Mobilna Android aplikacija za poboljšavanje korisničkog iskustva u igranju strateške društvene igre

Prpić, Jakov

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MOBILNA ANDROID APLIKACIJA ZA POBOLJŠAVANJE KORISNIČKOG ISKUSTVA U IGRANJU STRATEŠKE DRUŠTVENE IGRE

Završni rad

Jakov Prpić

Osijek, 2020.

SADRŽAJ

1. UVOD

Društvene igre često imaju fizičke komponente poput figurica, kockica i slično. Zamjena tih komponenti u mobilnoj aplikaciji te mogućnost igranja zahtijeva dobro razrađeno korisničko iskustvo.

Cilj ovog završnog rada je razvoj Android mobilne aplikacije koja će biti pomagalo pri igranju društvene igre *Dungeons and Dragons*. Navedena igra zahtijeva početno kreiranje lika za što je potrebna velika količina informacija. Kako bi se novi igrači upoznali s igrom, napisana je knjiga *Player's Handbook*. Razvijana aplikacije ima za cilj smanjiti potrebno vrijeme za kreiranje lika za igru bez gubitka informacija te korisniku ponuditi osnovne funkcije interakcije s kreiranim likom. S obzirom da se mobilna aplikacija razvija u svrhu poboljšanja korisničkog iskustva tijekom igranja društvene igre, u radu se proučava što sve utječe na korisničko iskustvo tijekom korištenja bilo koje aplikacije, a zatim i što utječe na korisničko iskustvo u aplikaciji razvijenoj za pomoć pri igranju društvene strateške igre. Za razvoj aplikacije, koristi se razvojna okolina *Android Studio* te programski jezik *Java*. Uz razvoj funkcionalnosti aplikacije, u *Android Studiju* dizajnirano je i korisničko sučelje jezikom *XML*. Kako bi se podaci mogli spremati za buduće korištenje, koristi se baza podataka i jezik *SQLite*. Način komuniciranja Android aplikacije i baze podataka ostvaren je implementacijom biblioteke *Room.*

Poglavlje 2 daje teorijsku podlogu potrebnu za poboljšanje korisničkog iskustva u igranju društvenih igara, a razmatrane su i trenutno razvijene aplikacije slične svrhe, kao i stanje u području. Poglavlje 3 sadrži zahtjeve na razvijenu aplikaciju, te model i arhitekturu njenog rješenja. U poglavlju 4 prikazano je i objašnjeno programsko rješenje, dok je u poglavlju 5 analiziran način rada aplikacije i ostvarena poboljšanja korisničkog iskustva tijekom igranja društvene igre.

1.1. Zadatak završnog rada

U završnom radu potrebno je analizirati i opisati zahtjeve i izazove pri igranju strateških društvenih računalnih igara s posebnim naglaskom na stolnu igru Dungeons and Dragons. Nadalje, potrebno je osmisliti načine poboljšavanja korisničkog iskustva u igranju omogućavanjem profila s likovima igraču, razvojem modela praćenja tijeka sredstava koja su na raspolaganju igraču, te modela usklađivanja njegove strategije igranja s dostupnim resursima u napadu i obrani. Za sve navedene zahtjeve, potrebno je osmisliti prikladnu bazu podataka, te opisati potrebne programske tehnologije i arhitekturu programskog rješenja. U programskom dijelu rada potrebno je ostvariti

odgovarajuće korisničko sučelje aplikacije, ostvariti bazu podataka, te implementirati navedene funkcionalnosti opisane Android mobilne aplikacije. Ostvareno programsko rješenje treba ispitati za prikladne uvjete korištenja i analizirati njegov utjecaj na korisničko iskustvo igrača.

2. PREGLED IZAZOVA VEZANIH ZA POBOLJŠAVANJE KORISNIČKOG ISKUSTVA U STRATEŠKIMA IGRAMA

Korisničko iskustvo je iznimno bitno područje aplikacije za razviti. Ono je direktno povezano sa samim uspjehom aplikacije na tržištu stoga je potrebno osigurati određenu kvalitetu. Kako bi se ostvarila kvaliteta, proučeni su obrasci i uvjeti na koje se treba paziti tijekom razvoja bilo kakve aplikacije te kakav je pristup najbolji za razvoj aplikacije za društvenu stratešku igru.

2.1. Korisničko iskustvo kod mobilnih aplikacija

Prema [1], brzina rasprostranjivanja pametnih telefona veća je od bilo koje druge tehnologije do sada. Veća je 10 puta od brzine širenja osobnih računala te duplo brža od proširenja interneta. Već lipnja 2012. godine *Apple App Store* je sadržavao 650,000 aplikacija te je na *Google Play*-u je broj aplikacija 2013. prešao preko 1,000,000. Kako bi aplikacija imala konkurentnost na tržištu treba ciljati prema korisničkoj angažiranosti i „ljepljivosti“. U [1] se također navodi da statistike *Flurry*-a, aplikacije za statistike iPhone aplikacija, ukazuju na to da besplatna aplikacija izgubi 95% svojih korisnika nakon 30 dana. Angažiranost se mjeri kao količina vremena provedenog u aplikaciji po mjesecu, a „ljepljivost“ kao mjera odanosti korisnika prema korištenju aplikacije. Kako bi se korisnici posvetili određenoj aplikaciji, od nje se očekuje da ima dobro razrađeno korisničko iskustvo. Loše korisničko iskustvo može uzrokovati da aplikaciju preko 70% korisnika prestane koristiti u kratkom vremenskom roku nakon prvog korištenja.

Korisničko iskustvo se definira kao svaka korisnikova interakcija s produktom ili uslugom. Dizajn korisničkog iskustva kao disciplina se bavi svim elementima koji zajedno čine sučelje aplikacije, uključujući vizualni dizajn, zvuk i interakciju. Dobro korisničko iskustvo se brine i za više funkcija nego što korisnik zahtijeva. Uključuje primjenu više disciplina kao što su inženjerstvo, marketing i grafički dizajn. Vidljivo iz navedenog, korisničko iskustvo ima puno prostora za rad.

Korisnici od aplikacije zahtijevaju intuitivno snalaženje u njoj i brzo savladavanje korištenja njenih funkcionalnosti. To područje aplikacije uvelike doprinosi hoće li ju korisnik nastaviti koristiti ili potražiti drugu.

2.2. Poboljšavanje korisničkog iskustva

Svaka aplikacija je namijenjena određenoj publici koju developeri trebaju razumjeti kako bi se osmislio optimalan pristup dizajniranju aplikacije. Ovisno o kojim se korisnicima predstavlja aplikacija, od korisnika je očekivano različito predznanje korištenja aplikacija. Iako postoje

kategorije korisnika te se aplikacija radi za specifičnu namjenu, postoje opći uvjeti razvoja aplikacije koji se moraju zadovoljiti kako bi se osigurala njena kvaliteta [8, 9, 10, 11, 12]. Navedeni uvjeti su podijeljeni na funkcionalne, koji se bave procesima koji se događaju u aplikaciji te dizajnerske koji se bave vizualnim aspektom aplikacije.

2.2.1. Funkcionalnosti mobilne aplikacije

Aplikacija u obavljanju svog osnovnog cilja treba biti brza i precizna. Prema izvješću *Localytics*-a [2], 21% korisnika prestane koristiti aplikaciju ako su imali negativno prvo iskustvo s njom. Tijekom njenog razvoja postoje određene osnovne funkcionalnosti zbog kojih je razvijena te se njima treba najviše posvetiti. Dodatne funkcionalnosti, iako povećavaju mogućnosti aplikacije, dodatno opterećuju korisnike svim mogućnostima. Razlog korištenja aplikacija je obavljanje složenih zadataka uz minimalnu potrebnu pažnju. Ako je aplikacija teška za korištenje te zahtjeva previše pažnje, korisnici će brzo odustati od nje.

Korisnika se treba osloboditi od što više bespotrebnog unosa podataka što je moguće. Unos podataka mora biti jednostavan. U slučaju tipkanja, korisniku se treba osigurati odgovarajuća tipkovnica pri samom unosu. Tipkanje je često moguće olakšati prijedlogom unosa na temelju do tad napisanog, ispravljanjem pogrešno napisanih riječi ili zaobići nekim drugim načinom unosa podataka.

Kako bi se olakšalo prvo korištenje aplikacije, pristup njenim funkcionalnostima treba biti intuitivan. U slučaju da rukovanje određenim područjima nije sasvim jasno na prvi pogled, od developera je očekivano pojašnjenje. *Onboarding* je proces koji korisnicima objasni kako koristiti određenu funkcionalnost kada se prvi put sretnu s njom. Iako se čini kao odlično pomagalo, treba se koristiti efikasno i samo kada je ono potrebno.

Navigacija postoji u svakoj aplikaciji te treba osigurati da je lako vidljiva, lagana za koristiti te da vodi negdje. Navigacija u aplikaciji ne bi trebala biti jedinstvena za tu aplikaciju, već napravljena na način s kojim se korisnik susretao u drugim aplikacijama. Također, u aplikaciji se nakon odabiranja jedne vrste navigacije treba pridržavati toj vrsti, odnosno izbjegavati korištenje više vrsta navigacije. Ne smije se dogoditi da neki put ima kraj bez rezultata, te u takvom slučaju se korisniku treba ponuditi slično rješenje problema.

Geste koje se koriste za korištenje dodatnih funkcionalnosti određenih objekata trebaju biti neke od standardnih. Trebaju biti korištene samo kao brži, ali već i postojani te lakše uočljiviji način korištenja aplikacije. Razlog tomu je jer su geste skrivene. Ako se developer previše oslanja na

geste, složenost aplikacije raste te će korisnik morati naučiti više o aplikaciji kako bi ju znao koristiti u potpunosti.

2.2.2. Dizajn mobilne aplikacije

Prva stvar koju korisnik vidi tijekom pokretanja aplikacije je njen dizajn. Prema [3], osobi treba 150ms kako bi formirao pouzdano mišljenje o aplikaciji. Za usporedbu, osobi je potrebno 200ms da pročita jednu riječ. Iz istraživanja je zaključeno da se mišljenje formirano u prvih 150ms ne razlikuje puno od mišljenja formiranog nakon 4 sekunde izloženosti dizajnu aplikacije. Znajući to, može se zaključiti da je prvi dojam jako bitan. Kako bi se osiguralo što bolje mišljenje, treba se pridržavati općih standarda dizajniranja aplikacije.

Dobar dizajn zahtijeva dobru istragu i plan izgleda aplikacije. Pošto na tržištu već postoji velik broj aplikacija, postoje i aplikacije koje su slične onoj koja se kreira. Developer treba proučiti te aplikacije kao korisnik i njihove dobre karakteristike koristiti kao smjernice, te razmisliti o mogućim poboljšanjima. Također treba uzeti u obzir koji će korisnici koristiti buduću aplikaciju.

Sučelje aplikacije je potrebno predstaviti na minimalistički način. Svako dodavanje gumbova, slika ili ikona povećava složenost sučelja. Korisniku se treba prikazati jednostavna interakcija s aplikacijom, pošto stavljanje previše mogućnosti preopterećuje korisnika. Jednostavnost povećava korisnikovu sposobnost razumijevanja aplikacije. Glavno pravilo je održavanje jedne primarne funkcionalnosti po aktivnosti.

Treba uzeti u obzir preglednost aplikacije i čitljivost prikazanog teksta. Za tekst se preporuča korištenje prirodnog fonta razvojne okoline, veličine iznad 11sp. Gumbovi trebaju biti dovoljno veliki da ih korisnik može stisnuti bez problema. Potrebno je paziti na kontrast korištenih boja, što uvelike utječe na preglednost. Od boja je dovoljno koristiti jednu primarnu i jednu sekundarnu te jedino mijenjati njene nijanse.

Kako zasloni mobitela postaju sve veći, teško je upravljati nečim što se nalazi na suprotnom kraju korisnikova palca. Iz tog razloga u obzir se uzima područje zaslona koje je najlakše dohvatljivo. Treba izbjegavati stavljanja interaktivnih gumbova ili slično pri vrhove zaslona te težiti držanju svega pri sredini zaslona.

2.2.3. Poboljšavanje korisničkog iskustva u strateškim društvenim igrama

Nakon osiguranja osnovnih zahtjeva o poboljšanju korisničkog iskustva u aplikaciji, treba se pobrinuti i da svoj cilj obavlja kvalitetno. Zadnjih godina, tvrtke su uz društvene igre krenule razvijati i aplikacije koje bi dodatno olakšale igranje igre kao pomagala ili ih učinile zabavnijima.

Na temelju tih aplikacija mogu se razviti smjernice kako napraviti dobru aplikaciju za proširenje igračkog iskustva te na što paziti. U [4] su se razgledale pozitivne i negativne strane korištenja aplikacije uz igranje društvene igre. Postoje razlozi na koje developeri aplikacija ne mogu utjecati, no ti se mogu zanemariti.

Glavna negativna strana je niska kvaliteta, a visoka složenost aplikacije. Treba ispitati aplikaciju u mnogim scenarijima i osigurati da i ako aplikacija padne, da se vrati na prijašnje stanje. Igraču treba omogućiti povratak na prijašnje stanje nakon obavljanja neke radnje u slučaju da ju je napravio slučajno. Ako je aplikacija previše složena, igrači će neizbježno morati posvetiti određeno vrijeme učenju aplikacije što je tijekom igranja samo distrakcija. Također, visoka složenost može uzrokovati da je jednostavnije igrati društvenu igru bez aplikacije.

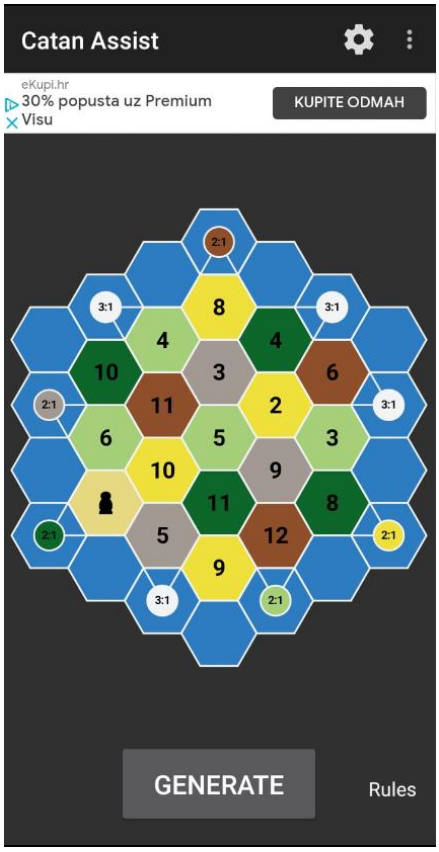
Aplikacija treba služiti samo kao pomoć pri igranju, a ne biti cijela zasebna igra. Oduzimanje dijelova igračkog iskustva automatizacijom radnji dovodi do pogoršanja igračkog iskustva. Prema [5], treba uzeti u obzir koje dijelove društvene igre je pogodno automatizirati, a koje oduzimaju osjećaj igranja. Iz istraživanja se može zaključiti da visok stupanj automatizacije dovodi do poteškoća praćenja igre, smanjenja strateškog razmišljanja te smanjenja kooperativnosti s partnerima. Automatizacija složenih događaja koji uvelike mijenjaju stanje igre može promaknuti novim igračima te uzrokovati zbunjenost i zahtijevati određeno vrijeme da igrač razumije što se dogodilo. U navedenom istraživanju su igrači izjavili da su se više zabavili tijekom igranja prave fizičke igre naspram aplikacije s visokom automatizacijom, no da im je niska automatizacija pomogla.

Davanjem aplikaciji preveliku kontrolu oduzima igračevu mogućnost stvaranja kućnih pravila. Određene društvene igre, već kao i *Uno*, igrači igraju po kućnim pravilima jer su na taj način pristupačnije većem broju igrača. Pokazano je i da novi igrači često odrede neka pravila kako bi se lakše privikli igranju. Forsiranjem pravila igre preko aplikacije smanjuje se igračeva sposobnost prilagodbe.

Igrači su izjavili da pretjerane funkcionalnosti koje oduzimaju korištenje fizičkih stvari društvene igre pridonose manjku osjećaja igranja društvene igre. Dobra aplikacija će se brinuti da se način igranja ne mijenja već da služi samo kao pomagalo.

Aplikacije mogu uvelike pomoći pri smanjenju potrebnog vremena za pripremu igre te pomažu u praćenju stanja. Aplikacija za Belu bi mogla držati rezultate nakon svake runde te zbrajati do sad osvojene bodove. Kako bi se pomoglo novim igračima, aplikacija može sadržavati video koji objašnjava pravila društvene igre. Složenije igre zahtijevaju određeno vrijeme za pripremu. U

društvenoj igri *Catan*, aplikacija poboljšava korisničko iskustvo tako što za svaku rundu daje jedinstvenu kombinaciju mape te uvelike pridonosi mogućnosti ponovnog igranja te smanjuje vrijeme pripreme jer igrači ne moraju pratiti pravila kako posložiti mapu. Na slici 2.1. prikazana je jednostavnost sučelja te aplikacije.



Slika 2.1. *Sičelje aplikacije za društevnu igru Catan*

Sukladno aplikaciji za društvenu igru *Catan*, aplikacije mogu pridonijeti igri dodavanjem novih mehanizama, novih načina za igranje igre. Neke igre imaju određenu gornju granicu koliko igrača može igrati, aplikacija može pružiti rješenje kako povećati taj broj. Igre koje se temelje na priči mogu sadržavati multimedijske efekte. Zvukovi, glazba, postojanje naratora mogu olakšati igraču da utone u priču. U takvim igrama developeri mogu igračima ažuriranjem aplikacije dati nove priče za odigrati. U ostalim igrama mogu dati nove karte, likove ili izgled određenim stvarima.

Glavno svojstvo društvenih igara je da su one igrane u društvu. Prema [6], sučelje koje podržava igru više igrača odjednom ne mora značiti da poboljšava interakciju među njima. Čak dolazi do pojave da svaki igrač bude zaokupljen onim što se njemu događa, dok ima malo znanja o situaciji drugih suigrača. U navedenom istraživanju igrači su za istim interaktivnim sučeljem i zajedno igraju igru. Razmatrali su koja svojstva pomažu u povećanju kooperativnog igranja. Najbitniji zaključak je da se igračima trebaju prezentirati stvari u igri koje nisu za njih, već za drugog

suigrača. Takav pristup igri podsjeća igrača da nije sam. Neki objekti na sučelju su zahtijevali interakciju više igrača odjednom. Stvaranje zadataka u kojima zajedničko rješavanje uvelike pridonosi brzini njegovog obavljanja također povećava kooperativnost. Samo dodavanje zvuka specifičnog za svakog igrača je pridonijelo osjećaju bivanja u grupi. U [7] razmatrano je koje stavke igre povećavaju empatiju i smanjenje negativnih osjećaja u kooperativnim igrama. U istraživanju su bila dva igrača koja su igrala potpuno digitalnu igru. Svakom je ponuđen isti zadatak, no u jednom slučaju se igrači međusobno nisu vidjeli, a u drugom slučaju jesu. Zaključeno je da se osjećaj empatije povećao u slučaju kada su se igrači međusobno vidjeli, no nije imalo znatnog utjecaja na pojavu negativnih osjećaja.

