

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Matija Kralj

Matija Lešković

Nikola Petanjek

Matija Pianec

Josip Marijanović

Maze Solver 1 – DKU dokumentacija

**PROJEKTNI ZADATAK IZ KOLEGIJA ANALIZA I RAZVOJ
PROGRAMA**

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Matija Kralj, 0016118416, IPI

Matija Lešković, 0016117989, IPI

Nikola Petanjek, 0016117307, IPI

Matija Pianec, 0016117151, IPI

Josip Marijanović, 0016117952, IPI

Maze Solver 1 – DKU dokumentacija

PROJEKTNI ZADATAK IZ KOLEGIJA ANALIZA I RAZVOJ PROGRAMA

GITHUB REPOZITORIJ : [MAZE SOLVER 1 TIM](#)

Mentor:

Doc. dr. sc. Zlatko Stapić

Varaždin, 2018.

Sadržaj

1. Uvod.....	1
2. Svrha dokumenta	1
3. Popis funkcionalnosti.....	1
4. Android Studio.....	2
4.1. O Android Studiu.....	2
4.2. Preuzimanje razvojnog okruženja.....	3
4.3. Izrada novog projekta	3
4.4. Android implementacija	9
4.4.1. loadingScreen	9
4.4.2. MainActivity.....	11
4.4.3. SensorSelectionFragment	13
4.4.4. ConnectionTypeSelectionFragment	16
4.4.5. ListOfDevicesFragment.....	27
4.4.6. ConnectedDialogFragment	34
4.4.7. Izrada animacija.....	39
5. Visual Studio.....	44
5.1. O Visual Studiu.....	44
5.2. Preuzimanje Visual Studia.....	44
5.3. Instalacija Visual Studia	45
5.4. Izrada novog projekta	45
5.5. Arduino implementacija	47
5.5.1. Kreiranje novog projekta u Arduinu	47
5.5.2. Početak implementacije	48
5.5.2.1. Setup	49
5.5.2.2. Loop.....	49
5.5.2.3. CitajBluetooth()	51
5.5.2.4. IzvrsiRadnjuBT().....	52
5.5.2.5. Metode za kretanje robota	54
5.5.3. Implementacija algoritama na robotu	56
5.5.3.1. Algoritam sa prioriternim skretanjem u desno	56
5.5.3.2. Algoritam sa prioriternim skretanjem u lijevo.....	57
5.5.3.3. Labirint	58
5.5.3.4. Implementacija u Arduino okruženju	59
5.5.4. Implementacija u Android Studio – algoritam	60

5.5.4.1. Prednji ultrazvučni senzor uz čitač crte	60
5.5.4.2. Slučaj sa više ultrazvučnih senzora bez čitača crte	62
6. Preuzimanje gotovog projekta	66
7. Dizajn mobilne aplikacije	67
7.1. Kreiranje responzivnog dizajna	71
7.1.1. Maze Solver – Responzivan dizajn.....	71
Popis literature.....	75
Popis slika.....	76
Popis tablica.....	79

1. Uvod

Aplikacija Maze Solver 1 nastala je u sklopu projekta na kolegiju „**Analiza i razvoj programa**“, na „**Fakultetu organizacije i informatike u Varaždinu**“, u suradnji sa udrugom IRIM. Svrha aplikacije je izrada novih digitalnih edukacijskih materijala u sklopu projekta STEM revolucija u zajednici koji će biti objavljeni na edukacijskom web portalu IZRADI. Aplikacija Maze Solver 1 implementira prolazak mBot robota kroz unaprijed izrađeni fizički labirint koristeći varijabilni broj ultrazvučnih senzora. Uz varijabilni broj ultrazvučnih senzora, aplikacija autonomno upravlja mBot robotom te je broj senzora na mBot – u modularno riješen. Podrška ovom projektu bili su naši mentori, **Doc. dr. sc. Zlatko Stapić** i **Dr. sc. Boris Tomaš** sa već navedenog kolegija, „**Analiza i razvoj programa**“.

Za izradu ovog projekta i društveno korisne dokumentacije je zaslužan **Maze Solver tim**:

- **Matija Kralj** (AIR)
- **Matija Lešković** (AIR)
- **Josip Marijanović** (AIR)
- **Nikola Petanjek** (AIR)
- **Matija Pianec** (AIR)

2. Svrha dokumenta

Svrha DKU dokumentacije je izrada digitalnih edukacijskih materijala koji će biti objavljeni na edukacijskom web portalu IZRADI. U DKU dokumentaciji detaljno je opisan postupak izrade novih projekata u razvojnim okruženjima koje je naš tim koristio prilikom izrade Maze Solver aplikacije, sam proces izrade Maze Solver aplikacije te ostale bitne stvari koje će biti naglašene.

3. Popis funkcionalnosti

Aplikacija Maze Solver 1 nudi sljedeće funkcionalnosti:

1. Uparivanje mBota i mobilne aplikacije pomoću Bluetooth tehnologije
2. Uparivanje mBota i mobilne aplikacije pomoću WiFi tehnologije
3. Autonomno upravljanje robotom preko mobilne aplikacije

4. Android Studio

U ovom poglavlju bit će opisan sam Android Studio kao jedan od najrasprostranjenijih razvojnih okruženja za izradu mobilnih aplikacija na Android operacijskom sustavu, preuzimanje i instalacija Android Studio paketa.



Slika 1. Android Studio logo

4.1. O Android Studiu

Android Studio, kao što je već i rečeno, jedan je od najrasprostranjenijih razvojnih okruženja za izradu Android mobilnih aplikacija. Temelji se na IntelliJ IDEA, također razvojnom okruženju temeljenom na Java programskom jeziku koji služi za izradu računalnog softvera. IntelliJ IDEA razvijan je od strane JetBrainsa, dok je Android Studio razvijan također od strane JetBrainsa uz kompaniju Google. Android Studio dizajniran je specifično za razvijane mobilnih aplikacija za Android uređaje. Neke službeno objavljene funkcionalnosti Android Studia su:

- Fleksibilno građen Gradle – based sustav
- Jednostavan, relativno brz i funkcionalno bogat emulator Android mobilnog uređaja
- Razvojno okruženje koje nudi mogućnost razvoja softvera na svim Android uređajima
- Mogućnost pokretanja nove verzije softvera bez izgradnje i pokretanja novog APK – a (engl. **Android Package Kit**)
- Podrška prema Google Cloud Platformi, preko koje je moguće integrirati Google Cloud Messaging.

(Meet Android Studio, b. d.)

4.2. Preuzimanje razvojnog okruženja

U ovom poglavlju bit će prikazana izrada novog projekta u Android Studio razvojnom okruženju. Kao prvi korak može se navesti preuzimanje i instalacija Android Studio paketa. Preuzimanje paketa može se pokrenuti na sljedećoj poveznici: https://developer.android.com/studio/?gclid=CjwKCAiAl7PgBRBWEiwAzFhmmsA69qfflcNZqErlowtT6iAsTes8EGeKWv9H0r0mj4C18_bnXz69cxoCTikQAvD_BwE.

Prilikom preuzimanja, korisnik mora prihvatiti uvjete i odredbe, no to je već standardan postupak prilikom preuzimanja bilo kojeg softverskog rješenja.

4.3. Izrada novog projekta

Nakon što je razvojno okruženje uspješno instalirano na računalo, korisnik može kreirati svoj prvi projekt. Na slici nalazi se početni prozor nakon instalacije Android Studio okruženja. Na prozoru su prikazane sljedeće opcije:

- Izrada novog Android Studio projekta
- Otvaranje postojećeg Android Studio projekta
- Preuzimanje projekta s nekog od alata za verzioniranje koda kao što su: CVS, Git, Google Cloud, Mercurial, Subversion
- Debugiranje APK – a
- Uvoz projekta
- Uvoz Android uzorka koda



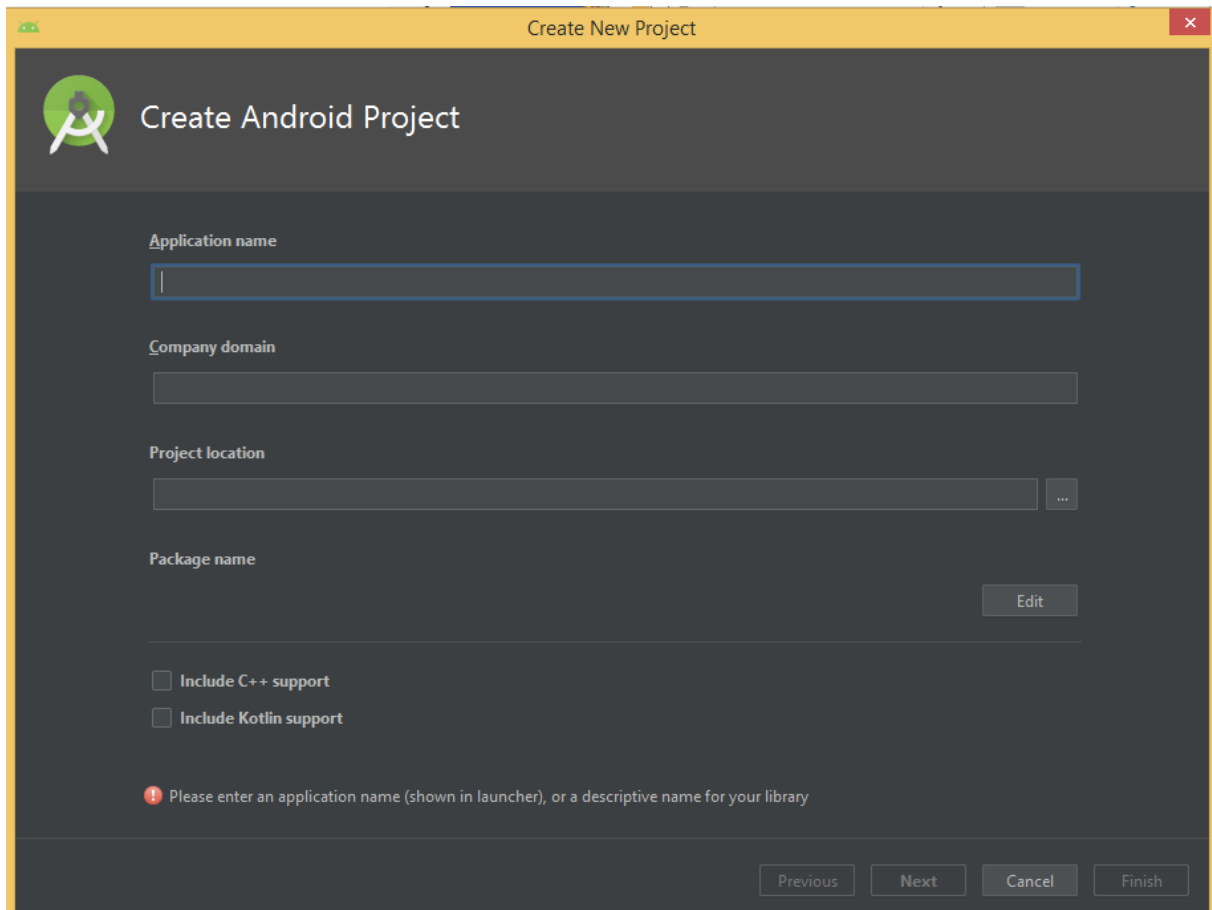
Slika 2. Početni prozor Android Studia

Ukoliko korisnik želi napraviti novi projekt, odabire prvu opciju, to jest *Start a new Android Studio project*.

Na slici prikazan je prozor koji se otvara nakon gore navedene opcije. U prvom tekstualnom okviru korisnik unosi željeni naziv aplikacije. Nakon upisivanja naziva aplikacije, korisnik može u sljedeći tekstualni okvir upisati domenu poduzeća. U trećem i posljednjem tekstualnom okviru korisnik odabire gdje će se novokreirani projekt spremiti na računalu. Također, u dnu prozora nalaze se dva odabira:

- Uključivanje podrške za C++ programski jezik
- Uključivanje podrške za Kotlin programski jezik

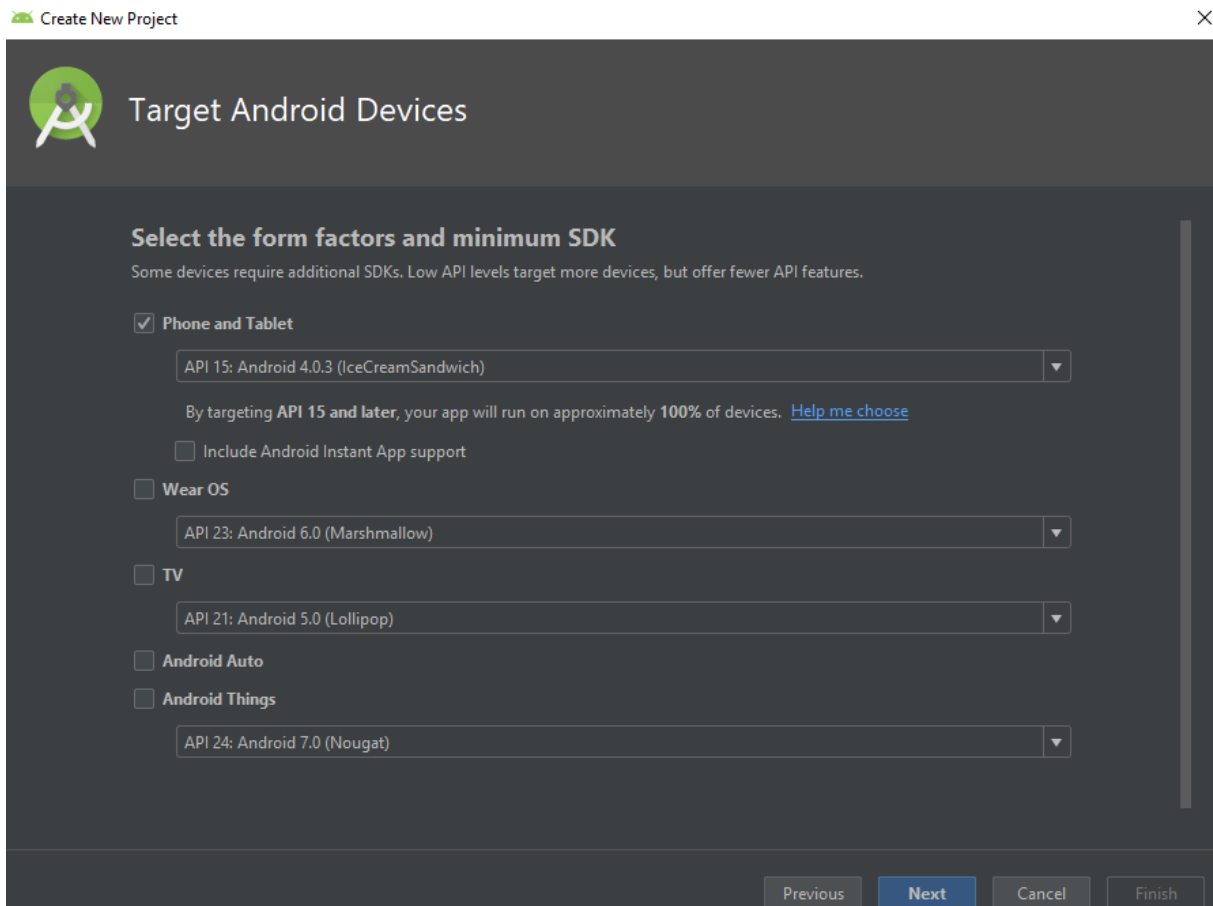
Nakon uspješno unesenih podataka, korisnik može pritisnuti na gumb Next koji ga vodi u daljnji postupak kreiranja novog projekta.



Slika 3. Kreiranje novog Android projekta

Na slici prikazan je prozor koji se otvara nakon pritiska na gumb Next. Korisniku se nudi odabir tipa uređaja za koji želi izraditi aplikaciju i verzija SDK – a (engl. **Software Development Kit**), pri čemu korisnik može proučiti koja verzija SDK – a mu je pogodnija. Starije verzije SDK – a nude veću pokrivenost uređaja dok novije verzije SDK – a nude više API (engl. **Application Programming Interface**) funkcionalnosti. Android Studio nudi sljedeće opcije koje su prikazane slikom:

- Mobilni uređaj i tablet – korisniku se prikazuje postotak pokrivenosti uređaja s odabranom verzijom SDK – a, te mogućnost pomoći koja se otvara pritiskom na poveznicu
- Wear OS
- TV
- Android Auto, te naposljetku
- Android Things

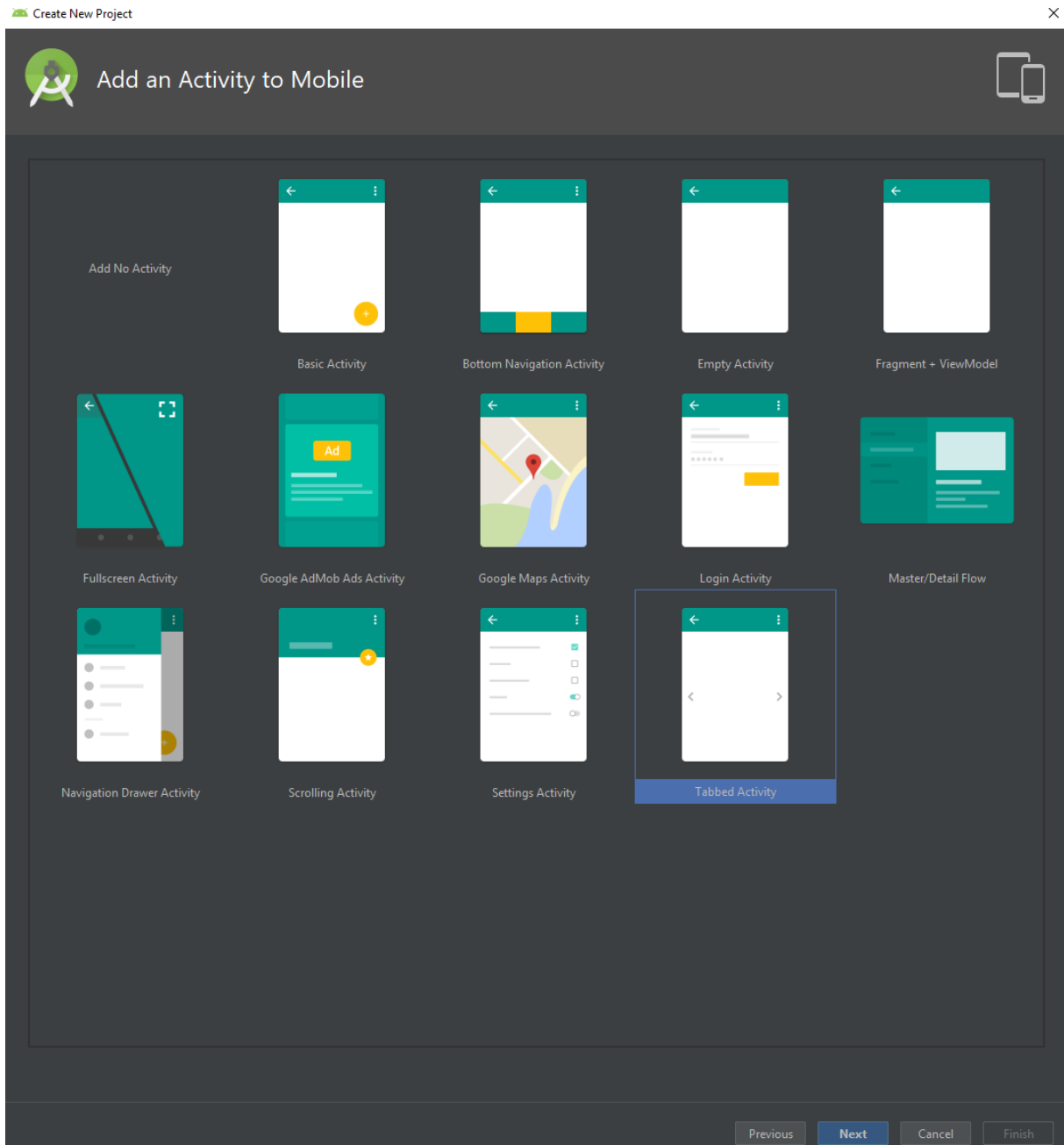


Slika 4. Odabir Android uređaja

Nakon što korisnik odabere uređaj za koji želi kreirati projekt i ponuđenu verziju SDK - a, može pritisnuti na gumb Next koji ga vodi u daljnji tijek kreiranja projekta.

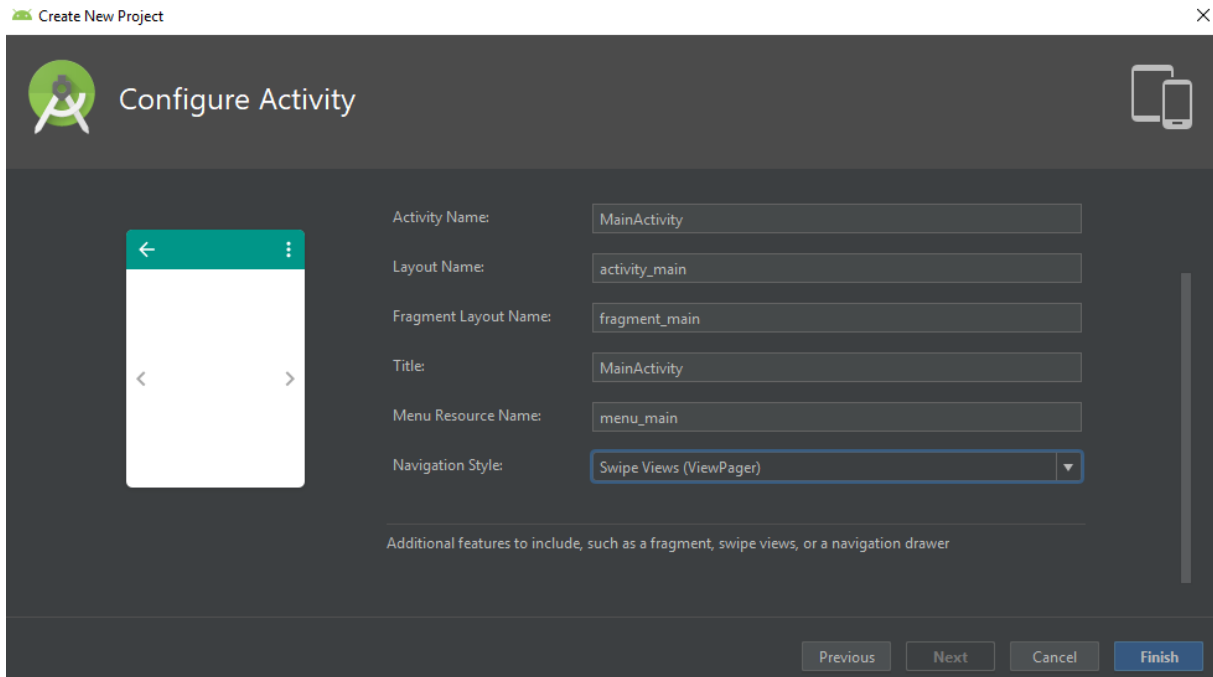
Pritiskom na gumb Next otvara se prozor na slici . U ovom prozoru, korisniku se nudi pregršt opcija što se tiče prikaza na mobilnom uređaju. Te opcije su:

- Bez kreiranja aktivnosti
- Opća aktivnost
- Aktivnost s navigacijom na dnu mobilnog uređaja
- Prazna aktivnost
- Izrada fragmenta i ViewModela
- Aktivnost koja se prostire kroz cijeli zaslon mobilnog uređaja
- Aktivnost koja sadrži Google AdMob reklamu
- Google Maps aktivnost
- Aktivnost koja nudi mogućnost prijave
- Aktivnost sa složenim obrascem
- Navigation drawer aktivnost
- Aktivnost koja nudi mogućnost scrollanja
- Aktivnost koja sadrži više kartica



Slika 5. Dodavanje početne aktivnosti u projekt

Nakon odabrane vrste aktivnosti, korisnik može pritisnuti gumb Next kako bi se preusmjerio na posljednji korak kreiranja novog projekta. Uz gumb Next, prozor sadrži i gumbove Previous i Cancel koji vraćaju na prijašnji korak i poništavaju proces kreiranja novog projekta.



