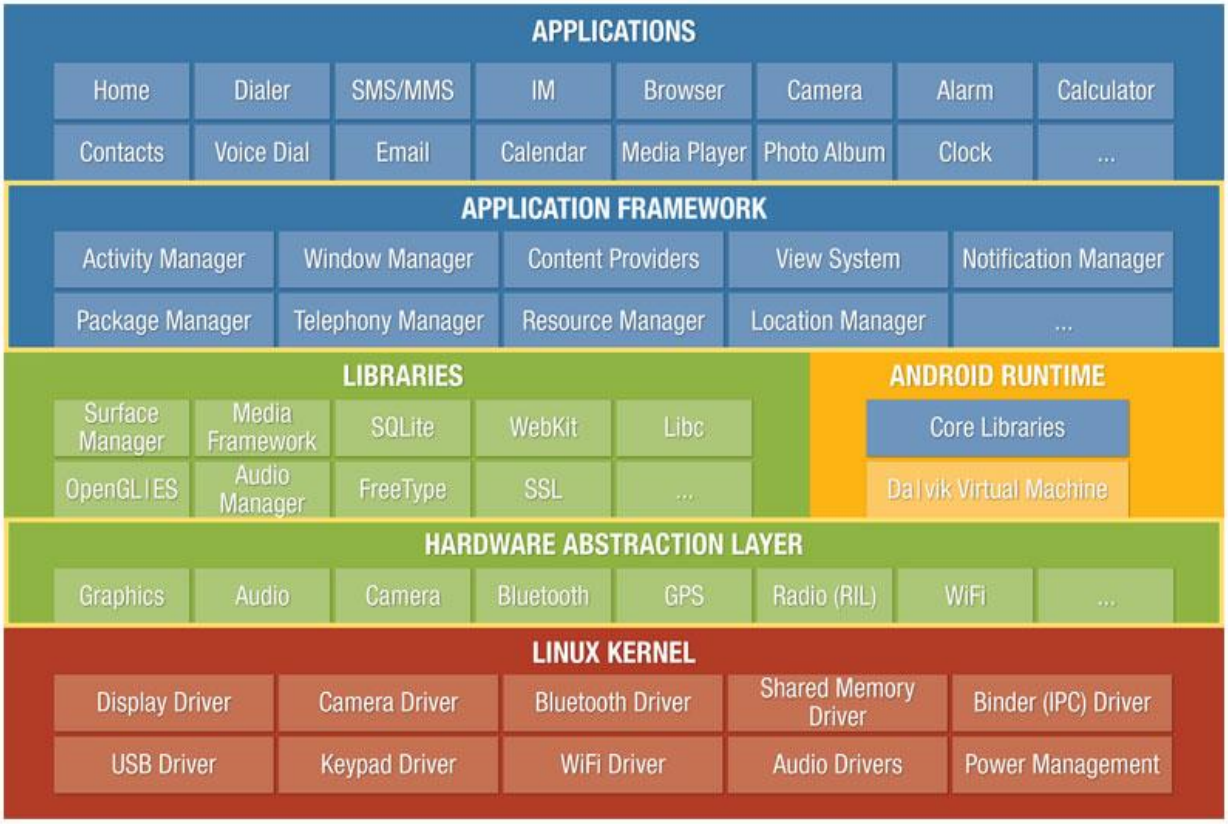
Android je otvoreni operacijski sustav koji na najbolji mogući način iskorištava sve resurse i mogućnosti uređaja na kojem je instaliran [5]. Svaka aplikacija može pristupiti svim osnovnim funkcijama uređaja poput *GPS*-a i kamere, a dodane su i razne podrške i biblioteke za njihovo upravljanje. Ovo razvojnim programerima znatno olakšava posao. Sustav je moguće značajno izmijeniti. Prilagođen je za uporabu na uređajima koji koriste 2D ili 3D grafičku biblioteku baziranu na OpenGL ES 2.0 specifikacijama. Za trajnu pohranu podataka implementiran je relacijski *SQLite* sustav za upravljanje bazama podataka (*DBMS*). Android uređaj se s drugim uređajima povezuje i prenosi informacije i datoteke preko *GSM, UMTS, Bluetooth, Wi-Fi, LTE, NFC* i drugih tehnologija. *SMS* i *MMS* servisi implementirani su za slanje i primanje poruka. Moguće je reproducirati razne formate multimedijalnih, tekstualnih i drugih vrsta datoteka. Neke od hardverskih podrški koje sadrži su podrške za ekran osjetljiv na dodir, *GPS*, brzinomjer, grafičko 3D ubrzanje i podrška za višedodirni ekran. Internet preglednik baziran je na *WebKit*-u i *V8 JavaScript engine*-u i naziva se Google Chrome. Iako je većina Android aplikacija napisana u Javi, platforma ne uključuje *Java Virtual Machine*, nego se koristi vlastito rješenje *Android Runtime* bazirano na *Dalvik* virtualnom stroju [6]. Sučelje sustava je bazirano na direktnoj manipulaciji, gdje korisnik gestama poput dodira i klizanja na prikazanim objektima upravlja uređajem i unosi podatke, uz virtualnu tipkovnicu. Reakcija na korisnikov unos dizajnirana je tako

da bude trenutna što omogućuje fluentnost u radu. Često uređaj daje vibracijsku povratnu informaciju (engl. *haptic feedback*) koja korisniku kazuje da je njegov unos registriran. Pri pokretanju sustav ulazi u prikaz zvan pokretač (engl. *launcher*). On nam omogućava pokretanje svih instaliranih aplikacija i interakciju s raznim grafičkim elementima (*widgetima*). Na vrhu ekrana vidimo statusnu traku koja prikazuje informacije o uređaju i njegovoj povezanosti, te obavijesti iz raznih aplikacija. Android podržava obavljanje više zadataka odjednom (engl. *multitasking*), što znači da možemo neku aplikaciju spustiti, raditi u drugoj neko vrijeme, pa opet otvoriti prvu i ne izgubiti podatke iz nje. Kada se aplikacija ne koristi sustav suspendira njene aktivnosti tako da ne koriste resurse poput baterije ili procesora, ali ostaju u memoriji dostupne za nastavak rada u njima. Preuzimanje aplikacija odvija se primarno na servisu Google Play s preko milijun aplikacija i 50 milijardi preuzimanja. Moguće ih je instalirati i iz raznih drugih izvora. Tu možemo pronaći sve aplikacije razvijene od strane Googlea i svih drugih razvojnih programera koji žele svoj rad podijeliti ili prodati. Pri tome, naravno, moraju poštovati Google-ove zahtjeve za kompatibilnost. Kako su većinski Android uređaji napajani baterijom, vrlo je važno svesti potrošnju električne energije na minimum. Google najavljuje implementaciju platforme virtualne realnosti u svom operativnom sustavu za jesen 2016. godine [4].