Iz navedenih istraživanja može se zaključiti da se u slučaju većeg digitaliziranja aplikacije za društvene strateške igre treba paziti na igračevu spoznaju igrača oko sebe. Will Wright, kreator popularne igre *The SIMS* je rekao da je cilj stvaranja igara da se naprave zajednice igrača.

2.3. Dungeons and Dragons

Kako bi se razvijana aplikacija mogla usporediti sa sličnim programskim rješenjima, prvo je potrebno shvatiti igru kojoj je namijenjena. *Dungeons and Dragons* je društvena strateška igra uloga [13] u kojoj je igrač predstavljen likom u fantastičnom svijetu kreiranom od strane *Dungeon Master*-a. *Dungeon Master* je igrač koji kreira fiktivan svijet te omogućuje interakciju ostalih igrača s tim svijetom. On je zadužen za kreiranje lokacija, događaja i likova koji žive u tom svijetu te glumljenje njih. Ostali igrači prave svojeg lika kojeg kontroliraju u stvorenom svijetu. Informacije o stvorenom liku su zapisane na papirima danim od strane osnivača igre nazvanim *Character Sheet*. Svaki lik može biti jedna od mnogih rasa i jedna od mnogih klasa. Svaka klasa ima svoje karakteristike poput oružja ili magije koju koristi te kakvu ulogu ima u bitkama. Rase osim što imaju utjecaja na prosječnu visinu, težinu i izgled, imaju utjecaja i na narav lika. Određene rase su u svom korijenu zle te mora postojati određen razlog da biće te rase ne bude svoje prirodne naravi. Svaki lik ima svoje ideale, vrline i mane te svoju povijest koja ga razlikuje od ostalih. Uspješnost u željenim akcijama s likom određuju njegove statistike i rezultat bačene kockice. Statistike se dijele na statistike lika kao što su snaga, inteligencija, karizma i ostale te statistike opreme koju lik posjeduje. Što se duže igra s određenim likom, s njim se skuplja iskustvo i vremenom se uče nove vještine. U nesretnim slučajima tijekom avanture, lik može poginuti te se tada stvara novi lik. U jednom se igranju može igrati zasebna avantura ili avantura koja je dio veće kampanje. Postoji skupina igrača koji svoju kampanju igraju već preko 30 godina. Kako bi se igrači snalazili u igranju, postoji knjiga *Player's Handbook* koja objašnjava sve rase, klase te

mogućnosti upravljanja likom. Također za *Dungeon Master*-a postoji *Dungeon Master's Guide*. Obje knjige sadrže otprilike 300 stranica.

2.4. Prikaz stanja u području i pregled sličnih programskih rješenja

Aplikacija je razvijena na Android platformi. Služi kao pomagalo u igranju društvene strateške igre *Dungeons and Dragons*. U trgovini *Google Play* postoji par sličnih aplikacija. Razmatrane su najsličnije aplikacije *5e Companion* i *DnD Session Assistant*.

2.4.1. 5e Companion

Aplikacija *5e Companion* nudi mnoštvo funkcionalnosti koje pomažu u igranju društvene igre *Dungeons and Dragons.* Aplikacija je namijenjena i običnom igraču i *Dungeon Master*-u. Kao pomagalo obojima postoji olakšani pristup informacijama iz knjige *Player's Handook*. Za igrače je olakšano držanje informacija o stvorenim likovima u aplikaciji kako bi se oslobodila ovisnost o papirima. Također je razvijena određena razina automatizacija rukovanja s unesenim informacijama. Navedena automatizacija pomaže i početnicima u skraćenju vremena traženja svih informacija oko kreiranja lika. Za *Dungeon Master*-a nudi više pomagala. Mogu se lakoćom generirati grupe protivnika koji napadaju igrače tijekom avanture. Sadržane su informacije o čudovištima iz knjige *Monsters Manual* te im se brzo pristupa. Omogućeno je kreiranje izmišljenog čudovišta i držanje informacija o njemu.

Najveći nedostatak navedene aplikacije je taj što postoji previše mogućnosti te većina nije dovoljno razrađeno. Kreiranje lika je digitalizirano, no i dalje složeno. Aplikacija je svojom veličinom i mogućnostima predstavljena kao da zamjenjuje potrebne knjige za uspješno igranje igre, no pri interakciji s unesenim podacima se teško dolazi do informacija. Unosi za oba tipa igrača su spori. Nedostaju ključna ograničenja koja dovode do zbunjenosti igrača, pogotovo novijih. Navedeni nedostatci ukazuju na loše korisničko iskustvo ili nedovršeni proizvod.

2.4.2. DnD Session Assistant

DnD Session Assistant aplikacija služi kao pomagalo za držanje informacija i upravljanjem lika tijekom igranja. Za razliku od aplikacije *5e Companion*, ograničenja su bolje razrađena i osnovne funkcionalnosti su bolje implementirane. Aplikacija također ima brz pristup informacijama napisanih u knjizi *Player's Handbook*. Kreiranje i upravljanjem likom je jedna od više funkcionalnosti navedene aplikacije. Problem je što ostale funkcionalnosti postoje, no nemaju veze s igranjem. Iako su opći unosi kao što su rasa i klasa lika dobro međusobno povezani i daju

strukturu, bilo što personalizirano nije podržano. Svaki ulazak u aplikaciju dočekan je reklamom što znatno pogoršava korisničko iskustvo.

3. IDEJNO RJEŠENJE I MODEL MOBILNE APLIKACIJE ZA POBOLJŠAVANJE KORISNIČKOG ISKUSTVA KOD IGRANJA STRATEŠKE DRUŠTVENE IGRE

3.1. Zahtjevi na mobilnu Android aplikaciju

Cilj aplikacije je pomoć pri igranju te poboljšanje korisničkog iskustva tijekom igranja društvene igre *Dungeons and Dragons*. Kako bi se zamijenio tradicionalan pristup igri gdje igrač treba nositi pribor za pisanje, dva papira s informacijama o liku te dodatne papire s bilješkama o priči, teži se digitaliziranju i automatizaciji pojedinih odgovornosti. Potrebno je omogućiti korisniku rukovanje s određenim likom kojeg odabere za planiranu avanturu. To zahtijeva mogućnost kreiranja lika te njegovo spremanje za kasnije korištenje. Kreiranje lika može potrajati i do dva sata. Digitaliziranje kreiranja može uvelike smanjiti potrebno vrijeme ako se na temelju unesenih informacija korisniku ponude druge relevantne informacije. Glavni parametri svakog lika su njegova rasa, klasa i pozadina. Na temelju njih igraču se dodjeljuje početna oprema, određeni dodaci na statistike kao što su snaga, agilnost i inteligencija te vještine u korištenju određenih alata i oružja. Također se treba pobrinuti za rase koje imaju „podrase“ te prikazati korisniku njihovu razliku. Nakon kreiranja lika, korisniku je potrebno omogućiti upravljanje njegovim iskustvom i opremom koju će koristiti. Kako bi se osigurala složnost unesenih podataka s pravilima igre, treba implementirati određena ograničenja na unose. Korisnik će koristiti iz igre već kreiranu opremu te je zahtijevana pravilna međusobna interakcija i prikaz odgovarajućih informacija korisniku. Kako se oprema može dobivati i gubiti tijekom igranja, potrebno je omogućiti dodavanje i brisanje stvari iz vlasništva. Igra se temelji na kreativnosti igrača te će neizbježno doći do kreiranja vlastitih stvari. Unos personaliziranih stvari treba biti lagano i produkti moraju biti dobro ukomponirani u već postojanu strukturu. Sve unesene stvari moraju se povezivati samo s likom koji ih posjeduje. Općenite radnje lika moraju biti ponuđene i zajedno sa svim do sad navedenim trebaju biti intuitivne za korištenje.

3.2. Model rješenja

Kako bi se mogli ispuniti navedeni zahtjevi, potrebno je imati i koristiti bazu podataka. Mogućnost kreiranja lika pri prvom pokretanju aplikacije zahtijeva stvaranje i spremanje podataka vezanih za kreiranje lika u bazu podataka za kasniji pristup. Navedeni proces treba se odraditi u pozadini, kako korisnik ne bi morao čekati da krene koristiti aplikaciju. Model rješenja sastoji se od tri glavna dijela: kreiranje lika te nakon odabiranja lika, bacanje kockica i kontroliranje posjedovanih stvari.

3.2.1. Kreiranje lika

Za samo kreiranje lika potrebno je imati sve vrijednosti glavnih parametara. Glavni parametri su sami po sebi složeni objekti i za svaki treba napraviti *Entity* u bazi podataka kao i za samog lika. Kako bi se obavještavalo korisnika što promjena određenog parametra znači, treba imati pristup svim informacijama koje su vezane za taj parametar te pratiti svaku promjenu koja se dogodi. Iz razloga što je velik broj parametara međusobno povezan, treba kontinuirano spremati korisnikove odabire za njihovo kasnije korištenje ili povratak na njih u slučaju želje za izmjenama. Kako se teži ubrzanju kreiranja likova, korisniku se za svaku odluku biranja parametra trebaju prikazati u tom trenutnu bitne informacije. Iz razloga što objekt lika ne treba držati sve informacije o njegovim složenim glavnim parametrima, treba postojati trajna veza između njihovih klasa kako bi se u kasnijim slučajima moglo brzo doći do potrebnih informacija. Pri završetku kreiranja lika, trebaju se napraviti i objekti opreme koje posjeduje te i njih spremiti u bazu podataka s vezom između njih. Nakon kreiranja lika, objekt lika se sprema u bazu podataka za kasnije korištenje. Odabiranjem kreiranog lika se učitavaju sve njegove vrijednosti te prikazuju korisniku. Na temelju tih informacija korisnik zna kako igrati sa svojim likom te zaključivati što raditi u određenim situacijama.

3.2.2. Kockice

Kao pomagalo pri igranju, korisniku se pruža mogućnost bacanja kockica u samoj aplikaciji. Time se uklanja fizička ovisnost o mogućnosti igranja *Dungeons and Dragons* društvene igre o posjedovanju kockica. Korištenje kockica treba biti brzo i intuitivno. Treba omogućiti bacanje više kockica istog broja uz određene dodatne bodove. U slučaju bacanja više kockica, trebaju se prikazati rezultati svih bacanja bez dodatnih bodova jer postoje određena događanja koja se temelje na bačenom broju. Također treba prikazati sumu svih bacanja kako bi se ubrzao proces dolaženja do ukupnog rezultata.

3.2.3. Posjedovanje stvari

Ovisno o odabranoj klasi, novo kreirani lik ima početnu opremu koje može koristiti. Igranjem korisnik dobiva ili gubi objekte koje posjeduje te treba sukladno ažurirati bazu podataka o takvim događanjima. Dodavanje stvari treba biti kontrolirano, novo pribavljena oružja i oprema trebaju biti jedni od objekata koji već postoje u bazi podataka. Razlog tome je što društvena igra sadrži isplaniranu, uravnoteženu kolekciju opreme, te dodavanje vlastito napravljenih objekata narušava takvu strukturu. Stvari koje su vezane za priču ili avanturu kroz koju lik napreduje također se trebaju moći dodavati i spremati za kasnije korištenje. Korisnički kreirane stvari u svrhu priče se

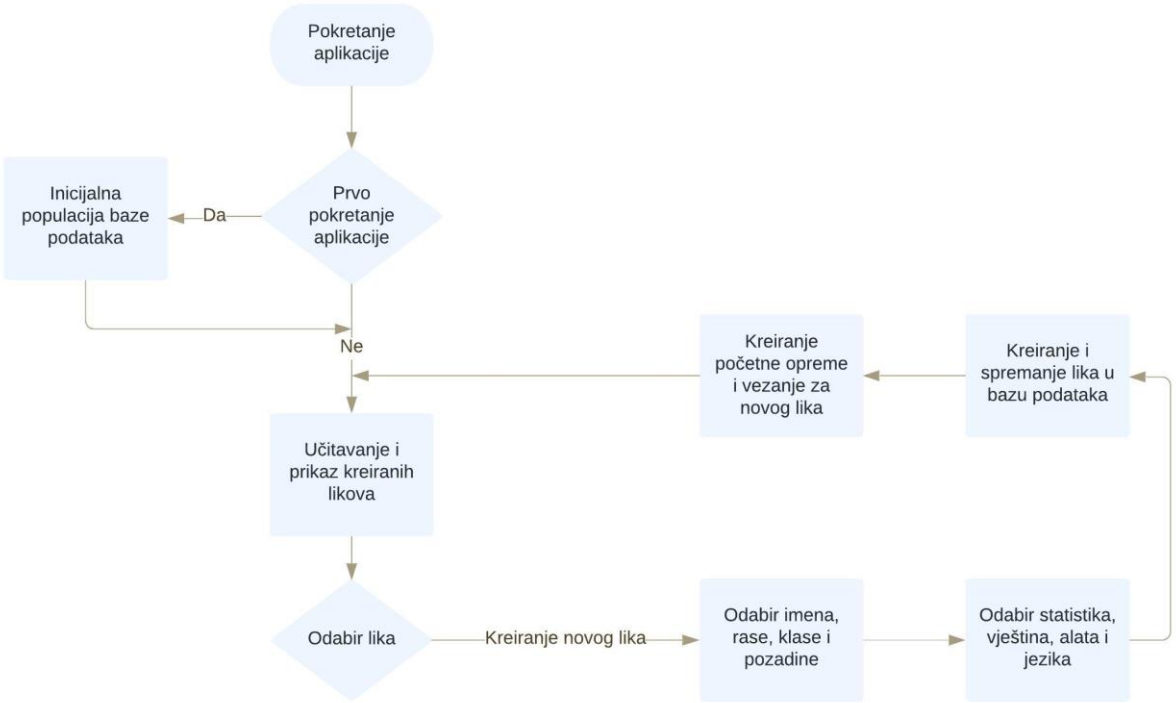
tretiraju kao alat, nešto što ne utječe izravno na karakteristike lika. Korisniku treba biti omogućen lagan prijelaz između odabira oružja, opreme ili alata koje posjeduje.

3.3. Prijedlog arhitekture rješenja

Aplikacija se sastoji od dva dijela, prvog u kojem se kreira lik i drugog u kojem se koristi odabrani lik za pomoć pri igranju.

3.3.1. Arhitektura rješenja kreiranja lika

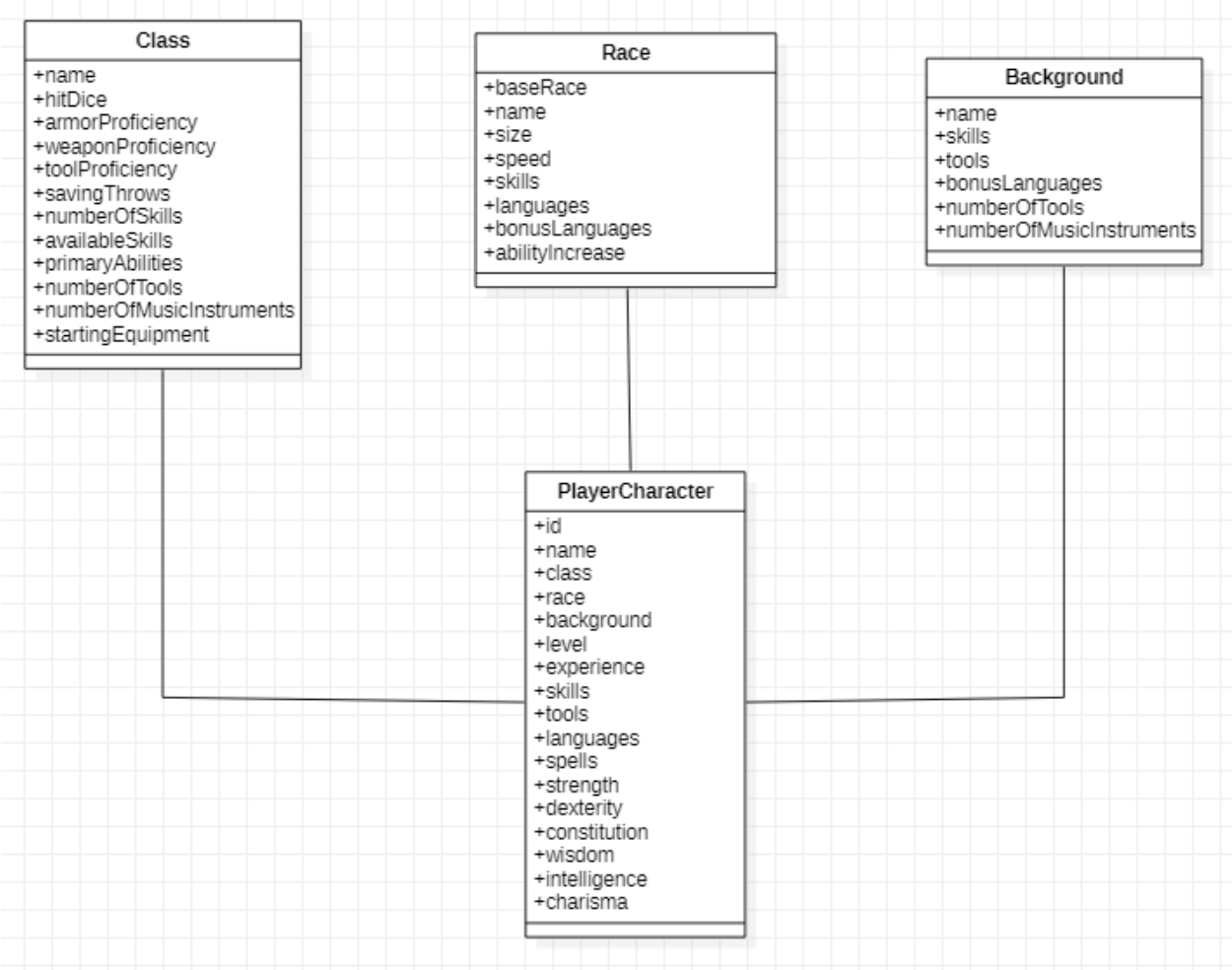
Idejno rješenje i dijagram tijeka prvog koraka aplikacije u kojem korisnik kreira lika kojeg se sprema u bazu podataka su prikazani slikom 3.1.