Slika 6. Konfiguriranje aktivnosti

Na slici prikazan je posljednji korak prilikom kreiranja novog projekta. Na lijevoj strani nalazi se fotografija odabranog stila aktivnost (u ovom slučaju aktivnost s više kartica) te se na desnoj strani nalazi nekolicina tekstualni okvira. U prvom od njih korisnik upisuje naziv aktivnosti. Nakon toga, nalazi se tekstualni okvir u kojem korisnik unosi naziv rasporeda, odnosno .xml dokumenta u kojem se nalazi XML kod za aktivnost. Uz raspored, korisnik može upisati i naziv rasporeda fragmenta za odabrani stil aktivnosti. Uz sve navedeno, korisniku se pruža opcija zadavanja naziva naslova, naziva resursa te stil navigacije. Nakon što korisnik unese sve potrebne nazive, nude mu se tri mogućnosti: Pritisak na gumb Previous koji korisnika vraća jedan korak unatrag, pritisak na gumb Cancel koji prekida cijeli proces izrada novog projekta te pritisak na gumb Finish koji kreira novi projekt.

4.4. Android implementacija

U ovom će poglavlju biti prikazana implementacija Maze Solver 1 aplikacije u Android Studio razvojnom okruženju.

4.4.1. loadingScreen

LoadingScreen je klasa koja proširuje klasu AppCompatActivity koja nam služi kao bazna klasa za izradu aktivnosti. Svrha aktivnosti loadingScreen je u prikazu zaslona za vrijeme učitavanja. Sadrži jednu metode: onCreate. onCreate je protected (pristup varijablama i metodama je dozvoljen iz svih klasa koji su u istom paketu kao i klasa u kojoj je metoda izrađena te svim naslijeđenim klasama) metoda koja ne vraća rezultat. Iznad metode nalazi se anotacija @Override koja govori razvojnom okruženju i kompajleru kako se metodom ispod anotacije želi izmijeniti način ponašanja iste metode iz bazne klase. Metoda kao svoju prvu liniju koda ima super.onCreate(savedInstanceState) koja nam služi za pozivanje metode onCreate iz bazne klase. Nakon toga, klasa nam prikazuje odgovarajući layout na pozadinu Android uređaja. Layout u Android Studiu deklarira vizualnu strukturu aplikacije. Layout je .xml dokument koji je izrađen od View (osnovna klasa za izradu korisničkog sučelja aplikacije) te ViewGroup (specifični tip View klase koji može sadržavati ostale podklase klase View) objekata. Nakon što je učitani layout, pokreće se novi Handler koji nam omogućuje slanje i procesiranje objekata tipa Message i Runnable koji su usko povezani sa MessageQueue – om dretve. MessageQueue predstavlja klasu koja sadrži listu poruka koje će biti poslane od strane Looper klase. Unutar Handlera instanciran je novi objekt tipa Intent koji predstavlja poruku što se želi učiniti u toku rada aplikacije. Intent prima dva argumenta: trenutni kontekst i klasu s kojom želimo nešto učiniti. Kao prvi argument stavlja se this, to jest, kontekst trenutne klase dok se kao drugi argument stavlja MainActivity – klasa koju želimo učitati. Nakon toga otvaramo MainActivity klasu koja je zapravo aktivnost, što znači da se otvara novi zaslon na pozadini uređaja. Na slikama se nalazi kod klase te pripadajućeg layouta.

```

package hr.foi.nj3m.androidmazesolver1;

import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;

public class loadingScreen extends AppCompatActivity {
    private static int SPLASH_TIME_OUT = 1000;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_loading_screen);

        //OtvaranjeGlavneForme
        new Handler().postDelayed(() - {
            Intent homeIntent = new Intent( packageContext: loadingScreen.this, MainActivity.class);
            startActivity(homeIntent);
            finish();
        }, SPLASH_TIME_OUT);
    }
}

```

Slika 7. Kod klase loadingScreen

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/ic_pocetniEkran"
    tools:context=".loadingScreen">
</android.support.constraint.ConstraintLayout>

```

Slika 8. activity_loading_screen.xml

4.4.2.MainActivity

Kao i `loadingScreen`, `MainActivity` je klasa koja proširuje `AppCompatActivity` klasu te pomoću anotacije `@Override` izmjenjuje ponašanje metode `onCreate`. Uz proširivanje neke bazne klase, `MainActivity` također implementira klasu `NavigationView`. Implementira njezinu metodu `OnNavigationItemSelectedListener` te ju izmjenjuje pomoću anotacije `@Override`. `MainActivity` klasa otvara `activity_main.xml` layout koji se razlikuje od layouta `loadingScreen` aktivnosti. `loadingScreen` layout sadrži `ConstraintLayout` koji omogućava prilagođavanje veličine i pozicije `ViewGroup` objekata, dok `activity_main.xml` layout sadrži `DrawerLayout` – kontejner koji omogućava korištenje izbornika koji se mogu „izvući“ iz vertikalnih krajeva zaslona android uređaja. Klasa `MainActivity` posjeduje jedno privatno svojstvo naziva `drawer` koji je tipa `DrawerLayout` – upravo kontejner koji je sadržan u `activity_main.xml` layoutu. Unutar `onCreate` metode nalazi se varijabla tipa `View` u koju se sprema `DecoivView` (sadrži standardni zaslon i dekoracije te sadržaj prikazan klijentu) najviše razine unutar aplikacije. Nakon toga se u cjelobrojnu varijablu sprema vrijednost zastavice `SYSTEM_UI_FLAG_FULLSCREEN` koja postavlja aktivnost preko cijelog zaslona. Pomoću metode `findViewById(id)` može se dobiti `View` bilo kojeg objekta u aplikaciji. U argument se upisuje `R.id.drawer_layout` te se tako prosljeđuje id `drawer_layout` layouta te se sprema u svojstvo `drawer`. Isti postupak se ponavlja i kod `NavigationView` objekta koji se nalazi u layoutu te koji služi za prikazivanje standardnog navigacijskog izbornika aplikacije. Poziva se metoda `setNavigationItemSelectedListener` iz `navigationView` objekta koja postavlja slušač koji će se okinuti kada je jedan odabrana jedna od stavki koje se nalaze u izborniku. Nakon što je postavljen slušač, otvara se fragment (jedan dio korisničkog sučelja aplikacije koji se može ukomponirati u aktivnost) `ConnectionTypeSelectionFragment` u `FrameLayout`u `FrameLayout` služi za „izdvajanje“ jednog dijela zaslona mobilnog uređaja. U taj dio uređaja mogu se staviti različiti objekti, no u ovom projektu su stavljeni samo fragmenti i to samo jedan u nekom vremenskom razdoblju. Uz metodu `onCreate`, aktivnost `MainActivity` sadrži još dvije metode: `public boolean onOptionsItemSelected (MenuItem)` i `onBackPressed()`. `onBackPressed` je izmijenjena metoda bazne klase te se u njoj provjerava otvorenost navigacijskog izbornika. To se može učiniti preko `drawer` objekta te njegove metode `isDrawerOpen` koja kao argument prima `View`. Na mjesto argumenta metode upisuje se `GravityCompat.START`, odnosno `START` konstanta klase `GravityCompat` koja gura objekt po x osi s lijeve strane po kontejneru. Ukoliko je izbornik izvučen, tada se poziva metoda `closeDrawer` koja zatvara izbornik. Ako izbornik nije izvučen, tada se poziva `onBackPressed` metoda iz bazne klase. Primarna djelatnost `onOptionsItemSelected` metode je učitavanje odgovarajućeg fragmenta na zaslon s obzirom na odabranu stavku u izborniku. U Izborniku se nalaze 3 stavke: `Gallery`, `Add` i `Connection type selection`.

```

drawer = findViewById(R.id.drawer_layout);
final NavigationView navigationView = findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);

if (savedInstanceState == null) {
    new Handler().post(() -> {
        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new GalleryFragment()).commit();
        navigationView.setCheckedItem(R.id.nav_gallery);
    });
}
new Handler().post(() -> {
    Fragment fragment = new ConnectionTypeSelectionFragment();
    FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
    transaction.replace(R.id.fragment_container, fragment);
    transaction.commitAllowingStateLoss();
});

```

Slika 9. Svojtvo drawer i korištenje slušača

```

@Override
public boolean onNavigationItemSelectedListener(@NonNull MenuItem menuItem) {
    switch (menuItem.getItemId()) {
        case R.id.nav_gallery:
            getSupportFragmentManager()
                .beginTransaction()
                .setCustomAnimations(R.anim.enter_left_to_right, R.anim.exit_left_to_right, R.anim.enter_right_to_left, R.anim.exit_right_to_left)
                .replace(R.id.fragment_container, new GalleryFragment())
                .commit();
            break;
        case R.id.nav_add_photo:
            getSupportFragmentManager()
                .beginTransaction()
                .setCustomAnimations(R.anim.enter_left_to_right, R.anim.exit_left_to_right, R.anim.enter_right_to_left, R.anim.exit_right_to_left)
                .replace(R.id.fragment_container, new AddFragment())
                .commit();
            break;
        case R.id.nav_info:
            Toast.makeText(context, this, "Info", Toast.LENGTH_SHORT).show();
            break;
        case R.id.nav_sensor_selection:
            getSupportFragmentManager().beginTransaction()
                .setCustomAnimations(R.anim.enter_left_to_right, R.anim.exit_left_to_right, R.anim.enter_right_to_left, R.anim.exit_right_to_left)
                .replace(R.id.fragment_container, new SensorSelectionFragment())
                .commitAllowingStateLoss();
            break;
    }
    new Handler().post(() -> { drawer.closeDrawer(GravityCompat.START); });
    return true;
}

```

Slika 10. onNavigationItemSelectedListener metoda

```

@Override
public void onBackPressed() {
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}

```

Slika 11. onBackPressed metoda

4.4.3. SensorSelectionFragment

SensorSelectionFragment je klasa koja proširuje Fragment klasu. Primarna djelatnost klase je odabir senzora koje korisnik želi koristiti prilikom pronalaska izlaza iz labirinta. Klasa sadrži 5 privatnih svojstava: 3 objekta IUltraSonic koji predstavljaju 3 senzora na robotu, listu IUltraSonic objekata koja će biti korištena prilikom slanja informacija o odabranim sensorima te objekta Button naziva mButton.

Klasa se sastoji od dvije javne metode: onCreateView te onStart. Na slici može se vidjeti onCreateView metoda koja vraća objekt tipa View. Kao što se može vidjeti, iznad metode nalazi se anotacija @Override, što znači da se ista metoda nalazi i u baznoj klasi Fragment. Metoda prima 3 argumenta:

- LayoutInflater – objekt koji instancira layout datoteku u odgovarajući View objekt
- ViewGroup – View roditelj, odnosno View u koji se postavlja fragment
- Bundle – ako nije null, fragment se rekonstruira pomoću posljednjeg spremljenog stanja fragmenta

Metoda vraća View dobiven preko LayoutInflater objekta i njegove metode inflate koja prima 3 argumenta:

- XmlPullParser – XML layout koji se želi inflateati
- ViewGroup – Roditelj u koji će se layout inflateati
- Boolean – ako je istinit, tada se layout datoteka iz prvog argumenta inflate i postavlja u ViewGroup objekt iz drugog argumenta. Ako je lažan, tada se layout datoteka iz prvog argumenta inflatea i vraća kao View.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_sensor_selection, container, attachToRoot false);
}
```

Slika 12. Metoda onCreateView u klasi SensorSelectionFragment

Druga metoda u klasi SensorSelectionFragment je onStart(). Kao i metoda onCreateView, i metoda onStart sadrži iznad sebe anotaciju @Override što znači da je njezina implementacija i u baznoj klasi Fragment. Na početku metode poziva se onStart metoda iz bazne klase te se u varijablu mButton koja je tipa Button dohvaća gumb Nastavi iz layouta fragment_sensor_selection čiji je ID btnNastavi. Nakon što je dohvaćen gumb, postavlja se slušaš koji će okinuti metodu onClick kad korisnik stisne na gumb. U Metodi onClick koja prima parametar View, dohvaćaju se checkboxovi. Svaki checkbox odgovoran je za jedan senzor robota. Ukoliko korisnik ne odabere niti jedan senzor, tada se korisniku prikazuje poruke „Niste

odabrali senzore!“. Ukoliko je korisnik odabrao senzor, tada se instancira odgovarajući objekt UltrasonicSensor klase koja implementira sučelje IUltraSonic. Nakon što je objekt instanciran, dodaje se u listu objekata te se proslijeđuje klasi.

U aplikaciji, kako bismo poboljšali SOLID dizajn i rastavili funkcionalnosti i odgovornosti klasa, kreirali smo uz druga sučelja i nekoliko sučelja koja se odnose na senzore.

```
public interface ISensors {  
  
    String getFullValue();  
  
    void setCurrentValue(String value);  
}
```

Prvo osnovno sučelje je ISensors, ovo sučelje je osnovno sučelje kojeg implementira svaki senzor, i čitač crte i ultrazvučni senzor. Implementiranjem ovog senzora, svaka klasa mora implementirati i metode koje su po ugovoru navedene u sučelju, a to su „getFullValue“ i „setCurrentValue“, to je napravljeno iz razloga jer svaki senzor mora imati neku vrijednost koja se može dohvatiti i zapisati na senzor.

Sljedeće sučelje, koje je malo opširnije i proširuje ISensors je IUltraSonic.

```
public interface IUltraSonic extends ISensors {  
  
    Sides getSensorSide();  
  
    double getNumericValue();  
  
    boolean seesObstacle();  
}
```

Ovo sučelje je kreirano specifično kako bi ga klasa UltrasonicSensor mogla implementirati. Kako ovo sučelje proširuje ISensors, klasa UltrasonicSensor po ugovoru mora imati klase koje ima ISensors sučelje, kao i klase koje ima IUltraSonic. Metode IUltraSonic sučelja su: „getSensorSide“ jer se svaki ultrazvučni senzor može nalaziti na nekoj strani mBota, „getNumericValue“ koja točno mora vratiti *double* kao rezultat jer je to stvarni tip podataka koji se stvara u arduino programu za mBota. Zadnja metoda je „seesObstacle“ koja je *true* ukoliko se ispred promatranog senzora nalazi prepreka, a inače je *false*.

Zadnje sučelje što se tiče senzora je ILineFollower koje proširuje isto kao i prethodno sučelje, ISensors.

```
public interface ILineFollower extends ISensors {  
  
    boolean isRightSideOut();  
    boolean isLeftSideOut();  
    boolean bothAreOut();  
}
```

Vrijedi isto kao i za prošlo sučelje, klasa koja implementira ovo sučelje, a to je LineFollower, mora implementirati metode sučelja ISensors i metode sučelja ILineFollower. ILineFollower sučelje ima metode „isRightSideOut“, „isLeftSideOut“ i „bothAreOut“. Iz imena metoda, lako je zaključiti što bi one trebale raditi, dakle prva vraća istinu ukoliko je desni senzor izvan linije

koju čitač čita, a lijevi je na liniji. Za drugu metodu vrijedi obrnuto, dok metoda „bothAreOut“ vraća istinu ukoliko nijedan od senzora ne vidi liniju. Sukladno tome, lako je provjeriti jesu li oba na liniju.

4.4.4.ConnectionTypeSelectionFragment

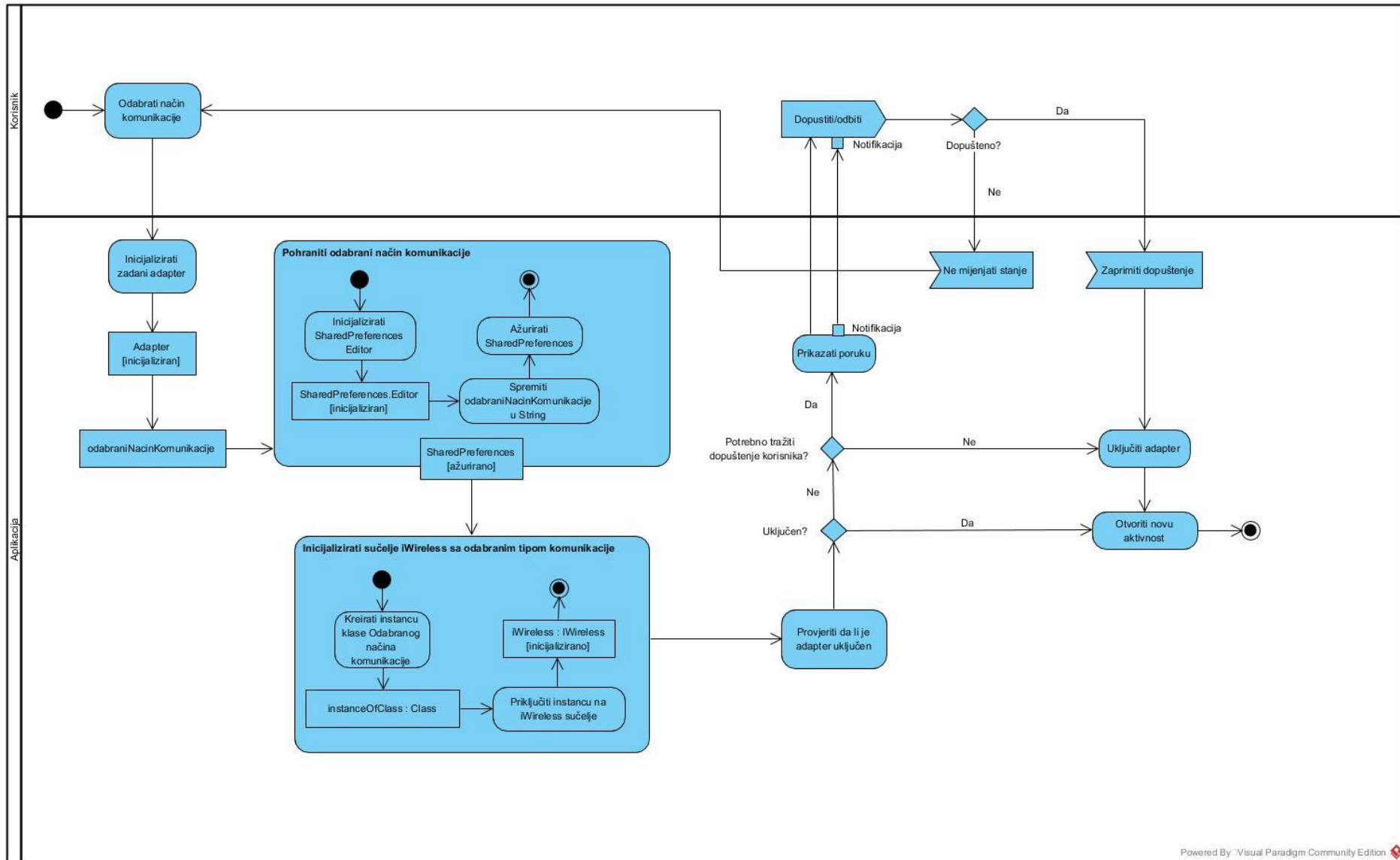
Prije nego krenemo s objašnjavanjem koda ovog fragmenta, prikazat ćemo i ukratko objasniti dijagram aktivnosti kako bismo dobili dojam o tome što želimo postići u ovom dijelu aplikacije. Slika dijagrama nalazi se na sljedećoj stranici.

Kada se korisniku otvori aplikacija, prvo mu se prikažu dva gumba; jedan za Bluetooth, a drugi za WiFi način komunikacije. Korisnik pritiskom na jednog od ta dva gumba pokreće slušatelj događaja koji prvo inicijalizira adapter za odabrani način komunikacije. Varijabla naziva *odabraniNacinKomunikacije* poprima vrijednost „bluetooth“ ili „wifi“, ovisno o tome koji gumb je pritisnut. Vrijednost te varijable sprema se u zajednički prostor kojemu se može pristupiti iz bilo kojeg dijela aplikacije, a to će nam trebati u daljnjem radu aplikacije u različitim fragmentima.

Ovisno o odabranom načinu komunikacije, potrebno je kreirati instancu klase koja sadrži metode za rad s uređajima (traženje, povezivanje). Ta instanca klase priključuje se na sučelje *IWireless*, a to znači da ćemo sada preko tog sučelja pozivati metode klase čija je to instanca.

No, sve ovo bitno je za rad same aplikacije, a korisnik to ni ne vidi te vjerojatno ni ne zna što se u pozadini događa. Jedino što korisnik vidi je da je pritiskom na, recimo, tipku sa znakom Bluetooth-a uključio Bluetooth na svom mobilnom uređaju (ako je prethodno bio isključen). Prije nego što se Bluetooth uključio, potrebna je potvrda korisnika da želi uključiti Bluetooth.

Nakon što je adapter za odabrani način komunikacije uključen, otvorit će se novi fragment naziva *ListOfDevicesFragment*.