2.1.2. Arhitektura

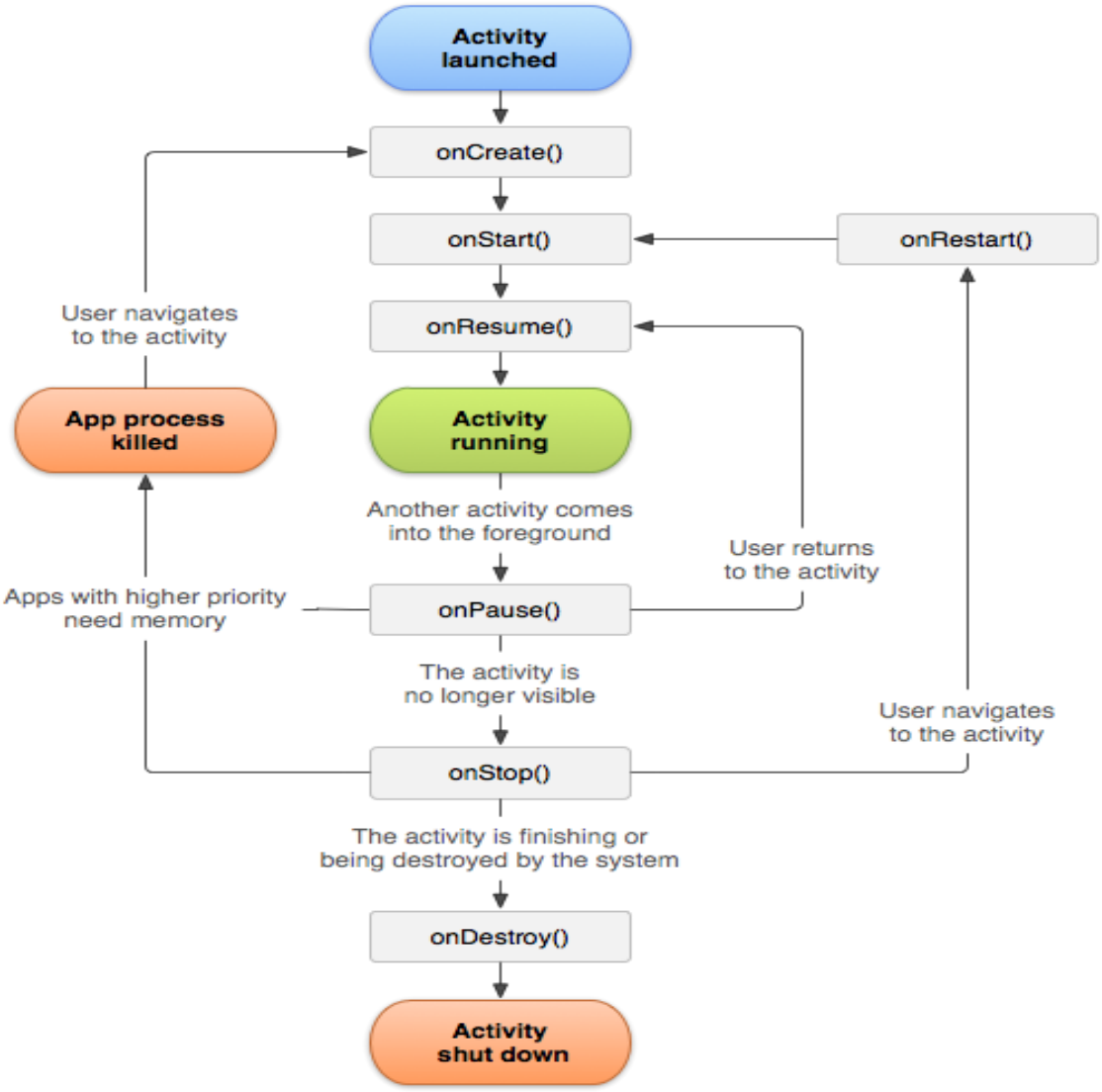
Android operacijski sustav je stog softverskih komponenata podijeljen u pet slojeva [5]. Svaki sloj grupira zajedno više programa, dijelova podrške, drivera i biblioteka. Na samom dnu nalazi se Linux jezgra. Ona zapravo nikada ne komunicira s korisnicima i razvojnim programerima, ali upravlja sve što se događa na uređaju. Njenu važnost vidimo iz zadataka koje obavlja. U to spada upravljanje memorijom, sigurnosne postavke, upravljanje napajanjem, podrška za dijeljene biblioteke, mrežni stog i ostali driveri. Sa svakom novom verzijom Androida poboljšava se i jezgra sustava donoseći nove mogućnosti i bolje performanse. Neki od bitnijih dijelova jezgre su driveri zaslona, kamere, dijeljene memorije, *Bluetootha*, napajanja, zvukovni i BIPC *driver*. Povrh jezgre nalazi se sloj za apstrakciju hardvera (*HAL*). Često se pripaja prvom sloju jezgre. On daje aplikacijama direktan pristup hardverskim resursima i osigurava im određenu razinu apstrakcije. Aplikacije tako ne moraju znati sva svojstva hardvera nego ga jednostavno mogu pozvati i upravljati njime. U trećem sloju nalazimo sve biblioteke nužne za rad aplikacija, *Android Runtime* (*ART*) i *Dalvik* virtualni stroj. Biblioteke sadrže skupove naredbi koje kazuju uređaju kako da

rukuje različitim tipovima podataka i bazirane su na Javi. Neke važnije su *Surface Manager* za kreiranje prozora na zaslonu, *SGL* za 2D grafiku, *Open GL* za 3D grafiku, *Media Framework* za multimediju, *WebKit* za Internet preglednik, *libc* za sistemske C biblioteke, *SQLite* za baze podataka i *Open SSL* za sigurni prijenos podataka. *ART* sadrži osnovne *C/C++* biblioteke koje omogućuju funkcionalnost navedenih Java biblioteka. Svaka pojedina pokrenuta aplikacija izvodi se u svojoj zasebnoj instanci *Dalvik* virtualnog sloja. Četvrti sloj jest sloj aplikacijskog okvira. Pruža servise više razine u obliku Java klasa. Razvojni programeri koriste te servise za izradu aplikacija. Aplikacije direktno komuniciraju s dijelovima ovog sloja. On je odgovoran za upravljanje aktivnostima, dijeljenjem podataka između aplikacija, telefonskim pozivima, lokacijom i resursima koje koristimo u aplikaciji. Najviši sloj je aplikacijski. On sadrži programska rješenja poput Internet preglednika, kalendara, telefona, igara i slično. Sva korisnička interakcija događa se na ovom sloju. U dodiru s nižim slojevima su većinom samo programeri. Aplikacije sastoje od četiri vrste komponenata: Aktivnosti (engl. *Activity*), Servisa (engl. *Service*), Pružatelja usluga (engl. *Content Provider*) i Prijemnika emitiranja (engl. *Broadcast Reciever*). Svaka Android aplikacija može pokrenuti komponentu druge aplikacije, pa stoga nije potrebno implementirati u svaku kodove za izvođenje te određene komponente. Aktivnost predstavlja jedan prikaz s korisničkim sučeljem. Jedna aplikacija sadrži jednu ili više aktivnosti koje se međusobno pokreću specijaliziranim porukama tzv. *Intentovima*. Servis je komponenta koja se pokreće u pozadini i izvodi dugotrajne operacije. On nema korisničko sučelje, a može biti pokrenut od strane druge komponente aplikacije. Pružatelj sadržaja upravlja pristup dijeljenom nizu aplikacijskih podataka. Preko njega aplikacija može zahtijevati i izmjenjivati te podatke. Na posljetku, prijemnik emitiranja je komponenta koja reagira na obavijesti emitirane kroz cijeli sustav. Također nema korisničko sučelje, ali može stvoriti obavijest u statusnoj traci na čiji pritisak otvaramo aplikaciju i pregledavamo sadržaj o kojem smo obaviješteni.



Slika 2.2. Arhitektura Android operativnog sustava [9]

Kako bismo razumjeli način na koji funkcionira aplikacija vrlo je važno znati što se događa s aktivnostima od pozivanja do brisanja iz memorije. Aktivnosti se tretiraju kao stog, te kad se nova pozove postavlja se na vrh tog stoga, pri tome da posljednja korištena aktivnost ostaje u pozadini [9]. One imaju četiri stanja: aktivne, pauzirane, zaustavljene i ugašene. Ako je aktivnost prikazana na zaslonu tada je aktivna, tj. na vrhu je stoga. Kada se preko aktivnosti pojavi druga transparenta ili aktivnost nepune veličine, ona je pauzirana. Sve informacije su sačuvane, a brišu se samo u slučaju da je sistemska memorija na vrlo niskoj razini. Ako se neka druga, *full screen* aktivnost prikaže preko prve, ona je zaustavljena. Ne vidimo je na zaslonu i često je sustav ugasi kako bismo iskoristili memoriju drugdje. Zadnje je potpuno gašenje aplikacije. To se događa nakon što je neko vrijeme provela zaustavljena ili pauzirana i kada ju korisnik opet otvori kompletno se resetira na početno stanje.