Slika 3.1. *Dijagrama tijeka kreiranja novog lika*

Prvim pokretanjem aplikacije u pozadini se kreiraju i spremaju podaci u bazu podataka za kasnije korištenje tijekom kreiranja lika. Nakon inicijalne populacije baze podataka, podaci ostaju spremljeni u njoj te se mogu koristiti kad god. Da se nakon pokretanja aplikacije uspiju prikazati likovi za odabir, aplikacija prvo treba pozvati bazu podataka i učitati kreirane likove iz nje. U slučaju da ne postoji niti jedan lik, jedina opcija je kreirati novog. Novi lik se kreira odlaskom na novu aktivnost koja je zadužena za njegovo postupno kreiranje. Na sljedećem zaslonu se biraju glavni parametri lika. Kako bi korisničko iskustvo bilo zadovoljeno, korisniku se treba pružiti mogućnost naknadne izmjene odabranih parametara pamćenjem korisnikovih odabira. Svaka

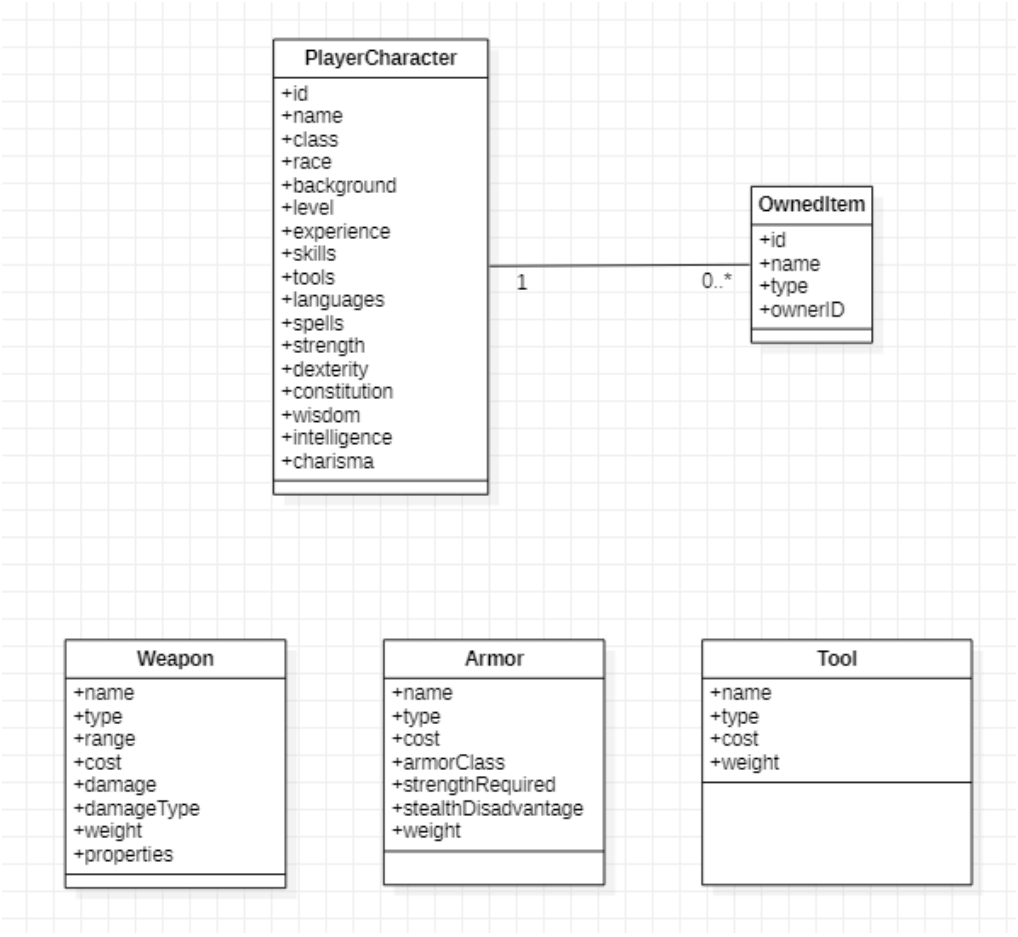
tranzicija između zaslona treba biti glatka. Na zaslonima odabira statistika, vještina, alata i jezika korisniku se prikazuju potrebne informacije temeljene na odabranim glavnim parametrima. Kada je korisnik zadovoljan konstruiranim likom, objekt u koje je kontinuirano spremao podatke se realizira i spremi u bazu podataka za daljnje korištenje. Njegov se identitet koristi za stvaranje početne opreme u bazi podataka. Da se sve bitne informacije zadrže, objekt lika u bazi podataka je definiran i povezan načinom prikazanim na slici 3.2.



Slika 3.2. *Struktura objekata potrebnih za kreiranje lika*

Velik broj atributa lika ovisi o složenim objektima klase, rase i pozadine. Kako se ne bi morale u objekt lika spremati varijable koje se ne mijenjaju, već ovise samo o odabranoj rasi, klasi ili pozadini te u korist puno efikasnijeg pretraživanja baze podataka, stvara se *Foreign key* (strani ključ). Objekti koji trebaju biti jedinstveni u bazi podataka imaju *Primary key* (glavni ključ) koji ih obilježava kao što OIB označava osobu. Kada jedan objekt spremi vrijednost glavnog ključa drugog objekta u svoju varijablu te postavi referencu na drugi objekt, nastaje veza između njih te se u prvom objektu ta varijable zove strani ključ. U slučaju klase lika sa svakom od klasa rase, klase i pozadine nastaje nova veza odnosa 1:1 te se objekti tih klasa dohvaćaju pri jednostavnoj

pretrazi baze podataka. Odabiranjem određene klase, liku se dodjeljuju vještine korištenja određenih oružja, opreme i alata. Uz to, svaka klasa nudi igraču odabir stručnosti u određenim vještinama kao što su sakrivanje, akrobatika, povijest i ostale. Također na temelju nje lik dobiva svoju početnu opremu s kojom kreće igrati te mu se daje informacija kojim statistikama se treba posvetiti kako bi povećao svoj potencijal. Sljedeće, korisnik bira rasu svog lika. Rasa određuje veličinu i brzinu lika, jezike koje razumije te broj jezika koje može dodatno znati i koje statistike mu se povećavaju. Kako bi korisnik imao bolju ideju koja mu rasa odgovara, na zaslonu se uz glavne informacije rase prikazuju glavne informacije odabrane klase. Nakon rase, korisnik odabire pozadinu svog lika na temelju čeg dobiva dodatne vještine te određene pozadine daju mogućnost učenja dodatnih jezika. Nakon odabira glavnih parametara klase, rase i pozadine, na korisnik detaljnije uređuje svog lika odabirom statistika, dodatnih vještina, alata i jezika. Kao kraj aktivnosti kreiranja novog lika, lik se sprema u bazu podataka i njemu dodijeljen identifikacijski broj se uzima za dodavanje novih objekata u bazu podataka koji predstavljaju njegovu početnu opremu. Na slici 3.3. prikazana je veza klase lika i klase koja svojim objektima predstavlja njegovu opremu te klase moguće opreme.

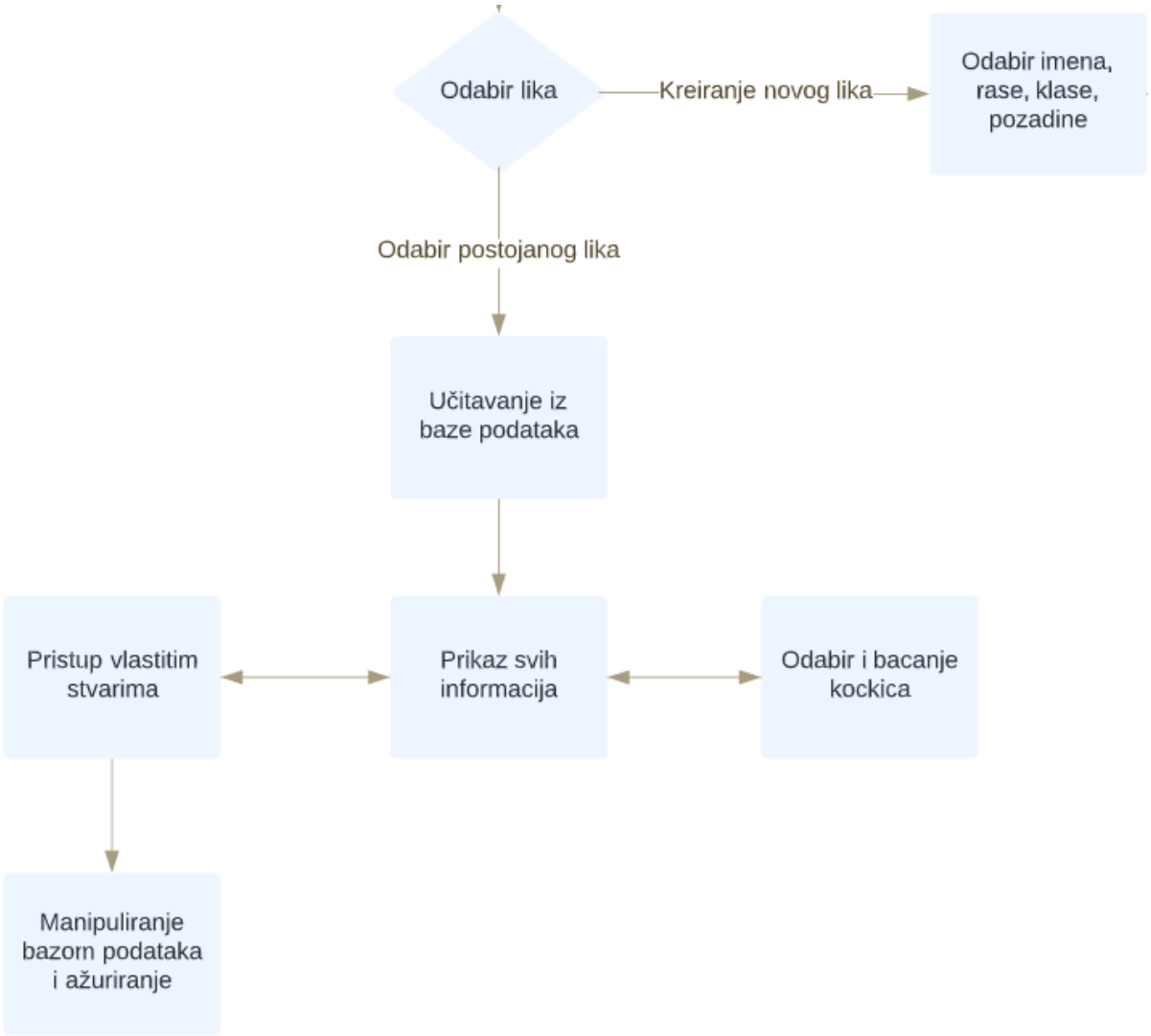


Slika 3.3. *Veza klase lika i klase „OwnedItem“ te klase moguće opreme*

Prijašnje spomenuta *Primary key – Foreign key* veza omogućuje još jednu vrstu veze odnosa 1:N. U takvoj vezi se više objekata veže za jedan objekt što na razini baze podataka znatno olakšava pretraživanje. Povratna vrijednost funkcije umetanja lika u bazu podataka je njegov identifikacijski broj koji se dodjeljuje parametru „ownerID“ u objektu klase „OwnedItem“. Tom poveznicom baza podataka se može lakoćom filtrirati da se dobiju svi „OwnedItem“ objekti određenog lika. Atribut „type“ navedene klase služi prepoznavanju što taj objekt predstavlja, oružje, oklop ili alat.

3.3.2. Arhitektura rješenja interakcije s likom

Na slici 3.4. prikazano je idejno rješenje i dijagram tijeka drugog koraka aplikacije u kojem korisnik bira lika i njime ima mogućnost igranja društvene igre.



Slika 3.4. *Dijagrama tijeka interakcije s likom*

Odabirom lika iz baze podataka se učitavaju njegove informacije detaljnije i na novoj aktivnosti prikazuju korisniku. Na temelju danih informacija korisnik može igrati društvenu igru i nakon

bitke ili obavljanja nekog zadatka si dodati bodove iskustva što vode do većeg nivoa. Svakim porastom nivoa, lik je jači i izdržljiviji. Sljedeći zaslon zadužen je za omogućavanje korisniku bacanje kockica. Kako bi se poboljšalo korisničko iskustvo, odabir količine kockica, dodatnih bodova i odabir koja se kockica baca mora biti brzo i jednostavno. Treći zaslon u trenutnoj aktivnosti služi za kontroliranje stvari koje lik posjeduje. Klikom na određen tip objekta, korisniku se prikazuju objekti tog tipa u vlasništvu lika. Na slici 3.3. mogu se vidjeti tipovi objekata koji se prikazuju. Objekt klase „OwnedItem“ služi kao poveznica između lika i različitih objekata te iz vrijednosti varijable „type“ može se znati koje objekte treba prikazati. Korisnik može dodavati nove stvari svom liku stvaranjem novog „OwnedItem“ objekta koji pokazuje na objekt neke druge klase. Kada korisnik obriše objekt, briše se objekt „OwnedItem“ koji je pokazivao na učitani objekt. Ovakav pristup omogućuje korisniku da ima više kopija istog objekta, npr. u slučaju da želi imati pet koplja za bacanje. Također se na takav način na istoj aktivnosti mogu prikazivati različiti objekti samo ovisno o tome što korisnik želi vidjeti. Svo brisanje i dodavanje se automatski mora ažurirati kako bi korisnik odmah dobio povratnu informaciju o trenutnom stanju njegovog lika.

4. PROGRAMSKO RJEŠENJE MOBILNE ANDROID APLIKACIJE

4.1. Korištene programske tehnologije, okviri i jezici

Razvoj aplikacije ostvaren je na platformi *Android Studio*. Programski jezik u kojem je pisan kod je *Java*, dok je sučelje pisano u *XML*-u. Kako bi se ostvarilo spremanje podataka za kasniji pristup, korištena je *Room* biblioteka za stvaranje baze podataka i njeno manipuliranje.

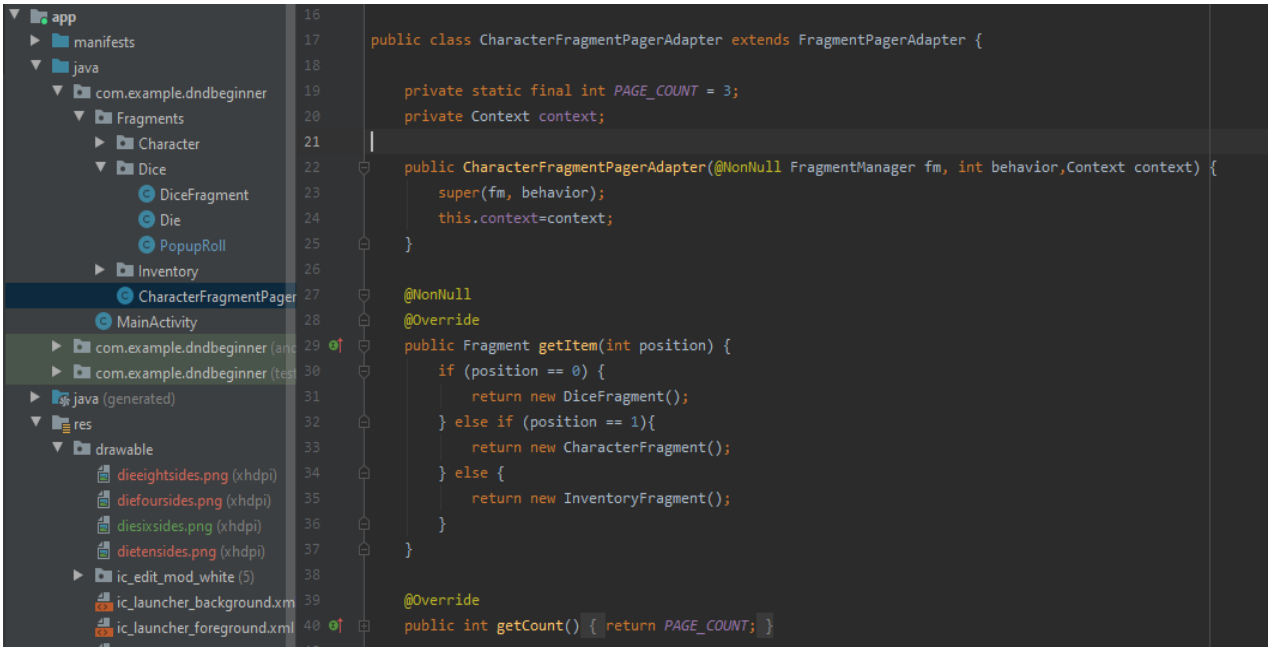
4.1.1. Razvojna okolina Android Studio

Android Studio je službena razvojna okolina namijenjena za aplikacije na *Googlovom* operacijskom sustavu Androidu [14]. Prva stabilna inačica objavljena je 2014. godine. Programski jezici koji su podržani su Java, C++ i nedavno razvijen *Googlov* programski jezik *Kotlin*. *Android Studio* nudi mogućnost kreiranja virtualnog uređaja na kojem se mogu testirati razvijene aplikacije. Različiti uređaji imaju različit *API* i inačicu Androida te treba paziti za koju publiku se razvija aplikacija. Također je podržan unos vanjskih biblioteka koje omogućuju dodatne već razvijene funkcionalnosti.

Izgrađen je na temelju *JetBeansovog* razvojnog okruženja *IntelliJ* koji također koristi programski jezik *Java*. Po uzoru na *IntelliJ*, u *Android Studiju* je također ostvarena pomoć pri pisanju koda u obliku predlaganja metodi na temelju napisanog, prikaza potrebnih argumenata za metodu, lakog preopterećivanje metoda i još mnogih svojstava. Logika je pisana u *Javi*, a vizualni dizajn određuju *XML* datoteke.