Slika 13: Dijagram aktivnosti fragmenta *ConnectionTypeSelectionFragment*

Već smo rekli da se nakon pokretanja aplikacije otvara zaslon s dvije tipke:

- Bluetooth i
- WiFi

Tipke dodajemo na način da dohvatimo referencu na tipku koju smo kreirali u dizajneru preko njezinog ID-a:

```
final ImageButton tipkaBluetooth = getView().findViewById(R.id.btnBluetooth);  
final ImageButton tipkaWifi = getView().findViewById(R.id.btnWifi);
```

Slika 14. Dohvaćanje tipki preko reference u dizajneru

```
tipkaBluetooth.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        BluetoothAdapter bluetoothAdapter = BluetoothAdapter.  
            getDefaultAdapter();  
        prefEditor.putString(„TypeOfConnection“, „bluetooth“);  
        prefEditor.commit();  
        iWireless = WirelessController.createInstance(getActivity(),  
            sharedPreferences.getString(„TypeOfConnection“, „DEFAULT“));  
        if (bluetoothAdapter.isEnabled())  
            openListOfDevices();  
        else  
            iWireless.enableDisable(broadcastReceiver);  
    }  
});  
  
tipkaWifi.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        WifiManager wifiManager = (WifiManager) getActivity().  
            getApplicationContext().getSystemService(Context.WIFI_SERVICE);  
        prefEditor.putString(„TypeOfConnection“, „wifi“);  
        prefEditor.commit();  
        iWireless = WirelessController.createInstance(getActivity(),  
            sharedPreferences.getString(„TypeOfConnection“, „DEFAULT“));  
        if (wifiManager.isWifiEnabled())  
            openListOfDevices();  
        else  
            iWireless.enableDisable(broadcastReceiver);  
    }  
});
```

Slika 15. Slušatelji koji su postavljeni na tipke

Na svaku od tih tipki potrebno je postaviti slušatelj događaja pritiska te tipke kako bi se izvele željene radnje ako je neka od tih dviju tipki pritisnuta.

Odabirom jednih od ovih dviju tipki odabiremo način komunikacije. Odabrani način komunikacije želimo pohraniti na neko mjesto na kojem će to biti dostupno na bilo kojem mjestu u aplikaciji. Za to će nam poslužiti sučelje zvano *SharedPreferences*. Kako bismo odabrani način komunikacije zapisali u *SharedPreferences*, potrebno je dohvatiti željenu datoteku u koju zapisujemo vlastite postavke u obliku ključ-vrijednost. Slijedeća linija koda dohvatiti će datoteku naziva „MazeSolver1“ u obliku objekta, a ako ta datoteka ne postoji, kreirati će se nova datoteka istog tog naziva:

```
SharedPreferences sharedPreferences = getContext().getSharedPreferences(
    „MazeSolver1“, Context.MODE_PRIVATE);
```

Slika 16. Dohvaćanje datoteke u obliku objekta

Sada kad imamo tu datoteku, želimo u nju kao postavku pohraniti odabrani način komunikacije kako bismo to mogli iskoristiti kasnije u aplikaciji jer će nam biti potrebno. Kako bismo mogli zapisivati u tu datoteku, također nam je potrebno jedno sučelje zvano Editor. Ono se poziva na sljedeći način:

```
SharedPreferences.Editor prefEditor = sharedPreferences.edit();
```

Slika 17. Pozivanje sučelja Editor

Ovom linijom koda smo nad objektom naše datoteke kreirali uređivač pomoću kojega možemo uređivati datoteku. Pritiskom na neku od tipki pozvat ćemo metodu *putString()* objekta *prefEditor*, a kao argumente ćemo joj proslijediti naziv vrijednosti koju želimo spremiti, kao ključ, i samu vrijednost koju želimo spremiti pod tim ključem. Npr. ako smo pritisnuli tipku Bluetooth, tada smo odabrali Bluetooth kao način komunikacije, pa možemo spremiti vrijednost „*bluetooth*“, a kao naziv te vrijednosti stavit ćemo „*TypeOfConnection*“:

```
prefEditor.putString(„TypeOfConnection“, „bluetooth“);
```

Slika 18. Spremanje željenog niza u datoteku

Trenutno se ovaj niz znakova nalazi u uređivaču, a kako bismo to pohranili u datoteku „*MazeSolver1*“, potrebno je nad objektom *prefEditor* pozvati metodu *commit()* koja će promjene koje smo napravili u editoru spremiti u našu datoteku:

```
prefEditor.commit();
```

Dakle, sada smo u datoteku naziva „*MazeSolver1*“ spremili jedan par ključ-vrijednost u sljedećem obliku:

MazeSolver1	
Ključ	TypeOfConnection
Vrijednost	bluetooth

Tablica 1. Oblik para ključ-vrijednost

Odabrani način komunikacije koristimo prvenstveno kada dohvaćamo instance klasa koje sadrže metode za povezivanje s uređajem ili za komunikaciju s uređajem. Bitno nam je koji smo način komunikacije odabrali jer se metode za povezivanje i komunikaciju razlikuju pa se, u ovisnosti o načinu komunikacije stvaraju i dohvaćaju odgovarajuće instance odgovarajućih klasa u različitim dijelovima aplikacije.

Klase za povezivanje i komunikaciju preko Bluetootha i preko WiFi-ja nalaze se u različitim modulima. Moduli su dijelovi aplikacije zvani „kontejneri“ koji sadrže izvorni kod i resurse potrebne za rad s određenim dijelovima aplikacije. Ova aplikacija podijeljena je na module kako bi se olakšalo snalaženje kroz aplikaciju, a ono što je još važnije, olakšala nadogradnja i izmjene koda aplikacije. Iz glavnog (*app*) modula, klasama različitih modula

pristupamo koristeći sučelja. Sučelje sadrži tzv. potpise metoda koje je potrebno implementirati u klasama koje implementiraju to sučelje. Možemo reći da sučelje predstavlja vrata kroz koja se pristupa implementiranim metodama u odgovarajućim klasama koje implementiraju to sučelje. Implementirati sučelje znači, u klasi/klasama koja/e implementira(ju) sučelje, dodijeliti metodama, čiji je potpis u sučelju, kod koji će se izvršavati prilikom poziva tih metoda. Ovo je važno kako bi se ostvarila modularnost aplikacije, odnosno ako bismo recimo željeli u budućnosti dodati novi način komunikacije, tada bismo samo dodali novi modul i u njemu potrebne klase za povezivanje i komunikaciju te u odgovarajućim klasama implementirali odgovarajuća sučelja tako da im dodijelimo kod koji će se izvršavati pozivom tih metoda. Potpisi metoda u sučeljima će ostati isti i funkcije će se pozivati na identičan način, samo će se izvoditi druge radnje, ovisno o odabranom načinu komunikacije i biti će im možda potrebno proslijediti različite argumente.

Prvo sučelje koje koristimo je *IWireless*. Potpisi metoda u tom sučelju izgledaju ovako:

```
public interface IWireless {  
    void enableDisable(BroadcastReceiver broadcastReceiver);  
    void discover(BroadcastReceiver broadcastReceiver);  
}
```

Slika 19. *IWireless* sučelje

Ovo sučelje sadrži opise dviju metoda:

- *enableDisable()* koja služi za uključivanje i isključivanje adaptera odabranog načina komunikacije
- *discover()* koja služi za traženje dostupnih uređaja za komunikaciju odabranim načinom komunikacije

Dakle, potpisi ovih dviju metoda trebali bi ostati ovakvi za bilo koji način komunikacije koji imamo i koji planiramo eventualno dodati, no implementacija tih metoda u klasama odabranog načina komunikacije se razlikuju. Pogledajmo te dvije metode za Bluetooth i za WiFi način komunikacije:

1) Bluetooth:

```
@Override
public void enableDisable(BroadcastReceiver broadcastReceiver){
    if (!bluetoothAdapter.isEnabled()){
        Intent enableBTIntent = new Intent(BluetoothAdapter.
            ACTION_REQUEST_ENABLE);
        context.startActivity(enableBTIntent);
        IntentFilter BTIntent = new IntentFilter(BluetoothAdapter.
            ACTION_STATE_CHANGED);
        context.registerReceiver(broadcastReceiver, BTIntent);
    }
}
```

Slika 20. Metoda *enableDisable* – Bluetooth

Ova metoda još jednom obavlja provjeru uključenosti Bluetootha. Ako nije uključen, pokreće se aplikacijski servis za uključenje Bluetootha (*ACTION_REQUEST_ENABLE*). Također se dodaje *IntentFilter* kojeg će prihvaćati *BroadcastReceiver* i reagirati definiranom radnjom prilikom promjene trenutnog stanja konekcije (*ACTION_STATE_CHANGED*) koje će se promijeniti nakon što navedeni servis uključivanjem Bluetootha promijeni njegovo stanje iz isključenog u uključeno.

```
@Override
public void discover(BroadcastReceiver broadcastReceiver){
    if (bluetoothAdapter.isDiscovering()){
        checkBTPermissions();
        bluetoothAdapter.cancelDiscovery();
        bluetoothAdapter.startDiscovery();
        Toast.makeText(context, „Tražim uređaje...“, Toast.LENGTH_LONG).
            show();
        IntentFilter discoverDevicesIntent = new IntentFilter(
            BluetoothDevice.ACTION_FOUND);
        context.registerReceiver(broadcastReceiver, discoverDevicesIntent);
    }
    if (!bluetoothAdapter.isDiscovering()){
        checkBTPermissions();
        bluetoothAdapter.startDiscovery();
        Toast.makeText(context, „Tražim uređaje...“, Toast.LENGTH_LONG).
            show();
        IntentFilter discoverDevicesIntent = new IntentFilter(
            BluetoothDevice.ACTION_FOUND);
        context.registerReceiver(broadcastReceiver, discoverDevicesIntent);
    }
}
```

Slika 21. Metoda *discover* – Bluetooth

Metoda *discover* provjerava verziju androida na uređaju na kojem se pokreće aplikacija te, shodno tome, provjerava je li aplikaciji dozvoljen pristup lokaciji uređaja jer je lokacija uređaja potrebna kako bi se s obzirom na lokaciju našeg uređaja pronašli drugi uređaji koji su u blizini. Metoda je identična i za provjeru za WiFi. Metoda *checkBTPermissions* izgleda ovako:

```
private void checkBTPermissions(){
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP){
        int permissionCheck = ContextCompat.checkSelfPermission(context,
            „Manifest.permission.ACCESS_FINE_LOCATION“);
        permissionCheck += ContextCompat.checkSelfPermission(context,
            „Manifest.permission.ACCESS_COARSE_LOCATION“);
        if (permissionCheck != 0)
```

```
        ActivityCompat.requestPermissions((Activity) context,  
        new String[]{Manifest.permission.ACCESS_FINE_LOCATION,  
        Manifest.permission.ACCESS_COARSE_LOCATION}, 1001);  
    }  
}
```

Slika 22. Metoda *checkBTPermissions*

Ako je korisnik odobrio pristup lokaciji, prekida se traženje uređaja ako je ono trenutno pokrenuto i pokreće se iznova. Korisniku se ispisuje tekst „Tražim uređaje...“ i dodajemo novi *IntentFilter* kojeg ćemo registrirati na *BroadcastReceiver* ako se dogodio događaj pronalaska novog uređaja.

2) WiFi

Kod WiFi-ja su nam potrebni objekti *WifiManager*, *WifiP2PManager* i *WifiP2pChannel*. Njih inicijaliziramo na sljedeći način:

```
WifiManager wifiManager = (WifiManager) context.getSystemService(Context.  
    WIFI_SERVICE);  
WifiP2pManager wifiP2pManager = (WifiP2pManager) context.getSystemService(Context.  
    WIFI_P2P_SERVICE);  
WifiP2pManager.Channel wifiP2pChannel = wifiP2pManager.initialize(context,  
    context.getMainLooper(), null);
```

Slika 23. Inicijalizacija objekata *WifiManager*, *WifiP2pManager* i *WifiP2pChannel*

WifiManager je upravljač za rad s WiFi konekcijom i njenim metodama, *WifiP2pManager* sadrži dodatne metode za komunikaciju s uređajima WiFi *peer-to-peer* načinom komunikacije, a *WifiP2pManager.Channel* je poveznica aplikacije i WiFi P2P okosnice.

```
@Override  
public void enableDisable(BroadcastReceiver broadcastReceiver) {  
    if (!wifiManager.isWifiEnabled()) {  
        wifiManager.setWifiEnabled(true);  
        IntentFilter wifiIntent = new IntentFilter(WifiManager.  
            WIFI_STATE_CHANGED_ACTION);  
        context.registerReceiver(broadcastReceiver, wifiIntent);  
    }  
}
```

Slika 24. Metoda *enableDisable* - Wifi

Ako WiFi trenutno nije uključen, ova metoda poziva funkciju *setWifiEnabled* koja kao argument prima *boolean* vrijednost *true* ili *false* i ovisno o toj vrijednost uključuje ili isključuje WiFi, respektivno. Zato smo ovdje prosljedili vrijednost *true*. Također stvaramo *IntentFilter* i registriramo na njega *BroadcastReceiver* koji će od tog *IntentFilter*-a dobiti obavijest kada se dogodi događaj promjene stanja WiFi-ja.

```

@Override
public void discover(final BroadcastReceiver broadcastReceiver){
    checkWifiPermissions();
    wifiP2pManager.discoverPeers(wifiP2pChannel, new WifiP2pManager.
        ActionListener(){
            @Override
            public void onSuccess(){
                IntentFilter intentFilter = new IntentFilter(WifiP2pManager.
                    WIFI_P2P_CONNECTION_CHANGED_ACTION);
                context.registerReceiver(broadcastReceiver, intentFilter);
            }
            @Override
            public void onFailure(int reason){
                //ako želimo, ovdje dodajemo radnju ako pretraživanje ne uspije
            }
        });
}

```

Slika 25. Metoda *discover* - Wifi

Kod ove metode važan nam je dio *OnSuccess()* gdje registriramo *BroadcastReceiver* na događaj *WIFI_P2P_CONNECTION_CHANGED_ACTION*, odnosno događaj promjene stanja WiFi-ja.

Objasnimo sada *BroadcastReceiver*. *BroadcastReceiver* je klasa koja prima obavijesti koje se šalju iz aplikacije kada se dogodi neki događaj. Mi možemo registrirati *BroadcastReceiver* da prima specifične obavijesti iz nekog dijela aplikacije kada nam je to potrebno i za to definirati kako će *BroadcastReceiver* odgovoriti na te obavijesti. Poruka obavijesti je „umotana“ u objekt zvan *Intent* čiji niz znakova akcije identificira događaj koji se dogodio. *IntentFilter* je specifični događaj na kojeg mi registriramo *BroadcastReceiver* i na koji želimo da on reagira nekom radnjom koju definiramo u *BroadcastReceiveru*.

Ovisno o tome koji smo način komunikacije odabrali, potrebno je dohvatiti zadani lokalni Bluetooth adapter uređaja u slučaju Bluetooth načina komunikacije ili upravljač za rad s WiFi konekcijom i njenim metodama u slučaju WiFi načina komunikacije.

a) Bluetooth način komunikacije:

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

Slika 26. Dohvaćanje lokalnog Bluetooth adaptera

b) WiFi način komunikacije:

```
WifiManager wifiManager = (WifiManager) getActivity().getApplicationContext.
    getSystemService(Context.WIFI_SERVICE);
```

Slika 27. Dohvaćanje Wifi managera

Nakon toga, potrebno je provjeriti da li je Bluetooth, odnosno WiFi, uključen na našem mobilnom uređaju. Za oboje se provjera obavlja na sličan način:

a) Bluetooth:

```
bluetoothAdapter.isEnabled()
```

Slika 28. Provjera uključenosti Bluetootha

b) WiFi:

```
wifiManager.isWifiEnabled()
```

Slika 29. Provjera uključenosti Wifi - a

Uzimajući u obzir uključenost, odnosno isključenost Bluetootha, odnosno, WiFi-ja, možemo izvesti neke radnje u našoj aplikaciji. Recimo da želimo da se nakon odabira i uključivanja našeg odabranog oblika komunikacije otvori zaslon na kojem možemo vidjeti dostupne uređaje sa kojima se želimo upariti, među kojima je i naš *mBot*, možemo iskoristiti ove gore navedene provjere kao uvjet pod kojim će se ili otvoriti novi zaslon ili će se u aplikaciji izvesti nešto drugo. Mi želimo da se, ukoliko je naš odabrani način komunikacije već uključen, dakle npr. ako je Bluetooth već uključen, otvori nova aktivnost koja će sadržavati prikaz svih dostupnih uređaja s kojima se možemo povezati koristeći Bluetooth oblik komunikacije, a ako je isključen, tada ćemo ga uključiti i tek će se nakon uključivanja otvoriti ta ista nova aktivnost. Prvo ćemo pokazati kreiranu funkciju naziva *openListOfDevices* koja otvara novi (sljedeći) fragment naziva „*ListOfDevicesFragment*“:

```
private void openListOfDevices() {  
    Fragment fragment = new ListOfDevicesFragment();  
    FragmentTransaction transaction = getActivity().getSupportFragmentManager().  
        beginTransaction();  
    transaction.replace(R.id.fragment_container, fragment);  
    transaction.addToBackStack(null);  
    transaction.commitAllowingStateLoss();  
}
```

Slika 30. Metoda openListOfDevices

Ako odabrani način komunikacije nije uključen, pozvat ćemo metodu *enableDisable*, čiji kod smo ranije naveli, koja će ga uključiti.

BroadcastReceiver će, nakon što primi obavijest da je promijenjeno stanje Bluetootha, provjeriti u kojem je Bluetooth adapter sada stanju. Stanje će dohvatiti od *Intenta* koji ga je postavio, točnije od našeg *BTIntent*-a. Kako bismo najjednostavnije provjeravali stanja i na temelju tih stanja izvršili zadane radnje u aplikaciji, koristit ćemo kontrolnu strukturu *switch-case*. Vrijednost koju ćemo ispitivati je, naravno, stanje (*state*), a prolazit ćemo kroz stanja koja nas zanimaju i, ovisno o tome koje stanje smo dohvatili, izvršit će se naredbe zadane za to stanje. Dakle, to bi izgledalo ovako:

```
private final BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String action = intent.getAction();  
        if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {  
            final int state = intent.getIntExtra(BluetoothAdapter.  
                EXTRA_STATE, BluetoothAdapter.ERROR);  
            switch (state) {  
                //ako je Bluetooth isključen  
                case BluetoothAdapter.STATE_OFF:  
                    //izvrši zadane naredbe  
                    break;  
  
                //ako je Bluetooth uključen  
                case BluetoothAdapter.STATE_ON:  
                    //izvrši zadane naredbe  
                    break;  
            }  
        }  
    }  
}
```

Slika 31. BroadcastReceiver

Nama su od interesa stanje kada je Bluetooth isključen (*STATE_OFF*) i kada je Bluetooth uključen (*STATE_ON*). Ako je Bluetooth uključen, rekli smo da želimo otvoriti novi (sljedeći) fragment, dakle, naredbe koje će se izvršiti su one u funkciji *openListOfDevices*, dakle case *BluetoothAdapter.STATE_ON* sada izgleda ovako:

```
...
case BluetoothAdapter.STATE_ON:
    Toast.makeText(context, „Bluetooth uključen“, Toast.LENGTH_LONG).show();
    openListOfDevices();
    break;
...
```

Slika 32. Slučaj kad je Bluetooth uključen

Dakle, korisniku će se na ekranu prikazati poruka „Bluetooth uključen“ i pozvat će se funkcija *openListOfDevices* koja će pokrenuti sljedeći fragment „*ListOfDevicesFragment*“. Ako je, pak, Bluetooth isključen, želimo ga uključiti, a to smo već izveli ranije kada smo prilikom pritiska na gumb provjeravali uključenost Bluetootha. Dakle, imali smo:

```
if (bluetoothAdapter.isEnabled())
    openListOfDevices();
else
    iWireless.enableDisable(broadcastReceiver);
```

Slika 33. Provjera uključenosti Bluetootha

Dakle, ako je Bluetooth uključen, poziva se metoda *openListOfDevices* koja otvara sljedeći fragment „*ListOfDevicesFragment*“, a ukoliko nije, poziva se metoda *enableDisable* sučelja *iWireless* koja uključuje Bluetooth i, ukoliko se Bluetooth uključio, prikazuje putem *BroadcastReceiver*a korisniku poruku „*Bluetooth uključen*“ i tek onda pokreće sljedeći fragment „*ListOfDevicesFragment*“.

Ukoliko se Bluetooth isključio, posao *BroadcastReceiver*a je, prilikom zaprimanja obavijesti da je Bluetooth isključen, prikazati korisniku na ekranu poruku „*Bluetooth isključen*“:

```
...
case BluetoothAdapter.STATE_OFF:
    Toast.makeText(context, „Bluetooth isključen“, Toast.LENGTH_LONG).show();
    break;
...
```

Slika 34. Slučaj kad je Bluetooth isključen

4.4.5.ListOfDevicesFragment

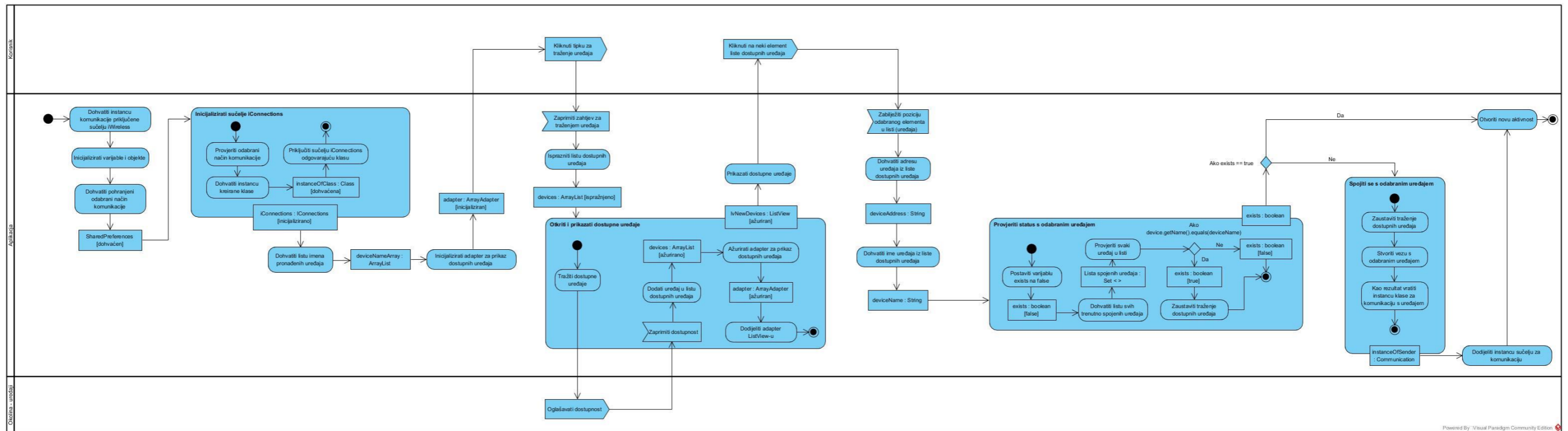
Prije nego krenemo s opisivanjem izvornog koda ovog fragmenta, također ćemo prikazati i ukratko objasniti dijagram aktivnosti, kao i kod prethodnog fragmenta, kako bismo dobili dojam o tome kako ovaj fragment radi i što je njegov zadatak. Slika dijagrama aktivnosti nalazi se na sljedećoj stranici.

Ovaj fragment prikazat će popis uređaja dostupnih za uspostavu veze za odabrani način komunikacije. Ovisno o odabranom načinu komunikacije, potrebno je inicijalizirati novo sučelje *IConnections* koje se implementira u istoj klasi kao i *IWireless*, ali ovo sučelje sadrži metode koje će nam biti potrebne za pronalaženje uređaja i njihovo prikazivanje u obliku liste.

