SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tim AIR1905

HerbertGame

PROJEKT IZ KOLEGIJA „ANALIZA I RAZVOJ PROGRAMA“

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Anđel
Emilio Grobenski
Filip Horvatić
Andreja Jurišić
David Koprek

Studij: ***Informacijsko i programsko inženjerstvo, Baze podataka i baze znanja***

HerbertGame

PROJEKT IZ KOLEGIJA „ANALIZA I RAZVOJ PROGRAMA“

Mentor/Mentorica:

Doc. dr. sc. Zlatko Stapić

Varaždin, studeni 2020.

Sadržaj

1. Uvod

U ovom radu opisana je projekta dokumentacija aplikacije HerbertGame u sklopu kolegija Analiza i razvoj programa. Veći dio ovog rada detaljno prikazuje funkcionalnosti mobilne igrice. Opisana je arhitektura sustava pomoću koje se lakše može razumjeti pravilno funkcioniranje svake zasebne komponente. Navedeni su svi opisi funkcionalnosti i najbitniji dijagrami koju su tu za pravilnu implementaciju.

Postoje tri glavne cjeline, to su Opis domene i specifikacije zahtjeva, Tehnologija i metodika programskog proizvoda te Specifikacije klasa. U prvom od navedenih poglavlja definirane su funkcionalnosti aplikacije, prikazane su skice predloženog wifreframe-a i dva dijagrama. ERA dijagram je namijenjen za implementaciju baze podataka s kojom aplikacija rukuje, a dijagram slučaja korištenja služi kao pomoć programerima pri implementaciji funkcionalnosti. Drugo poglavlje opisuje sve tehnologije i metodike rada korištene pri razvoju, a treće i najbitnije poglavlje opisuje sve implementirane klase, njihov odnos i korisnost.

2. Opis domene i specifikacija zahtjeva

Ideja mobilne aplikacije je izrada jednostavne igrice gdje je igračima zadatak riješiti male algoritamske zadatke pomoću jednostavnog jezika i animiranog robota koji obavlja te zadatke. Nakon uspješnog rješavanja prethodne zagonetke nastavlja se na sljedeću. Svaki level ima zadani mali labirint gdje je cilj pronaći najefikasniji put robota (Herberta). Igrač tijekom igre sakuplja bodove ovisno o uspješnosti rješenja, odnosno temeljem dužine algoritma i broju grešaka.

2.1. Definirane funkcionalnosti aplikacije

Popis funkcionalnosti za pravilni rad igrice:

F0 - Prijava u aplikaciju/Odjava (igračima prijava nije obvezna, ali im omogućuje sudjelovanje u onim aktivnostima za koje je nužna). Može biti Google/Facebook ili vlastita prijava.

F1 – Prikaz svih zadataka / skupina zadataka uz obvezno označavanje riješenih/neriješenih te ostvarenih rezultata.

F2 – Rješavanje zagonetke uz a) unos naredbi pomoću klasične tipkovnice, b) unos naredbi pomoću posebne Herbert tipkovnice.

F3 – Animacija kretanja Herberta po kreiranim uputama.

F4 – Preview kretanja robota tijekom samog programiranja.

F5 – Spremanje rješenja. Automatsko spremanje najboljeg rješenja (onog koje ostvaruje najviše podova).

F6 - Otvaranje riješenih zagonetki uz prikaz rješenja.

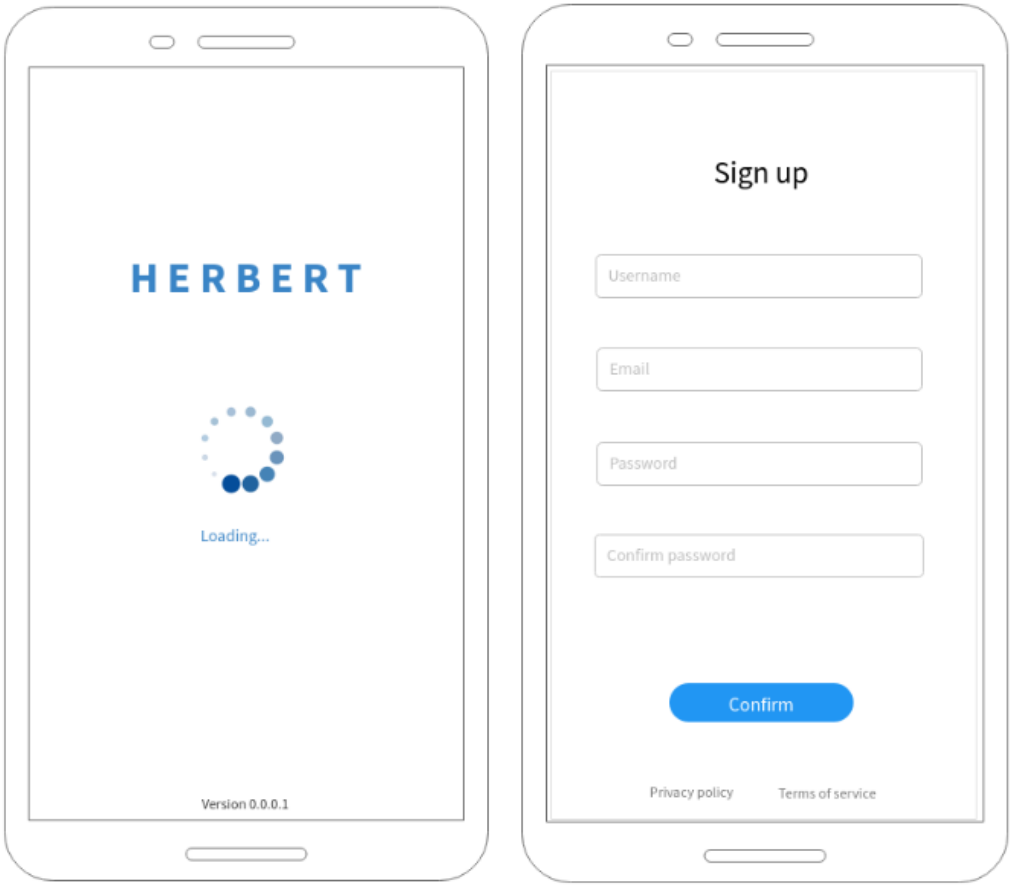
F7 – Integracija sa Google Games.

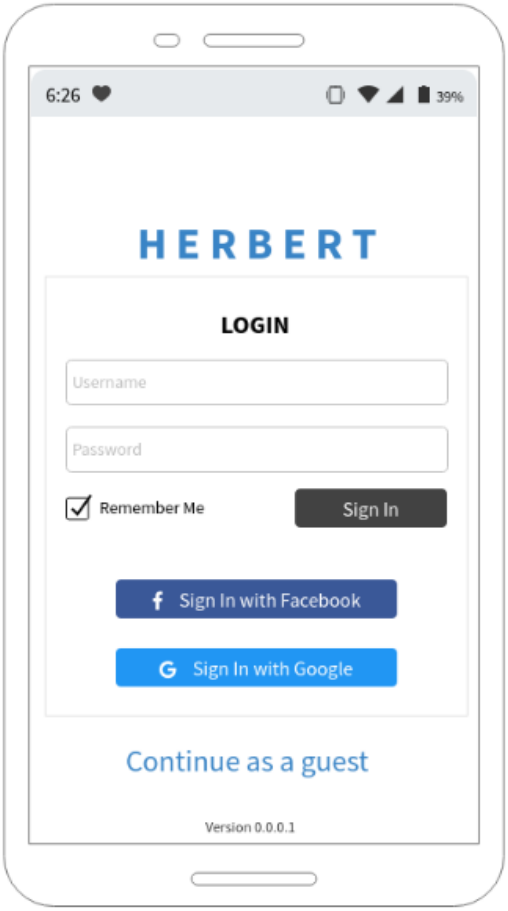
F8 – Tutorial za učenje igrača igranju igre / rješavanju zagonetki.