4.1.2. Programski jezik Java

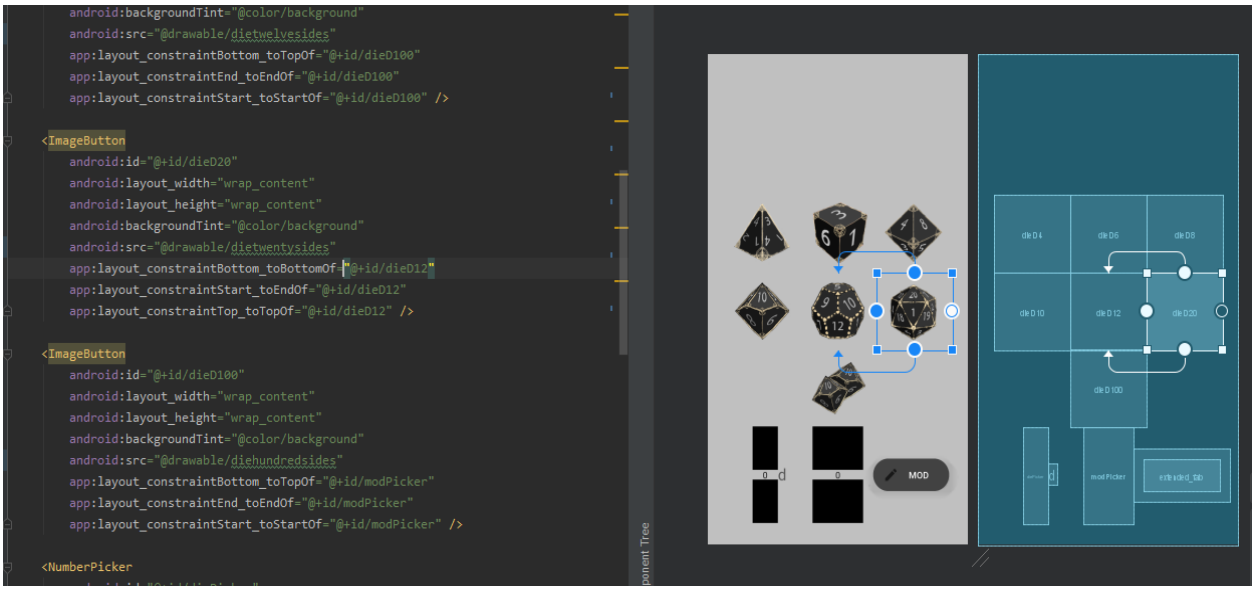
Java je objektno-orijentirani programski jezik koji se temelji na klasama. Napravljen je s ciljem da developeri aplikacija mogu napisati kod samo jednom i pokrenuti ga na bilo kojoj platformi koja podržava *Javu*, bez potrebe za ponovnim kompiliranjem. *Java Virtual Machine* omogućuje pokretanje *Java* aplikacija bez obzira o arhitekturi računala. 2019. *Github* je objavio da je *Java* najkorišteniji programski jezik. Sintaksa *Jave* je slična programskim jezicima C i C++ te je prikazana na slici 4.1.



Slika 4.1. *Sintaksa programskog jezika Java*

4.1.3. Jezik XML

Extensible Markup Language služi kao ljudima lagan kod za čitanje kojeg također i računalo može čitati. Cilj *XML*-a je jednostavnost, općenitost i lako korištenje. Ima jaku podršku za sve razne znakove u brojnim ljudskim jezicima preko *Unicode*-a. U *Android Studiju* se koristi kao jezik koji opisuje pravila kako će se određeni element prikazivati na zaslonu. Sintaksa *XML* jezika i dizajnirano sučelje prikazani su na slici 4.2.



Slika 4.2. *Sintaksa XML jezika i odgovarajuće sučelje*

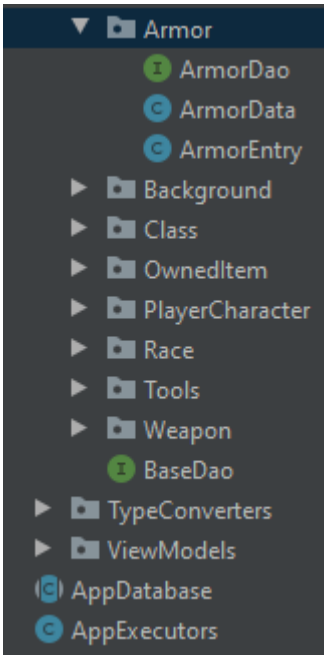
4.1.4. Biblioteka Room

Room je biblioteka koja funkcionira pretvaranjem objekata koji se žele spremiti u *Java* objekte. Na Android platformi implementiranje baze podataka zahtijeva puno posla. *Room* pojednostavljuje taj proces smanjivanjem ponovljenog koda za svaku aplikaciju i provjerava *SQL* strukturu baze tijekom kompiliranja aplikacije. To znači da kada se napiše struktura baze podataka u aplikaciji, prije pokretanja same aplikacije na uređaju, *Room* provjeri svoju strukturu i ako pronađe grešku onemogući njeno pokretanje dok se ta pogreška ne ispravi. Zbog toga se nikad neće dogoditi da nakon pregledavanja koda aplikacije ispostavi da je greška zapravo u strukturi baze podataka.

4.2. Arhitektura Room, ViewModel i LiveData

4.2.1. Arhitektura Room

Konfiguriranje baze podataka *Room* bibliotekom je jednostavno implementiranje klase baze podataka „AppDatabase“ te posebna klasa i sučelje *Dao* za svaki od *Entityja*. Za slučaj ove aplikacije, potrebne klase su prikazane na slici 4.3.



Slika 4.3. *Potrebne klase za implementaciju Room biblioteke*

Dao sučelja služe za uporabu baze podataka definiranjem funkcija koje izvršavaju *SQL* naredbe. Svako *Dao* sučelje sadrži osnovne funkcije za umetanje, ažuriranje i brisanje te se iz tog razloga može napraviti osnovno sučelje „BaseDao“ iz kojeg se sva ostala sučelja izvode. Na slici 4.4. prikazan je programski kod koji definira „BaseDao“ sučelje.

```java
public interface BaseDao<T> {

    @Insert
    void insert(T obj);

    @Insert
    void insertAll(T... obj);

    @Update(onConflict = OnConflictStrategy.REPLACE)
    void update(T obj);

    @Delete
    void delete(T obj);
}
```

Slika 4.4 *Programski kod BaseDao sučelja*

Data klase poput „ArmorData“ sadrže objekte potrebne za inicijalnu populaciju baze podataka. Klase poput „ArmorEntry“ predstavljaju *Entity* u *SQL* bazi podataka koje *Room* mapira kao objekte u *Javi*. Kao i u *SQL*-u, na atribute se mogu stavljati ograničenja te jedna od mogućnosti je stvaranje stranih ključeva. Slikom 4.5. prikazan je programski kod kojim se stvaraju strani ključevi „PlayerCharacterEntry“ klase.

```java
@Entity(tableName = "playerCharacters",
        foreignKeys = {@ForeignKey(entity = RaceEntry.class,
            parentColumns = "race_name",
            childColumns = "race",
            onDelete = ForeignKey.CASCADE),
        @ForeignKey(entity = ClassEntry.class,
            parentColumns = "name",
            childColumns = "class",
            onDelete = ForeignKey.CASCADE),
        @ForeignKey(entity = BackgroundEntry.class,
                parentColumns = "name",
                childColumns = "background",
                onDelete = ForeignKey.CASCADE)},
        indices = {@Index(value = {"race"}),
                @Index(value = {"class"}),
                @Index(value = {"background"})})
public class PlayerCharacterEntry {
    @PrimaryKey(autoGenerate = true)
    private int id;
    @NonNull
```

Slika 4.5 *Programski kod implementacije stranih ključeva PlayerCharacterEntry klase*

Stvoreni strani ključevi povezuju klasu lika s klasama likove rase, klase i pozadine. Nakon što je kreiran i spremljen objekt klase „PlayerCharacterEntry“, objektima složenih klasa s kojima je

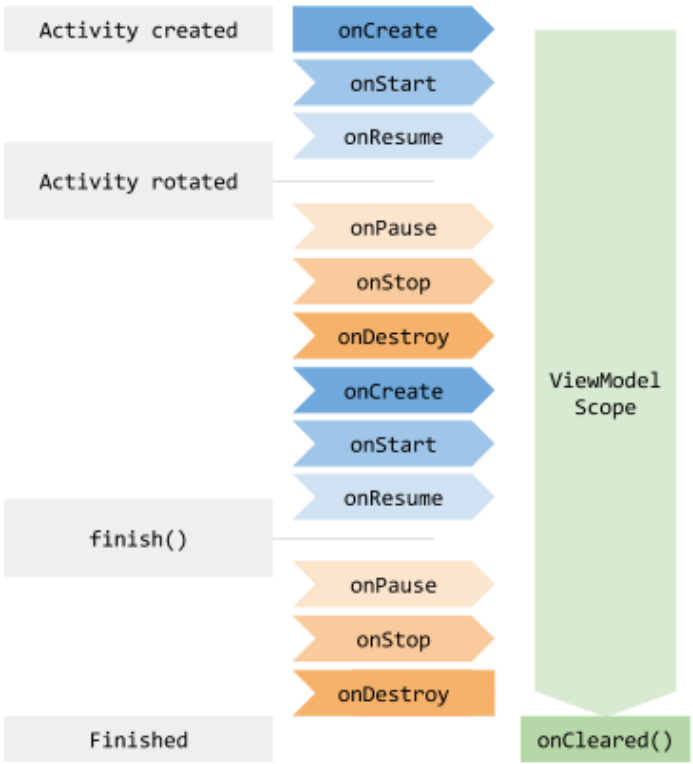
povezan može se brzo doći te izvući potrebne informacije. Sve što je potrebno je u *Dao* sučelju određene klase kreirati metodu koja *SQL* naredbom dohvaća objekt te klase. Na slici 4.6. prikazana je metoda sa *SQL* naredbom kojom se dohvaća objekt likove rase na temelju imena rase i identifikacijskog broja lika.

```
@Query("SELECT race.* FROM race "+
        "INNER JOIN playerCharacters ON race.race_name=playerCharacters.race WHERE playerCharacters.id LIKE :id")
LiveData<RaceEntry> getPlayerRace(int id);
```

Slika 4.6. *Programski kod metode sa SQL naredbom za dohvaćanje objekta rase lika*

4.2.2. ViewModel i LiveData

U aktivnosti kreiranja lika potrebno je pamtiti korisnikov odabir kroz više zaslona. Jedan zaslon aktivnosti može sadržavati jedan ili više fragmenata. Fragmenti predstavljaju skup komponenata korisničkog sučelja njima nadređene aktivnosti te imaju vlastiti životni ciklus. Kako je u aplikaciji potrebno pamćenje informacija izvan životnog ciklusa fragmenta, dolazi do potrebe korištenja *ViewModela* i *LiveData* objekata. Na slici 4.7. prikazan je životni ciklus *ViewModela* u usporedbi sa životnim ciklusom fragmenta ili aktivnosti.



Slika 4.7. *Životni ciklusi ViewModela i fragmenta ili aktivnosti*

Otvaranjem fragmenta ili aktivnosti poziva se metoda *onCreate* gdje se može kreirati *ViewModel*. Odlaskom s fragmenta ili aktivnosti poziva se metoda *onDestroy* u kojoj se oslobađa sva zauzeta memorija te se tako gube odabrani podaci. Kreiranjem *ViewModela* i spremanjem podataka u njega, podaci mogu preživjeti *onDestroy* metode fragmenata ili aktivnosti. Navedena činjenica omogućuje da se u aktivnosti kreiranja lika gdje su sadržana tri fragmenta *ViewModelom* mogu dohvaćati informacije s različitih fragmenata. Korištenje *ViewModela* i *LiveData* objekata omogućuje kontinuirano ažuriranje sučelja s kojim su povezani. *LiveData* je klasa koja sadržava samo jedan atribut kojeg se može promatrati. Promatranje funkcionira na principu *Observer* obrasca koji obavještava svoje *subscribere* o bilo kakvoj promjeni. Uz to, bilo kakvo čitanje podataka iz objekta *LiveData* obavlja se asinkrono, što znači da se stvara novi *Thread* (nova nit) na kojem se obavljaju sva ažuriranja korisničkog sučelja kako ne bi blokiralo bilo kakav korisnikov unos. Na slici 4.8. prikazana je klasa „CharacterClassViewModel“ koja sadržava *ViewModel* korišten za promatranje promjena odabira klase lika.

```
public class CharacterClassViewModel extends ViewModel {
    private MutableLiveData<ClassEntry> characterClass=new MutableLiveData<>(new ClassEntry());

    public void setCharacterClass(ClassEntry playerClass){
        this.characterClass.setValue(playerClass);
    }
    public LiveData<ClassEntry> getCharacterClass(){
        return this.characterClass;
    }
}
```

Slika 4.8. *Programski kod ViewModela korištenog za promatranje promjene odabrane klase*

Kao što je na prijašnjoj slici naveden *ViewModel* za promatranje promjenu odabrane klase, analogno su implementirani *ViewModeli* koji promatraju promjene odabrane rase i pozadine. Kad god korisnik promjeni jedan od tih parametara, sukladno se promjeni vrijednost odgovarajućeg parametra u objektu lika kojeg se kreira, te se također ažurira i dio korisničkog sučelja vezanog za taj parametar. U slučaju da neki dijelovi korisničkog sučelja nisu u trenutku promjene vidljivi, takvim pristupom korisnik neće promjenu ni vidjeti već će korisničko sučelje već biti ažurirano prije korisnikove pojave. Dodatna korist korištenja *LiveData* objekata je što se njenim uništavanjem oslobodi memorija svih *observera* koji su bili kreirani.

4.3. Programsko rješenje glavnih zahtjeva

4.3.1. Inicijalna populacija baze podataka

Roomova klasa „AppDatabase“ sadrži metodu „addCallback“ koja izvršava kod napisan u njoj tijekom prvog pokretanja aplikacije. Populacija baze podataka se treba odviti asinkrono te se u tu svrhu stvara klasa „AppExecutor“ za korištenje *Executor* objekata. *Executor* objekti se koriste kako bi se zaobišla odgovornost upravljanja poretkom odvijanja procesa. Za populaciju baze podataka napravljene su Data klase s jednom *public static* metodom koja se za svaki *entity* poziva unutar *addCallback* metode. Na slici 4.9. prikazan je programski kod metode *addCallback* i načina na koji se pozivanjem *Dao* sučelja svake klase ispunjava baza podataka.

```
    if (sInstance == null) {
        synchronized (LOCK) {
            Log.d(LOG_TAG, msg: "Creating new database instance");
            sInstance = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, AppDatabase.DATABASE_NAME)
                    .addCallback(new Callback() {
                        @Override
                        public void onCreate(@NonNull SupportSQLiteDatabase db) {
                            super.onCreate(db);
                            AppExecutors.getInstance().diskIO().execute(new Runnable() {
                                @Override
                                public void run() {
                                    AppDatabase instance=getInstance(context);
                                    populateDatabase(instance);

                                }
                            });
                        }
                    })
                    .build();
        }
    }
    return sInstance;
}

private static void populateDatabase(AppDatabase instance) {
    instance.weaponDao().insertAll(WeaponData.populateWeaponData());
    instance.armorDao().insertAll(ArmorData.populateArmorData());
    instance.RaceDao().insertAll(RaceData.populateRaceData());
    instance.toolDao().insertAll(ToolData.populateToolData());
    instance.backgroundDao().insertAll(BackgroundData.populateBackgroundData());
    instance.classDao().insertAll(ClassData.populateClassData());
    instance.ammunitionDao().insertAll(AmmunitionData.populateAmmunitionData());
}
```

Slika 4.9. *Programski kod za inicijalnu populaciju baze podataka*

4.3.2. Kreiranje lika

Kreiranje lika odvija se kroz tri fragmenta, u prvom se biraju ime, rasa i klasa, u drugom pozadina i statistike te u trećem vještine, alati, instrumenti i jezici.

Fragment za odabir imena, klase i rase

Korisnik unosi ime lika u *EditText* nakon čega se uneseni *String* sprema uzimajući u obzir ako se unese prazan prostor prije ili poslije imena da ga obriše. Kako fragment nema gumb za spremanje unosa, taj proces se odvija nakon što korisnik ode na drugi fragment. U slučaju da korisnik ne unese ime, liku se automatski dodjeljuje ime „Unknown Character“. Na slici 4.10. prikazan je programski kod kojim je takvo ponašanje ostvareno.

```
@Override
public void onPause() {
    if(characterName.getText()==null || characterName.getText().toString().equals("")){
        characterName.setText(R.string.unknown_name);
    }
    characterCreationViewModel.setCharacterName(characterName.getText().toString().trim());
    super.onPause();
}
```

Slika 4.10. *Programski kod funkcije spremanja imena lika*

Za odabir rase i klase, prvo je potrebno doći do svih objekata rasa i klasa. Pošto se operacije s *Room* bazom podataka ne mogu odvijati na *main threadu*, poziva se metoda klase „AppExecutor“ te se stvara nova nit. Nakon dohvaćanja podataka, vraća se na *main thread* kako bi se predali podaci funkcijama za postavljanje *NumberPickera* za rase i klase nazvanima *classPicker i racePicker* te kako bi se moglo ažurirati korisničko sučelje. Na slici 4.11. prikazan je programski kod za dohvaćanje imena klasa, rasa i pozadina unutar nove niti.

```
AppExecutors.getInstance().diskIO().execute(new Runnable() {
    @Override
    public void run() {
        final List<ClassEntry> classEntries = mDb.classDao().getClasses();
        final List<RaceEntry> raceEntries=mDb.RaceDao().getRaces();
        final List<String> distinctRaceNames=mDb.RaceDao().getDistinctNames();

        AppExecutors.getInstance().mainThread().execute(new Runnable() {
            @Override
            public void run() {
                setUpClassPicker(classEntries);
                setUpRacePicker(raceEntries,distinctRaceNames);
            }
        });
    }
});
```

Slika 4.11 *Programski kod za postavljanje classPickera i racePickera*

Na slici 4.12. prikazan je programski kod koji se izvršava pokretanjem *setUpClassPicker* metode.

```
private void setUpClassPicker(final List<ClassEntry> classEntries) {
    final List<String> classNames= new ArrayList<>();
    for(ClassEntry classEntry : classEntries){
        classNames.add(classEntry.getName());
    }
    classPicker.setDisplayedValues(classNames.toArray(new String[0]));
    classPicker.setMinValue(0);
    classPicker.setMaxValue(11);
    characterClassViewModel.getCharacterClass().observe(requireActivity(), new Observer<ClassEntry>() {
        @Override
        public void onChanged(final ClassEntry classEntry) {
            characterClassViewModel.getCharacterClass().removeObserver(this);
            requireActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    classPicker.setValue(classNames.indexOf(classEntry.getName()));
                    updateClassText(classEntry);
                }
            });
        }
    });
    classPicker.setOnValueChangedListener(new NumberPicker.OnValueChangeListener() {
        @Override
        public void onValueChange(NumberPicker picker, int oldVal, int newVal) {
            characterClassViewModel.setCharacterClass(classEntries.get(newVal));
            updateClassText(classEntries.get(newVal));
        }
    });
}
private void updateClassText(final ClassEntry classEntry) {
    primaryAbilitiesTextView.setText(classEntry.getPrimaryAbilities());
    String hitDieString = "d" + classEntry.getHitDice();
    hitDieTextView.setText(hitDieString);
}
```

Slika 4.12. *Programski kod za postavljanje classPickera*

Od svake klase uzima se njeno ime i dodaje u listu imena klasa. Dobivena imena stavljaju se kao vrijednosti za biranje u *classPickeru*. Promjenom odabrane klase ažuriraju se prikazane informacije te se obavještava *observere* koji slušaju promjenu odabrane klase. Prikaz odabrane klase nakon što se korisnik vrati na trenutni fragment zahtijeva pamćenje koje se ostvaruje *LiveData* objektom. Povratkom se učitava klasa i vrijednost *classPickera* se stavlja na ime prijašnje odabrane klase. Svaku promjenu vrijednosti prikazane na *classPickeru* sluša *observer* objekta „characterClassViewModel“ na zadnjem fragmentu te sukladno promjeni pamti odabranu klasu i ažurira korisniku trenutno nevidljive informacije. Najbitnije informacije klase su statistike koje svojim porastom više ojačaju lika u odnosu na druge te koliko izdržljivosti koja klasa nudi.