Slika 2.3. „Životni ciklus“ Android aktivnosti [4]

2.2. Android Studio

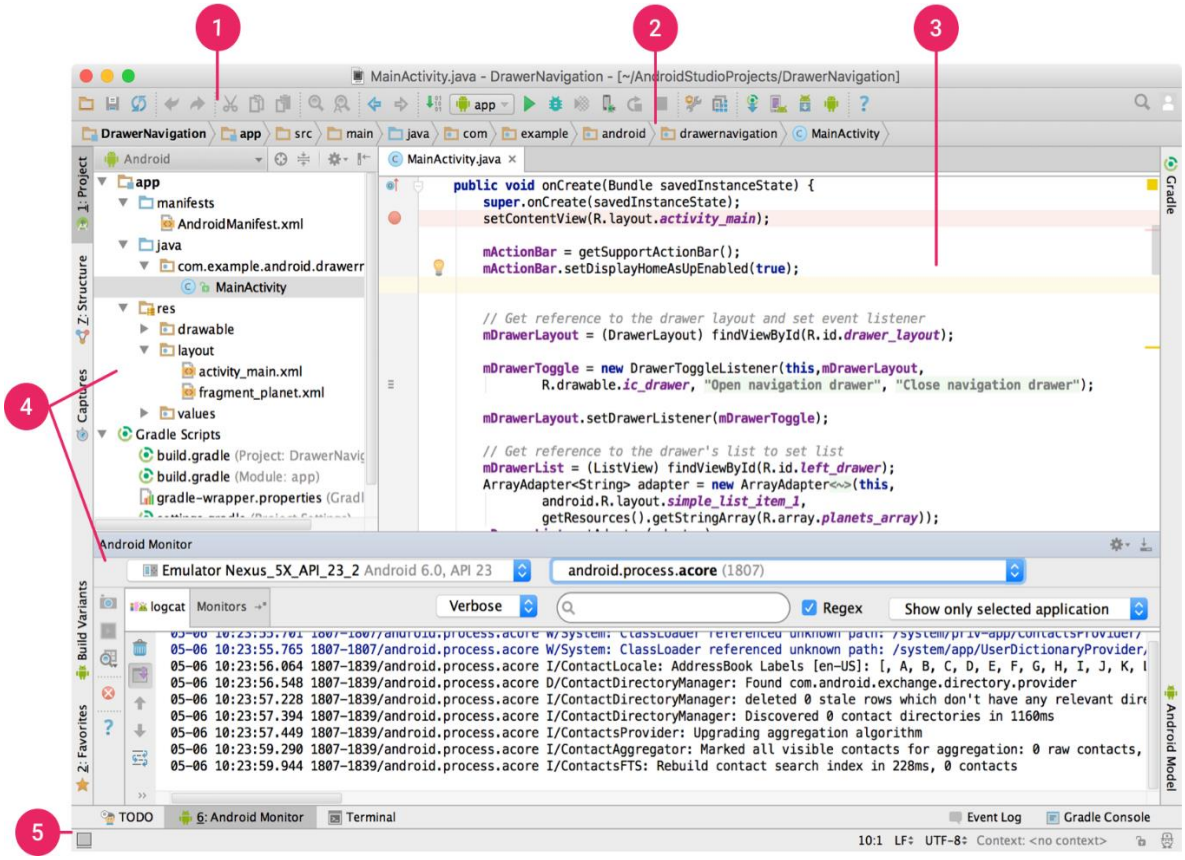
Ovo je Googleovo službeno integrirano razvojno okruženje (engl. *Integrated development environment* - IDE) za razvoj Android aplikacija [3]. Prva stabilna verzija 1.0 izdana je u prosincu 2014. godine, a najnovija jest 2.1.2. Baziran je na IntelliJ IDEA Java razvojnom okruženju čije mogućnosti dodatno nadograđuje. Neke bitnije značajke su fleksibilan Gradle *build system*, brz i bogat *Android Virtual Device* (AVD) emulator, ujedinjeno okruženje za razvoj za sve vrste Android uređaja, Instant Run funkcija za brze izmjene koda u aplikaciji bez ponovnog pokretanja i druge [4]. Svi potrebni alati su, kao i on, otvoreni i mogu se preuzeti s Interneta. Kako bi Android Studio mogao raditi, potrebno je instalirati i Android SDK alate, *Java Runtime Environment* te

Java Development Kit. Prije nastanka ovog alata, najviše se koristio *Eclipse IDE* s dodatkom za Android. Zbog mnogih prednosti i jednostavnosti masovno se prelazi na Android Studio. Aplikacije se izrađuju u Java programskom jeziku, a njihovo sučelje u ugrađenom editoru baziranom na *XML* jeziku. Pri izradi *layouta* na zaslonu se prikazuje pregled izgleda aplikacije. Ovo razvojno okruženje pri izradi novog projekta nudi nam automatsku izradu predložaka aktivnosti i sličnog. Projekti se sastoje od *manifest XML* datoteke koja sadrži osnovne informacije o aplikaciji i dopuštenja, java datoteka koje predstavljaju aktivnosti, klase i slično te svih resursa koji nisu kod poput *XML layouta*, slika i drugog. Kako bismo testirali napisan kod pokrećemo ga u ugrađenom AVD emulatoru ili na fizičkom uređaju. Koristeći Gradle *build system* izmjenjujemo, konfiguriramo i proširujemo proces stvaranja datoteke za pokretanje. Iz *softwarea* je moguće stvoriti APK datoteku koju korisnici mogu instalirati na uređaju ili generirati potpisani paket za učitavanje na Google Play. Također, Android Studio sadrži alate za otklanjanje pogrešaka i praćenje korištenja resursa uređaja.

Korisničko sučelje se sastoji od:

1. Alatne trake gdje pokrećemo razne operacije poput pokretanja aplikacije, emulatora i slično.
2. Navigacijske trake koja nam pomaže da navigiramo kroz projekt i otvaramo datoteke radi uređivanja.
3. Prozora *editora* gdje pišemo i uređujemo kod. Ako uređujemo *XML* datoteku tu se pojavljuje i prikaz izgleda aplikacije.
4. Alatni prozor s funkcijama za upravljanje projektom, otklanjanje grešaka, praćenje resursa itd.
5. Statusna traka prikazuje stanje projekta i razvojnog okruženja te izbacuje poruke i upozorenja.

Ovi dijelovi označeni su i vidljivi na slici 2.4.



Slika 2.4. Korisničko sučelje Android Studija [4]

Kada pokrećemo aplikaciju otvara se zasebni prozor ugrađenog AVD emulatora kojeg upravljamo pritiskom tipke miša i unosom tipkovnicom. Na njemu nije moguće testirati aplikacije koje koriste neke dijelove hardvera poput *GPS*-a, kamere, *Bluetootha* i sličnog.

Slika 2.5. Prozor pokrenutog AVD emulatora [4]

2.3. Java

Java je programski jezik visoke razine razvijen od strane kompanije Sun Microsystems i izdan 1995. godine. On je brz, siguran i pouzdan, a svrstava se u objektno-orijentirane programske jezike [2]. Pokreće velik broj uređaja i platformi te je u primjeni praktički svugdje. Pokušava programerima olakšati posao svojim principom „piši jednom, pokreni bilo gdje“. Aplikacije napisane u ovom programskom jeziku mogu se pokretati na bilo kojem uređaju koji podržava Javu bez potrebe za ponovnim stvaranjem izvršne datoteke. To radi pomoći Java virtualnog stroja (*JVM)* bez obzira na arhitekturu računala na kojem je on instaliran [10]. U razvoju Android aplikacija koristi se sintaksa Jave i velik broj već gotovih metoda specijaliziranih za Android. Na njemu se kod izvodi na posebnoj inačici JVM-a nazvanom *Dalvik* virtualni stroj, a sve više i na novom *Android Runtime*-u. Java je najpopularniji programski jezik i u 2016. godini ga koristi 20.8% programera, a slijede ga C i C++ sa 12.4% i 6.2% [12].

2.4. Potapanje brodova

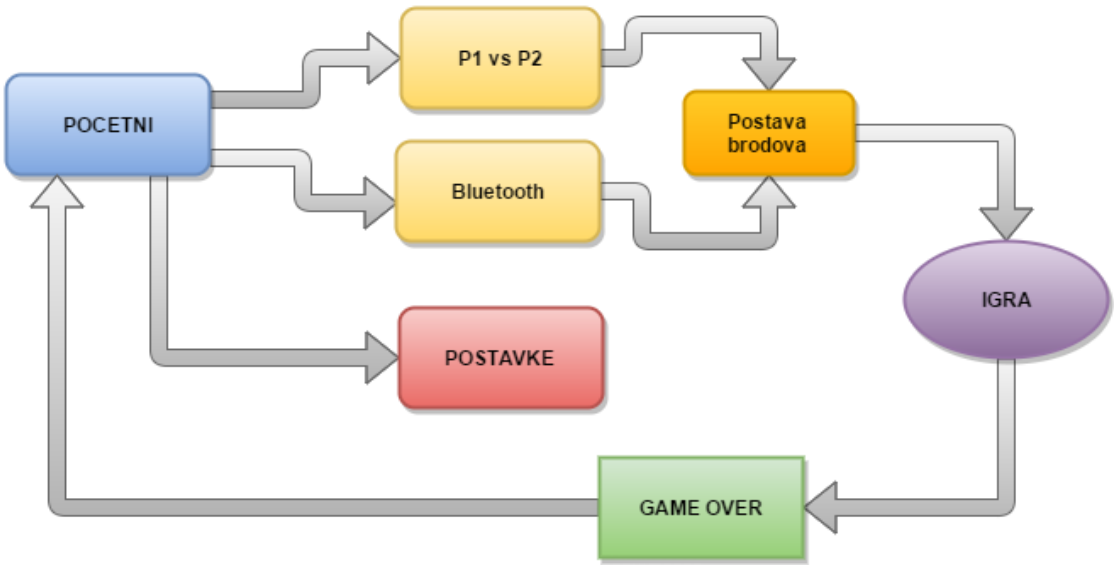
Potapanje brodova je popularna društvena igra kombinatorike. Igra se u dvoje. Cilj ove igre jest potopiti sve protivničke brodove, a da što više vaših ostane nepogođeno. Objavio ju je u Argentini 1982. godine Jaime Poniack, u svom časopisu Humor & Juegos. Za 11 godina se i pojavljuje u poznatom magazinu Games. Uobičajeno se igra pomoću olovke i papira, a u