Podaci će biti prikazani u pogledu naziva *ListView*, a njega će podacima popunjavati adapter. Adapter dohvaća popis uređaja iz liste u koju će biti spremljeni svaki otkriveni uređaj, a to su ti podaci kojima će se popuniti *ListView*.

Korisnik može odabrati jedan od tih uređaja te će se pokrenuti proces uparivanja s tim uređajem. Za uparivanje je potrebna adresa tog uređaja jer se ona proslijeđuje metodi za uparivanje te se na temelju te adrese stvori veza s tim uređajem. Također, postoji mogućnost da je korisnik već od prije uparen s tim uređajem, pa je to potrebno provjeriti te se u tom slučaju neće ponovno stvarati spona s tim uređajem.

Ako je povezivanje s uređajem proteklo uspješno, otvara se novi fragment naziva *ConnectedDialogFragment*.



Slika 35: Dijagram aktivnosti fragmenta *ListOfDevicesFragment*

Prilikom pokretanja fragmenta *ListOfDevicesFragment*, potrebno je dohvatiti neke objekte koje smo stvorili u prethodnoj aktivnosti, a neke je potrebno stvoriti u ovoj aktivnosti. Dohvatiti je potrebno instancu sučelja *IWireless* jer ćemo i u ovoj aktivnosti koristiti neke metode tog sučelja i instancu *SharedPreferences* jer ćemo u ovisnosti odabranog načina komunikacije izvoditi određene radnje.

Kako bismo dohvatili instancu sučelja *IWireless*, koristimo metodu *WirelessControllera*: *getInstanceOfIWireless*.

```
IWireless iWireless = WirelessController.getInstanceOfIWireless();
```

Slika 36. Dohvaćanja instance sučelja *IWireless*

Ova metoda, jednostavno, samo vraća instancu tipa *IWireless*, a ta instanca sadrži instancu odabranog načina komunikacije koju smo ranije kreirali. Također, rekli smo da treba dohvatiti instancu sučelja *SharedPreferences*, a to se radi kao i u prethodnoj aktivnosti:

```
SharedPreferences sharedPreferences = getContext().getSharedPreferences("MazeSolver1", Context.MODE_PRIVATE);
```

Slika 37. Dohvaćanje instance sučelja *SharedPreferences*

Ranije smo naveli da će se u ovom fragmentu prikazati lista dostupnih uređaja s kojima se možemo povezati odabranim načinom komunikacije. Za prikaz tih uređaja koristit ćemo *ListView*. *ListView* je ustvari popis elemenata koje smo spremili u neku listu. Da bismo prikazali uređaje u *ListView*-u, potrebno je stvoriti listu i popuniti ju pronađenim uređajima.

Prije nego bilo što dalje radimo, kreirat ćemo adapter koji će preko sučelja iz klase odabranog načina komunikacije dohvatiti polje koje sadrži imena uređaja koje smo pronašli. U nastavku ćemo vidjeti kako se popunjava to polje. Adapter kreiramo na sljedeći način:

```
ArrayAdapter<String> adapter = new ArrayAdapter<>(getContext(), android.R.layout.simple_list_item_1, iConnections.getDeviceArray());
```

Slika 38. Kreiranje adaptera za popunjavanje *ListView*-a

Drugi argument koji se prosjeđuje prilikom kreiranja adaptera je oblik elemenata koje prikazujemo u *ListView*. *simple_list_item_1* je XML prikaz koji vizualizira primljeno polje niza znakova, odnosno drugim riječima, u listi će biti prikazani uređaji, jednostavno, kao imena tih uređaja u čitljivom tekstualnom obliku.

Kreirat ćemo *ListView* i dodijeliti mu stvoreni adapter koji služi tome da listu popunjava podacima; jednostavnije rečeno, prilikom pronalaska nekog uređaja, on će uzeti elemente polja koje smo mu dodijelili i popuniti *ListView* tim elementima, odnosno imenima uređaja koje smo pronašli. Za popunjavanje liste pronađenim uređajima, ovdje ćemo također koristiti *BroadcastReceiver* koji će primiti obavijesti o pronađenim uređajima (ako su pronađeni) i dodavati ih u listu.

Budući da u ovoj aplikaciji koristimo dva moguća načina komunikacije, biti će potrebne dvije različite liste: jedna koja će biti kreirana ako koristimo Bluetooth način komunikacije i koja će sadržavati elemente tipa *BluetoothDevice*, a druga lista je ona koja će biti kreirana samo ako koristimo WiFi način komunikacije i koja će sadržavati elemente tipa *WiFiP2PDevice*. WiFi P2P uređaj je onaj uređaj koji podržava povezivanje preko WiFi *peer-to-peer* veze. To je oblik WiFi kriptirane veze koji ne zahtijeva povezivanje na stvarnu mrežu ili „vruću točku“ (eng. *Hotspot*), već su uređaji spojeni direktnom vezom.

Dakle, *BroadcastReceiver* će ovdje, slično kao i u prethodnom fragmentu, dohvaćati obavijest o nekim događajima koji su nam ovdje potrebni, odnosno od interesa. Rekli smo da želimo pronađene uređaje spremite u listu i prikazati ih, dakle *BroadcastReceiver* bi trebao primiti obavijest ako je pronađen jedan ili više uređaja i dodavati/dodati te uređaje u listu. Za Bluetooth se događaj o pronađenom uređaju registrira uspoređivanjem dohvaćenog događaja sa predefiniranim događajem *BluetoothDevice.ACTION_FOUND*. To znači da je u blizini pronađen neki Bluetooth uređaj na kojemu je uključena vidljivost, odnosno koji oglašava da je dostupan za uparivanje preko Bluetootha. Taj uređaj želimo dohvatiti i dodati ga u našu listu. U nastavku slijedi kod koji to i izvodi:

```
private BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //dohvaćamo primljeni događaj
        final String action = intent.getAction();
        //ako je događaj pronađeni uređaj
        if (action.equals(BluetoothDevice.ACTION_FOUND)) {
            //kreiramo listu za elemente tipa BluetoothDevice
            ArrayList<BluetoothDevice> bluetoothDevices = new
                ArrayList<>();
            //dohvaćamo uređaj preko Intenta
            BluetoothDevice device = intent.getParcelableExtra(
                BluetoothDevice.EXTRA_DEVICE);
            //dodajemo uređaj u listu
            bluetoothDevices.add(device);
            /*uređaje također dodajemo, preko sučelja IConnections u listu
            kreiranu u klasi Bluetooth*/
            iConnections.addDevices(bluetoothDevices);
            //postavljamo adapter ListViewa
            lvNewDevices.setAdapter(adapter);
            //na ListView dodajemo događaj pritiska na neki element te liste
            lvNewDevices.setOnItemClickListener(
                ListOfDevicesFragment.this);
        }
        ...
    }
}
```

Slika 39. BroadCastReceiver - Bluetooth

Ako koristimo WiFi način komunikacije, *BroadcastReceiver* će također osluškivati događaje pronađenih WiFi uređaja i dodavati ih u listu. To je izvedeno na sličan način; događaj na koji će reagirati je *WIFI_P2P_PEERS_CHANGED_ACTION*. To je događaj promjene dostupnih uređaja, odnosno to znači da se popis dostupnih uređaja promijenio na način da je pronađen novi uređaj, da su svojstva nekog dostupnog uređaja na neki način izmijenjena ili da je izgubljen neki uređaj (npr. da više nije u doseg ili je isključen WiFi na tom uređaju i sl.). Kada *BroadcastReceiver* primi navedeni događaj, izvršit će se sljedeće naredbe:

```
private BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();

        ...

        if (action.equals(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION) {
            final ArrayList<WifiP2pDevice> wifiP2pDevices = new
                ArrayList<>();
            WifiP2pManager wifiP2pManager = (WifiP2pManager) getContext().
                getSystemService(Context.WIFI_P2P_SERVICE);
            WifiP2pManager.Channel wifiP2pChannel = wifiP2pManager.
                Initialize(getContext(), Looper.getMainLooper(), null);
            if (wifiP2pManager != null)
                wifiP2pManager.requestPeers(wifiP2pChannel, new
                    WifiP2pManager.PeerListListener() {
                        @Override
                        public void onPeersAvailable(WifiP2pDeviceList peerList)
                        {
                            for(WifiP2pDevice device : peerList.
                                getDeviceList())
                                    wifiP2pDevices.add(device);
                        }
                    });
            iConnections.addDevices(wifiP2pDevices);
            lvNewDevices.setAdapter(adapter);
        });
        lvNewDevices.setOnItemClickListener(
            ListOfDevicesFragment.this);
    }

    ...

}
}
```

Slika 40. BroadCastReceiver - Wifi

Navedeni kod radi sljedeće:

- stvara listu tipa *WifiP2pDevice* u koju ćemo dodavati pronađene uređaje,
- stvara *WifiP2pManager*, kao i u prethodnoj aktivnosti,
- stvara kanal, *Channel*, koji služi kao poveznica između aplikacije i WiFi P2P okosnice. Taj kanal se koristi kao argument u brojnim funkcijama u radu s WiFi-jem,
- dohvaća listu dostupnih uređaja,
- svaki uređaj iz te liste dodaje u našu kreiranu listu,

- dodaje pronađene uređaje u listu klase *WiFi*,
- postavlja adapter za *ListView*,
- na *ListView* postavlja događaj pritiska na neki element liste.

Sada imamo listu koja će prikazati dostupne uređaje, no treba na neki način pokrenuti pronalaženje tih uređaja. U aktivnost dodamo referencu na gumb koji smo kreirali u dizajneru okvira aplikacije.

Pritiskom na taj gumb izvršit će se sljedeće naredbe:

```
btnDiscover.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        iConnections.clearList();  
        iWireless.discover(broadcastReceiver);  
    }  
})
```

Slika 41. Slušač nad gumbom Discover

Izvede se dvije metode: *clearList* i *discover*. *clearList* je metoda koja će isprazniti listu koja sadrži popis dostupnih uređajima i ponovno će se pokrenuti pronalazak dostupnih uređaja, odnosno drugim riječima, pritiskom na gumb „Traži uređaje“ osvježiti će se popis dostupnih uređaja.

```
@Override  
public void clearList(){  
    bluetoothDevices.clear();  
}
```

Slika 42. Metoda clearList

Metoda *clearList* u klasi Bluetooth zapravo isprazni listu Bluetooth uređaja, ako postoje, koje smo dodali pronalaskom dostupnih uređaja. Metoda *discover* već je opisana.

4.4.6.ConnectedDialogFragment

Prije nego krenemo s objašnjavanjem ovog fragmenta, objasniti ćemo metode klase *BluetoothCommunicator* potrebne za ostvarivanje Bluetooth komunikacije s robotom.

U konstruktoru ove klase se dohvaća zadani Bluetooth adapter koji nam je potreban kasnije za stvaranje *socketa* preko kojeg ćemo slati i primiti poruke od robota.

Prva metoda po redu je *initializeSocket* koja služi za uspostavljanje *Bluetooth socketa* za komunikaciju s robotom i prima argumente:

- *address* – adresa uređaja s kojim smo uspostavili vezu,
- *handler* – referenca na *Handler* koji služi za upravljanje porukama koje primamo od robota preko dretve.

```
@Override
public void initializeSocket(String address, Handler handler) {
    try {
        bluetoothSocket = bluetoothAdapter.getRemoteDevice(address).
            createRfcommSocketToServiceRecord(uuid);
        bluetoothSocket.connect();
    } catch (IOException e) {
        e.printStackTrace();
    }
    this.handler = handler;

    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    try {
        tmpIn = bluetoothSocket.getInputStream();
        tmpOut = bluetoothSocket.getOutputStream();
    } catch (IOException e) {
        e.printStackTrace();
    }

    inputStream = tmpIn;
    outputStream = tmpOut;
}
```

Slika 43. Metoda initializeSocket

Preko adrese koju smo prosljedili dohvaćamo povezani uređaj i stvaramo s njime sigurnu *peer-to-peer* vezu te se spajamo s tim uređajem preko *socketa* i time je stvoren naš komunikacijski kanal. Ovo povezivanje potrebno je staviti u *try-catch* kontrolnu strukturu jer može doći do iznimaka koje bi dovele do rušenja aplikacije ukoliko bi nešto pošlo po zlu prilikom pokušaja uspostavljanja komunikacijskog kanala. Još je potrebno dohvatiti ulazni i

izlazni tok podataka preko *socket*a. Ovo je isto potrebno staviti u *try-catch* jer, ako je ranije nešto pošlo po zlu, neće se ni ovo moći izvršiti i rezultat će neuhvaćenom iznimkom. Ako su tokovi podataka uspješno dohvaćeni, tada se oni dodjeljuju varijablama koje smo kreirali i koje ćemo koristiti kao objekte preko kojih ćemo slati i primiti poruke.

Sljedeća metoda je *sendCommand* koja kao argument prima podatkovni niz (poruku) koji ćemo slati robotu pozivom ove metode.

```
@Override
public void sendCommand(String command) {
    byte[] message = command.getBytes();
    try {
        outputStream.write(message);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Slika 44. Metoda sendCommand

Podatkovni niz potrebno je pretvoriti u niz bajtova jer se prijenos podataka kroz *socket* obavlja upravo u tom obliku (nizu bajtova). Za slanje poruke koristimo objekt *outputStream* kojeg smo postavili u prethodnoj metodi i nad tim objektom pozivamo metodu izlaznog toka podataka – *write*, koja kao argument prima naš niz bajtova koji se stavlja u taj izlazni tok podataka i time je spreman za slanje.

Zadnja metoda je metoda *receive* koja služi za primanje poruka.

```
@Override
public void receive() {
    byte[] buffer = new byte[1024];
    int bytes;

    while (bluetoothSocket != null){
        try {
            bytes = inputStream.read(buffer);
            handler.obtainMessage(1, bytes, -1, buffer).sendToTarget();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Slika 45. Metoda receive

Prvo se inicijalizira *buffer* proizvoljne veličine. Mi smo odabrali 1024 bajta. Varijabla *bytes* služi za spremanje broja pročitanih bajtova poruke koji dobijemo pozivom metode *read* nad objektom ulaznog toka podataka, prosljeđujući kao argument *buffer* u koji je zapisana pristigla poruka. Varijabla *bytes* prosljeđuje se kao argument metodi *obtainMessage* koja se

poziva nad *handlerom*, a služi za vraćanje objekta tipa *Message*. Taj objekt tipa *Message* prosljeđuje se *handleru* kao argument te u *handleru* možemo nad tim objektom izvršavati radnje koje su nam potrebne. Prvi argument metode *obtainMessage* je identifikator preko kojeg ćemo dohvatiti poruku u *handler*, drugi argument je duljina poruke (ranije spomenuti *bytes*), kao vrijednost trećeg argumenta smo stavili -1 jer nam nije potrebna u našem slučaju, a zadnji, četvrti argument je vrijednost koju dodjeljujemo objektu poruke koji prosljeđujemo *handleru*, a to je ustvari poruka koja se nalazi spremljena u *bufferu*.

Na ovome fragmentu nalazi se gumb s tekстом „Šalji“. Pritiskom tog gumba *mBot* će se početi kretati kroz labirint. Ovo je trenutak kada se počinje provoditi algoritam za pronalaženje izlaza iz labirinta. No, prije toga, aplikacija prilikom pokretanja ovog fragmenta obavi još neke radnje. Prva je dohvaćanje spremljenog odabranog načina komunikacije. Druga je pokretanje dretve *SendReceive*. *SendReceive* je zapravo klasa koja proširuje dretvu, što znači da se izvođenje metoda te klase prepušta dretvama kako bi se oslobodila glavna dretva koja upravlja svim ostalim operacijama aplikacije. Treća je pokretanje dviju dretvi; jedne koja će pokrenuti metodu klase *SendReceive* za primanje poruka od *mBot*-a i druge, također metode klase *SendReceive* koja će poslati prvu poruku *mBot*-u i započeti provedbu algoritma. Komunikacija se dalje odvija tako da *mBot* detektira prepreke i šalje aplikaciji poruke. Aplikacija na te poruke odgovara tako da *mBotu* šalje instrukcije što mu je činiti u kojem slučaju.

Pokretanje dretve *SendReceive* izvodi se na način da se stvori instanca klase *SendReceive*, prosljeđujući joj pritom potrebne parametre:

- *deviceAddress* – već ranije spomenuta adresa uređaja s kojim smo se povezali u prethodnom fragmentu i
- *handler*, kojeg smo ranije opisali.

```
private void connect(String deviceAddress){
    switch (sharedPreferences.getString(„TypeOfCeonnection, „DEFAULT“)){
        case „bluetooth“:
            sendReceive = new SendReceive(deviceAddress, handler);
            sendReceive.start();
            break;
        case „wifi“:
            sendReceive = new SendReceive(„adresa“, handler);
            sendReceive.start();
            break;
    }
}
```

Ovdje se provjerava odabrani način komunikacije te mu se prosljeđuje pripadajuća adresa Bluetooth, odnosno WiFi uređaja. Nažalost, budući da nismo imali funkcionalni WiFi modul za *mBot*a, nismo implementirali samu WiFi komunikaciju.

Dakle, da bismo uspjeli upravljati mBotom potpuno autonomno, potrebno je uspostaviti i održavati *bluetooth* komunikaciju između pametnog telefona i mBota. Također, kako bi pametni telefon mogao upravljati mBotom, potrebne su mu neke informacije. Spomenute informacije se primaju u sljedećoj metodi.

```
List<String> listOfRecvMessages = new ArrayList<>();

boolean errorAtSum = false;
Handler handler = new Handler(new Handler.Callback() {
    @Override
    public boolean handleMessage(Message msg) {

        if(msg.what == 1){
            byte[] readBuffer = (byte[]) msg.obj;
            String message = new String(readBuffer, 0, msg.arg1);

            if(message.startsWith("m") && message.endsWith("b"))
            {
                String mBotMsg = "";
                try {
                    mBotMsg = message.substring(message.lastIndexOf('m')+1,
message.lastIndexOf('b'));
                }
                catch(Exception ex) {
                    mBotMsg = "";
                }

                listOfRecvMessages.add(mBotMsg);

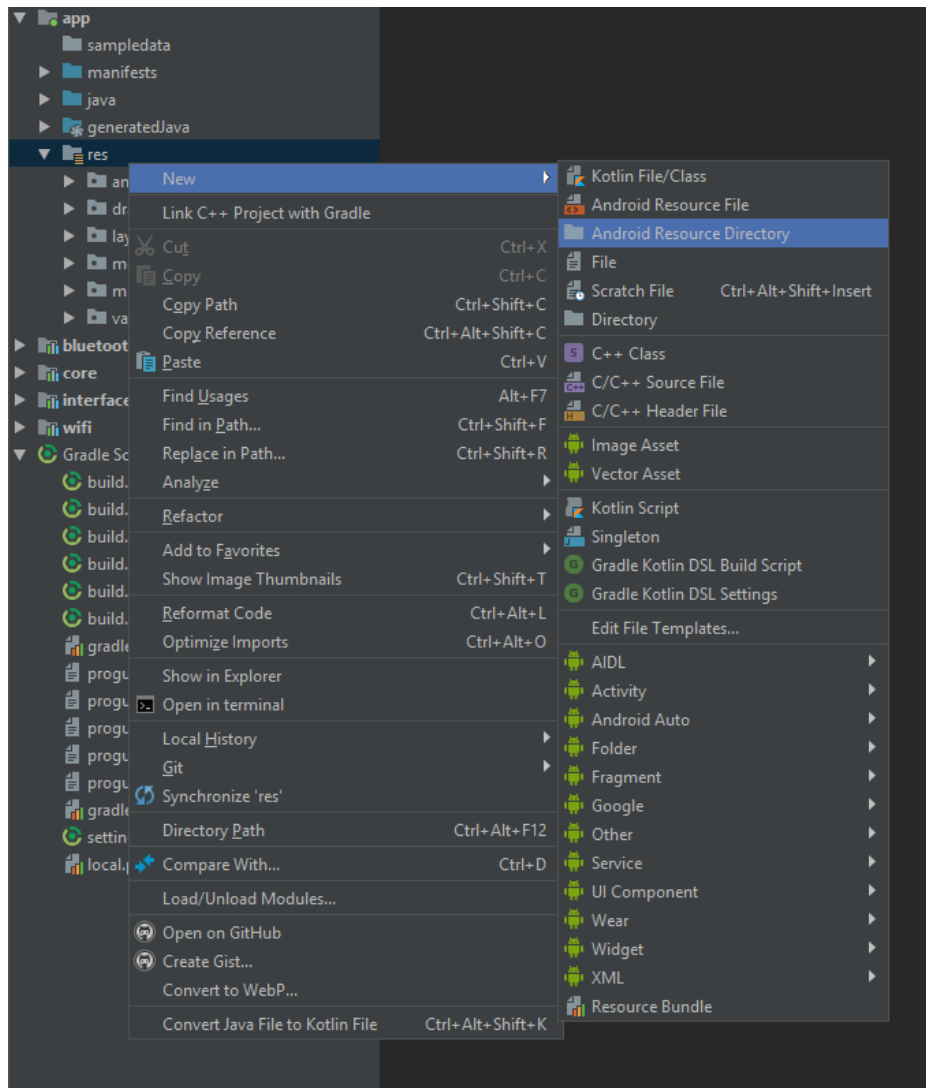
                if(mBotMsg.equals("OV"))
                {
                    for (String info:listOfRecvMessages)
                    {
                        SetInfoToSensor(info);
                    }
                    sendReceive.write(pathFinder.FindPath());
                    listOfRecvMessages.clear();
                }
            }
        }
        return true;
    }
});
```

Slika 46. Handler metoda

Primanje informacija se odvija sljedećim redoslijedom. Kako se ova metoda poziva svaki puta kada mobilni uređaj primi poruku s otvorenih vrata za komunikaciju (engl. *socket*), tako se i provjera primljene poruke odvija svaki put kada se primi poruka. Nakon toga, pokušava se odrediti ima li trenutno primljena poruka izvoriste od našeg mBota. Tj. na neki način treba provjeriti da li je novo-pristigla poruka zapravo neka poruka koju bismo trebali očekivati, pohraniti i daljnje obratiti u korisne informacije. To se odvija tako da se provjeri počinje li poruka slovom „m“ i završava li slovom „b“, ukoliko je ovo istina, možemo biti gotovo 100% sigurni da je ovo valjana poruka. Nakon toga, uzima se sam sadržaj te poruke bez maloprije spomenutih slova, jer ta slova služe samo kao omotači korisnog sadržaja. Nakon što se ovako uzme sadržaj, on se pohrani u listu koju ćemo koristiti kao skladište informacija. Sve ove radnje odvijaju se tako dugo dokle god ne pristigne poruka „OV“ (skraćena od „Over“), koja označava kraj slanja od strane mBota. Nakon primitka ove poruke, prolazi se kroz svaku poruku posebno, te se ta poruka dalje rastavlja na dijelove kako bi se izvukle vrijednosti senzora. U metodi „SetInfoToSensor“ svaka primljena poruka se provjerava ne bi li se saznalo o kojem se senzoru radi u toj informaciji. Kada se to sazna, informacija se zapiše u klasu određenog senzora kao statična varijabla. Nakon toga može započeti srž ove aplikacije, a to je algoritam za pronalazak izlaza iz labirinta. Metoda se zove „FindPath“, a poziva se iz imenovanog objekta „pathFinder“, klase „MBotPathFinder“.