Na slici 4.13. prikazan je programski kod koji se izvršava pokretanjem promjenom vrijednosti *racePickera* tj. promjenom odabrane rase.

```
boolean flag=false;
if(raceNames.contains(distinctRaces.get(newVal))){
    subRacePicker.setVisibility(View.INVISIBLE);
    flag=true;
}
if(!flag){
    raceViewModel.getSubRaces(distinctRaces.get(newVal)).observe(getViewLifecycleOwner(), (Observer) (strings) → {
            raceViewModel.getSubRaces(distinctRaces.get(newVal)).removeObserver(this);
            subRacePicker.setVisibility(View.VISIBLE);
            subRacePicker.setDisplayedValues(strings.toArray(new String[0]));
            raceViewModel.findRace(strings.get(subRacePicker.getValue())).observe(getViewLifecycleOwner(), (Observer) (raceEntry) → {
                    raceViewModel.findRace(strings.get(subRacePicker.getValue())).removeObserver(this);
                    updateChosenRace(raceEntry);
            });
            subRacePicker.setOnValueChangedListener(new NumberPicker.OnValueChangeListener() {
                @Override
                public void onValueChange(NumberPicker picker, int oldVal, final int newVal) {
                    final String[] displayedValues=picker.getDisplayedValues();
                    raceViewModel.findRace(displayedValues[newVal]).observe(getViewLifecycleOwner(), (Observer) (raceEntry) → {
                            raceViewModel.findRace(displayedValues[newVal]).removeObserver(this);
                            updateChosenRace(raceEntry);
                    });
                }
            });
    });
} else{
    raceViewModel.findRace(distinctRaces.get(newVal)).observe(getViewLifecycleOwner(), (Observer) (raceEntry) → {
            raceViewModel.findRace(distinctRaces.get(newVal)).removeObserver(this);
            updateChosenRace(raceEntry);
    });
```

Slika 4.13. *Programski kod za ažuriranje podataka vezanih za odabranu rasu*

Problem kod postavljanja *NumberPickera* za rasu je činjenica da postoje rase s podrasama. Zbog toga su za neke klase potrebna dva *NumberPickera*. Pamćenje odabrane rase napravljen je na identični način kao i kod pamćenja klase. Odabiranjem rase ispituje se sadrži li ta rasa podrase te se postavlja zastavica. Ako sadrži podrase, prikazuje se dodatni *subRacePicker* koji omogućuje njen odabir. Ako rasa ne sadrži podrase, *subRacePicker* se sakriva od korisnika. Odabiranjem nove rase se ažurira rasa lika i informacije koje su korisniku u trenutku bitne. Na temelju danih informacija, korisnik može odabrati kombinaciju klase i rase koje si međusobno pašu.

Fragment za odabir statistika i pozadine

Uz *classPicker* i *racePicker* postoji i *backgroundPicker* koji služi za odabir pozadine lika. Njegova funkcionalnost ostvarena je na način vrlo sličan *classPickera*. Promjena vrijednosti koju pokazuje *backgroundPicker* mijenja odabranu pozadinu što uzrokuje promjenu prikazanih informacija. Na slici 4.13. prikazana je metoda za postavljanje *NumberPickera* koja se zove za svaku statistiku.

```
    setUpPicker(strengthPicker,ID_STRENGTH);
    setUpPicker(dexterityPicker,ID_DEXTERITY);
    setUpPicker(constitutionPicker,ID_CONSTITUTION);
    setUpPicker(wisdomPicker,ID_WISDOM);
    setUpPicker(intelligencePicker,ID_INTELLIGENCE);
    setUpPicker(charismaPicker,ID_CHARISMA);

}

private void setUpPicker(final NumberPicker picker, final int picker_ID){
    picker.setDisplayedValues(new String[]{"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16","17","18","19","20"});
    picker.setMinValue(1);
    picker.setMaxValue(20);
    picker.setWrapSelectorWheel(false);

    characterCreationViewModel.getPlayerCharacter().observe(getViewLifecycleOwner(), (Observer) (playerCharacterEntry) -> {
            characterCreationViewModel.getPlayerCharacter().removeObserver(this);
            switch(picker_ID){
                case 1: picker.setValue(playerCharacterEntry.getStrength());
                    picker.setOnValueChangedListener((picker, oldVal, newVal) -> {
                            playerCharacterEntry.setStrength(newVal);
                    });
                    break;
                case 2:
```

Slika 4.13. *Programski kod metode za postavljanje NumberPickera statistika*

Za biranje vrijednosti statistika, korisniku je dano šest *NumberPickera* koji ovisno o identifikacijskom broju postavljaju određenu statistiku na odabranu vrijednost. Vrijednosti su ograničene od minimalne vrijednosti 1, do maksimalne vrijednosti 20.

Fragment za odabir vještina, alata, instrumenata i jezika

Kako bi se znalo koliko i koje vještine, alate i jezike korisnik smije odabrati, moraju se znati klasa, rasa i pozadina lika. Svaki od tih parametara znatno utječe na zadnji fragment tako da je postavljen *observer* na svaki od prijašnje kreiranih *NumberPickera* za te parametre. Na slici 4.14. prikazan je programski kod za kreiranje *observera* koji sluša promjenu odabrane klase na prvom fragmentu.

```
if(classEntryObserver==null){
    characterClassViewModel.getCharacterClass().observe(requireActivity(), classEntryObserver=(Observer) (classEntry) → {
            numberOfClassTools=classEntry.getNumberOfTools();
            numberOfClassMusic=classEntry.getNumberOfMusic();
            numberOfAvailableSkills=classEntry.getNumberOfSkills();
            if(classEntry.getToolProficiency()!=null){
                classTools=classEntry.getToolProficiency();
            }
            setUpToolsData();
            setUpMusicData();
            startingEquipment=changeStartingEquipment(classEntry);

            characterBackgroundViewModel.getCharacterBackground().observe(requireActivity(), (Observer) (backgroundEntry) → {
                    characterCreationViewModel.setCharacterSkills(backgroundEntry.getSkills());
                    setUpSkillsData(backgroundEntry);
            });
    });
}
```

Slika 4.14. *Programski kod observera za parametar klase*

Kao što je prije navedeno, *ViewModel* i *LiveData* imaju vlastiti ciklus koji ne ovisi o promjeni fragmenata. Tako postavljena struktura znatno poboljšava korisničko iskustvo tijekom navigacije kroz kreaciju lika. Tijekom stvaranja zadnjeg fragmenta, kreiraju se tri *observera* koja prate promjenu rase, klase i pozadine. Odlaskom na prvi fragment te promjenom klase, mijenjaju se informacije potrebne na zadnjem fragmentu i dok korisnik dođe na njega, promjene su već ostvarene. Ovisno o glavnim parametrima, korisniku je prikazano i ponuđeno na odabir što i koliko smije odabrati. Na sljedećoj slici 4.15. prikazan je programski kod primjera postavljanja gumba za biranje jezika. Uz njega, postoje još i gumb za biranje vještina, alata i instrumenta.

```
mBuilder.setMultiChoiceItems(listLanguages, checkedLanguages, (dialog, which, isChecked) → {
        if(isChecked){
            if(languageCounter>=numberOfBackgroundLanguages+numberOfRaceLanguages){
                checkedLanguages[which]=false;
                characterCreationViewModel.getPlayerCharacter().observe(getViewLifecycleOwner(), (Observer) (playerCharacterEntry) → {
                        characterCreationViewModel.getPlayerCharacter().removeObserver(this);
                        for(int i=0;i<checkedLanguages.length;i++){
                            if(checkedLanguages[i]){
                                if(!languagesSelected.contains(listLanguages[i])){
                                    languagesSelected.add(listLanguages[i]);
                                }
                            }
                        }
                        playerCharacterEntry.setLanguages(new ArrayList<>(languagesSelected));
                        languagesTextView.setText(StringListConverter.toString(languagesSelected));
                });
                dialog.dismiss();
            }
            else{
                languagesSelected.add(listLanguages[which]);
                languageCounter++;
            }

        }else{
            languageCounter--;
            languagesSelected.remove(listLanguages[which]);

        }
    });
mBuilder.setPositiveButton("OK", (dialogInterface, which) → {
        characterCreationViewModel.setCharacterLanguages(new ArrayList<>(languagesSelected));
        languagesTextView.setText(StringListConverter.toString(languagesSelected));
});

mBuilder.setNegativeButton("Dismiss", (dialogInterface, i) → {
        characterCreationViewModel.getPlayerCharacter().observe(getViewLifecycleOwner(), (Observer) (playerCharacterEntry) → {
                characterCreationViewModel.getPlayerCharacter().removeObserver(this);
                languagesSelected=new ArrayList<>(playerCharacterEntry.getLanguages());
                checkedLanguages=new boolean[checkedLanguages.length];
        });
        dialogInterface.dismiss();
});
```

Slika 4.15. *Programski kod za postavljanje gumba za odabir jezika*

Gumbovi rade tako da kreiraju *AlertDialog* koji pomoću *MultiChoiceItems* liste dodaje liku odabrano svojstvo. Nakon klika na gumb, korisniku se pojavi novi prozor s listom mogućih odabira i kučice koje označavaju stanja odabira. Bilo je potrebno ograničiti korisnika da može odabrati samo onoliko koliko mu je dozvoljeno te da se promjenom glavnih parametara resetiraju odabiri kako ne bi imao nedozvoljene mogućnosti. Također je trebalo pamtiti odabire odlaskom iz fragmenta te ih prikazati kao odabranima pri povratku. Uz gumb za potvrdu odabira, postavljen je gumb za odustajanje od izmjene te gumb za poništenje odabira i ponovnog biranja. U slučaju da korisnik ne smije birati niti jedno od opcija, gumb mu se ni ne prikazuje.

Nakon što je korisnik zadovoljan odabirom svih parametara koji su mu bili ponuđeni, klikom na krajnji gumb, objekt klase „PlayerCharacterEntry“ se nadopunjava dodatnim informacijama te se sprema u bazu podataka za daljnje korištenje. Kao povratna vrijednost funkcije umetanja lika u bazu, vraća se njegov identifikacijski broj te kreira i s njim povezuje njegova početna oprema. Na slici 4.16. se nalazi programski kod kojim se objekt lika nadopunjuje krajnjim informacijama, sprema u bazu podataka te se novo kreirana početna oprema povezuje s njim.

```
        playerCharacterEntry.setCharacterClass(classEntry.getName());
        playerCharacterEntry.setRace(raceEntry.getName());
        for(String skill : raceEntry.getAbilityIncrease()){
            if(skill.contains("ALL")){...}
            else{
                int abilityIncrease = Integer.parseInt(String.valueOf(skill.charAt(1)));
                if(skill.contains("STR")){
                    int strength=playerCharacterEntry.getStrength();
                    playerCharacterEntry.setStrength(strength+abilityIncrease);
                }
                if(skill.contains("DEX")){...}
                if(skill.contains("CON")){...}
                if(skill.contains("INT")){...}
                if(skill.contains("WIS")){...}
                if(skill.contains("CHA")){...}
            }
        }
});
playerCharacterEntry.setCharacterLevel(1);
playerCharacterEntry.setExperience(0);
playerCharacterEntry.setArmorClass(10);
playerCharacterEntry.setHitPoints(classEntry.getHitDice());
AppExecutors.getInstance().diskIO().execute(() -> {
        final long newPlayerID=mdB.playerCharacterDao().insertWithID(playerCharacterEntry);
        for(OwnedItemEntry ownedItemEntry:startingEquipment){
            OwnedItemEntry newOwnedItem=new OwnedItemEntry(ownedItemEntry.getName(),ownedItemEntry.getType(),(int) newPlayerID);
            mdB.ownedItemDao().insert(newOwnedItem);
        }
});
```

Slika 4.16. *Programski kod za završetak kreiranja lika*

4.3.3. Interakcija s likom

Dohvaćanje odabranog lika i prikaz informacija

Nakon što postoji bar jedan lik, može ga se odabrati te nastaviti na aktivnost gdje se prikazuju detaljnije sve njegove informacije. Na početnom zaslonu aplikacije su ponuđeni svi kreirani likovi, nakon klika na jednog od njih, stvara se novi *Intent* uz kojeg se šalje informacija o identifikacijskom broju odabranog lika. Odabirom se na novoj aktivnosti taj broj koristi za dohvaćanje odgovarajućeg lika iz baze podataka. Na temelju odabranog lika i njegovih glavnih parametara prikazuju se sve informacije potrebne za daljnju igru. Na slikama 4.17. i 4.18. prikazan

je programski kod kojim se kreira nova aktivnost s dodanim identifikacijskim brojem i njegovo kasnije dohvaćanje.

```
@Override
public void onItemClickListener(int characterID) {
    Intent intent = new Intent( packageContext: ChooseCharacterActivity.this, CharacterActivity.class);
    intent.putExtra(CharacterActivity.CHARACTER_ID, characterID);
    startActivity(intent);
}
```

Slika 4.17. *Programski kod za kreiranje intenta s ID-jem odabranog lika*

```
mCharacterID = intent.getIntExtra(CHARACTER_ID, DEFAULT_CHARACTER_ID);
PlayerCharacterViewModelFactory factory = new PlayerCharacterViewModelFactory(mDb, mCharacterID);
```

Slika 4.18. *Programski kod za dohvaćanje odgovarajućeg lika*

Na slici 4.18. prikazan je objekt klase „PlayerCharacterViewModelFactory“. Navedena klasa je kreirana kako bi se na temelju identifikacijskog broja mogao dohvatiti objekt odgovarajućeg lika.

Prikaz i interakcija s opremom

Na fragmentu s opremom, prikazana su četiri *ImageButttona* koji klikom otvaraju novu aktivnost ovisno o tome na kojeg se kliknulo. Gumbovi predstavljaju svaki tip opreme tj. oružja, oklop, štit i ostalo. Ulaskom u jedno od aktivnosti se prikazuju svi objekti klase „OwnedItemEntry“ koji na atributu „ownerID“ imaju vrijednost atributa „id“ objekta klase „PlayerCharacterEntry“ te su određenog tipa opreme. Kako objekti klase „OwnedItemEntry“ nisu sami po sebi objekti koje lik posjeduje, već samo veza s njima, trebaju se dohvatiti odgovarajući objekti koje lik zapravo posjeduje. Takva pretraga se radi po imenu objekata te je prikazana na sljedećoj slici 4.19.

```
playerCharacterViewModel.getOwnedWeapons(playerCharacterID).observe( owner: ItemsActivity.this, new Observer<List<OwnedItemEntry>>() {
    @Override
    public void onChanged(List<OwnedItemEntry> ownedItemEntries) {
        playerCharacterViewModel.getOwnedWeapons(playerCharacterID).removeObserver(this);
        final List<String> ownedWeaponNames=new ArrayList<>();
        ownedItems=ownedItemEntries;
        for(OwnedItemEntry ownedItemEntry: ownedItemEntries){
            ownedWeaponNames.add(ownedItemEntry.getName());
        }
        weaponViewModel.getOwnedWeapons(ownedWeaponNames).observe( owner: ItemsActivity.this, (Observer) (weaponEntries) -> {
                weaponViewModel.getOwnedWeapons(ownedWeaponNames).removeObserver(this);
                List<WeaponEntry> shownWeapons=new ArrayList<>();
                for(String ownedWeaponName:ownedWeaponNames){
                    for(WeaponEntry weaponEntry:weaponEntries){
                        if(ownedWeaponName.equals(weaponEntry.getName())){
                            shownWeapons.add(weaponEntry);
                        }
                    }
                }
                mWeaponAdapter.setWeaponEntries(shownWeapons);
        });
    }
});
```

Slika 4.19. *Programski kod za dohvaćanje oružja određenog lika*

Iako objekti koji su prvobitno traženi i oni koji su prikazani nisu isti objekti, brisanje nije problem. Brisanjem određenog oružja dohvaća se ID tog oružja, čime se dohvaća i ID objekta klase „OwnedItemEntry“. Nakon brisanja samo je potrebno ažurirati listu prikazanih oružja i njome ažurirati adapter. Analogno vrijedi za dodavanje. Na slici 4.20. prikazan je programski kod pomoću kojeg se brišu oružja.

```
    @Override
    public void onSwiped(@NonNull final RecyclerView.ViewHolder viewHolder, int swipeDir) {
        AppExecutors.getInstance().diskIO().execute(() -> {
                final int position = viewHolder.getAdapterPosition();
                final List<WeaponEntry> currentWeapons=mWeaponAdapter.getWeapons();
                mDb.ownedItemDao().delete(ownedItems.get(position));
                runOnUiThread(() -> {
                        currentWeapons.remove(position);
                });

        });
    }
}).attachToRecyclerView(mRecyclerView);
```