posljednje vrijeme često se implementira kao računalna igra. Svaki igrač crta dvije tablice veličine 10x10, jednu za praćenje hitaca na protivničke brodove, a drugu za praćenje protivničkih hitaca na njegovu tablicu u koju je postavio brodove. Igrači se dogovaraju o broju i duljini brodova koje postavljaju. Sva polja koja brod zauzima moraju biti poravnata horizontalno ili vertikalno duž jedne linije. Brodovi se smiju dodirivati, ali ne smiju preklapati. Kada oba igrača postave brodove, igra počinje. Odvija se kao niz poteza gdje igrači naizmjenično pogađaju protivničke brodove. Igrač pita protivnika za određeno polje u tablici, a drugi ga obavještava nalazi li se njegov brod u tom polju. Ako je pogodio, nastavlja igrati, a ako nije red je na protivnika. Igra se dok jedan od igrača ostane bez brodova. Prilično ju je jednostavno implementirati kao mobilnu aplikaciju zbog jednostavne logike igre i lako izvedivog sučelja.

Slika 2.6. Prvo izdanje igre u magazinu Humor & Juegos [11]

3. IDEJA

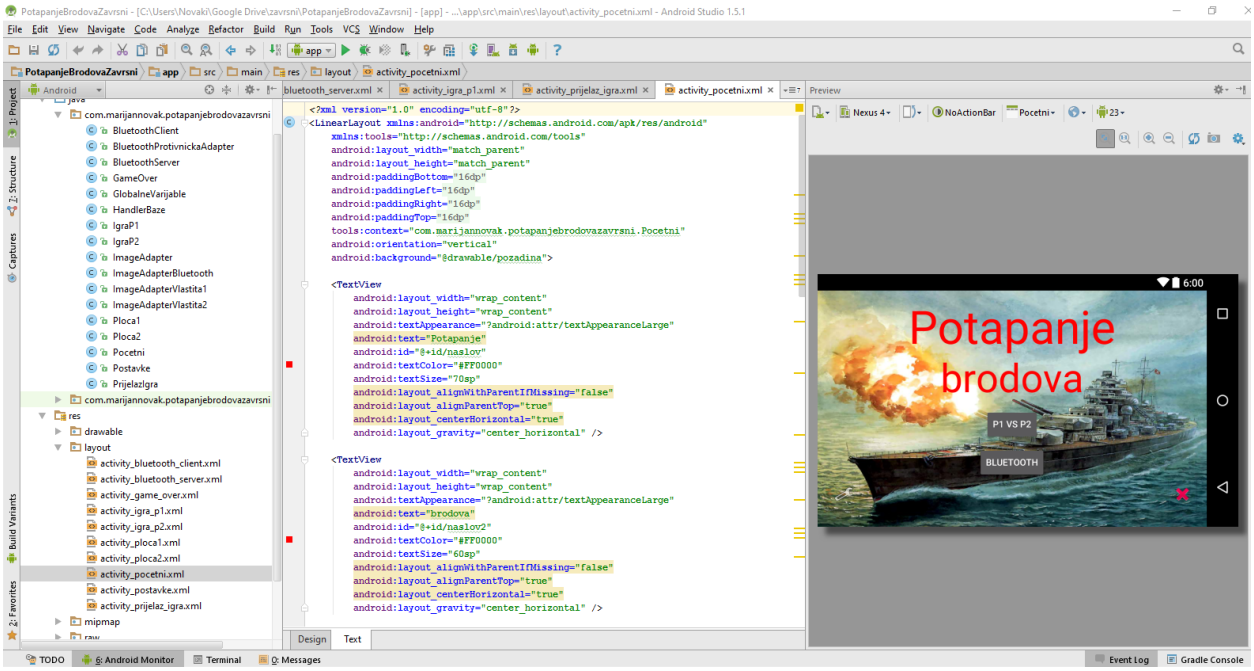
Ciljevi ovog rada postavljeni su u uvodu. Kako bi se uspješno napravila aplikacija treba prvo detaljnije isplanirati tijek njene izrade. Kao prvo, aplikacija će imati početni zaslon s naslovom igre „Potapanje brodova“ i tipkama koje vode u željeni način igre, postavke ili zatvaraju aplikaciju. Odabirom lokalne igre, aplikacija će prijeći na prikaz postave brodova prvog igrača, te ga tražiti da unese ime. Brodove će moći postaviti ručno ili jednostavno pritisnuti tipku za nasumičnu postavu. Kada završi, pritisnut će tipku dalje te predati uređaj drugom igraču da postavi brodove. Nakon toga, igra počinje. Igrač će na zaslonu imati tablični prikaz svoje postave brodova i prikaz na kojem će pogađati protivničke brodove. Ako pogodi nastavlja igrati, a ako promaši protivnik je na redu te aplikacija prikazuje prijelaznu aktivnost sve dok drugi igrač ne pritisne tipku za nastavak. Pogoci i promašaji bilježit će se i prikazivati na prikazanim tablicama Za predaju bit će dostupna tipka koja će upitati korisnika je li siguran, pa predati igru. Kada igra završi, predavanjem ili nečijim pogađanjem svih protivničkih brodova, prikazuje se aktivnost koja sadrži ime pobjednika i nudi korisniku povratak na početni zaslon. Pri tome bilježi se statistika i upisuje u bazu podataka. Igra preko *Bluetootha* odvijat će se u istom tijeku s izuzetkom prijelazne aktivnosti. Tijek aktivnosti igre vidljiv je na dijagramu sa slike 3.1. Odabiranjem postavki ponudit će se mogućnost uključivanja ili isključivanja zvuka, pregled statistike i informacija o autoru. Radit će se u Android Studiju i testirati na fizičkom uređaju Sony Xperia ™ M2.



Slika 3.1. Osnovni dijagram tijeka aktivnosti u aplikaciji

4. IZRADA APLIKACIJE

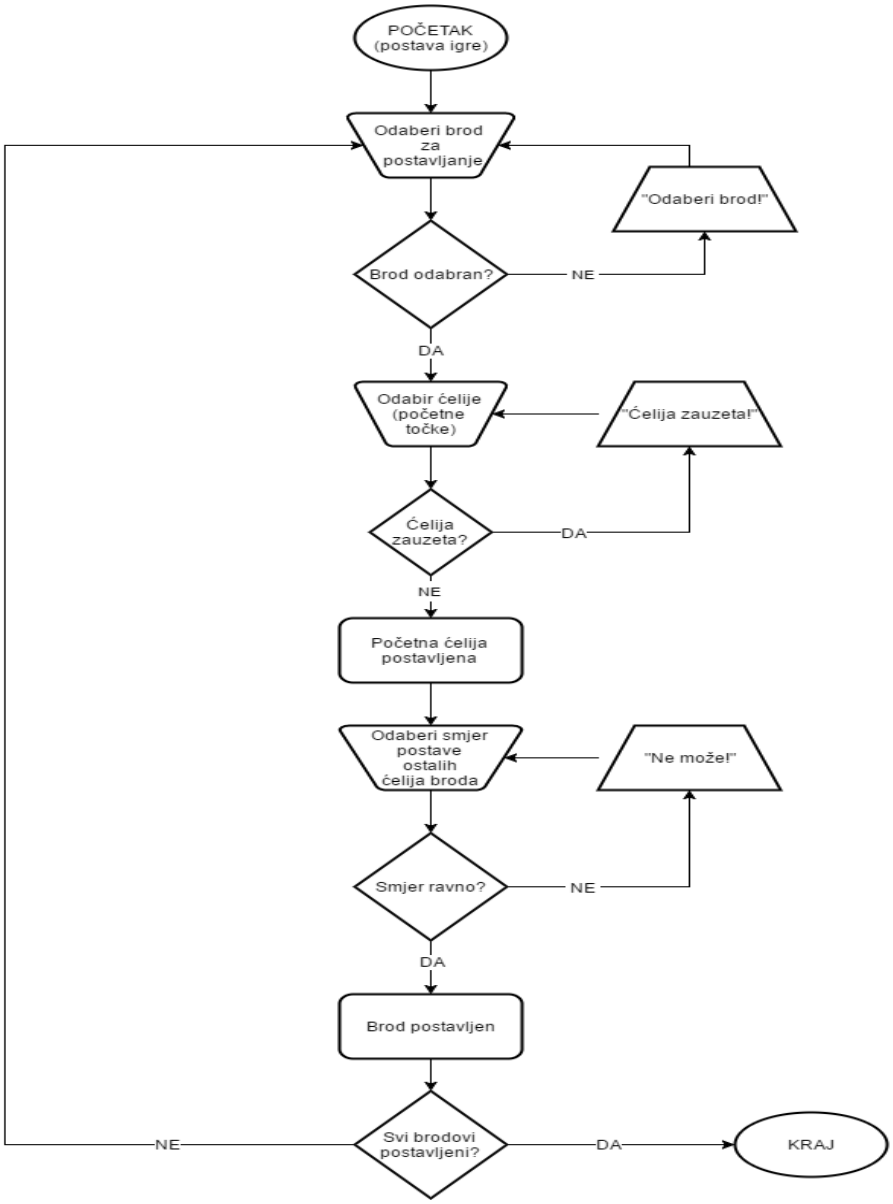
Izrađena aplikacija sastoji se od 10 aktivnosti, pripadajućih *layouta* i 7 Java klasa. Proces izrade *XML layouta* za početni zaslon i te sastavnice vide se na slici. Zbog vrlo velikog broja linija koda prikazat će se samo najbitniji dijelovi i metode. Nakon izrade početnog zaslona koji vodi na sve ostale aktivnosti trebalo je smisliti i implementirati logiku postave brodova i igre, te aktivnosti postavki i igre preko *Bluetootha*.