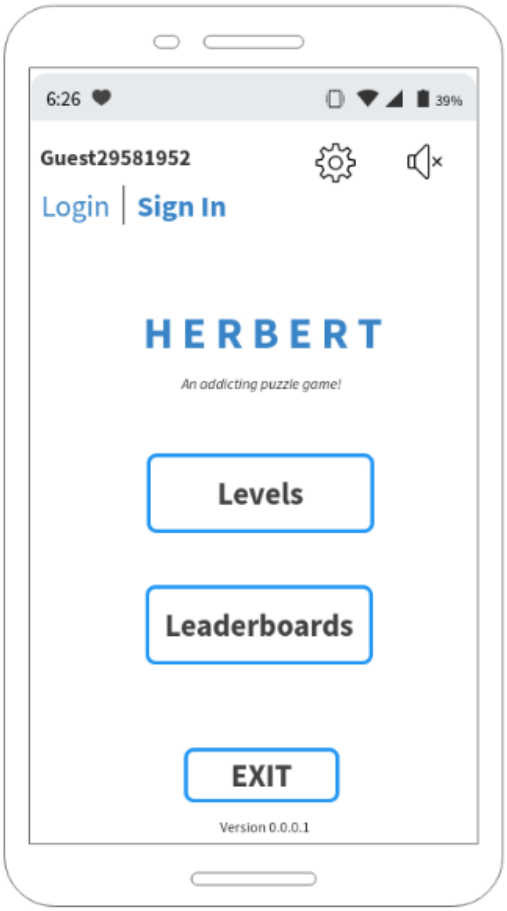
2.2. Wireframe

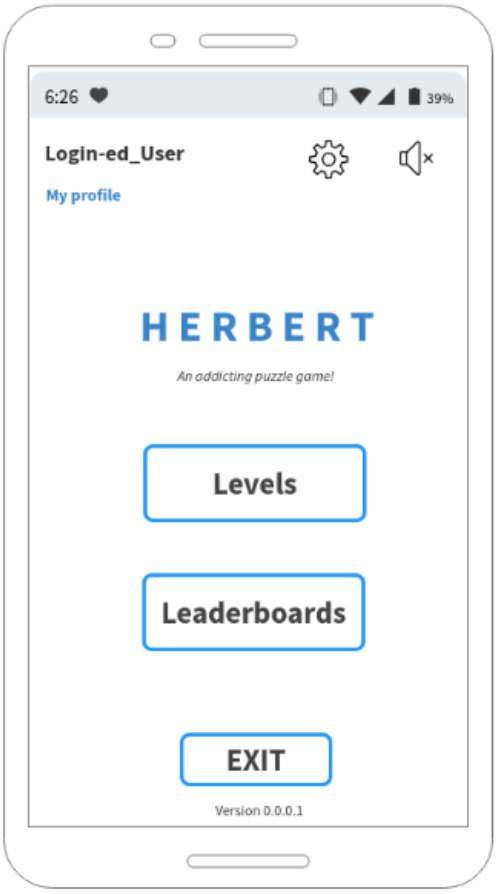
U cjelini su dane skice mogućeg wireframe-a osmišljenog na početku izrade samog projekta. Osmišljeni je app-view igrice s prijavom preko Facebook-a i Google-a te

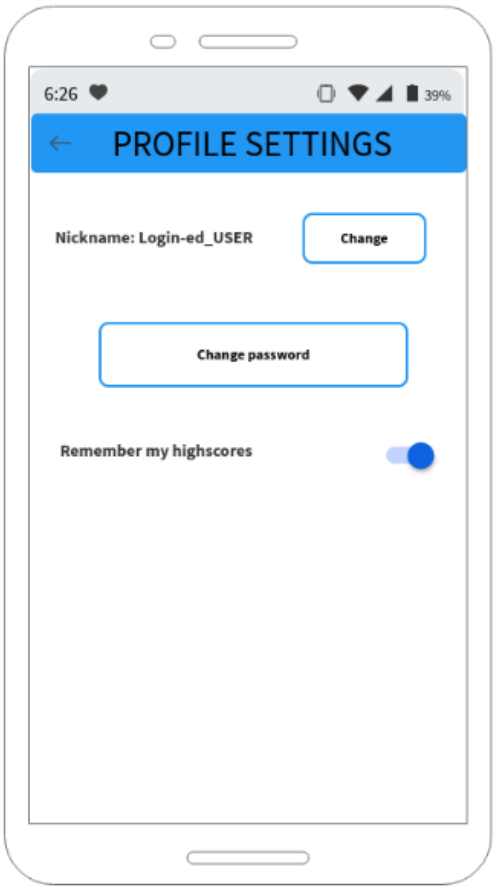
pripadajućom odjavom. Kasnije korisnik odabire level i počinje igrati igricu. Svaki level korisnik može riješiti pomoću Herbert tipkovnice ili unosom naredbi putem klasične tipkovnice. Igrač može vidjeti i ljestvicu rezultata i svoje najbolje moguće rješenje.

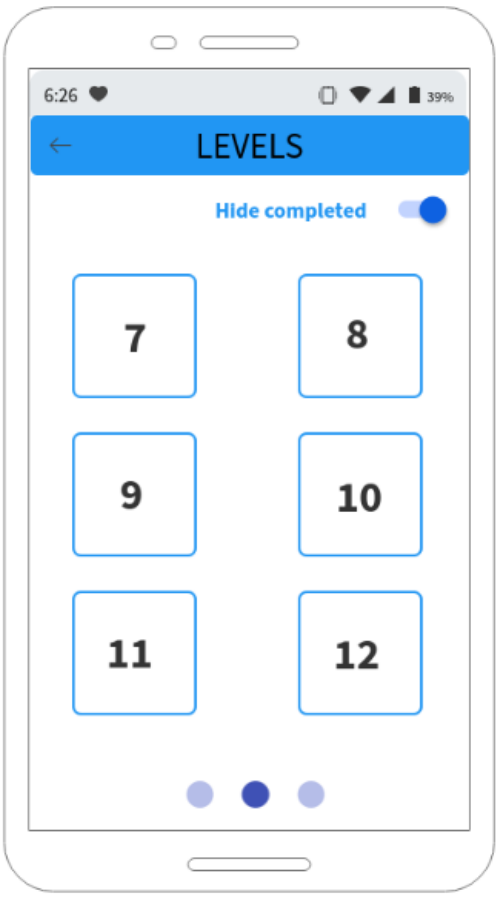


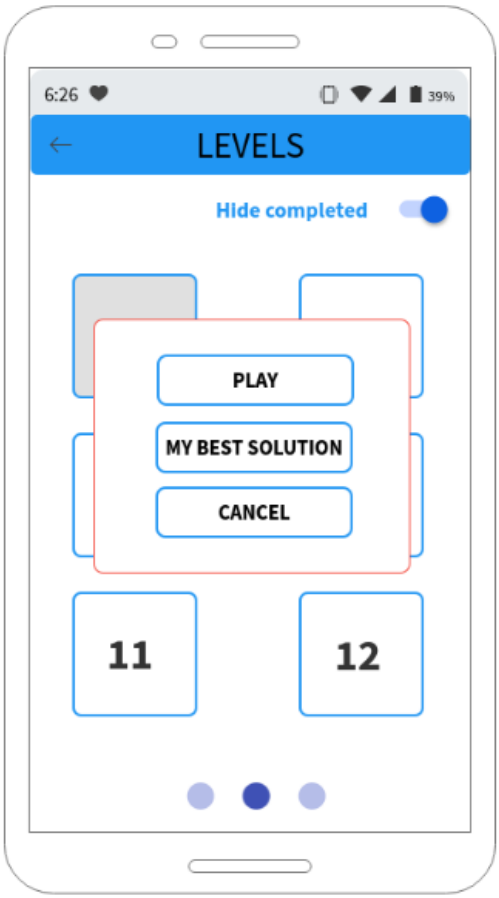

6:26
39%
HERBERT
LOGIN
Username
Password
Remember Me
Sign In
Sign In with Facebook
Sign In with Google
Continue as a guest
Version 0.0.0.1