Slika 4.20. *Programski kod za brisanje „OwnedItemEntry“ objekta iz baze*

Bacanje kockica

Fragment za bacanje kockica sadrži sedam različitih *ImageButtona* koji klikom kreiraju objekt kockice s određenim brojem strana i *PopupWindow* koji daje informacije o ishodu bacanja. Ishod bacanja ovisi o bačenoj kockici, količini bačenih kockica i *modifier* bodovima koji predstavljaju dodatne bodove za bacanje. Broj kockica i *modifier* bodovi se odabiru ponuđenim *NumberPickerima*. Na slici 4.21. prikazan je programski kod koji se izvrši svaki put kada se klikne na jedan od *ImageButtona.*

```
private void setUpDieOnClickListener(int numberOfSides,View v){
    Die die = new Die(numberOfSides);
    ArrayList<Integer> rolls = getRolls(die);
    PopupRoll popupRoll = new PopupRoll( rollDice: getDiePickerValue()+"d"+die.getNumberOfSides()+getModifierString(),
            getRollResult(rolls),
            getRollsAsString(rolls));
    popupRoll.showPopupWindow(v);
}
```

Slika 4.21. *Metoda kojom se bace kockice i prikaže rezultat*

5. NAČIN KORIŠTENJA I ANALIZA PROGRAMSKOG RJEŠENJA

5.1. Način korištenja mobilne Android aplikacije

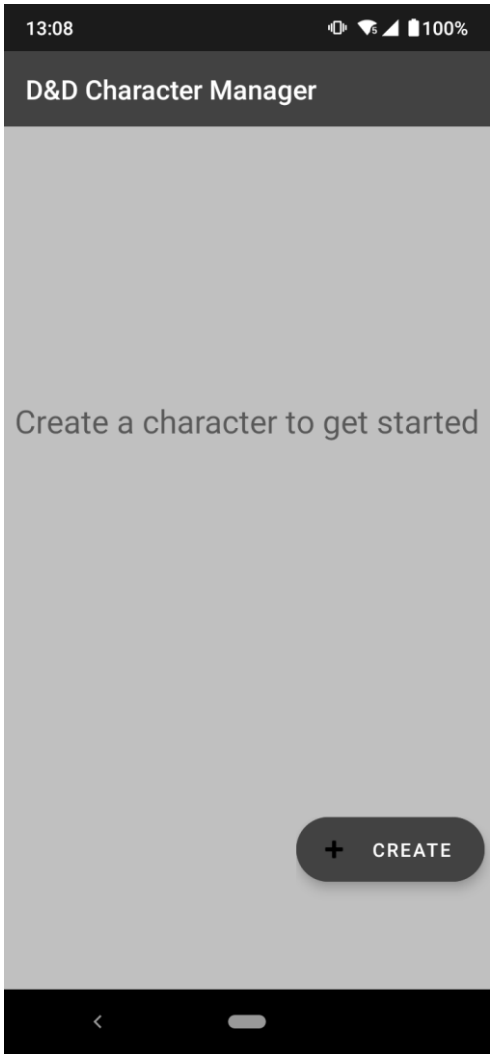
Aplikacija je podijeljena na dva glavna dijela te međukorakom koji ih povezuje. Pokretanjem aplikacije korisniku se na zaslon prikazuju kreirani likovi ako postoje i tipka za kreiranje novog lika. Klikom na gumb se otvara nova aktivnost s navigacijom kroz tri fragmenta kroz koje se koracima konstruira lik. Korisnik imenuje lika, odabire mu glavne parametre tijekom čega mu aplikacija pomaže pružanjem bitnih informacija. Na zadnjem fragmentu korisnik odabire zadnje karakteristike svog lika te se klikom na gumb lik sprema u *Room* bazu podataka. Korisnika se vraća na početni zaslon s ažuriranom listom koja prikazuje do sad kreirane likove s dodatkom novog lika. Klikom na jednog od likova, korisniku se otvara nova aktivnost s tri fragmenta gdje početni fragment prikazuje detaljne informacije o liku pomoću kojih se može igrati društvena igra. Odlaskom na desni fragment, korisniku je prikazan crtež lika gdje se klikom na mač ili neki drugi dio opreme korisnika vodi na listu stvari određenog tipa koje posjeduje. Za svaki objekt kojeg lik posjeduje, prikazane su potrebne informacije. Unutar te aktivnosti, korisnik može dodavati ili brisati objekte te se sukladno tome ažuriraju stanje baze podataka i korisničko sučelje. Klikom na gumb za povratak, korisnik se vraća na prvobitna tri fragmenta gdje odlaskom na prvi može bacati kockice. Kockice su prezentirane crtežom te klikom na njih se korisniku pojavljuje prozor s rezultatima bacanja kockice. Korisnik lakoćom može kontrolirati količinu kockica koje baca i dodatne bodove koji se primjenjuju na rezultat bacanja. Kao progres tijekom igranja s određenim likom, korisnik mu može dodavati bodove iskustva. Povećanjem iskustva liku raste nivo. Svakim nivoom liku je potrebno više iskustva za sljedeći te je jači u usporedbi s prijašnjim.

5.2. Primjeri korištenja mobilne aplikacije

U svrhu testiranja aplikacije kreirat će se dva lika te ispitati funkcionalnosti aplikacije. Prvim pokretanjem aplikacije unose se podaci bitni za kreiranje likova te će lista kreiranih likova biti prazna. Prvim likom će se provjeriti slijednost aplikacije te njena sposobnost da pomogne pri kreiranju lika. Također će se provjeriti funkcionalnost ažuriranja baze podataka i korisničkog sučelja pri brisanju i dodavanju opreme te će biti ispitana situacija bacanja kockice. Kreiranje drugog lika će biti testiranje mogućih unosa u aplikaciju, može li se ikako postići nešto izvan pravila.

5.2.1. Kreiranje prvog lika

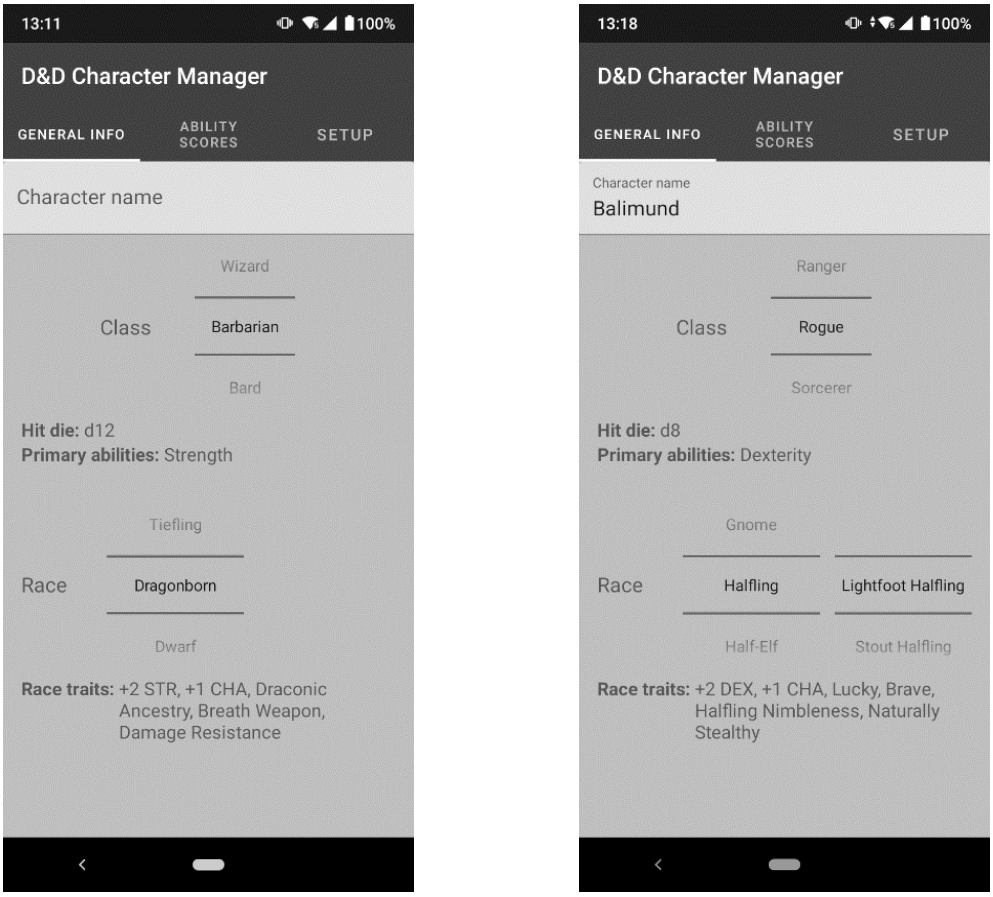
Proces kreiranja i njegova svrha su objašnjeni u prijašnje napisanom tekstu. Pokretanjem aplikacije se baza podataka treba inicijalno popuniti. Potvrda će biti vidljiva odmah tijekom kreiranja lika. Na slici 5.1. prikazan je početni zaslon aplikacije.



Slika 5.1. *Sučelje početnog zaslona aplikacije*

Aplikacija korisniku daje informaciju da ne postoji niti jedan lik s kojim se može igrati te ga se treba kreirati za nastavak.

Slika 5.2. prikazuje sučelje prvog fragmenta kreiranja lika i promjenu prikazanih informacija nakon promjene odabira klase i rase.



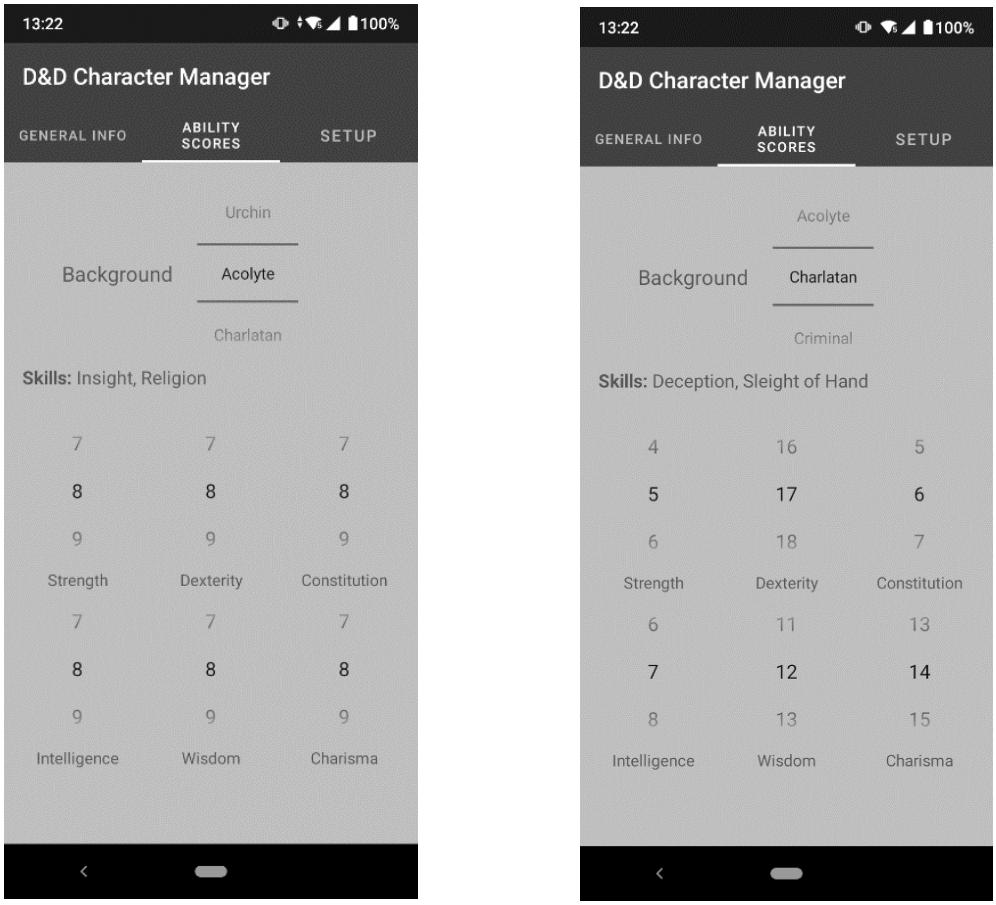
a) b)

Slika 5.2. *Početni fragment kreiranja lika i prikaz informacija nakon promjene*

Na slici 5.2. a) vidljivo je kako je inicijalno popunjavanje baze podataka uspješno tako što korisnik ima imena klasa i rasa te odgovarajuće informacije o odabranim.

Na slici 5.2. b) prikazano je kako se odabiranjem klase ažuriraju podaci navedeni pod „Hit die“ i „Primary abilities“ te se odabiranjem rase ažuriraju „Race traits“ podaci. Također je vidljivo kako rasa „Halfling“ ima dvije podrase, „Lightfoot Halfling“ i „Stout Halfling“ te se u tom slučaju pojavljuje *subRacePicker* koji omogućuje i odabir podrase.

Slika 5.3. prikazuje sučelje drugog fragmenta kreiranja lika i promjenu prikazanih informacija nakon promjene odabira pozadine.

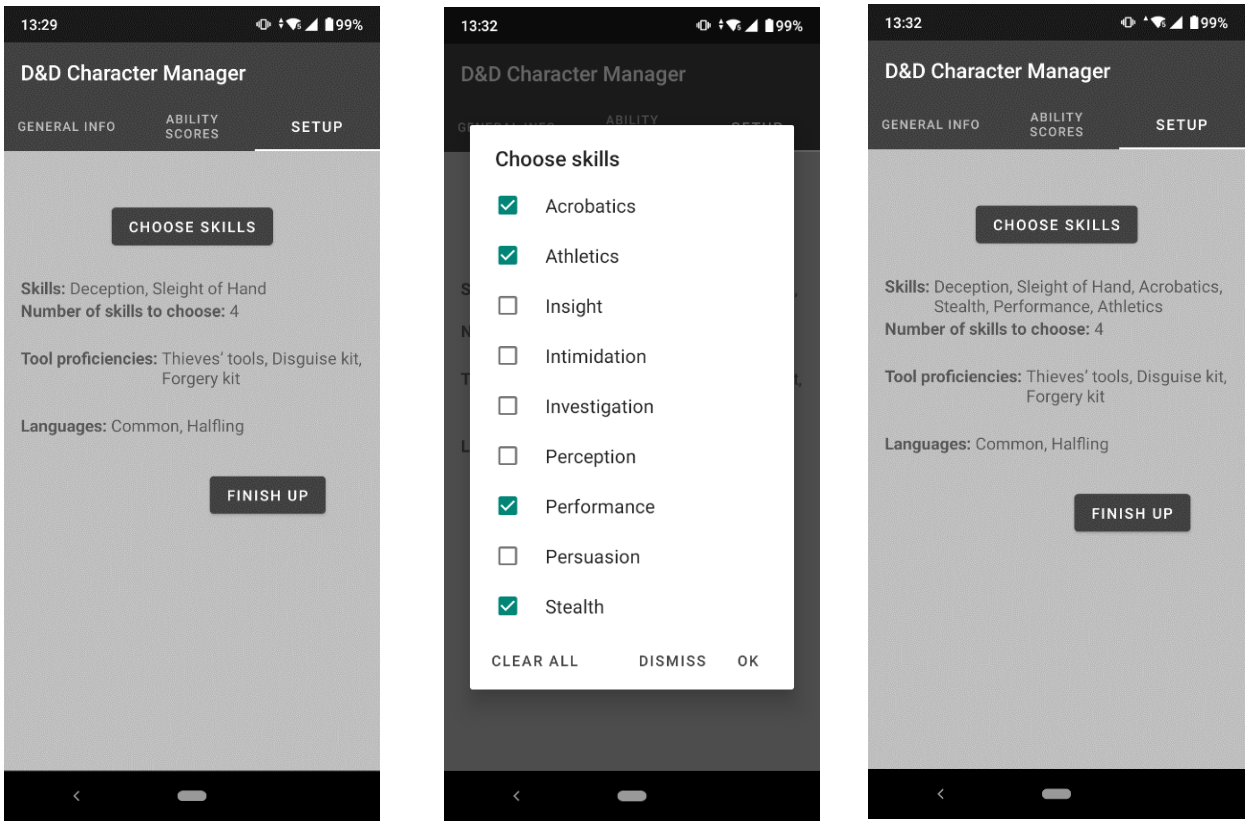


a) b)

Slika 5.3. *Početno stanje drugog fragmenta i prikaz informacija nakon promjene*

Na slici 5.3. b) vidljivo je kako se promjenom odabrane pozadine ažuriraju vještine. U slučaju zaborava kojim statistikama se treba posvetiti, povratkom na prošli fragment korisniku su prikazane informacije.

Na slici 5.3. prikazano početno stanje sučelja zadnjeg fragmenta, prozor koji se otvori klikom na gumb „Choose Skills“ i kako izgledaju označeni odabiri, te stanje zadnjeg fragmenta nakon što se odaberu određene vještine.



a) b) c)

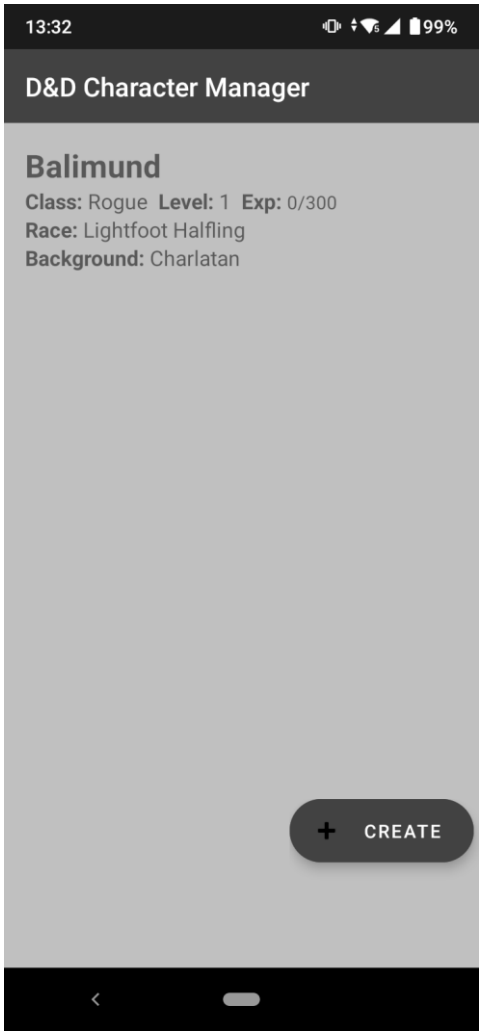
Slika 5.3. *Prikaz zadnjeg fragmenta*

Na slici 5.3. a) prikazane su početne informacije temeljene na odabranim parametrima klase, rase i pozadine. Izmjenom jednog od tih parametara se mijenjaju informacije prikazane na ovom fragmentu.