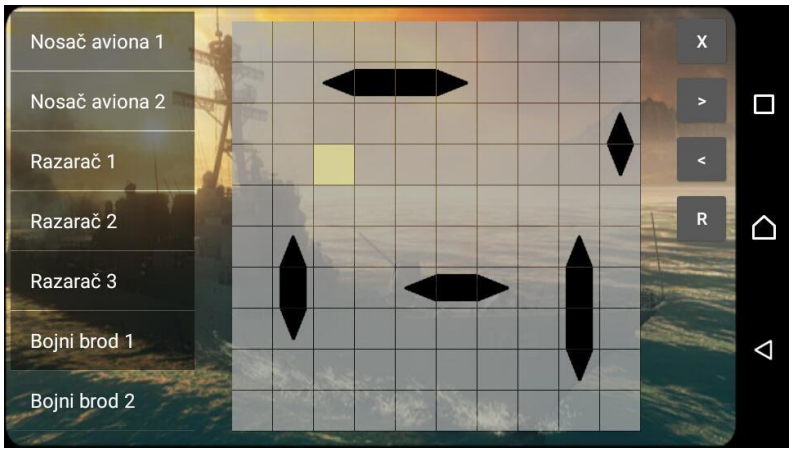
Slika 4.1. Izrada *layouta* za početni zaslon

4.1. Postavljanje brodova

Korisnik prvo treba odabrati određeni brod koji želi postaviti. Dok ga ne odabere pritisak na tablicu (*GridView*) za postavljanje samo će dati obavijest o neuspješnoj radnji. Kada odabere brod odabire početnu ćeliju, a ako je ona već zauzeta korisnik se o tome obavještava i čeka se drugi odabir. Nakon odabiranja početne ćelije, bira se smjer postavljanja broda. Sve ćelije moraju biti postavljene duž horizontalne ili vertikalne linije. Ako odaberemo smjer koji ne zadovoljava taj uvjet, aplikacija nam to javlja te očekuje drugačiji unos. Kada smo postavili sve brodove moguće je nastaviti na igru pritiskom na tipku „>“. Brodove je moguće postaviti nasumično pritiskom na tipku „R“. Ta metoda koristi istu logiku kao i ručno postavljanje, ali nasumično bira ćelije i smjerove. Identično se događa pri postavci brodova drugog igrača.



Slika 4.2. Dijagram toka programa pri postavljanju brodova



Slika 4.3. Aktivnost postavljanja brodova

Prvo inicijaliziramo tablicu za postavljanje brodova i njen adapter koji će ju popuniti.

```
final GridView tablicaBrodova = (GridView)
findViewById(R.id.tablicaBrodova);
final ImageAdapter celijeAdapter = new ImageAdapter(this);

tablicaBrodova.setAdapter(celijeAdapter);
tablicaBrodova.setOnItemClickListener(this);
```

Programski kod 4.1. Inicijalizacija tablice za postavljanje brodova

Spomenuta tablica jest *tablicaBrodova*, a *celijeAdapter* je instanca klase *ImageAdapter* napisane tako da popunjava tablicu željenim ikonama. Naredba *setOnItemClickListener* nam omogućava pritiskanje ćelija i pozivanje željene metode pri pritisku. Unutar te metode pišemo kod za postavljanje brodova.

```
if (brododabran) {

        switch (i) {

            case 0:

                if (pozicije1[position]) {
                    tost("Ćelija zauzeta!", 1000);
                } else {
                    i++;
                    temp = position;
                    pozicije1[position] = true;
                }
                break;

            case 1:

                if ((provjeriRedak(temp) ==
    provjeriRedak(position)) && ((temp -       position) < 0) &&
    (provjeriStupac(temp) <= (10 - vrstabroda - 1))) {

                    for (int j = 1; j <= vrstabroda; j++) {

                        if (pozicije1[temp + j]) {
                            nemoze = true;

                        }
                    }
                    if (!nemoze) {

                        for (int k = 0; k <= vrstabroda; k++) {

                            pozicije1[temp + k] = true;

                            if (k == vrstabroda) {

                                brododabran = false;
                                i = 0;
                            }
                        }
```

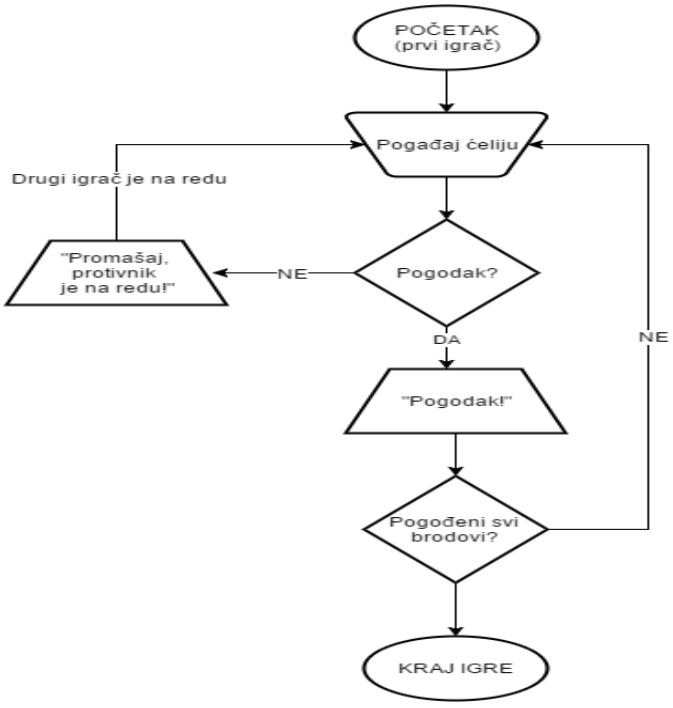
Programski kod 4.2. Postavljanje broda – slučaj kada je smjer desno od početne ćelije

Ako je brod s liste odabran varijabla *brododabran* je istinita i možemo odabrati početnu ćeliju. Nju u vektoru *pozicije1* upisujemo ako već nije zauzeta, spremamo varijablu posljednje pozicije u

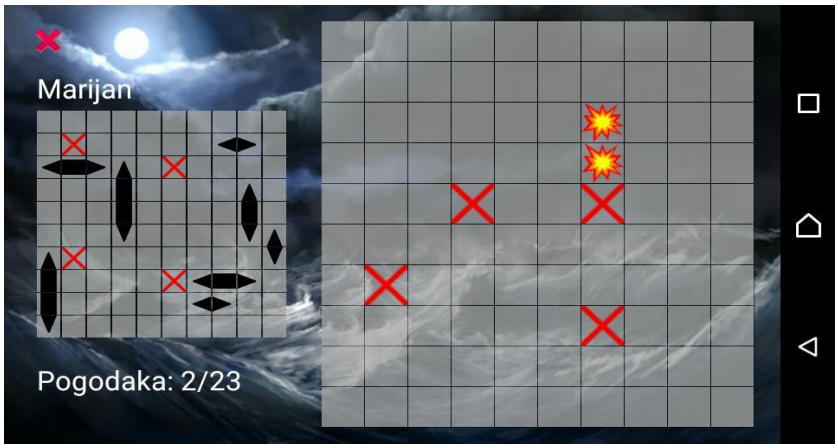
temp te povećavamo brojač *i* kako bi naznačili da je početna ćelija postavljena i možemo birati smjer. Kada se zadovolji uvjet da je smjer u horizontalnoj ili vertikalnoj ravnini s početnom ćelijom, kontrolna varijabla *nemoze* se postavlja u *false* i brod se postavlja. Ostale ćelije upisuju se u vektor, a kontrolna varijabla *brododabran* i brojač *i* se resetiraju. *Tost* je metoda koja prikazuje upisani tekst u notifikaciji na određeno vrijeme, a *provjeriRedak* i *provjeriStupac* vraćaju redak, odnosno stupac pozicije odabrane ćelije.