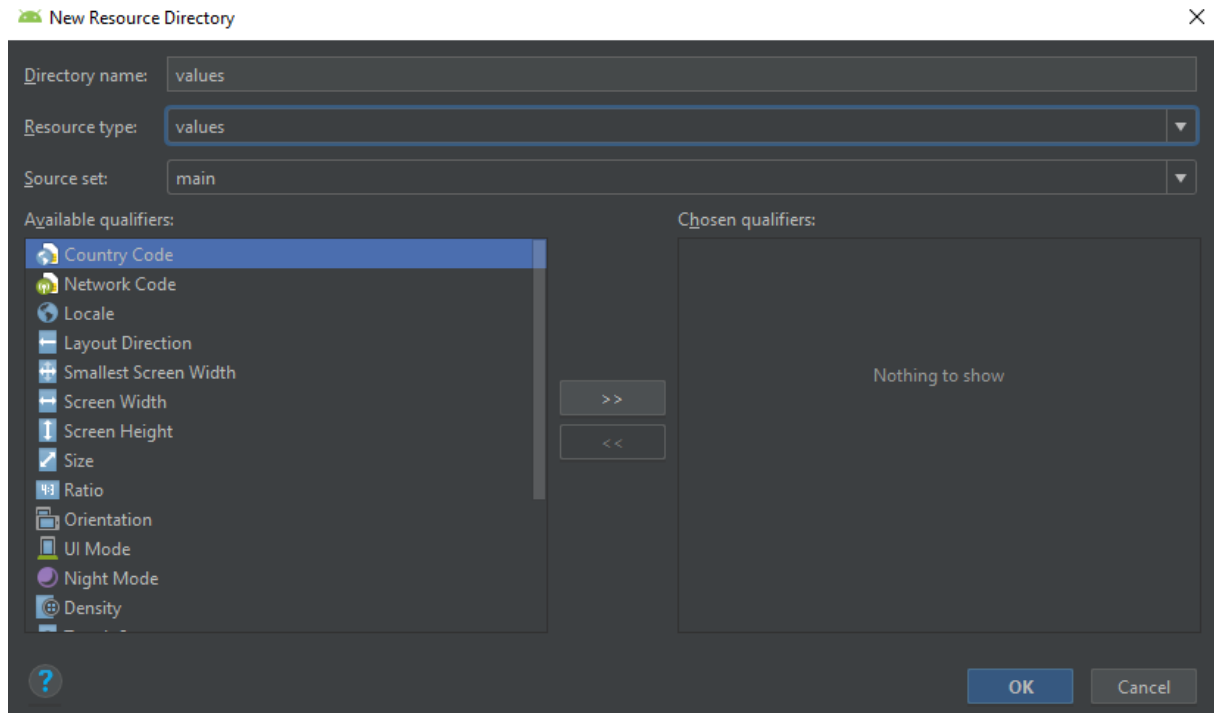
4.4.7. Izrada animacija

Kako bi se mogle izraditi animacije u Android Studio, potrebno je u modulu kreirati direktorij anim u poddirektoriju modula res. Pritiskom desnog klika miša na direktor res otvara se padajuća lista. U padajućoj listi potrebno je odabrati New -> što dovodi do nove padajuće liste iz koje je potrebno odabrati opciju Android Resource Directory. Na donjoj slici prikazan je dosadašnji postupak.



Slika 47. Padajući izbornici za animacije

Nakon što smo odabrali potrebne stavke iz padajućih lista, otvara se prozor kao što je na donjoj slici. U padajućoj listi Resource type odabiremo prvu opciju, to jest, opciju anim. Nakon toga klikom na gumb OK provodimo postupak do kraja te je sad izrađen Resource direktorij koji je potreban za animacije.



Slika 48. Novi Resource direktorij

Nakon uspješno kreiranog anim direktorija, može se krenuti u realizaciju XML datoteka koje će poslužiti kao animacija. U svrhu izrade Maze Solver aplikacije, korištena je 6 XML datoteka:

- Enter_left_to_rigth.xml
- Enter_rigth_to_left.xml
- Exit_left_to_right.xml
- Exit_right_to_left.xml
- From_bottom.xml
- From_top.xml

Enter_left_to_right.xml datoteka kao što i naziv sugerira, omogućuje ulaz fragmentu ili nekom drugom objektu ulaz s lijeve na desnu stranu. Takve ulaze omogućuju nam translate tag i njegovi atributi fromXDelta, toXDelta, fromYDelta i toYDelta. fromXDelta atribut postavljen je na -100% dok su svi ostali atributi postavljeni na 0%. Uz navedene attribute, u tagu se nalazi i atribut duration koji označava koliko milisekundi je potrebno kako bi se vrijednosti iz from atributa prebacile u to attribute. Na donjim slikama prikazan je kod svih 6 animacija.

```
enter_left_to_right.xml x
1  <?xml version="1.0" encoding="utf-8"?>
2  <set xmlns:android="http://schemas.android.com/apk/res/android">
3    <translate
4      android:fromXDelta="-100%"
5      android:toXDelta="0%"
6      android:fromYDelta="0%"
7      android:toYDelta="0%"
8      android:duration="500"/>
9  </set>
```

Slika 49. Enter_left_to_right.xml

Na donjoj slici prikazana je enter_right_to_left.xml datoteka. Atribut fromXDelta atribut postavljen je na 100% dok su preostali atributi identični onima iz enter_left_to_right.xml datoteke. Koristi se kada se fragment pojavljuje s desne strane zaslona te putuje prema lijevoj strani zaslona u trajanju od 500 milisekundi.

```
enter_right_to_left.xml x
1  <?xml version="1.0" encoding="utf-8"?>
2  <set xmlns:android="http://schemas.android.com/apk/res/android">
3    <translate
4      android:fromXDelta="100%"
5      android:toXDelta="0%"
6      android:fromYDelta="0%"
7      android:toYDelta="0%"
8      android:duration="500"/>
9  </set>
```

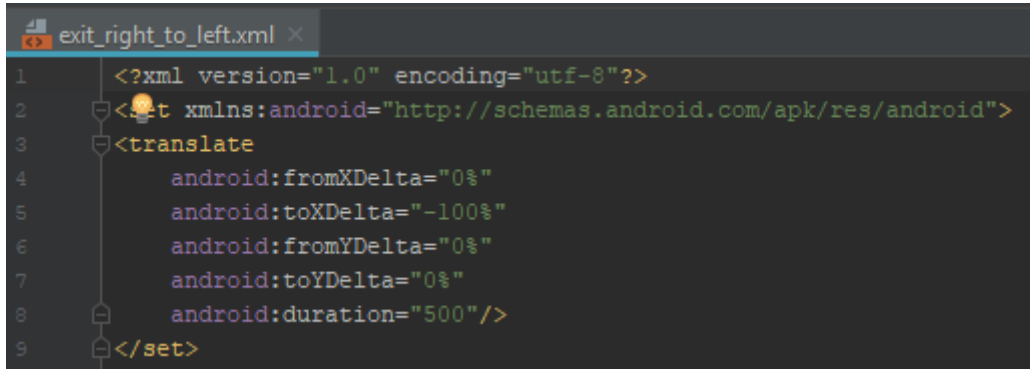
Slika 50. Enter_right_to_left.xml

Na donjoj slici prikazana je exit_left_to_right.xml datoteka. Atribut fromXDelta postavljen je na 0%, toXDelta na 100%. Ostali su atributi identični onima iz prijašnjim xml datoteka. Koristi se kada fragment izlazi iz zaslona s lijeve na desnu stranu u trajanju od 500 milisekundi.

```
exit_left_to_right.xml x
1  <?xml version="1.0" encoding="utf-8"?>
2  <set xmlns:android="http://schemas.android.com/apk/res/android">
3    <translate
4      android:fromXDelta="0%"
5      android:toXDelta="100%"
6      android:fromYDelta="0%"
7      android:toYDelta="0%"
8      android:duration="500"/>
9  </set>
```

Slika 51. Exit_left_to_right.xml

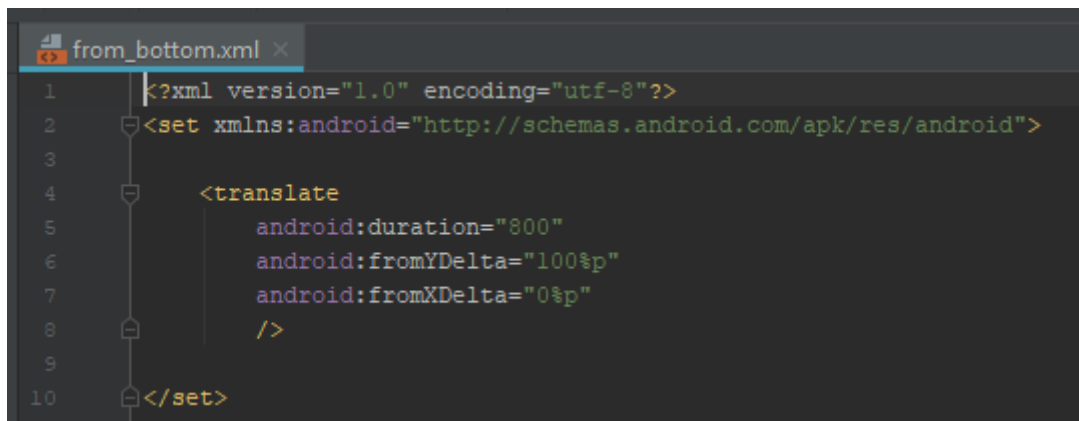
Na donjoj slici prikazana je datoteka `exit_right_to_left.xml`. Atribut `fromXDelta` ima istu vrijednost kao prethodna datoteka, dok se atribut `toXDelta` razlikuje u tome što je u ovoj datoteci postavljen na 100%. Ostali atributi su isti kao i u svim ostalim datotekama. Koristi se kada fragment izlazi iz zaslona s desne na lijevu stranu u trajanju od 500 milisekundi.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <set xmlns:android="http://schemas.android.com/apk/res/android">
3   <translate
4     android:fromXDelta="0%"
5     android:toXDelta="-100%"
6     android:fromYDelta="0%"
7     android:toYDelta="0%"
8     android:duration="500"/>
9 </set>
```

Slika 52. `Exit_right_to_left.xml`

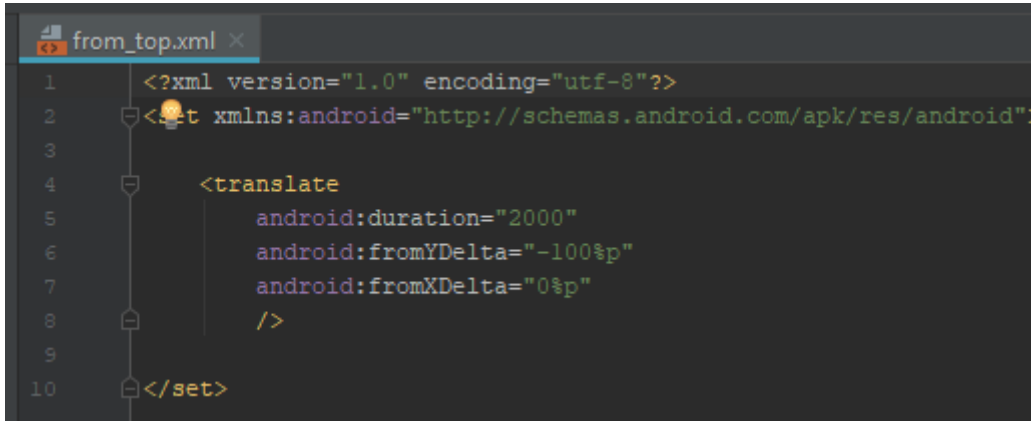
Na donjoj slici prikazana je datoteka `from_bottom.xml`. Njome se postiže ulazak fragmenata ili objekata u zaslon odozdo. Animacija, odnosno dolazak objekta na njegovo predviđeno mjesto traje 800 milisekundi. `fromYDelta` atribut postavljen je na 100%p dok je atribut `fromXDelta` postavljen na 0%p.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <set xmlns:android="http://schemas.android.com/apk/res/android">
3
4   <translate
5     android:duration="800"
6     android:fromYDelta="100%p"
7     android:fromXDelta="0%p"
8     />
9
10 </set>
```

Slika 53. `From_bottom.xml`

Na donjoj slici prikazana je from_top.xml datoteka. Njezina zadaća je animacija objekata ili fragmenata tako da ulaze u zaslon odozgo. Animacija traje 2000 milisekundi, to jest, 2 sekunde. Atribut fromYDelta postavljen je na -100%p dok je atribut fromXDelta postavljen na 0%p.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <set xmlns:android="http://schemas.android.com/apk/res/android">
3
4     <translate
5         android:duration="2000"
6         android:fromYDelta="-100%p"
7         android:fromXDelta="0%p"
8     />
9
10 </set>
```

Slika 54. From_top.xml

5. Visual Studio

U ovom poglavlju bit će opisan Microsoft Visual Studio (MSVS) kao jedan od najrasprostranjenijih razvojnih okruženja za izradu Windows programa, web stranica, aplikacija i usluga, preuzimanje MSVS te njegova instalacija.



Slika 55. Visual Studio logo

5.1. O Visual Studiu

Microsoft Visual Studio je integrirano razvojno okruženje koje je razvilo tvrtka Microsoft. On se koristi za razvoj računalnih programa, web stranica, web aplikacija, web servisa te mobilnih aplikacija. Visual Studio sadrži IntelliSense, komponentu koja dovršava naš kod tijekom pisanja. Visual Studio podržava 36 različitih programskih jezika, od kojih su neki: C, C++, C#, F#, JavaScript. Također, postoje još i različita izdanja Visual Studia: Community, Professional, Enterprise, Test Professional te Express. Community izdanje je besplatno, a Express se nakon Visual Studio Express 2017 neće koristiti. Visual Studio sadrži dosta značajki: uređivač koda, debugger, dizajner za Windows forme, klase, web, podatke te za mapiranje, zatim ima uređivač svojstava za sve objekte, preglednik objekata i mnoge druge.

5.2. Preuzimanje Visual Studia

Kao što je navedeno u prošlom potpoglavlju, Visual Studio ima jedno besplatno izdanje: Community. To izdanje je usmjereno prema individualnim korisnicima te manjim timovima.

Visual Studio može se preuzeti na službenoj Microsoftovoj stranici, na ovom linku: <https://visualstudio.microsoft.com/vs/community/>. Korisnik može odabrati koristi li Windows ili

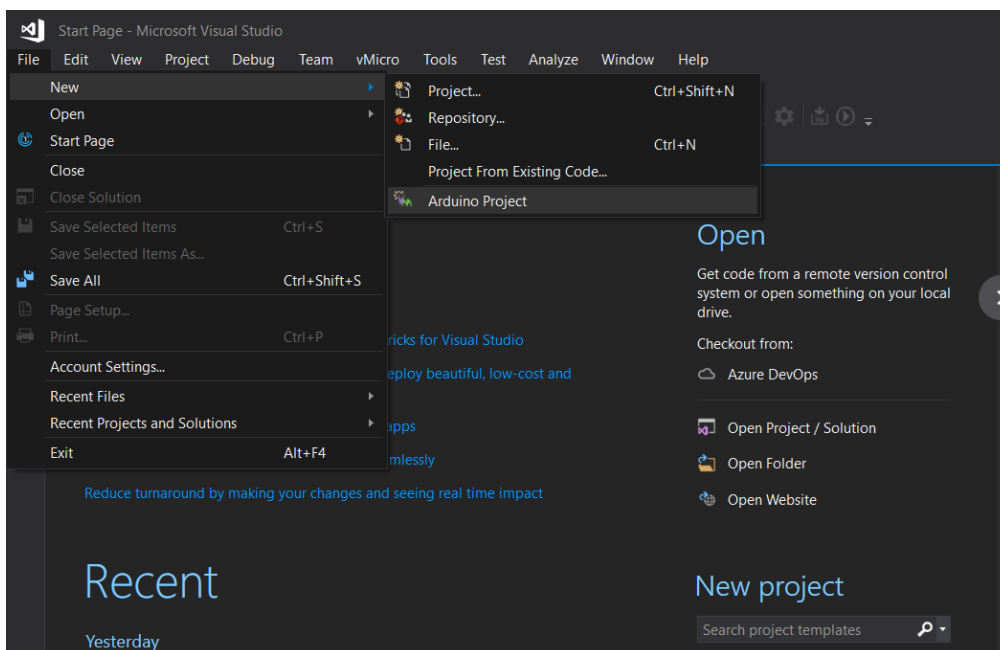
macOS te preuzme klikom na download. Trenutna verzija koja se može preuzeti je Community 2017.

5.3. Instalacija Visual Studia

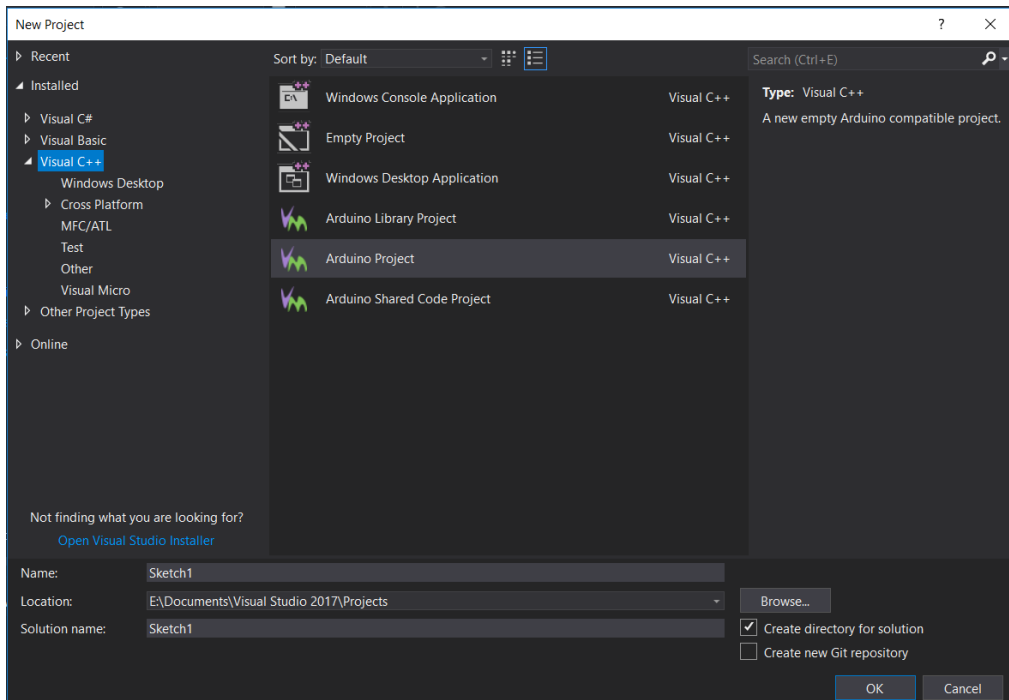
Nakon što se instalacija preuzela možemo ju pokrenuti. Potrebno je prihvatiti sve uvjete korištenja kao i kod svih ostalih programa koje instaliramo. Nakon toga nam se otvara prozor u kojem biramo koje sve programske jezike želimo koristiti. Odaberemo C++ za potrebe ovog projekta, ali mogu se i ostali jezici preuzeti. Nakon toga treba pričekati da se preuzme sve što treba i instalira.

5.4. Izrada novog projekta

Nakon što smo instalirali Visual Studio možemo krenuti na izradu projekta. Za izradu Arduino projekta treba nam dodatak Arduino IDE fo Visual Studio (možemo programirati i u samom Arduino, ali ovako je jednostavnije). Arduino IDE možemo preuzeti na ovoj poveznici: <https://marketplace.visualstudio.com/items?itemName=VisualMicro.ArduinoIDEforVisualStudio> , te samo instaliramo. Sada konačno možemo kreirati Arduino projekt. Odemo na File->New->Arduino Project ili File->New->Project... te odaberemo pod sekciji C++ što želimo raditi za Arduino.



Slika 56. Kreiranje novog Arduino projekta

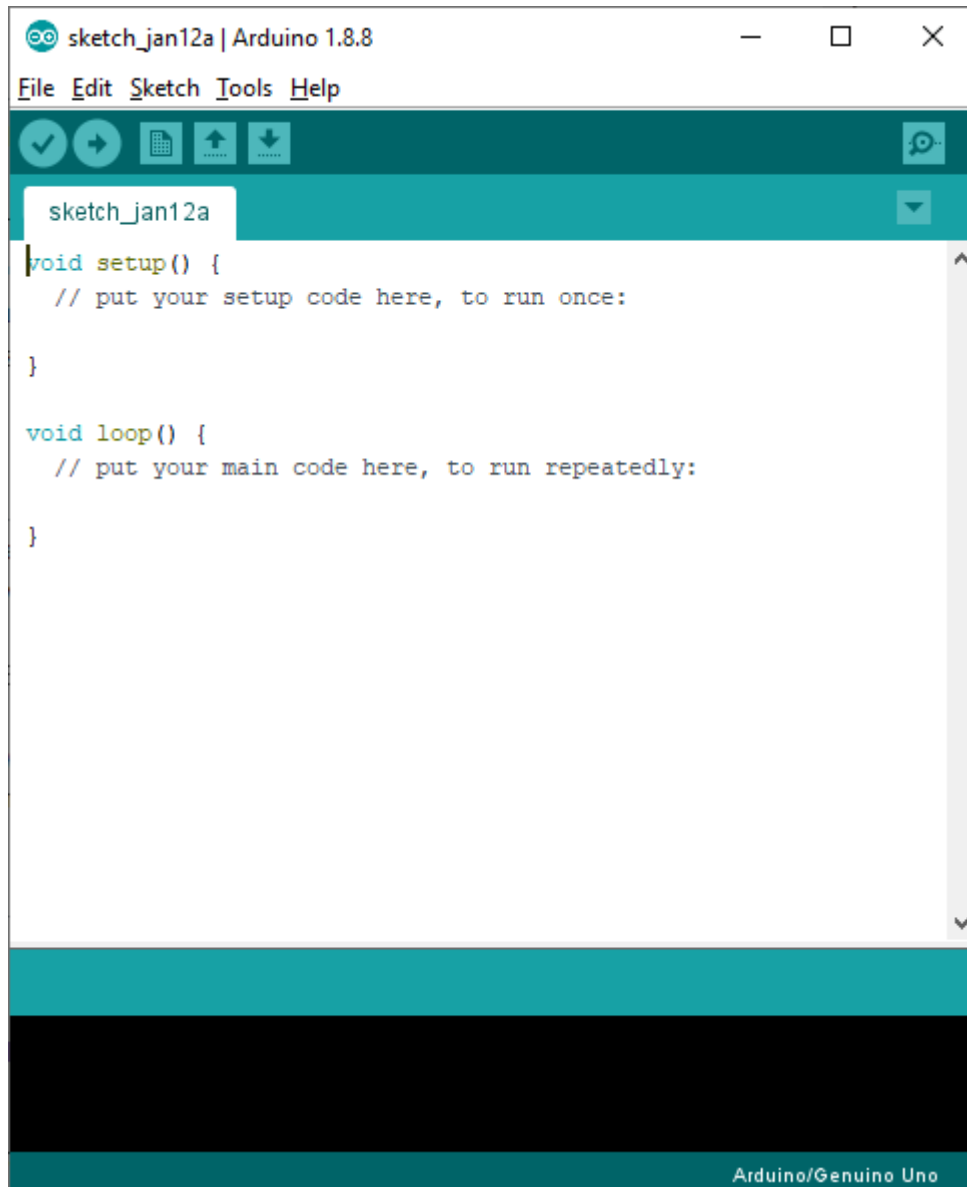


Slika 57. Odabir Arduino projekta

5.5. Arduino implementacija

Programski jezik korišten za implementaciju programskog koda algoritma na robotu je C++. C++ sadrži podršku za objektno orijentirano programiranje te je u ovom slučaju, jednostavan za savladati. U sljedećih nekoliko slika i komentara, opisati ćemo kako implementirati ovakav algoritam u Arduinou.

5.5.1. Kreiranje novog projekta u Arduinou



Slika 58. Incijalni projekt u Arduino IDE

Odabiremo File -> New i zatim će se otvoriti novi projekt, a može se nastaviti i raditi na već otvorenome.

5.5.2. Početak implementacije

Na samome početku potrebno je uključiti potrebne biblioteke koje su već unaprijed predviđene za rad sa Arduino pločicama. Zato ćemo u početku uključiti biblioteke sljedeće biblioteke :

- **MeMCore.h** – Driver za mCore ploče koja sadrži metode predefinirane za rad sa robotom (Upravljanje motorima, tipke, bluetooth i sl.)
- **SoftwareSerial.h** – podrška za serijsku komunikaciju
- **Arduino.h** – Dozvoljava korištenje programa koji se pišu za Arduino

```
#include "MeMCore.h"  
#include <SoftwareSerial.h>  
#include "Arduino.h"
```

Slika 59. Uključivanje potrebnih biblioteka

Također, na samome početku potrebno je inicijalizirati nekoliko objekata kako bismo imali kostur za daljnji rad sa robotom.