Na slici 5.3. b) prikazan je prozor koji se otvori nakon što korisnik stisne gumb „Choose Skills“. Da je korisnik uzeo druge parametre klase, rase i pozadine, možda bi imao i gumbove za odabir alata i jezika, no u ovom slučaju dodatni alati i jezici nisu dozvoljeni. Odlaskom s ovog fragmenta i kasnijim povratkom te klikom na gumb, odabiri će ostati isti. Klikom na „Clear All“ se brišu svi odabiri, a klikom na „Dismiss“ se poništavaju promjene napravljene nakon otvaranja prozora za odabir.

Na slici 5.3. c) prikazano je stanje nakon što je korisnik kliknuo na „OK“ i time potvrdio svoj odabir vještina.

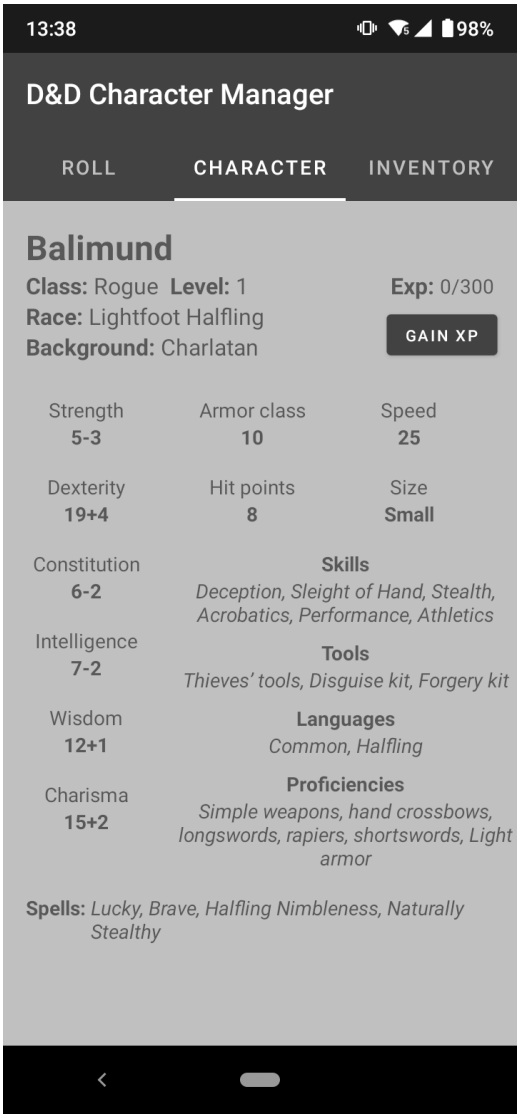
Klikom na gumb „Finish up“ aplikacija sprema kreiranog lika i vodi korisnika na sučelje prikazano na slici 5.4.



Slika 5.4. *Ažurirani početni zaslon*

Nakon kreiranja i spremanja lika otvara se početni zaslon aplikacije s dodanim likom. Progres s tim likom vidljiv je preko informacija o njegovom nivou i bodovima iskustva. Dodatno su prikazane informacije o odabranoj klasi, rasi i pozadini.

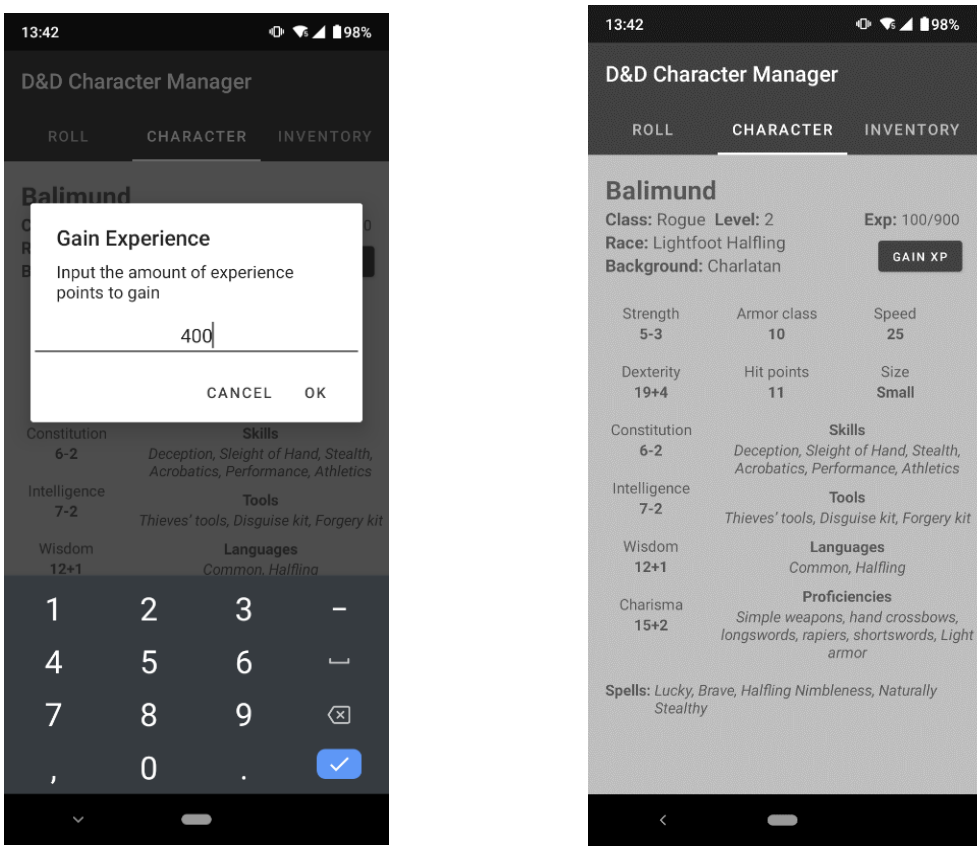
Klikom na napravljenog lika, baza podataka se pretraži i dohvati se odabrani lik. Otvara se nova aktivnost s detaljnijim prikazom o njegovim informacijama što je vidljivo na slici 5.5.



Slika 5.5. *Detaljnije informacije o odabranom liku*

Klikom na lika, korisnika se vodi na novu aktivnost s nova tri fragmenta od kojih je početni fragment s detaljnije napisanim informacijama o liku. Iz ovih se informacija može zaključiti uz koje dodatne bodove se mogu bacati kockice. Uz to, korisnik zna kojom se brzinom njegov lik kreće, koliko je velik, koliko ima života, koje sve sposobnosti ima i koja mu je vrijednost obrane. Također je ponuđen gumb „Gain XP“ koji klikom otvara novi *AlertDialog* za unos novih bodova iskustva.

Na slici 5.6. prikazan je prozor koji se otvara klikom na „Gain XP“ gumb, te što se mijenja u fragmentu lika porastom nivoa.



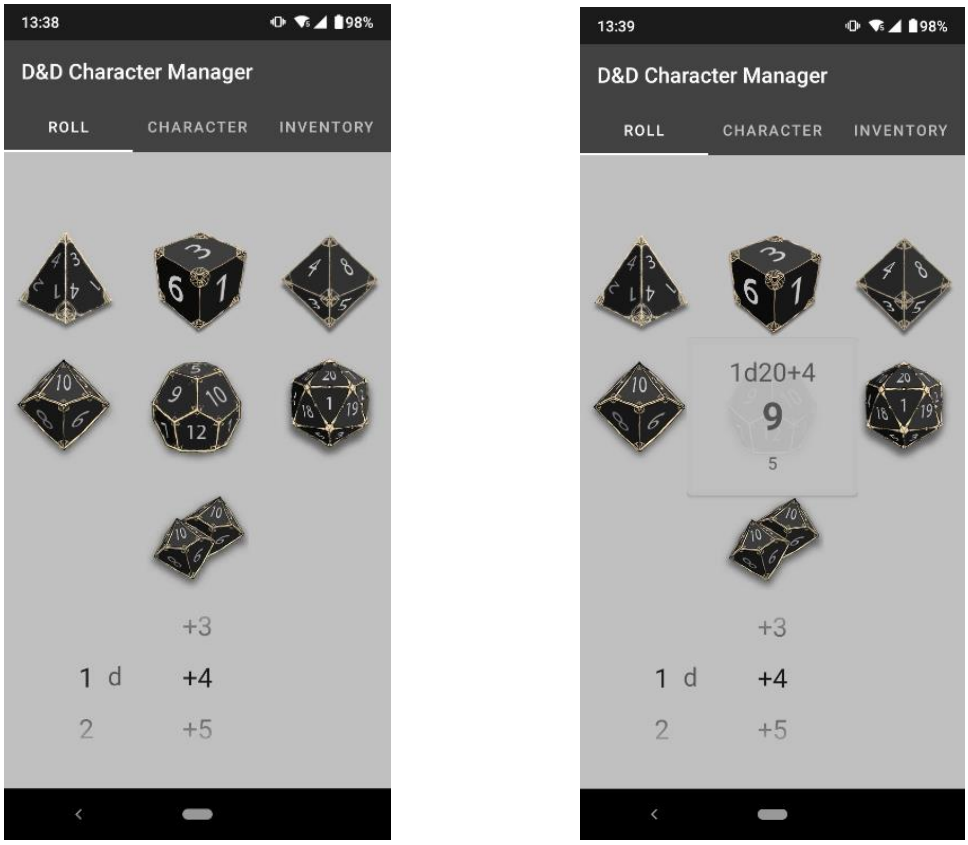
a) b)

Slika 5.6. *AlertDialog gumba „Gain XP“ i stanje fragmenta nakon unosa bodova*

Na slici 5.6. a) vidljivo je kako se otvaranjem prozora za unos bodova iskustva odmah otvara tipkovnica prilagođena za predviđeni unos. Unosom bodova iskustva te klikom na „OK“ se tipkovnica spušta i informacije o liku se ažuriraju.

Na slici 5.6. b) prikazano je stanje lika nakon porasta bodova iskustva. Sa slike 5.5. vidljivo je kako je bilo potrebno 300 bodova iskustva, te prijelazom preko te granice se trenutni bodovi umanjuju, nivo se povećava kao i „Hit Points“ vrijednost.

Slika 5.7. prikazuje sučelje fragmenta u kojem je omogućeno bacanje kockica, te prozor na kojem je prikazan rezultat bacanja.



a) b)

Slika 5.7. *Sučelje fragmenta s kockicama i PopupWindow s rezultatima bacanja*

Odlaskom na lijevi fragment omogućeno je bacanje kockica. U slučaju da se želi baciti 1d20 kockica u svrhu vještine u kojoj trenutno odabrani lik nije stručan, informacijom sa slike 5.5. vidljivo je da dodatni bodovi za tu akciju iznose +4.

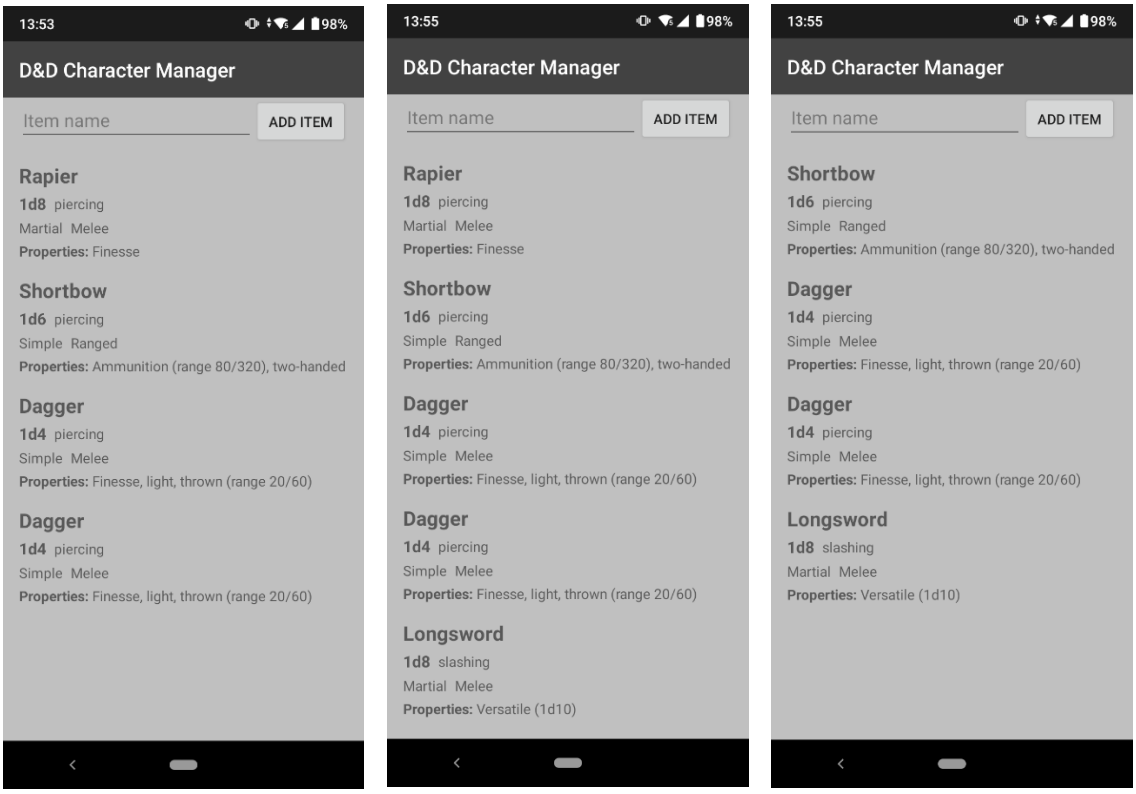
Na slici 5.7. a) prikazane su kockice različitih strana. Klikom na jedne od njih baca se onaj broj kockica koji je na prvom *NumberPickeru* uz dodatne bodove očitane s drugog *NumberPickera*. Rezultat bacanja kockica pojavi se na prozoru prikazanom na slici 5.7. b).

Odlaskom na najdesniji fragment pojavljuje se fragment čije je sučelje prikazano na slici 5.8.

Slika 5.8. *Sučelje fragmenta za provjeru opreme*

Na navedenom fragmentu se nalazi crtež lika. Klikom na jedan od uokvirenih pravokutnika pokreće se nova aktivnost gdje se prikazuju posjedovani objekti tog tipa. Način dohvaćanja opreme točno od odabranog lika je objašnjen u prijašnjem poglavlju.

Klikom na crtež mača otvara se nova aktivnost na kojoj je prikazana lista oružja u vlasništvu odabranog lika, što je prikazano na slici 5.9.



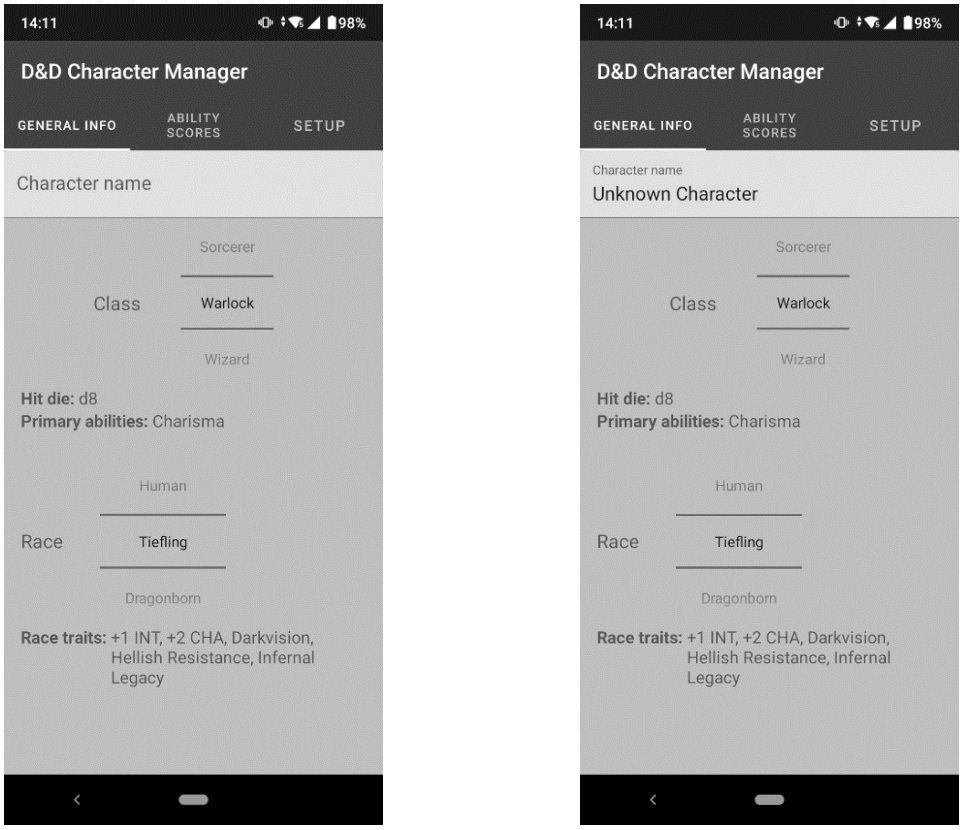
Slika 5.9. *Aktivnost koja prikazuje oružja u vlasništvu odabranog lika te stanje nakon brisanja ili dodavanja*

Na slici 5.9. b) prikazano je ažurirano stanje aktivnosti nakon dodavanja objekta „Longsword“. Dodavanjem se i sučelje i baza podataka automatski ažuriraju. Analogno vrijedi i za brisanje objekata što je vidljivo u slici 5.9. c) brisanjem objekta „Rapier“.

5.2.2. Kreiranje drugog lika

Kreiranjem drugog lika ispituju se granice unosa i pamćenja aplikacije i provjerava se hoće li što dovesti do njenog iznenadnog rušenja.

Na slici 5.10. prikazano je stanje sučelja prvog fragmenta aktivnosti kreiranja lika, nakon ispitivanja ponašanja aplikacije u slučaju ostavljanja polja imena praznim.