4.2. Igra

Kada igra započne, igrač na tabličnom prikazu (*GridView*) neprijateljskih brodova vidljivom na slici 4.5. pritišće ćeliju koju želi gađati. Ako se u njoj nalazi protivnički brod, dolazi obavijest o pogotku te je igrač i dalje na redu. U protivnom, ako promaši, aktivira se prijelazni prikaz i drugi igrač je na redu. Igra se tako odvija sve dok se svi brodovi jednog igrača ne potope. Tada aplikacija prelazi na aktivnost GameOver koja sadrži ime pobjednika. U manjem *GridViewu* s lijeve strane igrač prati gdje je protivnik gađao i koje mu je brodove pogodio. Ispod njega nalazi se ispis koji kazuje koliko je pogodaka postignuto. Igru je moguće predati pritiskom na crveni iksić u lijevom gornjem rubu zaslona. Tijek igre odvija se po dijagramu toka na slici ispod.



Slika 4.4. Dijagram toka igre



Slika 4.5. Zaslon tijekom igre

Slika 4.6. Prijelazna aktivnost

Kod zaslužan za provjeru pogotka ili promašaja prvog igrača te upisivanje tih pokušaja u vektor radi njihovog simboličkog prikaza u odabranim ćelijama slijedi ispod. Ako odabrana ćelija nije prije odabrana, tj. nije niti pogođena niti promašena što vidimo iz vektora *pokusajina2*, ispitujemo je li na toj ćeliji protivnički brod. To vidimo čitanjem iz vektora *drugePozicije* koji sadrži ćelije postavljenih brodova protivnika. Ako je uvjet ispunjen povećavamo varijablu *pogodaka* za 1, ispisujemo to na zaslon te upisujemo pokušaj u vektor *pokusajina2*. Isključujemo mogućnost pritiskanja ćelija tablica na neko vrijeme kako ne bi došlo do više slučajnih pritisaka naredbom *protivnicka.setEnabled(false).* Ako je broj pogodaka jednak ukupnom broju ćelija brodova, igrač je pobijedio te se *Intentom* veže aktivnost GameOver i pokreće. Ona ispisuje ime pobjednika te nudi odlazak na početnu aktivnost. Pri njenom pokretanju podaci o upravo odigranoj igri spremaju se u bazu podataka. U protivnom, ako je igrač promašio upisuje se pokušaj, mijenja varijabla *naredu*, što označava da je protivnik na redu, pa pokreće prijelazna aktivnost PrijelazIgra.

```
if((GlobalneVarijable.pokusajina2[position] != 1) &&
(GlobalneVarijable.pokusajina2[position] != 2)) {

        if (GlobalneVarijable.drugePozicije[position]) {

            tost("Pogodak!", 2000);
            pogodakap1++;

           igr1.setText("Pogodaka: " + pogodakap1 + "/23");
                 GlobalneVarijable.pokusajina2[position] = 1;


                  protivnicka.setEnabled(false);

                  if (pogodakap1 >= 23) {

                      GlobalneVarijable.pobjedio = 0;
                      Intent gotovo = new
      Intent(getApplicationContext(), GameOver.class);
                      startActivity(gotovo);
              } else if (!GlobalneVarijable.drugePozicije[position])
      {

                  tost("Promašaj!", 2000);
                  GlobalneVarijable.pokusajina2[position] = 2;

                          GlobalneVarijable.naredu = 1;
                          Intent prebaci = new
      Intent(getApplicationContext(), PrijelazIgra.class);
                          startActivity(prebaci);
                      }
                  }, 2000);
              }

          }else{

              tost("Već odabrana ćelija!", 1000);
          }
      }
```

Programski kod 4.3. Metoda koja provjerava je li igrač pogodio protivnički brod

Slika 4.7. Aktivnost GameOver

4.3. Statistika

Kako bismo statistiku mogli spremati u bazu, potrebno je napisati Java klasu koja će ju stvarati, ažurirati i iščitavati podatke iz nje. Ta klasa je u ovom slučaju HandlerBaze. Samo treba stvoriti instancu te klase, te primjenjivati metode koje smo u njoj napisali.

```
@Override

public void onCreate(SQLiteDatabase db) {

    String query = "CREATE TABLE " + IME_TABLICE + " ( " +
            COL_1 + " INTEGER PRIMARY KEY AUTOINCREMENT," +
            COL_2 + " TEXT," + COL_3 + " TEXT," +
            COL_4 + " TEXT," + COL_5 + " INTEGER " +  ");";

    db.execSQL(query);}
```

Programski kod 4.4. Stvaranje baze podataka

Vidimo da u *String query* upisujemo SQL naredbe, te ih naredbom *db.execSQL* pokrećemo. Naredba *CREATE TABLE* stvara tablicu sa stupcima *COL1* do *COL5*. Nakon toga potrebno je stvoriti metodu koja će parametre stupaca tablice ažurirati. Tablica se briše, te stvara nova s izmijenjenim parametrima:

```
@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {

    db.execSQL("DROP TABLE IF EXISTS " + IME_TABLICE);
    onCreate(db);

}
```

Programski kod 4.5. Ažuriranje stupaca baze

Slijedeća bitna metoda je dodavanje statistike u bazu. Pozivamo bazu u načinu za pisanje i stvaramo *ContentValues* varijablu *vrijednosti* kako bismo u nju umetnuli vrijednosti za upisivanje u stupce tablice. Iz svakog stupca umećemo vrijednost pod određenim navedenim imenom. Varijabla *result* daje nam indikaciju je li upis u tablicu bio uspješan s *true* ili *false* ako nije.

```
public boolean dodajStatistiku(String Igrac1, String Igrac2,
String pobjedio, long vrijemeIgre){

    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues vrijednosti = new ContentValues();

    vrijednosti.put(COL_2, Igrac1);
    vrijednosti.put(COL_3, Igrac2);
    vrijednosti.put(COL_4, pobjedio);
    vrijednosti.put(COL_5, vrijemeIgre);


    long result = db.insert(IME_TABLICE, null, vrijednosti);
    if(result == -1)
        return false;
    else
        return true;
}
```

Programski kod 4.6. Dodavanje statistike igre u bazu podataka

Metodom *ukupnoVrijeme* upisujemo u kursor *res* zbroj svih vremena igara iz tablice te ga vraćamo radi ispisa. Na sličnom principu rade i ostale metode unutar klase.

```
public Cursor ukupnoVrijeme(){

    SQLiteDatabase db = this.getWritableDatabase();

    Cursor res = db.rawQuery("SELECT SUM (vrijemeIgre) FROM " +
     IME_TABLICE, null);

     return res;
}
```

Programski kod 4.7. Metoda koja vraća ukupno vrijeme igre iz statistike

U klasi HandlerBaze nalaze se još metode *najvisePobjeda*, *najmanjePobjeda*, *brojIgara* i *najduzaIgra*. Ispis podataka iz baze unutar postavki pokazat ćemo na primjeru za ispis najviše pobjeda.

```
public void najvisePobjeda(){

    Cursor najvisePobjeda = baza.najvisePobjeda();

    if(najvisePobjeda.getCount() == 0){

        Toast.makeText(Postavke.this, "Nema podataka u bazi!",
Toast.LENGTH_SHORT).show();
        return;
    }

    while(najvisePobjeda.moveToNext()){

        statistike.append("Najviše pobjeda: " +
najvisePobjeda.getString(0) + " (" +najvisePobjeda.getString(1)+")"
+ "\n\n"); }

}

showMessage("Statistika", statistike.toString());
```