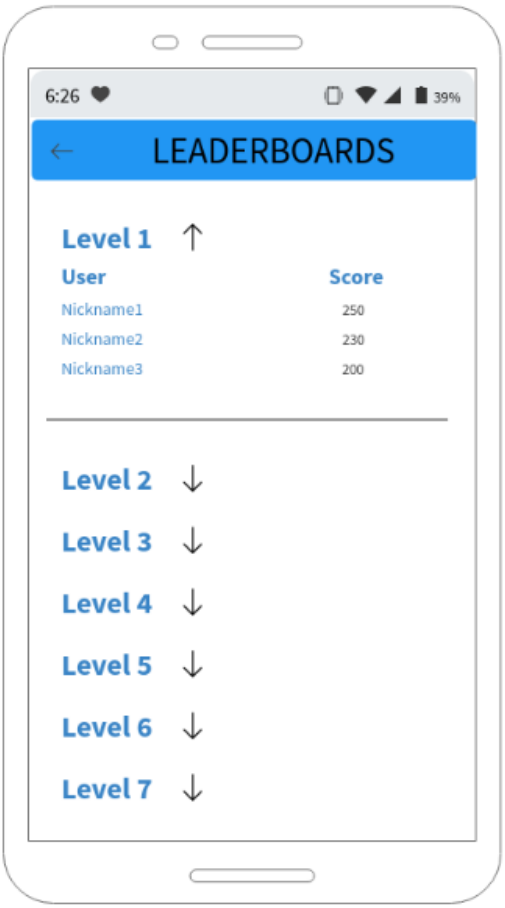

6:26
39%
Guest29581952
Login | Sign In
HERBERT
An addicting puzzle game!
Levels
Leaderboards
EXIT
Version 0.0.0.1

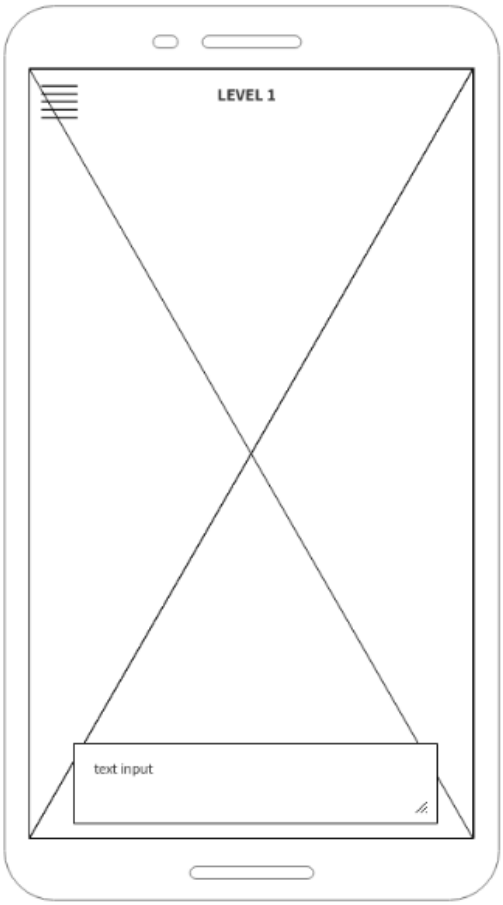

6:26
39%
Login-ed_User
My profile
HERBERT
An addicting puzzle game!
Levels
Leaderboards
EXIT
Version 0.0.0.1

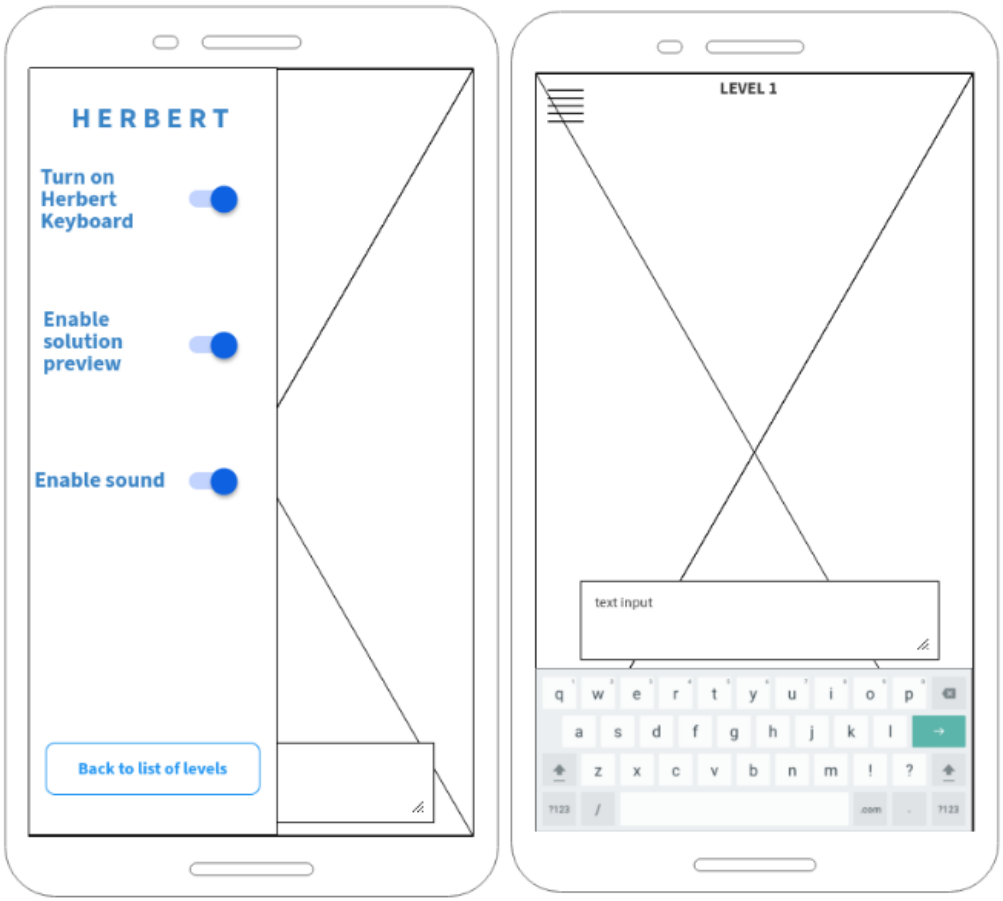

6:26
39%
PROFILE SETTINGS
Nickname: Login-ed_USER
Change
Change password
Remember my highscores


6:26
39%
LEVELS
Hide completed
7
8
9
10
11
12


6:26
39%
LEVELS
Hide completed
PLAY
MY BEST SOLUTION
CANCEL
11
12


6:26
39%
LEADERBOARDS
Level 1
User
Score
Nickname1
250
Nickname2
230
Nickname3
200
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7

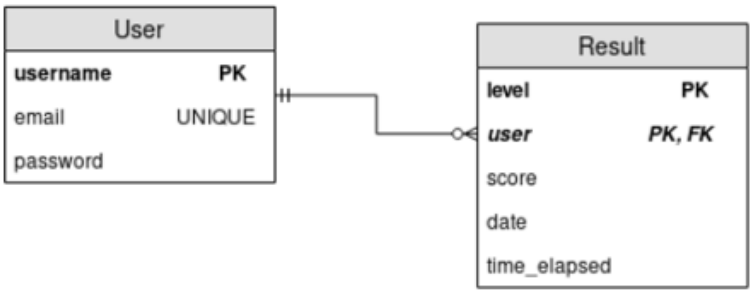

LEVEL 1
text input



2.3. ERA dijagram

ERA model nam služi za implementaciju vlastite prijave i registracije. Osim toga, model služi i za pamćenje najboljih rješenja korisnika za određen zadatak.