```
int index = -1;  
  
typedef struct objektPrimljenePoruke {  
    String sadrzaj;  
};  
  
const int velicinaPolja = 10;  
objektPrimljenePoruke poljeRadnji[velicinaPolja] = {""};  
  
MeBuzzer buzzer = MeBuzzer();  
String myString;  
MeDCMotor leftMotor(M1);  
MeDCMotor rightMotor(M2);  
  
MeUltrasonicSensor ultraSonic(3);  
MeUltrasonicSensor ultraSonicRight(4);  
  
MeLineFollower lineFollower(1);  
  
uint16_t brzinaKretanja = 127;  
  
Me4Button button = Me4Button();  
  
MeBluetooth bluetooth = MeBluetooth();
```

Slika 60. Inicijaliziranje globalnih varijabli

Inicijalizacija „index“ tipa „int“ potreban je za rad sa poljem i pamćenjem trenutno posljednjeg zauzetog mjesta u polju. Zatim smo u sljedećem bloku kreirali svoju vrstu tipa koji je „objektPrimljenePoruke“ koji ima u sebi objekt tipa „String“ koji predstavlja sadržaj poruke. Zatim smo inicijalizirali konstantu za veličinu polja na vrijednost „10“.

Kreiramo objekt polja, strukture koji smo samostalno kreirali tipa objektPrimljenePoruke naziva „poljeRadnji“ koji je veličine prethodno zadane vrijednosti „velicinaPolja“. „MeDCMotor“ objekti su objekti koji služe za upravljanje motorima, jedan je M1, a drugi M2. Također koristimo i senzore za čitanje udaljenosti koji su tipa MeUltrasonicSensor i imamo senzor za praćenje linije tipa MeLineFollower. Postavljamo brzinu kretanja na konstantnu vrijednost, no to se može vrlo lako promijeniti, čak i do 255 (inicijalno ovdje je postavljena na 127 kako bi robot sporije i preciznije radio). Me4Button je objekt koji je zapravo jedina tipka na robotu kojom upravljamo koji se trenutno program izvršava na Arduino Uno pločici. MeBluetooth objekt služi za komunikaciju bluetoothom.

5.5.2.1. Setup

U metodi setup inicijaliziramo sve potrebne varijable koje su nam potrebne za daljnji rad, a to su sljedeće :

```
void setup()
{
  button.setpin(A7);
  bluetooth.begin(115200);
  bluetooth.setTimeout(2);
}
```

Slika 61. Setup metoda Arduino IDE

Postavljamo glavnu tipku na mBotu na pin A7 i postavljamo serijsku komunikaciju (Bluetooth komunikaciju) na tzv. „Baud“ (brzinu slanja signala po sekundi) na vrijednost 115200 te „Serial.setTimeout(2)“ je vrijednost maksimalnog vremena za čekanje

5.5.2.2. Loop

Kako nam je kao timu, glavni zadatak bio kreirati program koji rješava problem pronalaska izlaza iz labirinta, ali tako da sve bude autonomno, tj. da mBotom upravlja pametni telefon. Bili smo u potrebi kreirati dvije aplikacije, jednu u Android Studiju koja će generirati naredbe i slati ih na neki način mBotu i to na temelju podataka sa senzora koje mBot šalje pametnom telefonu. Dakle, aplikacija koja čita i izvršava naredbe, te šalje informacije o sensorima biti će opisana u ovom podpoglavlju. Slijedi slika koda aplikacije, tj. slika glave metode – „loop“.

```
procitanaBTPoruka = CitajBluetooth();  
if (procitanaBTPoruka.startsWith("a") && procitanaBTPoruka.endsWith("d"))  
{  
    String zadnjaPoruka = ZapisiPorukuUPolje(procitanaBTPoruka);  
  
    if (zadnjaPoruka.equals("OV"))  
    {  
        IzvrsiRadnjuBT();  
        InicijalizirajPoljePrimljenihRadnji();  
        PosaljiInfoSenzora();  
    }  
}
```

Slika 62. Glava metode Loop

U suštini, spomenuta glava metoda ima vrlo jednostavan zadatak. Prvo što je potrebno izvršiti jest pročitati poruku s *bluetooth* modula ukoliko je to moguće, tj. ukoliko postoji skupina bajtova u mBotovom *bluetooth* spremniku (međuspremniku, engl. buffer). Nakon što je poruka pročitana (ili nije, ovisno o tome da li je bilo što dostupno za čitanje), poruka se zapisuje u varijablu „procitanaBTPoruka“ koja je tipa *String*. Ukoliko ta poruka počinje slovom „a“ i završava slovom „d“, poruka se zapisuje u spremnik primljenih poruka jer to znači da je to poruka koju je poslala naša aplikacija. Spremnik je polje koje prima tekst. Spomenute radnje se odvijaju tako dugo dok god se ne primi poruka „OV“ (skraćeno od „Over“), koja označava da je aplikacija na pametnom telefonu poslala sve naredbe koje mBot treba izvršiti. Kada se je ustanovilo da je navedena selekcija istinita, tj. primljena je poruka „OV“, mBot može odmah krenuti na izvršavanje naredbi. Po završetku naredbi, polje u koje su se zapisivale naredbe primljene od pametnog telefona mogu se izbrisati i inicijalizirati na početno stanje. Zadnja metoda u ovome dijelu je obavljanje slanja informacija sa senzora. Slanje se obavlja tako da mBot pročita vrijednosti sa svih senzora koje ima, te onda te informacije šalje *bluetoothom*, i to svaku u razmaku od 250ms, te svakoj poruci na početak dodaje slovo „m“ i na kraj slovo „b“, kako bi aplikacija na pametnom telefonu znala odrediti da je to poruka našeg mBota. Na kraju se pošalje i informacija za kraj slanja „OV“.

5.5.2.3. CitajBluetooth()

Ova metoda služi kao što i sam naslov asocira, na čitanje podataka koje primamo preko bluetootha. Bitno je napomenuti kako se bežičnim putem Bluetootha poruke šalju u obliku polja byte-ova. Implementirana metoda izgleda u sljedećem obliku :

```
String CitajBluetooth()
{
    String btPoruka = bluetooth.readString();

    if (btPoruka.startsWith("MS:"))
    {
        index++;
        String gotovaPoruka = btPoruka.substring(btPoruka.lastIndexOf(':') + 1, btPoruka.length());

        poljeRadnji[index].sadrzaj = gotovaPoruka;
        return gotovaPoruka;
    }

    return "";
}
```

Slika 63. Metoda CitajBluetooth()

Inicijaliziramo varijablu btPoruka tipka „String“ koja u sebe prima podatak koji se šalje preko Bluetooth komunikacije kojega čitamo pomoću „bluetooth.readString()“. Pošto se preko bluetooth komunikacije šalju poruke cijelo vrijeme koje služe za čekanje za vezom, potrebno je bilo na neki način označiti poruku koju ćemo prepoznati da je naredba za robota, a mi smo to napravili na način da početak poruke započne sa „MS:“ i zbog toga razloga u „if“ grananju provjeravamo da li poruke započinje sa „MS:“ i ako započne, povećavamo indeks polja za jedan kako bismo zapisali novu poruku na prazno mjesto u polju i zatim ponovno provjeravamo da li je zadnji znak poruke „:“ i ako je, to je kraj poruke. Tu poruku zatim spremamo na trenutni indeks i vraćamo poruku u obliku „String“.

5.5.2.4. IzvršiRadnjuBT()

Metoda služi za čitanje primljene poruke preko bluetootha i njeno izvršavanje, ovisno o obliku i sadržaju koji je u njoj sadržan. U metodi se nalazi jednostavna „for“ petlja koja prolazi kroz sve elemente polja i ovisno koji je njezin sadržaj, izvršava tu radnju. Metoda je sljedećeg oblika :

```
void IzvršiRadnjuBT()
{
    for (int i = 0; i < index; i++)
    {
        String radnja = poljeRadnji[i];

        if (radnja.equals("RL"))
            Skreni('l', 90, brzinaKretanja);

        else if (radnja.equals("RR"))
            Skreni('d', 90, brzinaKretanja);

        else if (radnja.equals("RM"))
            Kreni(brzinaKretanja);

        else if (radnja.equals("SM"))
            ZaustaviMotore();

        else if (radnja.equals("SUL"))
        {
            leftMotor.run(-brzinaKretanja-75);
            rightMotor.run(brzinaKretanja-10);
        }
        else if (radnja.equals("SUR"))
        {
            rightMotor.run(brzinaKretanja + 75);
            leftMotor.run(-brzinaKretanja + 10);
        }

        else if (radnja.equals("RF"))
        {
            Skreni('l', 90, brzinaKretanja);
            Skreni('l', 90, brzinaKretanja);
        }

        delay(100);
    }
}
```

Slika 64. Metoda IzvršiRadnjuBT()

Metoda provjerava kroz „if“ grananja čemu je jednak sadržaj poruke i zatim ga izvršava. Na primjer, ako je primljena poruka „RotateLeft“, on će ući u tu granu i izvršiti metodu „Skreni('l',90,brzinaKretanja)“ što u prijevodu znači da skrene lijevo, za 90 stupnjeva i brzinom

kretanja koja je na početku inicijalizirana na vrijednost „127“. Metode za izvršavanje kretnji robota biti će opisane u nastavku.

5.5.2.5. Metode za kretanje robota

Metode koje su implementirane za kretanje robota su :

- **Kreni** – Jednostavna metoda koja zapravo postavlja brzine kretanja motora na iste vrijednosti

```
void Kreni(uint16_t brzinaKretanja)
{
    leftMotor.run(-brzinaKretanja);
    rightMotor.run(brzinaKretanja);
}
```

Slika 65. Metoda za kretanje robota unaprijed ili unazad

- **Skreni** - metoda prima argumente smjer koja može biti 'l' za lijevo, a bilo koja druga će okretati robota na desnu stranu.

```
void Skreni(char smijer, uint16_t stupnjevi, uint16_t brzina)
{
    if (smijer == 'l')
    {
        leftMotor.run(brzina);
        rightMotor.run(brzina);
    }
    else
    {
        leftMotor.run(-brzina);
        rightMotor.run(-brzina);
    }
    delay(IzracunajVrijemeRotacije(stupnjevi, brzina));
    ZaustaviMotore();
}
```

Slika 66. Metoda za promjenu smjera kretanja robota

- **lineFollow** – Metoda koja prati liniju kroz labirint i koja ovisno o vrijednosti senzora (S1_IN označava da senzor vidi crtu, S1_OUT znači da senzor je izašao sa crte)

```
void lineFollow() {
    int sensorStateCenter = lineFollower.readSensors();

    switch (sensorStateCenter)
    {
        case S1_IN_S2_IN:
            //senzori su na centru, kreći se ravno
            bluetooth.print("OnLine");
            break;
        case S1_IN_S2_OUT:
            //senzor 2 je van linije (desni senzor)
            bluetooth.print("RightOut");
            break;
        case S1_OUT_S2_IN:
            //senzor 1 je van linije (lijevi senzor)
            bluetooth.print("LeftOut");
            break;
        case S1_OUT_S2_OUT:
            //oba senzora su van linije
            bluetooth.print("BothOut");
            break;
    }
}
```

Slika 67. Metoda za praćenje linije labirinta

Za početak je potrebno postaviti da se nešto čita iz senzora, to se radi s prvom linijom.

Nakon toga se gledaju mogući slučajevi očitavanja senzora:

1. Oba senzora su na crti.
2. Lijevi senzor je na liniji, a desni je van linije.
3. Lijevi senzor je van linije, a desni je na liniji.
4. Oba senzora su van linije.

Sad u svakom slučaju definiramo ono što želimo da se pošalje aplikaciji. Najbolje je poslati string za koji ćemo biti sigurni da ćemo ga i kasnije razumjeti. Za slučaj kada su oba na liniji zato uzimamo „OnLine“, kada je desni van linije „RightOut“, kada je lijevi van linije „LeftOut“ te kada su oba van linije „BothOut“.

5.5.3. Implementacija algoritama na robotu

Algoritam za izlazak iz labirinta u ovome slučaju je implementiran kao cjelokupno rješenje programirano na strani robota koji bez potrebe mobilnog uređaja može izaći iz labirinta, a služi kao testno okruženje potrebno za odvajanje odgovornosti sa robota na mobilni uređaj (podloga za autonomno upravljanje). Navedeni algoritam implementiran je za testiranje uspješnosti algoritma i sam njegov pronalazak izlaza, a također kao što je već prethodno opisano, kostur za nastavak sa radom u obliku mobilnog uređaja kao glavnog koordinatora robota koji samo odašilje vrijednosti svojih signala, a mobilni uređaj se nalazi kao donositelj odluka o kretanju. Samostalnom maštom i kreativnošću, implementirane su dvije verzije algoritma, a to su :

- Algoritam sa prioritetskim skretanjem u desno
- Algoritam sa prioritetskim skretanjem u lijevo

5.5.3.1. Algoritam sa prioritetskim skretanjem u desno

Algoritam je implementiran na sljedeći način razmišljanja i opisan je u sljedećih nekoliko točaka :

- Kretanje robota unaprijed sve dok ne dođe do prvog zida i pri čemu vrijednost očitane vrijednosti je ispod zadane (u našem slučaju, nekih 20cm)
- Provjera algoritma može li nastaviti svoju kretanju u smjeru u desno
- U slučaju nemogućnosti skretanja u desno, algoritam skreće robota još dva puta u desno kako bi nastavio svoju putanju u smjeru lijevo (u slučaju ako lijevo ne postoji zid, u suprotnome, robot se vraća otkuda je došao što implicira da je ulica bila zapravo slijepa ulica)

5.5.3.2. Algoritam sa prioritetnim skretanjem u lijevo

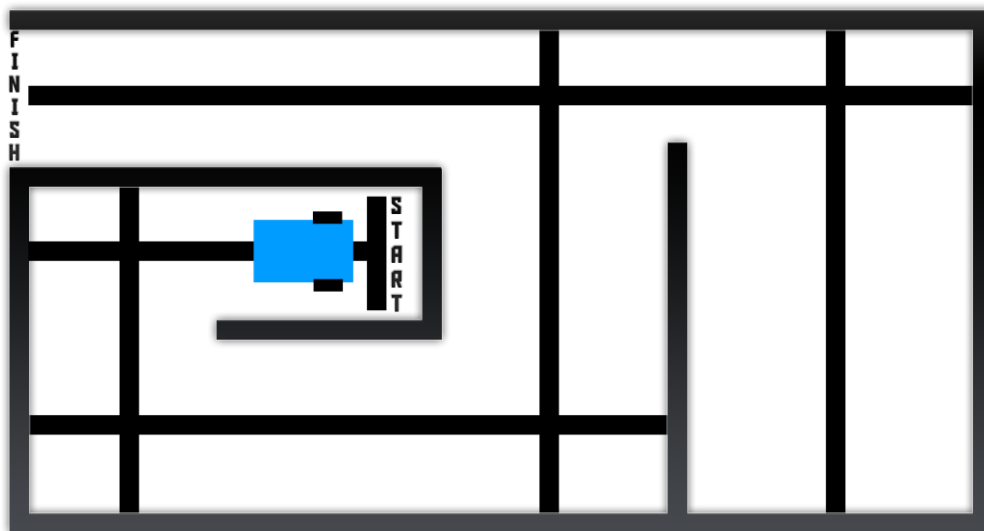
Algoritam je implementiran na isti način kao i prethodno opisani, no razlikuje se u tome što prioritizira skretanje u lijevo

- Kretanje robota unaprijed sve dok ne dođe do prvog zida i pri čemu vrijednost očitane vrijednosti je ispod zadane (u našem slučaju, nekih 20cm)
- Provjera algoritma može li nastaviti svoju kretanju u smjeru u lijevo
- U slučaju nemogućnosti skretanja u lijevo, algoritam skreće robota još dva puta u lijevo kako bi nastavio svoju putanju u smjeru desno (u slučaju ako desno ne postoji zid, u suprotnome, robot se vraća otkuda je došao što implicira da je ulica bila zapravo slijepa ulica)
- Algoritam je jednostavan te je testiran na skici labirinta koji će biti prikazan u nastavku

Valja napomenuti da u pozadini cijelog algoritma se konstantno provjerava položaj robota na crti, tj. nalazi li se robot na crti. Ukoliko robot izlazi malo sa crte, robot traži pomacima lijevo ili desno crtu kako bi se ponovno repositionira na odgovarajuću poziciju. Navedeno je implementirano na takav način zbog nekonzistentnosti skretanja zbog pražnjenja baterija, dok je isto moglo biti jednostavno riješeno pomoću kreiranja robota sa koračnim motorom koji bi uvijek radio isti korak i na takav način bi i samo skretanje bilo uvijek 100% točno, a time bi i sama implementacija algoritma bila jednostavnija, bez nepotrebnih zastoja (eng. „Delay“).

5.5.3.3. Labirint

Labirint je većih dimenzija radi lakših testiranja, kao i problema sa kojima smo se susreli radi spore responzivnosti danih senzora, ali i nekonzistentnosti skretanja u smjerovima lijevo ili desno zbog baterija koje ukoliko se isprazne malo, skretanje već nije konzistentno, već postotak točnosti pada. Labirint se sastoji od klasičnih skretanja koja su dobra za testiranja za oba algoritma sa prioritetima skretanja desno ili lijevo, te slijepo ulice koja testira okret i nastavak traženja izlaska iz labirinta. Labirint bi naravno mogao biti i teži, no zbog njegove veličine, kreiran je jednostavan, no dovoljan za testiranja. Slika slijedi u nastavku :



Slika 68. Labirint za provođenje testiranja algoritma

5.5.3.4. Implementacija u Arduino okruženju

Prethodno navedeni i opisani algoritmi slijede u obliku slika sa gotovom implementacijom. Programirano u Arduino IDE, jezik C++.

```
void rjesenjeProblemaDesno() {
  unsigned long sensorStateCenter = lineFollower.readSensors();
  unsigned long udaljenostDoZida = ultraSonic.distanceCm();
  switch (sensorStateCenter)
  {
    case S1_IN_S2_IN:
      if (udaljenostDoZida <= 22) {
        Skreni('d', 66, brzinaKretanja);
        delay(1300);
        unsigned long novaDistanca = ultraSonic.distanceCm();
        if (novaDistanca <= 22) {
          Skreni('d', 66, brzinaKretanja);
          delay(1300);
          //Oprez, slijepa ulica i vrati se unatrag
          if (ultraSonic.distanceCm() > 22) {
            Skreni('d', 66, brzinaKretanja);
            delay(1300);
            if (ultraSonic.distanceCm() <= 22) {
              Skreni('d', 66, brzinaKretanja);
              delay(1300);
              Skreni('d', 66, brzinaKretanja);
              delay(1300);
              Skreni('d', 66, brzinaKretanja);
              delay(1300);
            }
          }
        }
      }
      else {

      }
    }
    else {

    }
  }
  else {
    Kreni(brzinaKretanja);
  }
}
break;
```

```
case S1_IN_S2_OUT:
  //senzor 2 je van linije (desni senzor)
  ZaustaviMotore();
  Skreni('l', 5, 40);
  break;
case S1_OUT_S2_IN:
  //senzor 1 je van linije (lijevi senzor)
  ZaustaviMotore();
  Skreni('d', 5, 40);
  break;
case S1_OUT_S2_OUT:
  Skreni('d', 5, 40);
  break;
}
```

Slika 70. Algoritam prioriteta "desno"

Slika 69. Algoritam prioriteta "desno"

Algoritam prikazuje očitavanje trenutnog stanja senzora za praćenje linije što možemo vidjeti u prvoj liniji koda u metodi „rjesenjeProblemaDesno()“ kao i vrijednosti prednjeg senzora za učitavanje udaljenosti robota od prepreke (zida). Nakon toga, ulazi se u „Switch“ gdje ovisno o vrijednosti senzora za praćenje linije, algoritam se izvršava. Većina algoritma izvršava se dok je robot na crti, tj. oba senzora očitavaju vrijednost da se robot nalazi sa oba senzora na crti. Algoritam za lijevu stranu nećemo objavljivati jer je identičan navedenome, jedina razlika što se kao parametar unutar poziva metode „Skreni“ umjesto parametra „d“, upisuje „l“ i dalje radi na isti način. Za izvršenje algoritma, u metodu Arduina pod imenom „loop“ poziva se rjesenjeProblemaDesno()“.

5.5.4. Implementacija u Android Studio – algoritam

Ova metoda se sastoji od četiri dijela ili četiri scenarija, jedan scenarij je da je korisnik odredio neka se koriste samo prednji ultrazvučni senzori (uz koje je obavezan i čitač linije), drugi i treći scenariji su da je odabra samo prednji i desni / lijevi ultrazvučni senzor, i konačno zadnja mogućnost je da su odabrani svi senzori. Metoda kao rezultat na kraju izvođenja vraća listu naredbi koje će poslati mBotu.

5.5.4.1. Prednji ultrazvučni senzor uz čitač crte

Metoda se sastoji od dva dijela, prvi za poravnanje mBota na sredinu staze labirinta, a drugi za određivanje smjera kretanja. Metoda za centriranje mBota izgleda ovako:

```
private CommandsToMBot centerMBotFrontSensor()  
{  
    CommandsToMBot returnCmd = Null;  
  
    if(lineFollower.isLeftSideOut())  
        returnCmd = SpeedUpLeft;  
    else if(lineFollower.isRightSideOut())  
        returnCmd = SpeedUpRight;  
  
    return returnCmd;  
}
```

Dakle, metoda je vrlo jednostavna i vraća jednu naredbu, naredbu „Null“, koja zapravo ne znači ništa, tj. ukoliko se vrati spomenuta naredba, ona se ni ne šalje nakon što algoritam završi, jer se ni ne upisuje u polje za slanje. Ukoliko se nalazimo u ovoj metodi, aplikacija se je već kod primanja poruke „OV“ (zadnje poruke od mBota), zapisala sve informacija sa mBotovih senzora, pa tako i informaciju o čitaču crte. Ova metoda provjerava da li je čitač crte izvan linije koju prati s desne ili s lijeve strane. Ukoliko s lijevim senzorom nije na crti, potrebno je ubrzati lijevi motor mBota kako bi se on centrirao. Ukoliko se uspostavi drugačije, tj. da je desna strana mBota izvan linije, ubrzava se desni motor.

Nakon završetka ove metode, poziva se sljedeća metoda, a to je konkretna metoda za pronalazak izlaza iz labirinta.