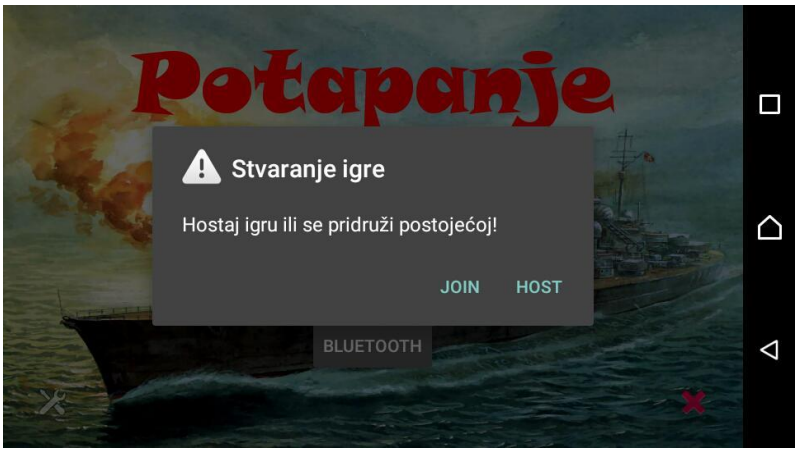
Programski kod 4.8. Ispis imena igrača s najviše pobjeda

Kreiramo kursor *najvisePobjeda* i postavljamo ga na vrijednost koju vraća metoda instance klase *HandlerBaze* imena *baza*. Ako u bazi nema podataka, o tome smo obavješteni, a ako ima, svi podaci se dodaju u *StringBuffer statistike*. Kada je unos podataka gotov, pozivamo metodu *showMessage* koja kreira *AlertDialog* i ispisuje statistiku kao *String*.

Slika 4.8. *AlertDialog* s podacima iz baze

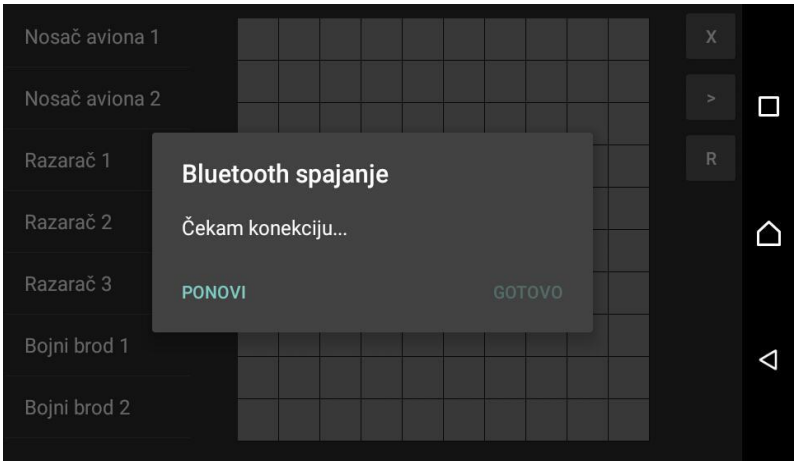
4.4. Bluetooth igra

Za igranje na dva uređaja, treba ih spojiti preko *Bluetootha*. Jedan uređaj se ponaša kao *server*, a drugi kao klijent. Opciju koji će biti što izabiremo u skočnom prozoru, te nas ovisno o tome aplikacija vodi u pripadnu aktivnost *BluetoothClient* ili *BluetoothServer*. Igra se odvija istim tijekom i logikom kao u lokalnoj igri.



Slika 4.9. Stvaranje Bluetooth igre

Implementacija logike igre i *layouti* su u obje aktivnosti identični, a jedina razlika je u kodu za spajanje *Bluetoothom*. U slučaju da pokrećemo igru kao *server*, uređaj automatski uključuje *Bluetooth* i vidljivost. Tada se pokreće novi *Thread spajanje* i čeka se da se klijent spoji. Kada su uređaji spojeni, pokreće se drugi *Thread reader* koji tokom cijele igre čita dolazne cjelobrojne varijable, tj. *integere* i ovisno o njima reagira. Kada je potrebno nešto poslati, tj. u slučaju da igrač pogađa brodove ili uređaj mora javiti drugome je li pogodio, pokreće se *Thread writter*, pošalje podatke te se odmah ugasi do idućeg pozivanja.



Slika 4.10. Spajanje uređaja kao *Bluetooth servera*

Thread spajanje nakon uspjeha pokreće čitanje koje ostaje aktivno do kraja igre. Varijablu *spajaj* koja služi kako bismo mogli ograničiti vrijeme čekanja *servera* postavljamo *u true*. Stvaramo varijable *start* i *end* za štopanje vremena čekanja *servera*. Tada pozivamo *Bluetooth* uređaj naredbom *BluetoothAdapter.getDefaultAdapter*. Stvaramo *socket* koji klijent treba prihvatiti i čekamo njegovo priključivanje. Kada je spajanje uspješno pokreće se *Thread reader*. Ove naredbe moraju biti obuhvaćene unutar *try-catch* bloka jer mogu izbaciti grešku.

```
private Runnable spajanje = new Runnable() {
    @Override
    public void run() {

        spajaj = true;
        BluetoothAdapter adapter =
BluetoothAdapter.getDefaultAdapter();
        UUID uuid = UUID.fromString("4e5d48e0-75df-11e3-981f-
0800200c9a66");

        long start = System.currentTimeMillis();
        long end = start + 120*1000;

        while (spajaj)
        {
            try {

            BluetoothServerSocket serverSocket =
adapter.listenUsingRfcommWithServiceRecord("BLTServer", uuid);

            socket = serverSocket.accept();

            new Thread(reader).start();

            } catch (IOException e) {
            e.printStackTrace();
            }
        }
    }
};
```

Programski kod 4.9. *Thread spajanje*

Thread reader sadrži uvjete i naredbe koje kazuju aplikaciji kako da reagira pri određenim primljenim podacima. Ako prima brojeve između 0 i 99, to znači da protivnik pokušava pogoditi vaše brodove. Tada treba provjeriti nalazi li se igračev brod na toj poziciji i preko *Threada writter* poslati potvrdu sa brojem 100 ili negaciju sa 101. Drugi uređaj, tj. klijent unutar identičnog *reader Threada* čita te brojeve i zaključuje je li pogodio. Kada uređaj pročita broj 103, znači da je na redu i može igrati, a ako je ulaz broj 104 protivnik se predao. Zbog velikog broja linija koda u *Threadu* pokazat će se samo primjer kada protivnik pogađa brodove. Ako je u ćeliji koju protivnik pogađa čiji je broj sadržan u varijabli *citanje* igračev brod, taj pokušaj se upisuje u vektor *bluetoothPokusaji*, te se inkrementira broj potopljenih brodova *potopljenih*. Ako je taj broj jednak

23, izgubili ste i preko *Intenta* se pokreće aktivnost GameOver. Varijabla *pobjedio* postavljena u 3 naznačava da je igrač izgubio. Protivniku se *Threadom writter* šalje broj 100 kako bi znao da je pogodio. U protivnom, ako je protivnik promašio šalje mu se 101, te varijabla *naRedu* označava da je korisnik na redu i može igrati. Broj pogodaka se ispisuje na ekranu preko varijable *pogodakaserver*.

```
    else if (citanje <= 99) {

        if (bluetoothPozicije[citanje]) {

                GlobalneVarijable.bluetoothPokusaji[citanje] = 1;

                Toast.makeText(BluetoothServer.this, "Vaš brod je
pogođen!", Toast.LENGTH_SHORT).show();
                potopljenih++;

                if(potopljenih == 23){

                    GlobalneVarijable.pobjedio = 3;

                    Intent gameover = new Intent(BluetoothServer.this,
GameOver.class);
                    startActivity(gameover);
                }

            pisanje = 100;
            WRITE = true;
            new Thread(writter).start();

        } else if (!bluetoothPozicije[citanje]) {

            runOnUiThread(new Runnable() {
                @Override
                public void run() {

                    GlobalneVarijable.bluetoothPokusaji[citanje] = 2;
                    Toast.makeText(BluetoothServer.this, "Protivnik je
promašio, na redu ste!", Toast.LENGTH_SHORT).show();

                    naRedu = true;
                    pogodakaserver.setText("Pogodaka: " +
String.valueOf(pogodaka));
                }
            });

            pisanje = 101;
            WRITE = true;
            new Thread(writter).start();

        }
```