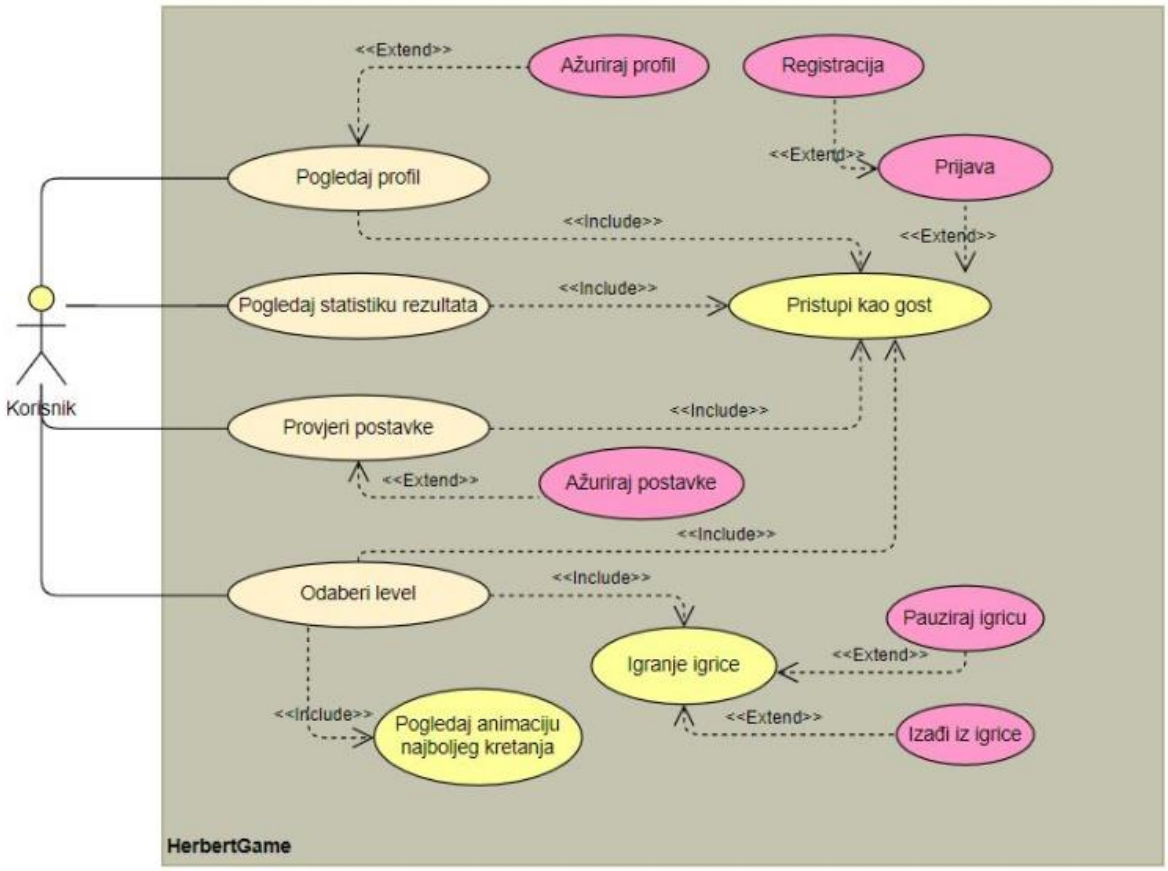
- `score` atribut predstavlja broj koraka koji je bio potreban za rješenje
- `time_elapsed` atribut predstavlja vrijeme koje je bilo potrebno za rješavanje zadatka
    - služi kao dodatno mjerilo za najbolje rješenje - igrači s jednakim brojem koraka razlikuju se po vremenu koje je bilo potrebno za rješavanje zadatka



Slika 1: ERA model

2.4. Dijagram slučaja korištenja

Dijagram slučaja korištenja za HerbertGame je vrsta dijagrama ponašanja koja prikazuje osnovne funkcionalnosti sustava (u ovom slučaju igrice) i njegovog odnosa s okolinom koju razumije korisnik (igrač). U donjem slučaju imamo primarnog učesnika s lijeve strane korisnika-igrača i još jednog sekundarnog učesnika s desne strane - Web Service označen stereotipom actor. Na početku interakcije s aplikacijom, korisnik ima na odabir Prijavu, Pristupanje kao gost, Provjeru postavki i Izlaz iz aplikacije. U prijavu je uključena i registracija vezom include ukoliko se korisnik prvi put prijavljuje. Pri prijavi i registraciji Web Service mora provjeravati podatke. Sljedeća moguća akcija korisnika je odabir da korisnik Pristupi kao gost. U tom slučaju nije potrebna prijava ili registracija ali se u sljedećim podakcijama moraju provjeravati podaci preko Web Service-a. Ako korisnik pristupa kao gost, ima ista prava kao i prijavljeni korisnik. Naredna moguća akcija korisnika je Pregledaj profil koji se proširuje na Ažuriraj profil, slijedi Pogledaj statistiku rezultata i Odaberi level. Sve tri akcije uključuju Provjeru podataka korisnika koju obavlja Web Service. Kod odabira levela, korisniku je uključena akcija Pogledaj animaciju najboljeg kretanja i Igranje igrice. Dostupan je pregled korisnikovog najboljeg rješenja. Akcije koje proširuju Igranje igrice su Pauziraj igricu i Izađi iz levela. Pri početnoj akciji Provjeri postavke omogućeno je opcionalno Ažuriranje postavki (misli se na osnovne Android-ove postavke).



Slika 2: Use case dijagram

3. Tehnologija i metodika razvoja programskog proizvoda

U ovom poglavlju navedene su sve tehnologije koje su korištene u izradi ovog projekta. Također, opisana je agilna metoda razvoja po kojoj smo radili, odnosno Scrum.

3.1. Tehnologija

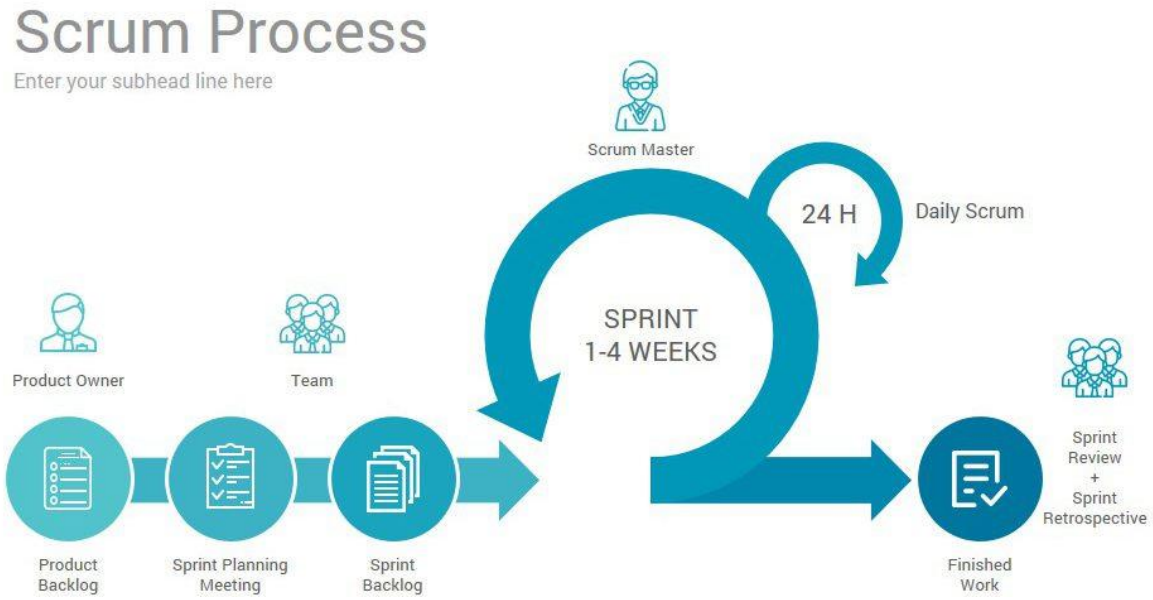
Za izradu projekta biti će korišteno nekoliko alata koji su nam dostupni u besplatnoj verziji. Neki od njih korišteni su online, a neki izravno preuzeti. To su alati: Android Studio 3.5.3, GitKraken, GitHub Desktop, FileZilla, Putty, Visual Studio Code, ZenHub, Visual Paradigm 15.2 Community Edition, draw.io. Sama aplikacija (igrica) izrađena je u Android Studiu 3.5.3. Za dohvaćanje i verzioniziranje koda koristima GitKraken i GitHub Desktop. Za komunikaciju sa serverom korišteni su FileZilla i Putty. Visual Studio Code služio je ispravljanju konflikata prilikom spajanja grana na GitHub-u. Za ERA dijagram i Use Case korišteni su alati draw.io online i Visual Paradigm. Wireframe je također crtan u online alatu - x.

3.2. Metodika

Metodikom koju slijedimo u razvoju našeg programskog proizvoda je agilni razvoj. Glavne karakteristike agilne metode jesu ispreplitanje procesa specifikacije, dizajna i implementacija, sustav se razvoja nizom verzija, a korisnici imaju veliku ulogu u izmjeni sljedeće verzije. Agilne metode žele postići jednostavnost, brzinu i učinkovitost te smanjiti gubitak vremena na dokumentaciju i nepotrebne procese. Postoje brojene metode zasnovane na agilnim principima i vrijednostima. Jedna od njih korištena je u razvoju ovog proizvoda, a to je Scrum.