```
private boolean rotated = false;  
  
private CommandsToMBot findPathFrontSensor()  
{  
    CommandsToMBot returnCmd = Null;  
  
    if(!FrontSensor.seesObstacle())  
    {  
        returnCmd = RunMotors;  
        rotated = false;  
    }  
}
```



```
}  
else if(rotated)  
{  
    returnCmd = RotateFull;  
    rotated = false;  
}  
else if(FrontSensor.seesObstacle())  
{  
    returnCmd = RotateRight;  
    rotated = true;  
}  
  
return returnCmd;  
}
```

Metoda također vraća jednu naredbu ovisno o vrijednostima senzora na koje naiđe tijekom izvršavanja. Ukoliko prednji senzor ne vidi prepreku, tj. može se kretati, naredba koja će biti vraćena je *RunMotors*, tj. kretanje naprijed. Metoda također koristi i jednu globalnu varijablu tipa *boolean* koja se postavlja na istinitu vrijednost (engl. *True*) samo ako je mobilna aplikacija odlučila da je potrebno poslati naredbu za rotiranjem udesno. Takva naredba će se poslati ukoliko prednji senzor spazi prepreku.

Na ovaj način, ukoliko dođe mBot do slijepe ulice, slati će se sljedeće radnje. Prednji senzor će vidjeti prepreku i poslati će se mBotu naredba neka se rotira za 90° udesno (vrijedi zadnja if selekcija jer je „rotated“ na *false*, također je i laž da prednji senzor ne vidi prepreku). Dakle ukoliko je slijepa ulica, a mBot se je rotirao desno, opet će prednji senzor vidjeti prepreku ispred sebe, ali se neće izvršiti normalna rotacija kako se je izvršila u prethodnom koraku, nego će se izvršiti druga if selekcija jer jer „rotated“ sada *true*, a ovaj uvjet je jači od zadnjeg, dok ovaj prvi (najjači) nije istinit. Tada će se poslati naredba koja okreće mBota za 180°, sada se mBot nalazi okrenut prema zidu s njegove lijeve strane u odnosu na smjer iz kojeg je došao. Prednji senzor će i dalje vidjeti prepreku, tada prva if selekcija ponovno neće vrijediti, „rotated“ je zbog prošlog koraka postavljen na lažnu vrijednost, te je zadovoljen treći if uvjet koji govori mBotu da se rotira ponovno za 90° udesno. U ovoj situaciji smo već bili, no sada je „rotated“ postavljen na *true*, a ne kao prije na *false* i sada se mBot ponovno može rotirati za 180°, te će u sljedećem koraku biti zadovoljen prvi uvjet, tj. da prednji senzor ne vidi prepreku, te će se mBotu slati da krene naprijed, tj. kretati će se istim putem kojim je došao do slijepe ulice.

Ovaj algoritam se može nadograditi tako da čitač linije očita križanje, te onda mobilni uređaj, tj. aplikacija mobilnog uređaja odlučuje kamo se kretati na križanju.

5.5.4.2. Slučaj sa više ultrazvučnih senzora bez čitača crte

Za sva tri prije spomenuta scenarija s više ultrazvučnih senzora, pozivaju se iste metode za rad, tj. za rješavanje problema. Jedina stvar u kojoj se razlikuju je očitavanje / izračunavanje vrijednost drugog senzora. Algoritam za svaki od senzora izgleda ovako:

```
CommandsToMBot cmdCenterMBot =  
centerMBotTwoOrMoreSensors(rightWallDistance, leftWallDistance);  
if(cmdCenterMBot != Null)  
    finalCommandList.add(cmdCenterMBot);  
  
CommandsToMBot findPathCmd =  
findPathTwoOrMoreSensors(rightWallDistance, leftWallDistance);  
if(findPathCmd != Null)  
    finalCommandList.add(findPathCmd);
```

Dakle, ukoliko je naredba metode za centriranje mBota vratila „Null“ naredbu, ne upisujemo je u listu za slanje, inače je upisujemo. Isto vrijedi i za metodu „findPathTwoOrMoreSensors“. Argumenti i jedne i druge metode su udaljenost lijevog senzora do zida i udaljenost desnog senzora do zida. U slučaju s trima sensorima, lako je pročitati vrijednost svakog senzora od zida jer nam je mBot tu informaciju poslao. Ukoliko imamo samo dva senzora, drugu vrijednost izračunamo, ako znamo kolika nam je širina staze i znamo jedan bočni senzor, lako možemo izračunati udaljenost drugog senzora od zida.

Metoda centriranja mBota s više senzora:

```
private CommandsToMBot centerMBotTwoOrMoreSensors(double  
rightWallDistance, double leftWallDistance)  
{  
    CommandsToMBot returnCommand = Null;  
  
    double sensorDistanceSum = rightWallDistance + leftWallDistance;  
  
    if(!CrossroadManager.checkIfCrossroad(sensorDistanceSum))  
    {  
        if(leftWallDistance > rightWallDistance)  
            returnCommand = SpeedUpRight;  
  
        else if(rightWallDistance > leftWallDistance)  
            returnCommand = SpeedUpLeft;  
    }  
  
    return returnCommand;  
}
```

Prvo i glavno što se radi je izračunavanje sume primljenih argumenata, tj. udaljenosti lijevog i desnog senzora od zida. Tada provjeravamo nalazi li se mBot na raskrižju ili ne. MBot se nalazi na raskrižju ukoliko je izračunata suma, zbrojena sa širinom mBota veća od širine same staze labirinta. Tada, ukoliko se mBot ne nalazi na križanju možemo ga centrirati, a inače moramo odlučiti u drugoj metodi kojim smjerom se bi mBot trebao kretati. Metoda centrira mBota tako da provjeri koji senzor mjeri veću udaljenost. Ukoliko lijevi senzor mjeri veću udaljenost od

desnoga, znači da je mBot s desne strane bliže zidu, što dalje znači da je potrebno ubrzati desni motor mBota kako bismo ga malo odmaknuli od zida. Isto vrijedi i za drugu stranu. Nakon završetka ove metode, poziva se metoda koja daje naredbe u samom kretanju mBota kroz labirint.

```
double lastSensorDistanceSum = 0;
private CommandsToMBot findPathTwoOrMoreSensors(double
rightWallDistance, double leftWallDistance)
{
    CommandsToMBot returnCmd = Null;

    double sensorDistanceSum = rightWallDistance + leftWallDistance;

    if(CrossroadManager.checkIfCrossroad(sensorDistanceSum) &&
lastSensorDistanceSum != sensorDistanceSum )
        returnCmd =
CrossroadManager.manageCrossroad(rightWallDistance,
leftWallDistance);

    else if(!FrontSensor.seesObstacle())
        returnCmd = RunMotors;

    lastSensorDistanceSum = sensorDistanceSum;

    return returnCmd;
}
```

Metoda sama po sebi izgleda dosta jednostavno i kratko, ali to nije slučaj, nego je razdvojena u više klasa i metoda. Metoda počinje isto kao i ona za centriranje mBota – računanjem sume udaljenost jednog i drugog senzora od zida. Sada ukoliko se mBot nalazi na križanju, a već je bilo rečeno kod prošle metode kako se odredi radi li se o križanju ili ne, pozivamo metodu koja određuje na koju stranu i kako će se mBot kretati. Ukoliko se mBot ne nalazi na križanju, neka se samo kreće naprijed.

Metoda koja određuje radnje na križanju izgleda ovako:

```
static boolean rotated = false;
static double distanceAfterRotate = 0;
public static CommandsToMBot manageCrossroad(double
rightWallDistance, double leftWallDistance)
{
    CommandsToMBot returnCmd = CommandsToMBot.Null;
    Sides sideToTurn = null;
    double sideSensorsDistanceSum = rightWallDistance +
```

```

leftWallDistance;

    if(rotated)
    {
        distanceAfterRotate = sideSensorsDistanceSum;
        rotated = false;
    }

    if(distanceAfterRotate > sideSensorsDistanceSum + 15 ||
sideSensorsDistanceSum > distanceAfterRotate + 15)
    {
        if(CrossroadManager.checkCrossroadSide(rightWallDistance))
        {
            returnCmd = RotateRight;
            sideToTurn = Right;
            rotated = true;
        }
        else
if(CrossroadManager.checkCrossroadSide(MBotPathFinder.FrontSensor.ge
tNumericValue()))
        {
            returnCmd = RunMotors;
            sideToTurn = Front;
            rotated = true;
        }
        else
if(CrossroadManager.checkCrossroadSide(leftWallDistance))
        {
            returnCmd = RotateLeft;
            sideToTurn = Left;
            rotated = true;
        }
        else
if(CrossroadManager.CheckIfDeadEnd(sideSensorsDistanceSum))
        {
            returnCmd = RotateFull;
            sideToTurn = FullRotate;
        }
    }
    return returnCmd;
}

```

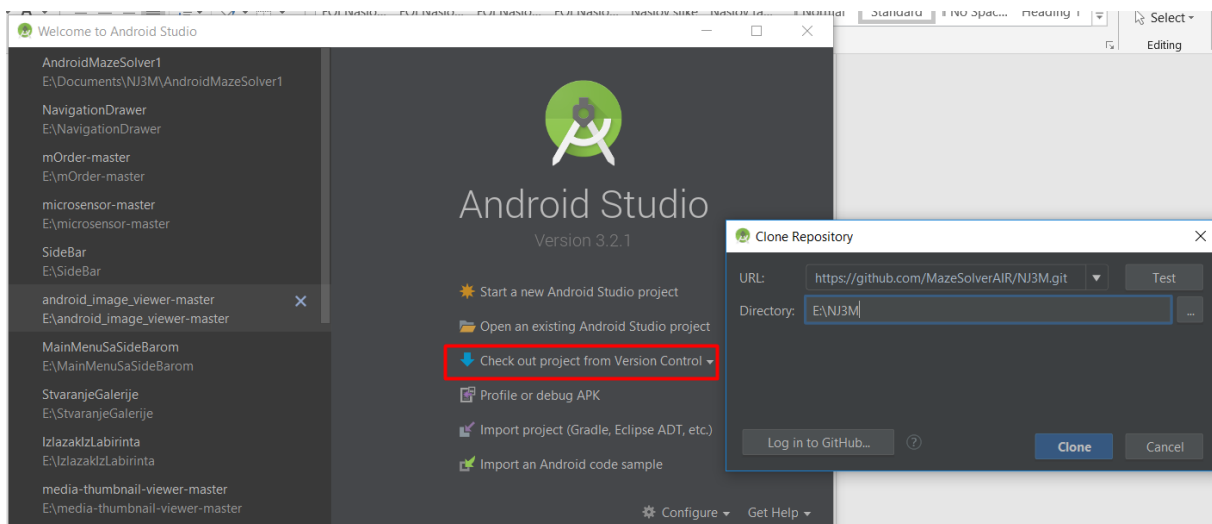
Ponovno se računa ona ista suma i to zbog toga kako bi ova metoda radila i za dva senzora, jer nam treba udaljenost svakog senzora od zida, pa ne prosljeđujemo već izračunatu sumu. Dakle, metoda prolazi kroz nekoliko if selekcija koje provjeravaju što se nalazi na križanju. Metoda „checkCrossroadSide“ prima udaljenost nekog senzora od zida i vraća istinu ukoliko se na tu stranu može skrenuti, inače vraća laž. Metoda će vratiti istinu ukoliko je promatrani argument, tj. vrijednost promatranog senzora veća od širine labirinta umanjene za širinu mBota. Kako kroz labirint prolazimo tako da prioritiziramo skretanje udesno kada god je to moguće, tako i ovdje. Ukoliko je prva if selekcija uspješna, druge se i ne provjerava, samo se skreće udesno, inače ravno ili ulijevo na kraju. Također ukoliko nijedan od ovih uvjeta nije ispunjen, znači da se radi o slijepoj ulici, te da je potrebno rotirati se za 180°. Također, pamte

se i dvije dodatne varijable, zadnja vrijednost sume udaljenosti i da li se je već na tom križanju izvršilo rotiranje. Jer kada se mBot nalazi na križanju, rotira se, još će neko vrijeme biti na istome križanju, te bi se cijelo vrijeme rotirao u krug. Zbog toga se moramo pobrinuti da se odlučuje o stranama za skretanje samo kada više nije na tom križanju. To se riješi tako da kada se skrene, varijabla „rotated“ se postavi na *true* tako da znamo da se je na tom križanju rotirao. Tada se u varijablu „distanceAfterRotate“ zapamti trenutna suma udaljenosti, a „rotated“ se opet postavlja na *false*. Tada se svaki puta na križanju provjerava razlika između udaljenosti na zadnjem rotiranju i sume na trenutnom raskrižju. Ako se te dvije vrijednosti razlikuju za više od 15 centimetara, vrlo je velika vjerojatnost da se radi o drugom križanju.

6. Preuzimanje gotovog projekta

Da bi se koristio već gotovi projekt potrebno ga je preuzeti s interneta. Naš projekt se nalazi na našem GitHub repozitoriju: <https://github.com/MazeSolverAIR/NJ3M.git>, te ga je klikom na download moguće preuzeti. Nakon toga raspakiramo zip datoteku koju smo preuzeli i možemo pokrenuti projekt. U Android Studiu odemo na File->Open... te nađemo preuzeti projekt, ili u slučaju da se po otvaranju Android Studia ne nalazimo u već nekom projektu nego na početnom zaslonu odaberemo Open an existing Android Studio project te pronađemo projekt. U Visual Studiu odemo na Open->Project/Solution... te pronađemo željeni projekt.

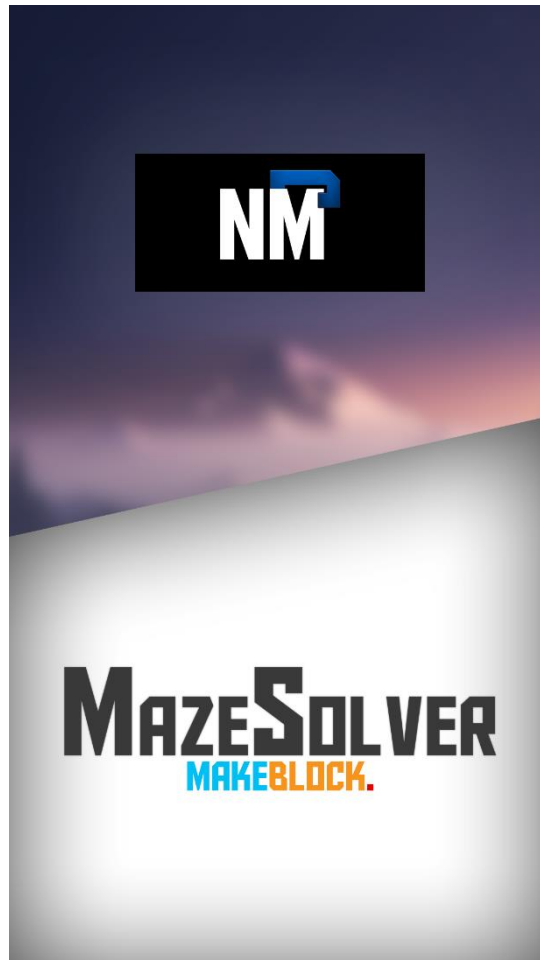
Na donjoj slici prikazan je postupak otvaranja projekta u Android Studiu preko GitHub – a.



Slika 71. Otvaranje projekta u Android Studiu preko GitHububa

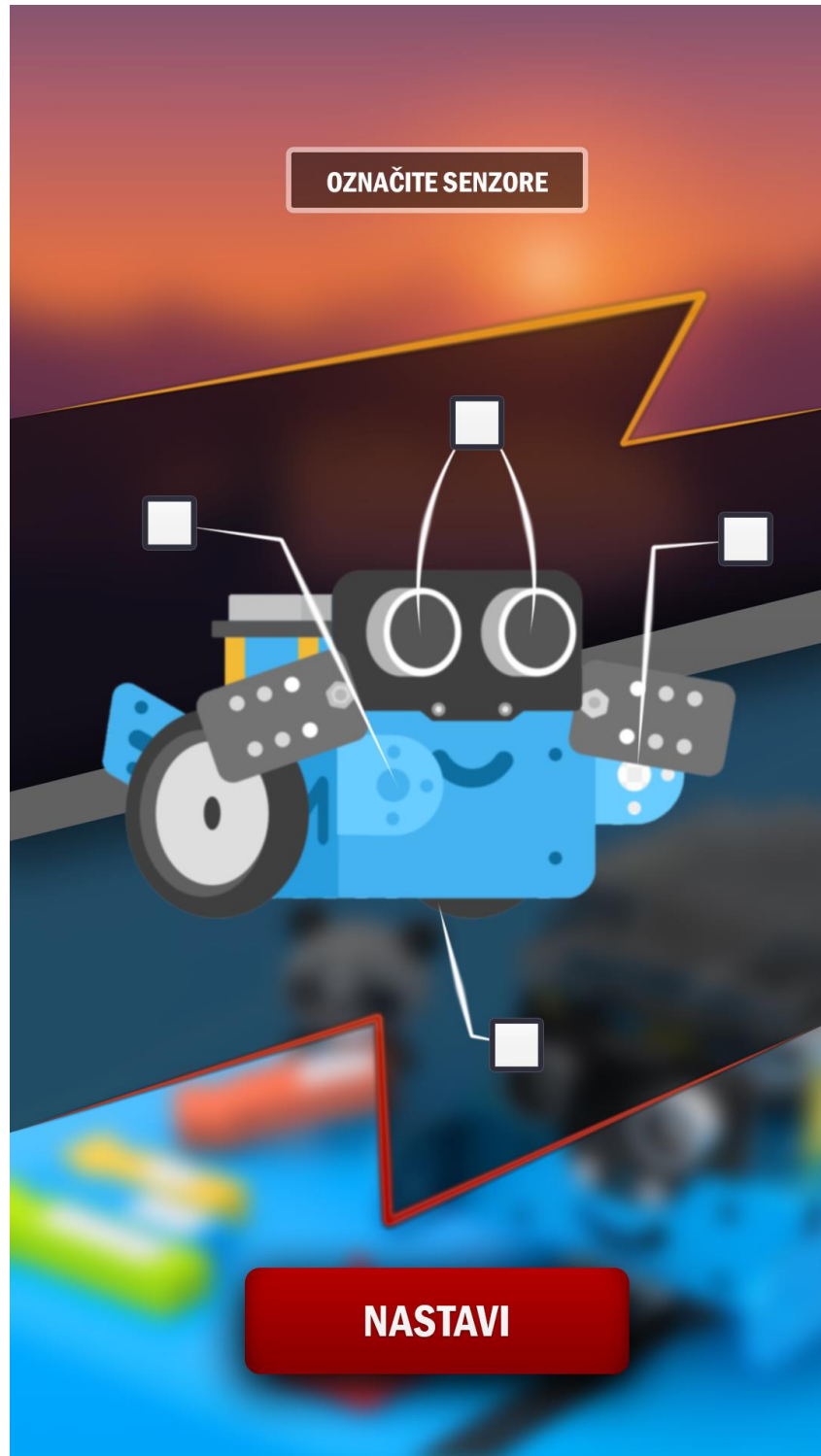
7. Dizajn mobilne aplikacije

U ovome poglavlju prikazat će se fotografije mobilne aplikacije za Maze Solver 1 projekt. Na prvoj slici prikazan je logo aplikacije. Logo aplikacije može se vidjeti prilikom pokretanja mobilne aplikacije.



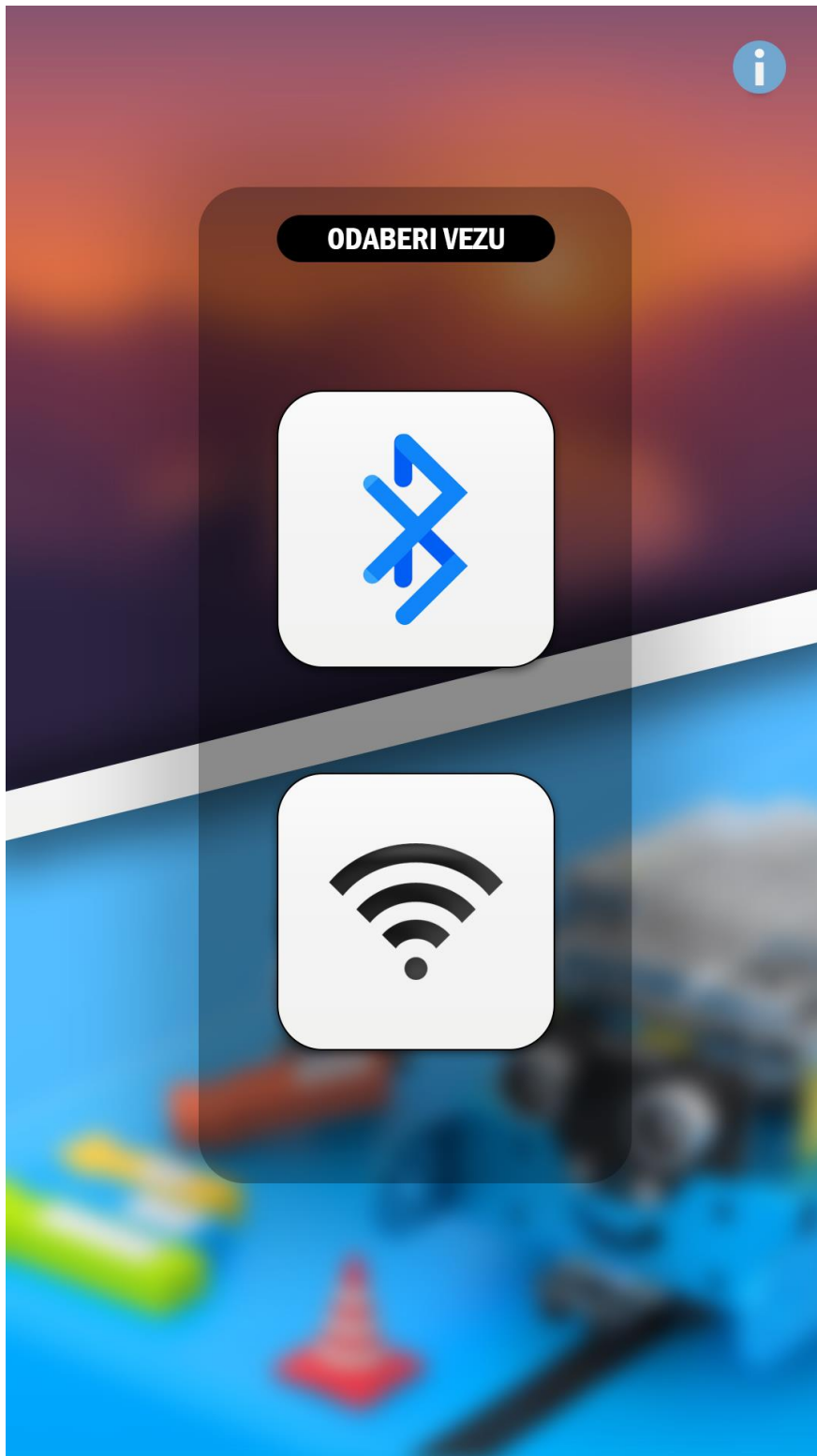
Slika 72. Logo aplikacije i zaslon otvaranja aplikacije

Na drugoj slici prikazan je zaslon na kojem korisnik može označiti senzore koje želi koristiti za pronalazak izlaza iz labirinta. Korisniku su ponuđeni sljedeći senzori: lijevi senzor, desni senzor te prednji senzor. Pritiskom na gumb „Nastavi“ korisniku se otvara zaslon upravljanja robotom, tj. početka prolaska kroz labirint.



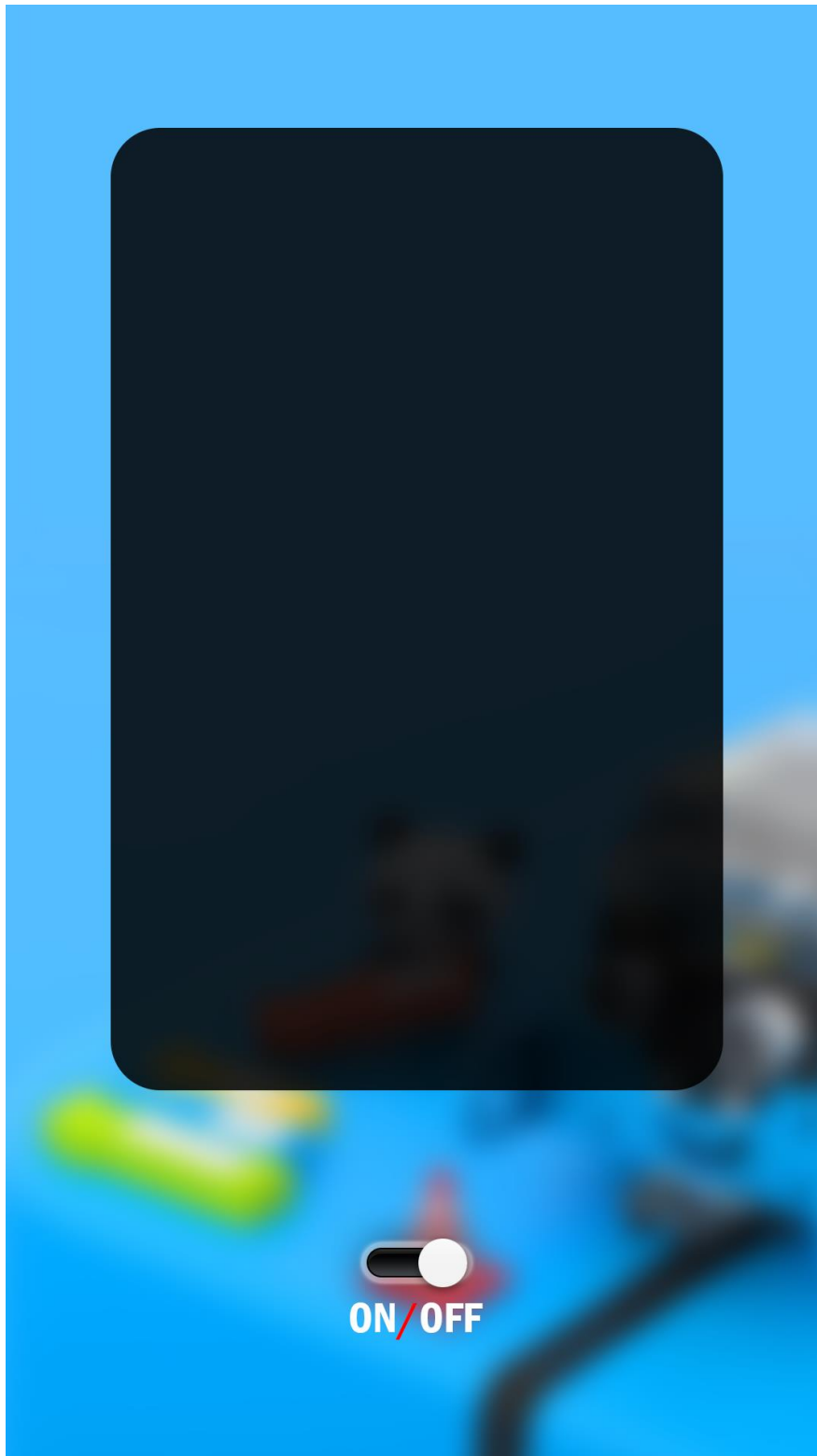
Slika 73. Zaslon odabira senzora

Na trećoj slici nalazi se zaslon koji se pojavi nakon pokretanja aplikacije. Na njemu je moguće odabrati želimo li se na mBota spojiti pomoću Bluetooth ili WiFi veze. Klikom na neki od gumba nas prebacuje na idući zaslon na kojem biramo na mBota na kojeg ćemo se spojiti.



Slika 74. Odabir načina veze

Na četvrtoj slici nalazi se zaslon na kojem će biti prikazan put mBota kroz labirint, tj. koje je kretnje napravio tokom svog prolaska kroz labirint.



Slika 75. Kretnje mBota kroz labirint

7.1. Kreiranje responzivnog dizajna

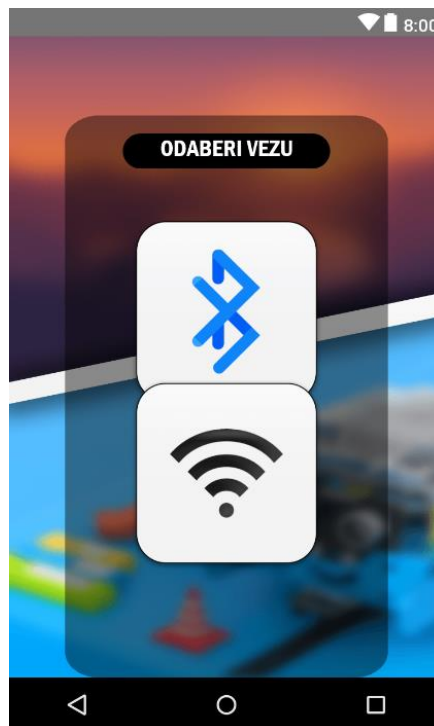
Responzivan dizajn danas je veoma bitna stavka bilo kakvog programskog rješenja kako bi raznim korisnicima na razno raznim uređajima sa različitim veličinama ekrana, sadržaj bio jasan i čitljiv, ali ujedno i jednak kao i svima drugima. Za kreiranje responzivnog dizajna nije potrebno razvijati zasebnu, novu mobilnu verziju, već se responzivnim tehnikama izgledi aktivnosti automatski prilagođavaju Vašem uređaju.

7.1.1. Maze Solver – Responzivan dizajn

Responzivan dizajn kreira se u nekoliko koraka, a to su :

- Postaviti željene veličine elemenata na zaslon koji želimo prikazati
- Postaviti elemente na određeno mjesto gdje ih želimo imati za sve buduće korisnike
- Postaviti fiksne granice (margine) i postaviti ovisnosti sa zaslonom „roditelj“
- Ako je to potrebno, dodatno prilagoditi
- Testirati na različitim veličinama ekrana

Za prikaz realiziranja kreiranja responzivnog dizajna, uzeti ću za primjer stranicu gdje se bira vrsta komunikacije sa robotom, bilo to Bluetooth ili WiFi. Aktivnost izgleda na sljedeći način :



Slika 76. Neispravan raspored elemenata na ekranu

Vidimo da na uređaju „Nexus 4“ koji je veličine 768x1280, tipke se preklapaju i cjelokupan dizajn ne izgleda baš najbolje. Potrebno je zatim prilagoditi izgled i slijediti prethodno navedene korake.

U prvome koraku ćemo u .XML datoteci aktivnosti, tipke WiFi i Bluetooth i njihove vrijednosti postaviti na fiksnu veličinu.

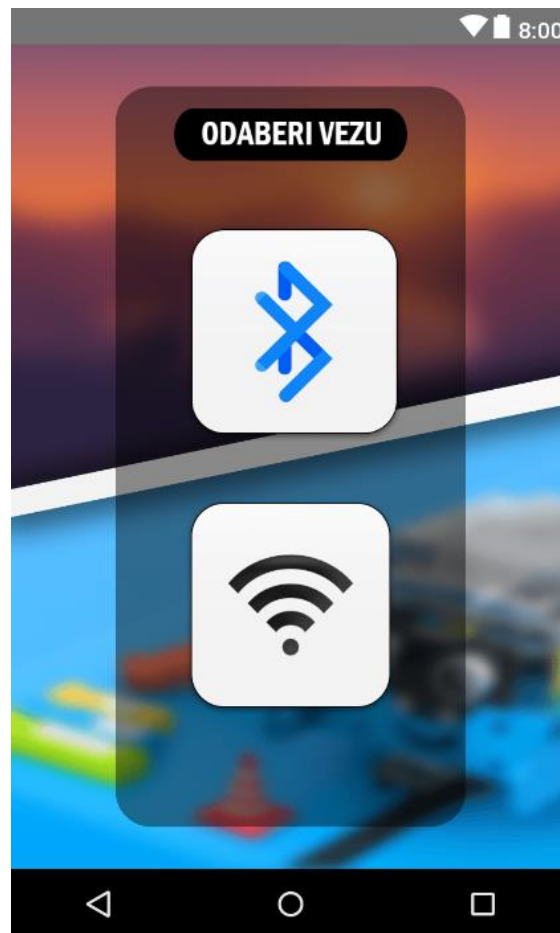
```
<ImageButton  
    android:id="@+id/btnWifi"  
    android:layout_width="150dp"  
    android:layout_height="150dp"
```

Slika 77. Postavljanje fiksnih veličina tipke WiFi