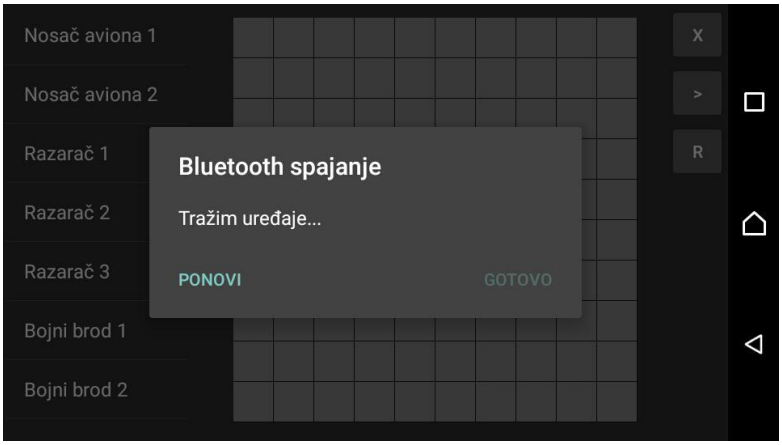
Programski kod 4.10. *Thread reader*

Slijedeći *Thead*, *writter*, je mnogo jednostavniji. Na *OutputStream*, tj. odlazni niz *os* šalje ono što upišemo u varijablu pisanje.

```
private Runnable writter = new Runnable() {

    @Override
    public void run() {

        if (WRITE) {

            try {

                os.write(pisanje);
                os.flush();
                Thread.sleep(10);

            } catch (Exception e) {

                android.util.Log.e("Exception", e.toString());

            }

            WRITE = false;
        }
    }
};
```

Programski kod 4.11. *Thread writter*

Aktivnost uređaja koji je klijent sadrži jednake *Threadove* za čitanje i pisanje, ali se onaj za spajanje razlikuje. Naredba *adapter.listenUsingRfcommWithServiceRecord* sa strane uređaja koji radi kao *server* prelazi u *createRfcommSocketToServiceRecord*, a *socket.accept* prelazi u *socket.connect*. Kada se ta aktivnost pokrene automatski se počinju pretraživati *Bluetooth* uređaji te se na onaj s istim *UUID*-om odmah spaja. Igra dalje izgleda i teče kao i u lokalnoj igri.



Slika 4.11. Spajanje uređaja kao *Bluetooth* klijenta

5. ZAKLJUČAK

Mobilna aplikacija „Potapanje brodova“ uspješno je izrađena, ispunjava zadane ciljeve, te sadrži sve tražene funkcionalnosti. Radi na operativnom sustavu Android kao što je i traženo u zadatku, a izrađena je u razvojnom okruženju Android Studio. Igra se može odvijati na jednom uređaju ili dva preko *Bluetootha,* a njen tijek se može pratiti obavijestima koje se ispisuju na ekranu. Statistika se sprema kao datoteka u obliku *SQLite* baze podataka i moguće ju je iščitati unutar aplikacije. U okviru ovog rada opisan je Android operacijski sustav, te njegove značajke i arhitektura. Dan je uvod u razvojno okruženje za izradu aplikacija i korišteni programski jezik. Sam tijek izrade aplikacije je opisan na što jednostavniji način. Njen izgled je pokazan slikama, a objašnjenje korištene logike dijelovima koda i dijagramima.

Izradom ovog završnog rada stečena su znanja o Android operacijskom sustavu, objektno orijentiranom programiranju u Javi i snalaženju u Android Studiju. S tim znanjem moguće je dalje napredovati i usavršavati se u pripadnom polju znanosti. Aplikacija je dobra zamjena za klasičnu igru potapanja brodova na papiru, te ju znatno olakšava i poboljšava iskustvo. Prilikom izrade aplikacije pojavljivali su se mnogi problemi, od kojih je većina nastala zbog nepostojećeg prijašnjeg iskustva u području. Problemi su otklonjeni istraživanjem dostupne literature i lekcija u programiranju. Proces izrade je zbog potrebe prvotnog učenja sintakse i stjecanja svih potrebnih znanja trajao nešto duže. Aplikacija, iako ne jako utjecajne, sadrži greške. Jedna od bitnijih je ta da tablični prikaz brodova na nekim određenim rezolucijama ekrana uređaja nema isti oblik. Taj problem moguće je riješiti ponovnom i kvalitetnijom izradom *layouta* te izradom ikonica raznih rezolucija, za svaku posebno. Dalje je moguće aplikaciju unaprijediti u vidu grafičkog izgleda, veće funkcionalnosti i prilagodljivosti korisniku.

LITERATURA

[1] Hello, Android: Introducing Google's Mobile Development Platform (Fourth Edition), Ed Burnette, The Pragmatic Programmers, LLC., 2015.

[2] Head First Java (Second Edition), Kathy Sierra, Bert Bates, O'Reilly Media,Inc., 2005.

[3] Android Studio Development Essentials (Android 5 Edition), Neil Smyth, eBook Frenzy, 2015.

[4] Android službena web stranica https://developer.android.com/ (lipanj 2016.)

[5] Tutorials Point – tutorijali iz programiranja http://www.tutorialspoint.com/ (lipanj 2016.)

[6] Besplatni online udžbenici https://en.wikibooks.org/wiki/Android (svibanj 2016.)

[7] Serijal Android video tutorijala https://goo.gl/hqtUCF (travanj 2016.)

[8] International Data Corporation http://www.idc.com/ (lipanj 2016.)

[9] Objašnjenja strukture Androida https://manifestsecurity.com/android-application-security/part-2 (lipanj 2016.)

[10] Java službena stranica https://java.com/en/download/faq/whatis_java.xml (lipanj 2016.)

[11] Povijest igre potapanje brodova http://www.conceptispuzzles.com/index.aspx?uri=puzzle/battleships/history (lipanj 2016.)

[12] Zastupljenost Jave i drugih programskih jezika http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (lipanj 2016.)

SAŽETAK

Tema ovog završnog rada jest izrada mobilne aplikacije „Potapanje brodova“. Zadatak je bio izvesti ju tako da se aplikacija može igrati na jednom uređaju ili dva putem bežične veze. Lokalna igra teče tako da poslije svog reda korisnik dodaje uređaj protivniku, a preko veze se igrači spajaju Bluetoothom. Nadalje, tijek igre može se pratiti obavijestima koje nam aplikacija ispisuje na zaslonu. Trenutni rezultati također su vidljivi. Kada igra završi u implementiranu bazu podataka se spremaju statistički podaci o njoj za naknadno ispisivanje na ekranu. Statistika sadrži podatke o ukupnom vremenu igre, broju igara, igračima sa najviše i najmanje pobjeda te najdužoj odigranoj igri. Rad opisuje potrebne teorijske podloge i tijek izrade aplikacije. Izvedba aplikacije je uspješna te sadrži sve zadane funkcionalnosti, iako postoje određene manje greške. Logika igre je objašnjena i popraćena dijelovima koda i dijagramima. Izgled aplikacije je prikazan slikama. Čitatelj iz rada dobiva generalne smjernice za izradu svoje aplikacije te najbitnije dijelove teorijske podloge.

KLJUČNE RIJEČI: Android, Android Studio, Java, objektno orijentirano programiranje, potapanje brodova, *Bluetooth*, aktivnost, statistika

ABSTRACT

In this thesis the theme is development of a mobile application named „Battleships“. The main task was to implement a local multiplayer game on a single device as well as a game on two wirelessly connected devices. Regarding the local game, players pass the device to their opponent after playing their turn, whereas the wireless game was implemented using Bluetooth. Furthermore, the game course can be monitored thanks to notifications displayed on the application screen. Current results are also visible. When the game is finished, the implemented database saves statistical data about it. These statistics contain information like total time played, best and worst player, as well as the longest lasting game. This paper describes the necessary theoretical background and the application development workflow. The application was made successfully and it includes all the required functionality, although there are some minor bugs. The game logic was explained and accompanied by parts of the code and diagrams. The layout is shown on the phone's screenshots. Paper's reader gets a general guideline to make his own application and the most important parts of theory.

KEYWORDS: Android, Android Studio, Java, object-oriented programming, battleships, *Bluetooth*, Activity, statistics

ŽIVOTOPIS

Marijan Novak rođen je 19.7.1994. godine u Virovitici. Osnovnu Školu Josipa Kozarca upisao je u Slatini 2001. godine i završio 2009. godine. Iste godine upisuje smjer elektrotehnika u Srednjoj Školi Marka Marulića u Slatini. Nakon završetka srednje škole upisuje Elektrotehnički Fakultet u Osijeku 2013. godine. Opredjeljuje se za smjer Komunikacije i Informatika.

MARIJAN NOVAK

_____