Scrum je procesni okvir koji ima fokus u iterativnom razvoju. Temelj njihove metode je empirijski pristup i znanje iz iskustva, odnosno donošenje odluka na temelju onog što je znano. Scrum je zamišljen kao rad s malim i agilnim timovima (u našem slučaju pet osoba), od kojih je jedan Scrum master. Proces izrade podijeljen je na sprint-ove, vremenske okvire u kojem se razvija jedan inkrement iterativnog procesa (slika 3). U našem slučaju postoje tri sprinta u kojem su zadani pojedini zadaci (issues). Svaki sprint je okvirno isplaniran i pregledan, a odvijaju se i Scrum meetings radi održavanja pravilnosti, izjašnjavanja nejasnoća, dogovora i sl. Sprint-evi su definirani u gore navedenom alatu – ZenHub.

Prema Vodiču kroz Scrum (Listopad 2017.), Scrum razotkriva učinkovitost našeg načina upravljanja razvojem proizvoda i tehnikama rada tako da kontinuirno unapređujemo proizvod, tim i radno okruženje.



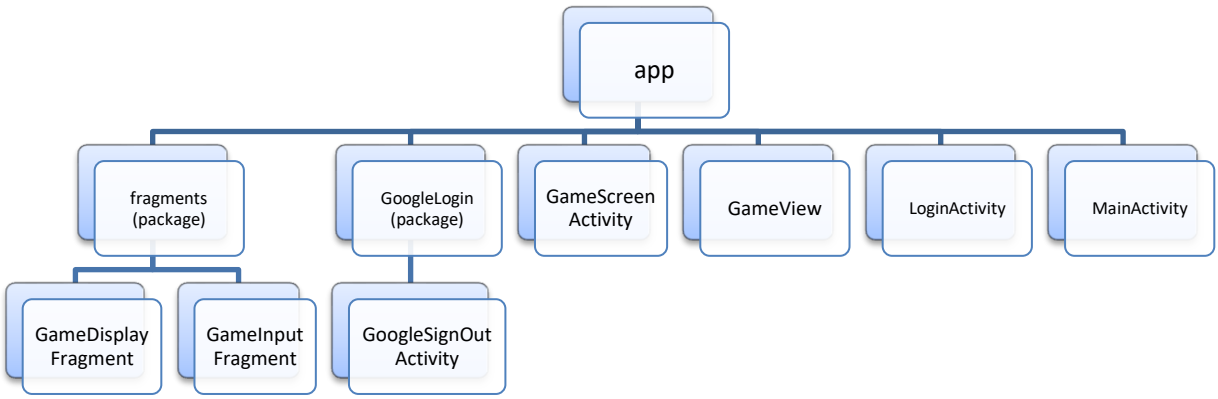
Slika 3: Scrum process

4. Specifikacije klasa

Aplikacija HerbertGame ostvarena je pomoću dva modula (app i engine). Modul app je osnovni modul. Sve klase objašnjene su kroz funkcionalnosti za koje su namijenjene. Hijerarhija klasa i paketa (package) prikazana je dijagramom za svaki modul.

U app nalaze se klase:

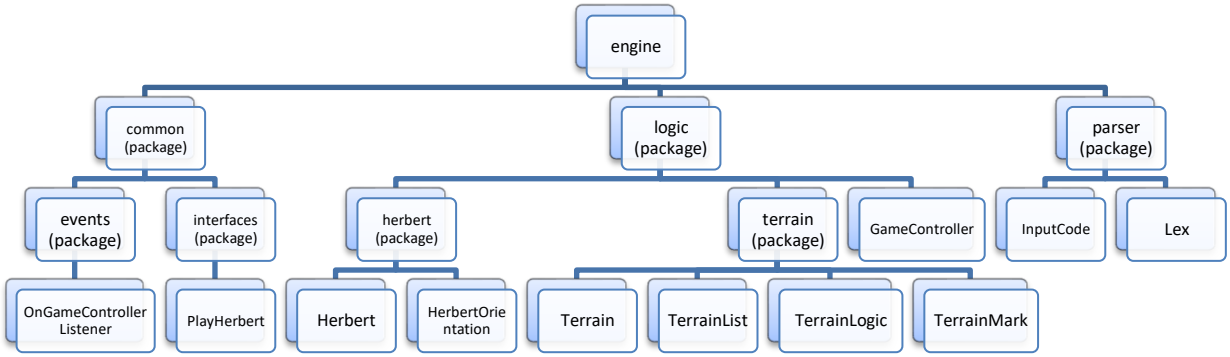
- GameDisplayFragment
- GameInputFragment
- GoogleSignOutActivity
- GameScreenActivity
- GameView
- LoginActivity
- MainActivity

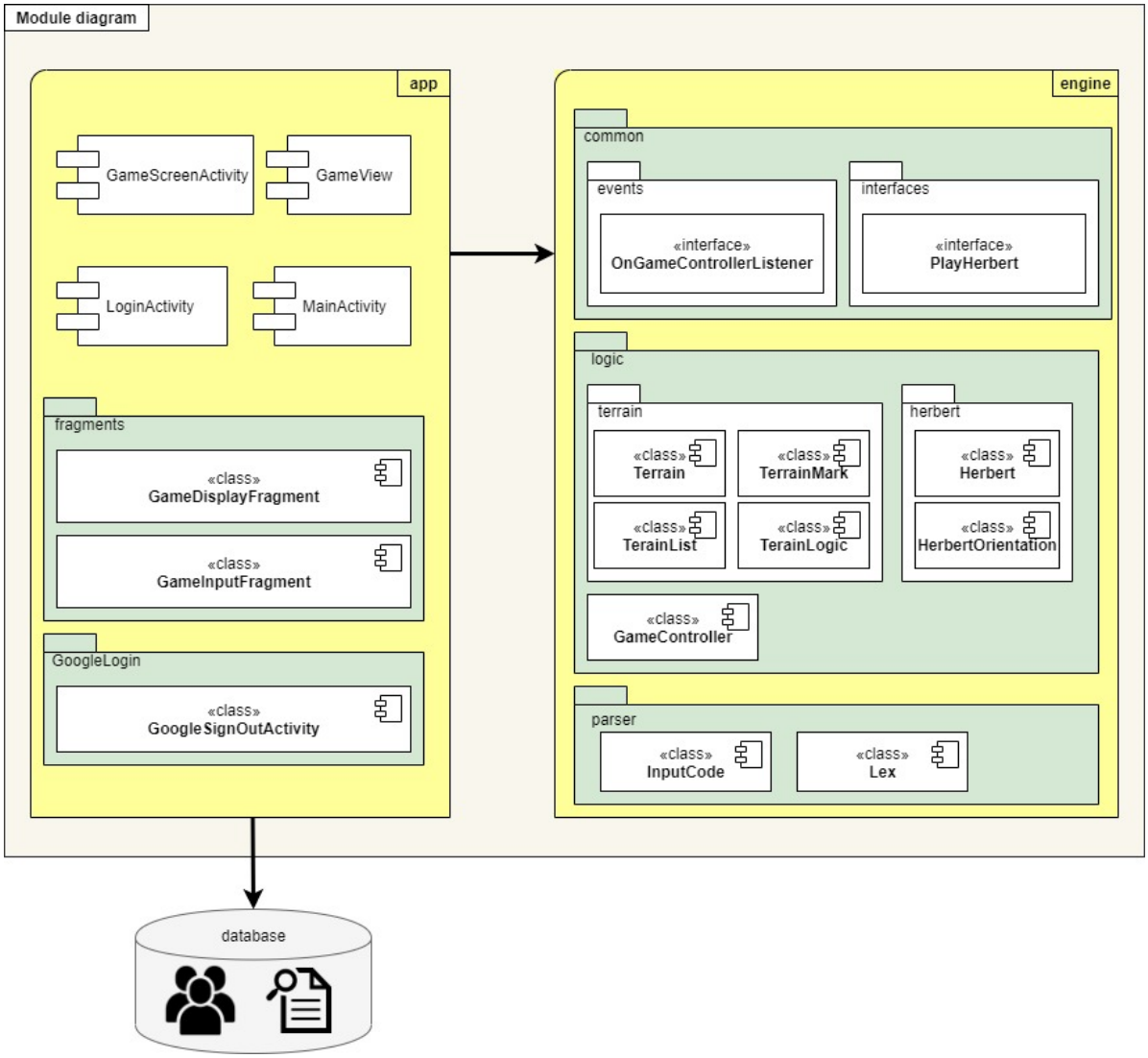


U engine nalaze se klase:

- OnGameControllerListener
- PlayHerbert
- Herbert
- HerbertOrientation
- Terrain
- TerrainList
- TerrainLogic

- TerrainMark
- GameController
- InputCode
- Lex





Slika 4: Module diagram

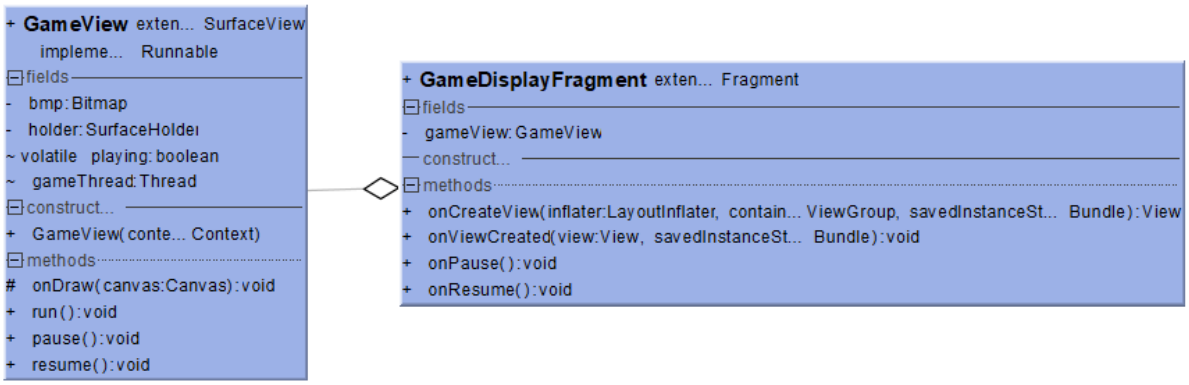
4.1. Modul app

Modul app je glavni modul u kojem se nalazi sedam klasa, gdje je četiri samostalnih klasa, a tri klase su podjeljene u dva paketa (package). U package-u *fragments* postoje dvije klase *GameDisplayFragment* i *GameInputFragment*.

Četiri zasebne klase redom su *GameScreenActivity*, *GameView*, *LoginActivity* i *MainActivity*.

4.1.1. GameDisplayFragment

Ideja klase je korištenje fragmenata pri prikazu sučelja igrice. Sadrži javnu *onCreateView* metodu koja stvara instancu objekta *GameView* u kojoj se prilikom kreiranja prosljeđuje kontekst.



Slika 5: Diagram klasa GameView i GameDisplayFragment

U metodi *onViewCreated* označeno je da sistemska klasa *Fragment* daje svojoj nadređenoj klasi ovlaštenje da izvrši metodu s istim nazivom. Postoje još dvije lifecycle metode *onPause* i *onResume* koje također daju ovlaštenja nadklasi.

4.1.2. GameInputFragment

Klasa namijenjena registriranju korisničkih ulaznih podataka i izvršavanje odgovarajućih operacija nad igricom.

4.1.3. GoogleSignOutActivity

Klasa *GoogleSignOutActivity* sadrži metode potrebne za odjavu korisnika prijavljenog preko Google-a. Klasa ima i svoj zasebni xml file gdje se otvara novi zaslon. Postoji lifecycle metoda *OnCreate* i zasebna privatna metoda *SignOut*.

U metodi *OnCreate* stvara se nova instanca(builda) za prikaz opcija prijave putem Google-a koja se konfigurira tako da prihvaća samo email (requestEmail). Dohvaćaju se korisnikovi teksni podaci preko *findViewById*. Prilikom klika na gumb *Odjava*, gumb reagira na metodu(listener) koja na klik gumba vrši odjavu korisnika prijavljenog putem Google-a u kojoj se nalazi metoda *signOut* koja vrši operaciju odjave. U slučaju da postoji prethodna prijava preko Google-a, postoje spremljeni podaci korisnika za sljedeću prijavu.

U metodi *SignOut* briše se račun povezan s aplikacijom, prikazuje se poruka uspješne odjave korisnika.

```
private void signOut() {
    mGoogleSignInClient.signOut()
            .addOnCompleteListener(this, new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    Toast.makeText(GoogleSignOutActivity.this,"Signed out
successfully!", Toast.LENGTH_SHORT).show();
                    finish();
                }
            });
}
```

4.1.4. GameScreenActivity

Na *GameScreenActivity* ima pristup povezanom *game_screen_activity.xml* zaslonu koji ima dvije komponente (FrameLayout). Gornja komponenta sa id-em *gameView* namijenjena je prikazu levela, odnosno igrice, a prikaz se vrši putem fragmenta (*GameDisplayFragment*) koji se u njega postavlja. Donja komponenta zaslona sa id-em *inputView* namijenjena je unosu korisnikovih naredbi (kratki algoritam). Putem fragmenta *GameInputFragment* vrši se upravljanje korisničkih unosa.

4.1.5. GameView

Klasa proširuje sistemsku *SurfaceView* klasu i implementira sistemsko *Runnable* sučelje. Pojedinosti *SurfaceView* klase su kontrolirane od strane *SurfaceHolder* apstraktnog sučelja. Postoji konstruktor *GameView* klase kojem je potreban parametar context, a taj context se prosljeđuje do nadklase (*SurfaceView*). *Runnable* sučelje posjeduje metodu *Run* koju klasa koristi.

Metoda *onDraw* s parametrom tipa *canvas* prosljeđuje dobiveni parametar nadređenoj klasi *View*. Metoda je namijenjena crtanju putanje.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
}
```

Metoda *run* definira 4 varijable tipa *float* koje se odnose na poziciju robota. Ako se u metodi uoči promjena kretnji robota, onda se pozicija mijenja na zaslonu (while petlja).

Definirane su dvije metode *pause*() i *resume*() za korisnikovu mogućnost pauziranja igrice. Postoji dretva *gameThread* koja je ovisno o operaciji pokrenuta ili zaustavljena.

4.1.6. LoginActivity

LoginActivity je klasa sa pripadajućom xml datotekom *login_activity.xml*. U klasi je implementirana funkcionalnost F0 – Prijava/odjava preko Facebook-a i Google-a. Prijava preko Facebook-a ostvarena je preko dvije lifecycle metode (*onCreate* i *OnStart*) i metode *loaduserProfile* (AccessToken newAccessToken).

```
private void loaduserProfile(AccessToken newAccessToken) {

    GraphRequest request = GraphRequest.newMeRequest(newAccessToken, new
GraphRequest.GraphJSONObjectCallback() {
        @Override
        public void onCompleted(JSONObject object, GraphResponse response)
{

            try {
                String first_name = object.getString("first_name");
                String last_name = object.getString("last_name");
                String email = object.getString("email");
                String id = object.getString("id");

                String image_url = "https://graph.facebook.com/" + id +
"/picture?type=normal";

                txtEmail.setText(email);
                txtName.setText(first_name + " " + last_name);
                RequestOptions requestOptions = new RequestOptions();

                requestOptions.dontAnimate();

Glide.with(LoginActivity.this).load(image_url).into(circleImageView);


            } catch (JSONException e) {
                e.printStackTrace();
            }

        }
    });

    Bundle parameters = new Bundle();
    parameters.putString("fields", "first_name,last_name,email,id");
    request.setParameters(parameters);
    request.executeAsync();

}
```

Za prijavu preko Google-a, u prvom je koraku potrebno konfigurirati Google prijavu da zahtjeva podatke koje korisnik unosi.