```
<ImageButton  
    android:id="@+id/btnBluetooth"  
    android:layout_width="150dp"  
    android:layout_height="150dp"
```

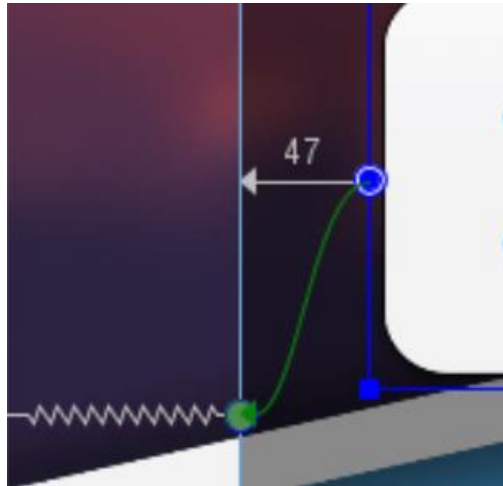
Slika 78. Postavljanje fiksnih veličina tipke Bluetooth

Zatim ćemo odabrane tipke premjestiti tako da odgovaraju trenutno odabranome zaslonu uređaja Nexus 4. To bi otprilike trebalo izgledati ovako :



Slika 79. Ispravan raspored elemenata na ekranu

Obje tipke tipke ćemo centrirati horizontalno kako bi se cijelo vrijeme nalazile u sredini, a to se realizira naredbom „*android:layout_centerHorizontal="true"*“. Dalje ćemo kreirati konstante udaljenosti od elementa tamne pozadine koja je ispod tipki kako bi u svim daljnjim povećanjima ili smanjenjima ekrana, tipke bile na istom položaju. To se realizira na sljedeći način :



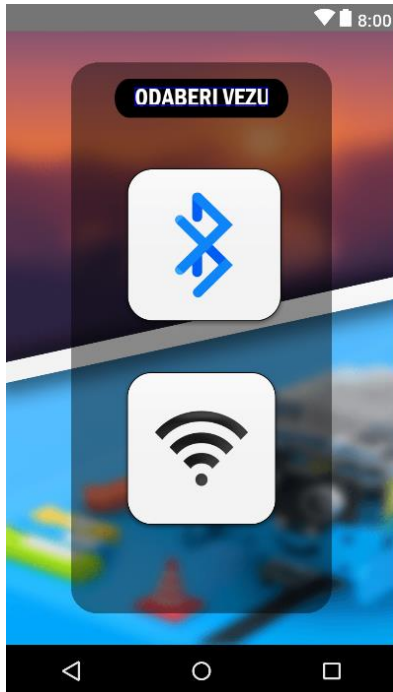
Slika 80. Postavljanje fiksnih udaljenosti od željenog elementa

Povuče se linija sa lijeve strane tipke i spoji se na točku tamne pozadine i tako za gornju i desno stranu. Zatim XML kod izgleda na sljedeći način :

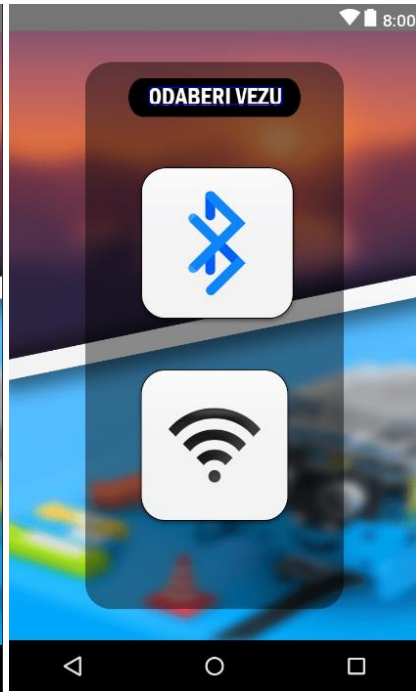
```
<ImageButton
    android:id="@+id/btnBluetooth"
    android:layout_width="150dp"
    android:layout_height="150dp"
    android:layout_alignStart="@+id/pozadinaTipkiOdabirVeze"
    android:layout_alignTop="@+id/pozadinaTipkiOdabirVeze"
    android:layout_alignEnd="@+id/pozadinaTipkiOdabirVeze"
    android:layout_marginStart="47dp"
    android:layout_marginTop="95dp"
    android:layout_marginEnd="42dp"
    android:layout_marginBottom="69dp"
    android:background="@drawable/ic_bluetoothikona">
</ImageButton>
<ImageButton
    android:id="@+id/btnWifi"
    android:layout_width="150dp"
    android:layout_height="150dp"
    android:layout_alignStart="@+id/pozadinaTipkiOdabirVeze"
    android:layout_alignEnd="@+id/pozadinaTipkiOdabirVeze"
    android:layout_alignBottom="@+id/pozadinaTipkiOdabirVeze"
    android:layout_marginStart="47dp"
    android:layout_marginEnd="47dp"
    android:layout_marginBottom="76dp"
    android:background="@drawable/ic_wifiikona">
</ImageButton>
```

Slika 81. Konačan XML kod za responzivan dizajn tipki

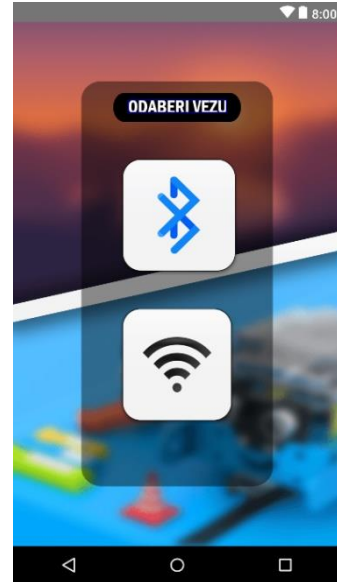
Na isti način, fiksirano sve preostale elemente na pozadini kao što su tamna pozadina iza tipki, naslov i svi preostali elementi koje želimo dodati na aktivnost. Izvršen responzivan dizajn izgleda na sljedeći način i prikazan je na uređajima sljedećim redoslijedom :



Slika 83. Nexus 5X



Slika 82. Nexus 5



Slika 84. Nexus 4

Popis literature

[1] *Meet Android Studio* (b. d.), Developers, Preuzeto 09.12.2018. s <https://developer.android.com/studio/intro/>

[2] *StackOverflow*, <https://stackoverflow.com>

Popis slika

Slika 1. Android Studio logo.....	2
Slika 2. Početni prozor Android Studia.....	4
Slika 3. Kreiranje novog Android projekta	5
Slika 4. Odabir Android uređaja	6
Slika 5. Dodavanje početne aktivnosti u projekt.....	7
Slika 6. Konfiguriranje aktivnosti	8
Slika 7. Kod klase loadingScreen.....	10
Slika 8. activity_loading_screen.xml	10
Slika 9. Svojtvo drawer i korištenje slušaća	12
Slika 10. onNavigationItemSelectedListener metoda	12
Slika 11. onBackPressed metoda	12
Slika 12. Metoda onCreateView u klasi SensorSelectionFragment.....	13
Slika 13: Dijagram aktivnosti fragmenta <i>ConnectionTypeSelectionFragment</i>	17
Slika 14. Dohvaćanje tipki preko reference u dizajneru	18
Slika 15. Slušatelji koji su postavljeni na tipke.....	18
Slika 16. Dohvaćanje datoteke u obliku objekta	19
Slika 17. Pozivanje sučelja Editor.....	19
Slika 18. Spremanje željenog niza u datoteku.....	19
Slika 19. <i>IWireless</i> sučelje	20
Slika 20. Metoda <i>enableDisable</i> – Bluetooth.....	21
Slika 21. Metoda <i>discover</i> – Bluetooth	21
Slika 22. Metoda <i>checkBTPermissions</i>	22
Slika 23. Inicijalizacija objekata <i>WifiManager</i> , <i>WifiP2pManager</i> i <i>WifiP2pChannel</i>	22
Slika 24. Metoda <i>enableDisable</i> - Wifi.....	22
Slika 25. Metoda <i>discover</i> - Wifi	23
Slika 26. Dohvaćanje lokalnog Bluetooth adaptera	23
Slika 27. Dohvaćanje Wifi managera.....	23
Slika 28. Provjera uključenosti Bluetootha	24
Slika 29. Provjera uključenosti Wifi - a	24
Slika 30. Metoda <i>openListOfDevices</i>	24
Slika 31. <i>BroadcastReceiver</i>	25
Slika 32. Slučaj kad je Bluetooth uključen	26
Slika 33. Provjera uključenosti Bluetootha	26
Slika 34. Slučaj kad je Bluetooth isključen.....	26

Slika 35: Dijagram aktivnosti fragmenta <i>ListOfDevicesFragment</i>	28
Slika 36. Dohvaćanja instance sučelja <i>IWireless</i>	29
Slika 37. Dohvaćanje instance sučelja <i>SharedPreferences</i>	29
Slika 38. Kreiranje adaptera za popunjavanje <i>ListView</i> -a.....	29
Slika 39. <i>BroadCastReceiver</i> - Bluetooth	31
Slika 40. <i>BroadCastReceiver</i> - Wifi.....	32
Slika 41. Slušač nad gumbom Discover.....	33
Slika 42. Metoda <i>clearList</i>	33
Slika 43. Metoda <i>initializeSocket</i>	34
Slika 44. Metoda <i>sendCommand</i>	35
Slika 45. Metoda <i>receive</i>	35
Slika 46. Handler metoda	37
Slika 47. Padajući izbornici za animacije.....	39
Slika 48. Novi Resource direktorij	40
Slika 49. <i>Enter_left_to_right.xml</i>	41
Slika 50. <i>Enter_right_to_left.xml</i>	41
Slika 51. <i>Exit_left_to_right.xml</i>	41
Slika 52. <i>Exit_right_to_left.xml</i>	42
Slika 53. <i>From_bottom.xml</i>	42
Slika 54. <i>From_top.xml</i>	43
Slika 55. Visual Studio logo.....	44
Slika 56. Kreiranje novog Arduino projekta	45
Slika 57. Odabir Arduino projekta	46
Slika 58. Inicijalni projekt u Arduino IDE	47
Slika 59. Uključivanje potrebnih biblioteka.....	48
Slika 60. Inicijaliziranje globalnih varijabli	48
Slika 61. Setup metoda Arduino IDE.....	49
Slika 62. Glava metode <i>Loop</i>	50
Slika 63. Metoda <i>CitajBluetooth()</i>	51
Slika 64. Metoda <i>IzvršiRadnjuBT()</i>	52
Slika 65. Metoda za kretanje robota unaprijed ili unazad	54
Slika 66. Metoda za promjenu smjera kretanja robota.....	54
Slika 67. Metoda za praćenje linije labirinta.....	54
Slika 71. Otvaranje projekta u Android Studiu preko GitHububa	66
Slika 72. Logo aplikacije i zaslon otvaranja aplikacije	67
Slika 73. Zaslon odabira senzora.....	68
Slika 74. Odabir načina veze.....	69

Slika 75. Kretanje mBota kroz labirint	70
Slika 76. Neispravan raspored elemenata na ekranu	71
Slika 77. Postavljanje fiksnih veličina tipke WiFi	72
Slika 78. Postavljanje fiksnih veličina tipke Bluetooth.....	72
Slika 79. Ispravan raspored elemenata na ekranu	72
Slika 80. Postavljanje fiksnih udaljenosti od željenog elementa	73
Slika 81. Konačan XML kod za responzivan dizajn tipki.....	73
Slika 82. Nexus 5	74
Slika 83. Nexus 5X.....	74
Slika 84. Nexus 4	74

Popis tablica

Tablica 1. Oblik para ključ-vrijednost.....	19
---	----