```
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestEmail()
        .build();
```

Također je potrebno stvoriti objekt s navedenim podacima. Sve to obavlja se u metodi onCreate.

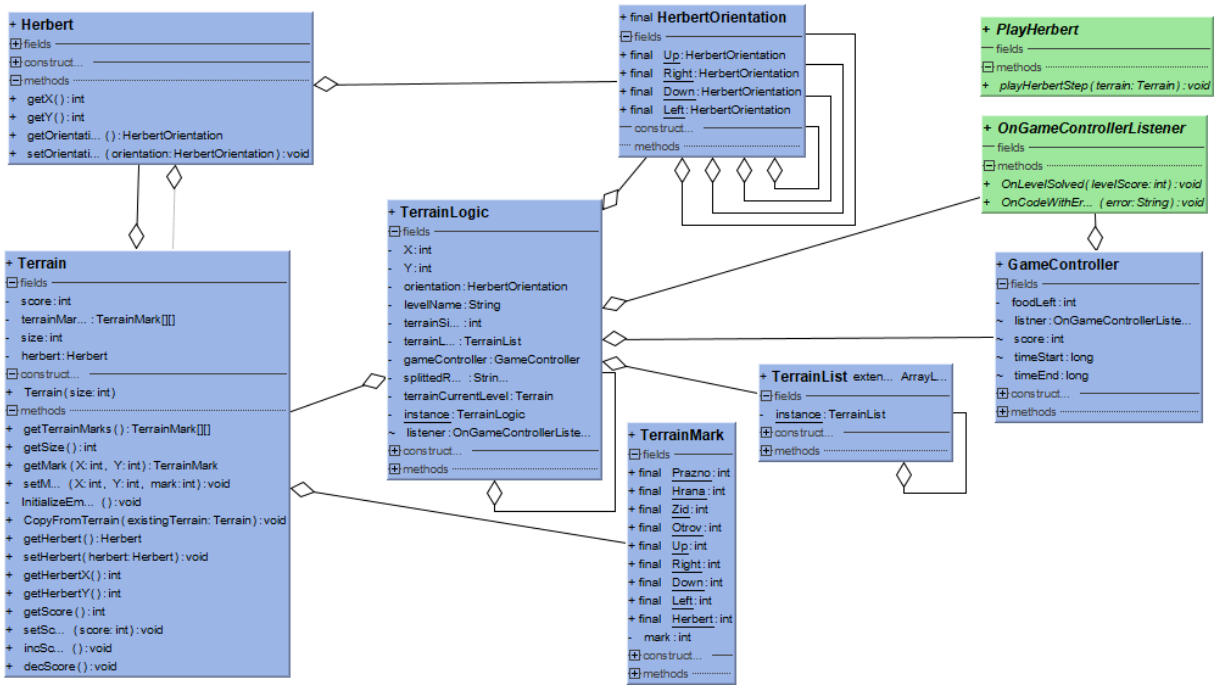
```
mGoogleSignInClient = GoogleSignIn.getClient(this, gso);
```

4.1.7. MainActivity

Glavna i početna aktivnost proširuje sistemsku klasu AppCompactActivity. Na glavnom screen-u postoje dvije nadjačane metode (override) iz sistemske klase koje su ujedno spadaju i u lifecycle metode aplikacije. To su onCreate() i onResume().

4.2. Modul engine

Cijeli engine modul implementiran je radi samog pokretanja igrice, odnosno kretanja robota Herberta koji je glavna srž aplikacije. Modul je podijeljen u tri package-a: *common*, *logic*, *parser*. *Common* package kod je raspoređen u svoja dva niža package-a, a to su events i interfaces.



Slika 6. Dijagram klasa engine modula (bez klasa Lex i InputCode)

4.2.1. OnGameControllerListener

Sučelje koje definira dvije metode *OnLevelSolved* sa parametrom tipa integer i metoda *OnCodeWithError* s parametrom tipa string. Klasa koja implementira to sučelje mora

sadržavati i te dvije metode s pripadajućim tijelom metode. *OnLevelSolved* bi trebala koristiti prilikom dovršavanja razine.

4.2.2. PlayHerbert

Sučelje sa metodom *playHerbertStep* s parametrom tipa terrain koje je objašnjeno u kasnijem potpoglavlju.

4.2.3. Herbert

Klasa predstavlja robota sa svojim konstruktorom. U klasi se nalazi njegova orijentacija kretnji na terenu. U ovoj klasi se ne kontrolira orijentacija nego se samo može dobiti ovisno gdje se nalazi na terenu.

```
public int getX() {
    return terrain.getHerbertX();
}

public int getY() {
    return terrain.getHerbertY();
}
```

4.2.4. HerbertOrientation

Klasa je enumeracijska sa četiri moguća kretanja - konstante (Up, Down, Left, Right).

4.2.5.Terrain

Klasa pokazuje stanje terena u svakom potezu za vrijeme kada se robot pomiče. Sadrži privatne varijable score, size, herbert i terrainMarks. Score predstavlja broj bodova prilikom svakog robotovog koraka, a size je jednak širini i visini terena. Herbert predstavlja robota na terenu koji se kreće. Ta varijabla je u interakciji s terain-om. TerainMarks je kvadratna matrica koja se postavlja na veličinu size (gore zadano).

Prva metoda je konstruktor koji prima argument size i postavlja matricu na tu veličinu i score na nulu. U konstruktoru se još stvara nova instanca Herbert-a i stavalja se u gornju privatnu varijablu.

Slijede dvije metode getHerbert() i setHerbert() koje služe za pristup gore navedenoj varijebli herbert dok naredna metoda getSize() tipa integer pristupa varijabli size,

Metoda setMark() postavlja točku na mogućoj površini terena ovisno o vrijednosti X i Y koordinate, a getMark() daje kvadratnu matricu TerrainMark s tim poljem.

InitializeEmpty() metoda ima zadatak označavanja prazne površine. Stvara kvadratnu matricu bez označenih polja. CopyFromTerrain metoda s parametrom existingTerrain kopira postojeće stanje terena. Puni varijablu *score* i matricu *terrainMark* s prijašnjom verzijom Terrain-a.

U klasi se još nalaze dvije metode koje dohvaćaju vrijednost integer-a pozicije robota prema X i Y koordinati na terenu. To se traži unutar kvadratne matrice TerrainMark (dupla for petlja).

```
public void setScore(int score) {
    this.score = score;
}
public void incScore(){
    this.score+=100;
}
public void decScore(){
    this.score-=1;
}
```

Gore navedene zadnje tri metode govore same za sebe. Postavlja veličinu, povećava za 100 ili oduzima za jedan.

4.2.6. TerrainList

Klasa nasljeđuje ArrayList<> u kojoj prihvaća objekt tipa Terrain, odnosno TerrainList predstavlja listu objekata tipa Terrain. U klasi je deklarirana jedna privatna statička varijabla koja drži referencu na objekt TerrainList. Slijedi konstruktor koji vraća instancu klase. TerrainList je zamišljen kao singleton klasa, zbog toga postoji Metoda *getInstance*() koja vraća novu instancu ako postoji, odnosno kreira ako ne postoji.

Iz proširene klase, nadjačana je add metoda s parametrom tipa Terrain. Metoda dodaje terrain u superklasu. Slijedi metoda getTerrain() koja vraća objekt terrain na nekom indeksu.

Metoda playHerbertStepByStep() je metoda koja prolazi kroz listu svih terena u gore zadanoj listi i putem sučelja PlayHerbert reproducira korak po korak na terenu.

Zadnja metoda getSize() je tipa integer i vraća veličinu liste.

4.2.7. TerrainLogic

TerrainLogic klasa je najbitnija klasa u unutarnjoj logici kretanja robota na terenu. Deklarirane su privatne varijable od kojih su tri tipa integer (X, Y i terrainSize), slijedi orijentacija tipa HerbertOrientation, varijabla levelName koja je tipa string, terrainCurrentLevel tipa Terrain i gameControler istog tipa. Deklarirano je i polje stringova splittedRow i terrainList koji sadrži svoju instancu.

```
public int getX() {
    return X;
}
public void setX(int x) {
    X = x;
}
public int getY() {
    return Y;
}
public void setY(int y) {
    Y = y;
}
```

Gornje četiri metode služe za postavljanje i dohvaćanje X i Y koordinata. Slijedi emptyTerrainList() metoda koja prazni listu terena. Nakon nje je konstruktor sa parametrom listener (sučelje OnGameControllerListener). Konstruktor postavlja početnu orijentaciju herberta i stvara novu instancu GameController.

TerrainLogic koristi singleton uzorak za upravljanje stvorenog objekta (vidljivo u metodu getInstance()). Jedna od istaknutijih metoda tipa string je parserUserCode sa ulaznim parametrom tipa string. Metoda je zadužena za čitanje korisnikovog unosa i šalje ga u metodu getMoves() koja se nalazi u klasi InputCode. Naredna metoda parseCodeToTerrains s ulaznim parametrom tipa string iz rezultata parsiranog unosa, to se prosljeđuje u listu znakova i provjerava da li znak „s“ pravilan. Znak se prosljeđuje u metodu parseSingleMove() gdje se teren uspoređuje s prijašnjim stanjem za pronalazak lokacije herberta. Znak se uspoređuje sa slovom i u tri slučaja postoje daljnji postupci. Znak „s“ pomiče herberta na temelju koordinata smjera, „l“ ka okreće lijevo i „r“ desno. Metoda move() objašnjava način na koji se herbert rotira i pokreće. Metoda također vidljivo objašnjava da li herbert pronađe hranu, otrov ili naleti na zid. Ukoliko pronađe hranu, score se povećava.

Dvije metoda za rotaciju rotateRight() i rotateLeft() služe za promjenu orijentacije gdje uzimaju konstante iz klase HerbertOrientation (Right, Down, Left, Up).

Metoda loadCurrentLevelTerrain s ulaznim parametrom context stvara trenutni Terrain level tako da čita opis tog levela iz zasebne .txt datoteke.

4.2.8. TerrainMark

U klasi su većinom definirane statične konstantne tipa integer za Hranu, Zid, Otrov, te konstante za kretanje (Up, Right, Down, Left i Herbert). Definirana su dva kontruktora gdje jedan ima ulazni parametar tipa int i vraća jedno puno polje u rasteru terena, a drugi kontruktor vraća prazno polje.

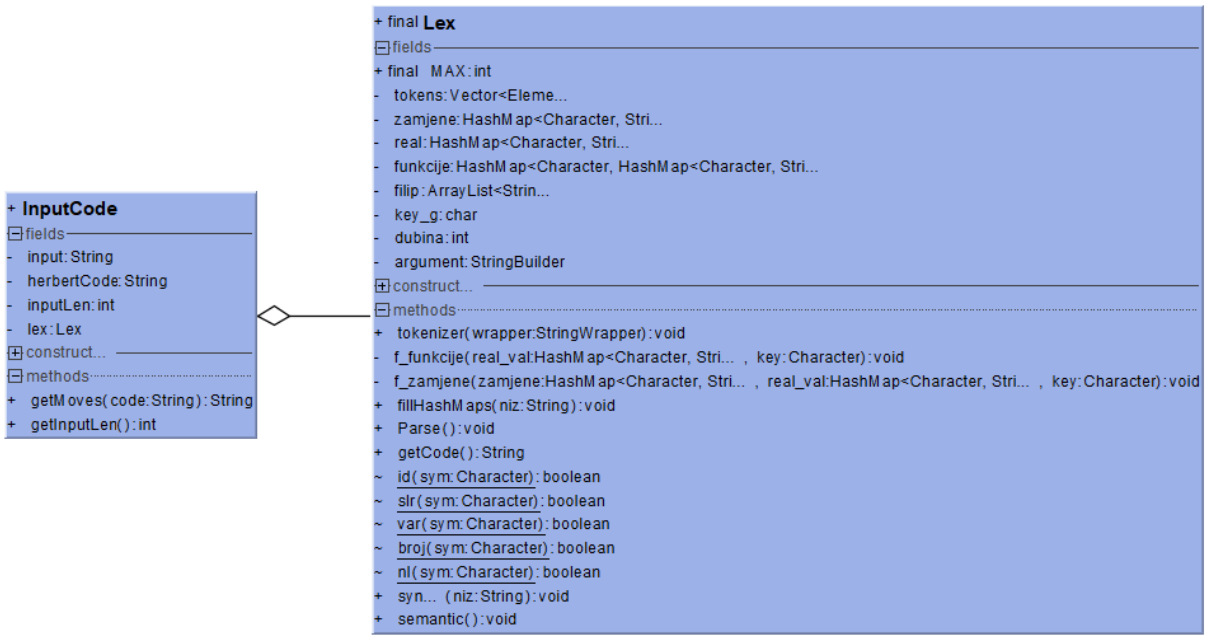
4.2.9. GameController

Klasa GameController je glavna klasa za praćenje stanja igre. Funkcija countFood() sadrži sumu sve hrane koja se zbraja u konačne bodove. Privatne metode startGameTimer() i endGameTimer() uzimaju sistemske metode za mjerenje i zaustavljanje vremena.

4.2.10. InputCode

Definirana je ugnježđena klasa StringWrapper s jednom varijablom string i tri metode. Kontruktor vraća string, metoda getString() tpa string vraća taj string, a setString() ga postavlja. Kod konstruktora se ta vrijednost postavlja samo jednom.

Kao što vidimo na grafu dolje postoje dvije metode i jedan konstruktor. U metodi *getMoves*() tipa string, sređuje korisnikove ulazne vrijednosti i stvara objekt StringWraper s kojim se upravlja u Lex klasi. Na grafu se vidi agregacija Lex klase.



4.2.11. Lex

Definirana je enumeracija Token za označavanje znakova korisničkog unosa (algoritma) i enumeracija za status grešaka. U klasi postoje razne ugnježđene klase: Element, Semantex, HerbertException, LexerException i Syntaxer.

Klasa Lex sadrži svoj konstruktor koji ne prima nikakav argument i služi za stvaranje raznih novih instanci, ponajviše lista i hash mapa koje su potrebne za rad.

Klasa Element zapravo predstavlja neki znak na nekoj poziciji iz korisničkog unosa, gdje je znak predstavljam elementom iz enumeracije Token. Klasa sadrži jednu metodu, odnosno konstruktor pomoću kojeg se gradi jedan objekt tipa Element.

Metoda tokenizer() za argument prima objekt tipa StringWrapper, odnosno prima korisnički unos, a metoda se brine o tome da svaki znak iz unosa interpretira tako da stvara objekte tipa Element te ih unosi u listu pod imenom *tokens*. Lista *tokens* je tipa Vector<Element>.

Klasa *HerbertException* nasljeđuje klasu *Exception* te omogućuje prilagođeno prepoznavanje greške kod robota i prikaz iste ako dođe do takvo nečeg. Klasa *LexerException* isto nasljeđuje *Exception* te služi za prilagođeno stvaranje greške kod *Lex* klase.

Lex klasa sadrži dvije metode pod imenom *f_funkcije()* i *f_zamjene()*. Prva metoda provjerava ulazne korisničke znakove, odnosno o kojem se znak radi te ih stavlja u pripadajuće hash mape. Ako dođe do greške kod metode onda ona stvara grešku tipa *HerbertException.* Metoda *f_funkcije()* prepozna i znakove koje je potrebno zamijeniti s ispravnim znakom, tada se ti svi znakovi prosljeđuju u novu metodu pod imenom *f_zamjene()* gdje se vrši zamjena tih znakova.

Iduća metoda je *fillHashMaps()* koja kao argument prima string zapis koji pretvara u hash map, odnosno puni hash mape.

Klasa *Semantex* je ugniježđena klasa koja se brine jeli korisnički unos argumenata ispravnog formata, odnosno strukture. Za takvo nešto brinu se tri metode pod imenom *semnatic()*, *checkzamjene()* i *check_funkcije()*.

Metoda *Parse()* služi za pokretanje metode *f_zamjene()* te u slučaju greške stvara grešku tipa *HerbertException().*

Metoda *getCode()* vraća iz liste filip prvi element, lista je tipa *ArrayList<String>* .

Slijede četiri metode pod imenom *id(), slr(), var()* i *nl().* Prva metoda provjerava dali je argument koji je tipa string u obliku malih slova. Druga metoda provjerava dali je argument u obliku velikih slova. Treća metoda provjerava jeli argument tipa broj, a četvrta metoda provjerava jeli se radi o kraju zapisa.

Zadnje dvije metode *syntax()* i *semantic()* pokreću pripadajuće metode unutar ugniježđene klase *Syntaxer*, ako je došlo do greške unutar tih metoda onda se stvara greška tipa *HerbertExeception. Syntaxer* klasa provjerava i bilježi prevedeni korisnički unos koji se nalazi u listi *tokens* koja je tipa *Vector<Elements>.*

Popis literature

[1] Schwaber K, Sutherland J. (Listopad 2017.), Vodič kroz Scrum, Sveobuhvatni vodič kroz Scrum: Pravila igre, dostupno 19.12.2019. na https://tododoingdone.com/vodic-kroz-scrum/