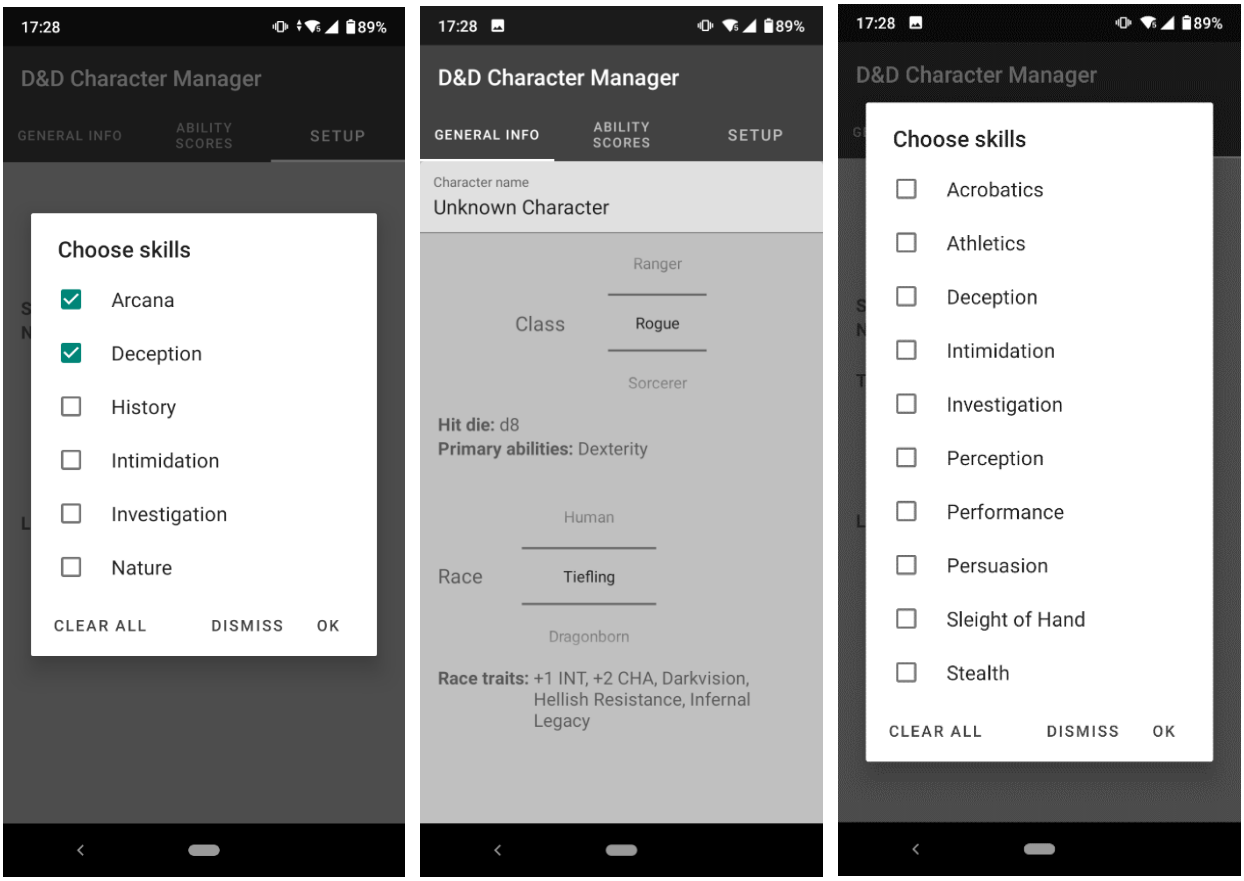


a) b)

Slika 5.10. *Unos praznog Stringa za ime lika te stanje nakon povratka na fragment*

Odlaskom s fragmenta bez unosa imena te kasnije vraćanje na prvi fragment se liku automatski dodjeljuje ime „Unknown Character“, što je vidljivo na slici 5.10. b).

Ispitivanje aplikacije u slučaju pokušaja odabira nedozvoljenih vještina ostvareno je i prikazano na slici 5.11.

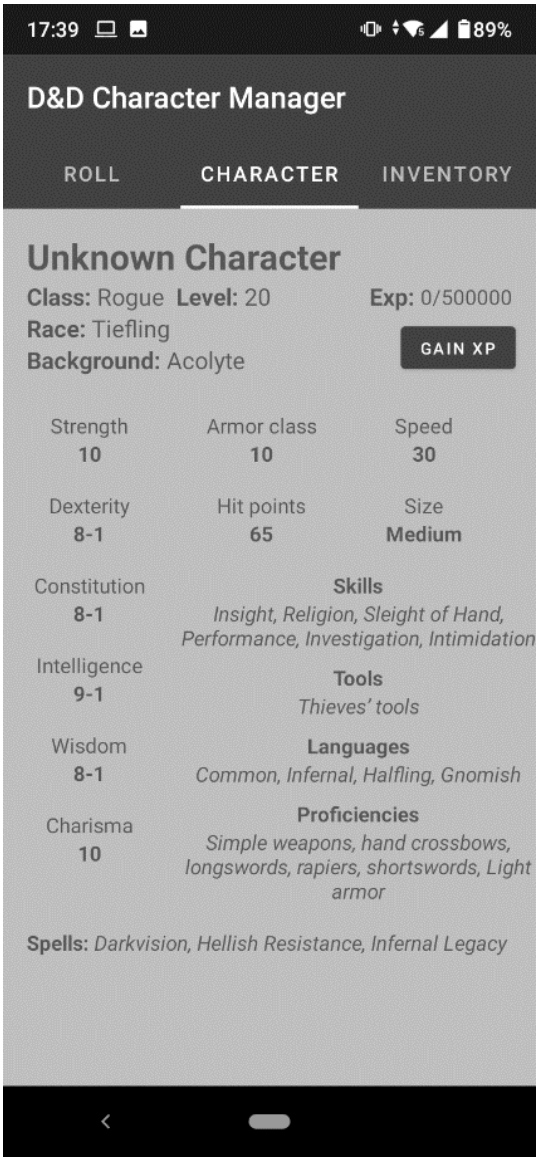


a) b) c)

Slika 5.11. *Odabir vještina, izmjena klase, te prikaz novog izbora vještina*

Slikom 5.11. a) prikazano je trenutno stanje odabranih vještina koje ostaju zapamćene tokom kretanja između fragmenata. Promjenom odabrane klase kao što je prikazano na slici 5.11. b) od aplikacije se očekuje da se odabiri obrišu i da se postavi nova lista dozvoljenih odabira. Navedeno je potvrđeno slikom 5.11. c).

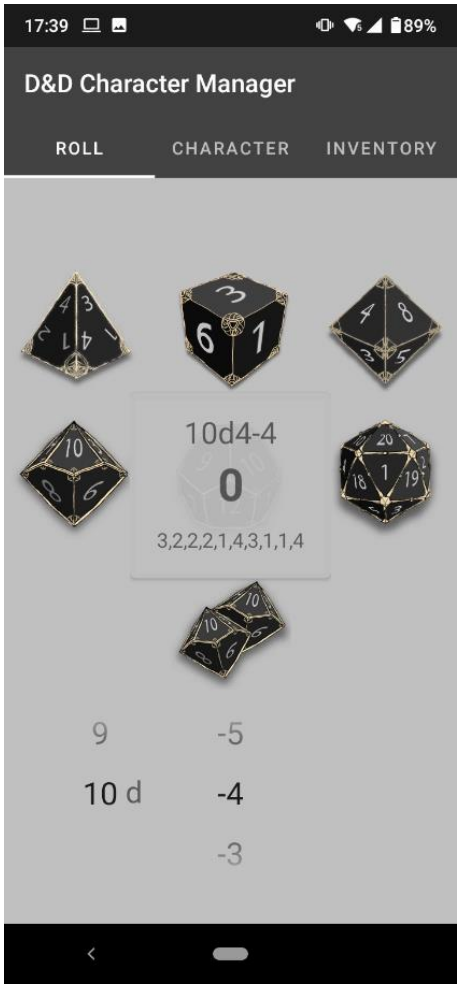
Pokušaj prelaska na nivo 21 je isproban i ponašanje aplikacije u tom slučaju prikazano je na slici 5.12.



Slika 5.12. *Ograničenje unosa novih bodova iskustva nakon nivoa 20*

Društvena igra *Dungeons and Dragons* dozvoljava liku da bude maksimalno nivo 20 bilo koje klase. Pokušavanjem dodavanja dodatnih bodova iskustva nakon prelaska 20. nivoa stanje bodova iskustva konstantno ostaje na 0.

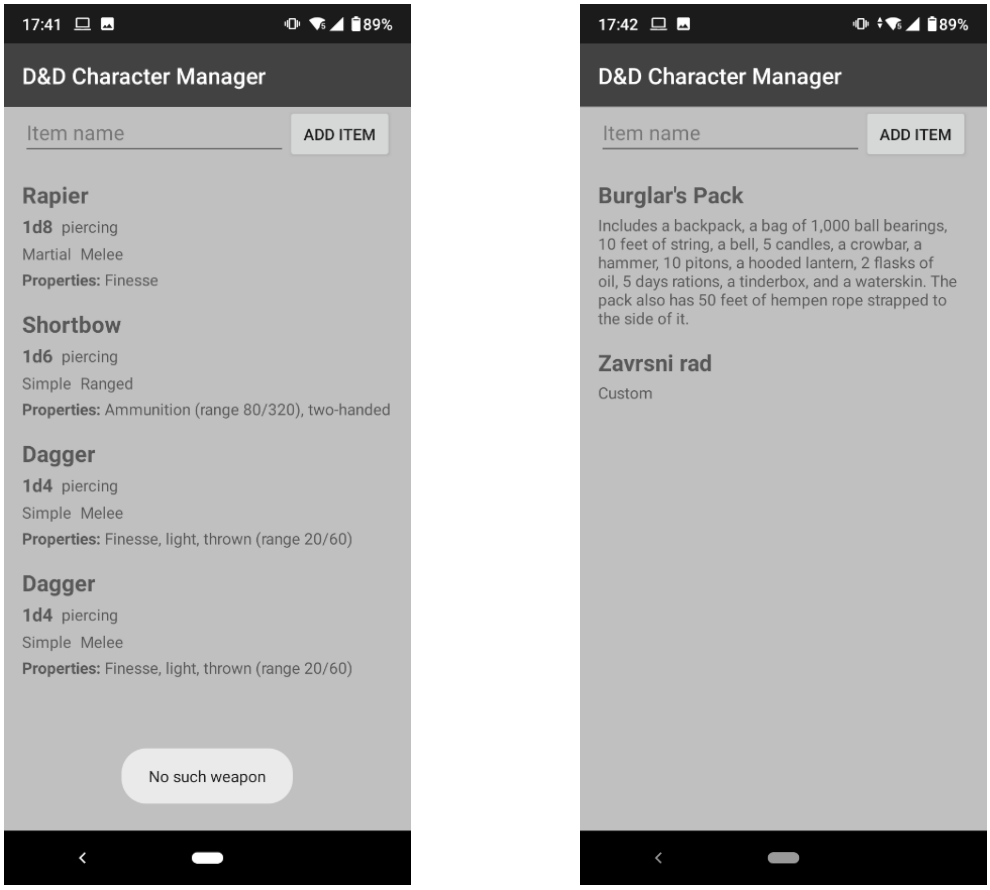
Ispitivanje aplikacije u slučaju da korisnik baca kockicu niskog broja s visokim negativnim bodovima prikazano je na slici 5.13.



Slika 5.13 *Ispitivanje pojave negativnih brojeva na kockicama*

U društvenoj igri *Dungeons and Dragons* minimalna vrijednost koja se može dobiti na kockici je nula. Bez obzira na to koja se kockica baca uz koliko negativne dodatne bodove, ne može se dobiti negativan rezultat bacanja.

Mogućnost dodavanja personaliziranih objekata je ispitano i prikazano na slici 5.14.



a) b)

Slika 5.14. *Dodavanje nepostojanog oružja i dodavanje personaliziranih stvari*

Odlaskom na aktivnost u kojoj se prikazuju oružja u vlasništvu određenog lika, te pokušavanje dodavanja oružja koje ne postoji u bazi podataka rezultira u poruci da takvo oružje ne postoji. Navedeni ishod prikazan je slikom 5.14. a).

Unos personaliziranih objekata omogućen je samo u aktivnosti gdje su prikazani alati i ostale stvari, tj. klikom na ruksak na crtežu lika. Unosom bilo kakvog imena, objekt se pojavljuje uz opis „Custom“ označujući da je objekt stvoren od strane korisnika, što je vidljivo na slici 5.14. b).

5.3. Analiza rada mobilne aplikacije

Iz scenarija kreiranja prvog lika gdje korisnik poduzima predviđene korake, vidljivo je da aplikacija obavlja svoj cilj. Pri prvom kreiranju lika, baza podataka u sebi već ima učitane podatke. Odabirom lika nakon njegovog kreiranja učitavaju se svi ispravni podaci. Iskustvo prelaskom granice pravilno ažurira nivo lika. Početna oprema je bila spremljena kako je i predviđeno. Razmatranjem drugog scenarija vidljivo je da su ograničenja na unose uzeta u obzir. Na slikama 5.10. i 5.11. b) vidljivo je kako vraćanjem na prijašnje fragmente aplikacija uspijeva u pamćenju parametara te radi potrebna ažuriranja.

5.3.1. Analiza korisničkog iskustva

Drugo poglavlje ovog završnog rada bavi se poboljšanjem korisničkog iskustva u aplikacijama i posebno u društvenim strateškim igrama. Iz navedenog poglavlja moguće je pratiti smjernice kako bi se što više poboljšalo korisničko iskustvo.

Područje funkcionalnosti aplikacije

Potreba za unos podataka uvelike je smanjena upotrebom *NumberPickera*. Korisniku su ponuđeni svi glavni parametri te jedino što mora je pomicati odabrani. Odabir vještina i sličnih parametara olakšan je kreiranjem liste dopuštenih odabira te kontroliranje količine odabira od strane aplikacije. Unosom bodova iskustva automatski se otvara tipkovnica s brojevima kako korisnik ne bi morao mijenjati vrstu unosa svaki puta. Navigacija kroz aplikaciju je konzistentna, koriste se fragmenti te je prijelaz između njih gladak.

Područje dizajna aplikacije

Svaka aktivnost posvećena je jednoj funkcionalnosti npr. aktivnost za kreiranje lika, aktivnost za interakciju s likom. Veličina slova je konzistentna za određene namjere te je dovoljno velika u svakom dijelu. Radi preglednosti su margine u svim slučajevima višekratnici broja osam. Kontroliranje unosa podataka je u većini slučajeva *NumberPickerima* tako da se teži istom načinu unosa. Postoji jedna primarna boja u aplikaciji te se za sve koriste njene nijanse. U crtežima se pojavljuje sekundarna boja koja je odabrana na temelju primarne. Pošto je primarna boja siva, dana je pažnja na situacije kada je boja pozadine pretamna te se u tim slučajima koristila bijela boja za tekst. Također je u fragmentu s kockicama uzeta u obzir veličina ekrana te čovjekov doseg palcem.

Poboljšanje korisničkog iskustva u društvenoj igri

Prije razvoja aplikacije isplanirana je njena namjena. Aplikacija služi kao pomagalo pri ubrzanju kreiranja lika te rukovanjem njegovim osnovnim funkcijama. Interakcija s likom ne zahtjeva puno

vremena, ima samo par funkcionalnosti osim same kreacije. Unos i brisanje objekata opreme je brzo. Osim kontroliranja opreme i bacanja kockica, igraču je ostavljeno da se brine za ostale mogućnosti svog lika. Glavni nedostatak ostalim aplikacijama razvijenim u ovu svrhu je što sama aplikacija zahtijeva puno pažnje i vremena što nije svrha aplikacije za društvenu igru.

5.3.2. Osvrt korisnika

Kako bi se prikazala uspješnost aplikacije, dvije osobe su ju ispitale te su dale povratnu informaciju.

Osoba ženskog spola 21 godine ostavila je sljedeći dojam. Aplikacija je jako intuitivna. Dizajn je lijep pogotovo crteži kockica i čovjeka. Jako je lagano kontrolirati dodatne bodove i količinu kockica koja se baca. Prijelaz između kockica, lika i opreme je lagan. Fragment s detaljnim informacijama daje puno informacija, a nije natrpan „Gain XP“ gumb je nešto što nije vidjela u ostalim aplikacijama te izgleda slatko. Sve što aplikacija radi, radi dobro.

Osoba muškog spola 21 godine napomenula je sljedeće. Navigacija je jednostavna i prijelaz u njoj je brz, aplikacija se ne čini preopterećenom. Kombinacija crne i sive boje je lagana za oči, sve informacije su lijepo opisane. Jedino što se može dodati je mogućnost postavljanja opreme aktivnom.

Korisnici su bili zadovoljni dizajnom i laganom navigacijom. Bili su zadovoljni količinom i rasporedom informacija koje su im dane. Za sljedeće inačice aplikacije moguće je dodati još jednostavnih stvari poput spomenute da se oprema može postaviti aktivnom, tj. da korisnik zna što trenutno koristi.

6. ZAKLJUČAK

U ovom završnom radu razvijena je mobilna Android aplikacija za pojednostavljenje igranja društvene igre *Dungeons and Dragons*. Istraživanjem navedene literature proučen je poželjan stupanj digitalizacije sadržaja igre te učinkovit stupanj automatiziranja kreiranja likova. Također, izvedene su smjernice za poboljšanje korisničkog iskustva u području dizajna i području funkcionalnosti mobilne aplikacije. Oslanjajući se na navedeno, predložen je model rješenja kako bi se zadovoljili zahtjevi na aplikaciju. Prema njemu, programski je ostvarena Android aplikacija. Na kraju, način rada mobilne aplikacije i njena uspješnost u ostvarivanju cilja su analizirani, pri čemu je poseban osvrt dan na analizu korisničkog iskustva.

Razvijena mobilna aplikacija nudi brzo kreiranje likova uz prikaz bitnih informacija na temelju kojih korisnik može lakše donositi odluke. Unos i odabir podataka koji definiraju lika je ograničen kako bi se jasno definiralo što je dozvoljeno tijekom njegovog kreiranja. Bazom podataka ostvareno je pamćenje kreiranih likova, te je tim omogućeno njihovo buduće korištenje. Uz to, omogućeno je dodavanje već postojećih objekata opreme za vlastito korištenje i njihovo brisanje te unos personaliziranih objekata. Također, uklonjena je ovisnost igranja igre o posjedovanju kockica za društvenu igru kreiranjem zaslona kojem je funkcija jednostavno biranje broja kockica za bacanje, kao i dodatnih bodova. Analizom aplikacije utvrđeno je da je došlo do poboljšanja korisničkog iskustva u smislu dizajna i funkcionalnosti. Razvijena aplikacija može pomoći ne samo trenutnim igračima društvene igre, već i novim igračima olakšati kreiranje lika te ostale aktivnosti.

LITERATURA

[1] M. Gencer, G. Bilgin, Ö. Zan, T. Voyvodaoğlu. „A New Framework for Increasing User Engagement in Mobile Applications Using Machine Learning Techniques“, *Proceedings of the Second international conference on Design, User Experience, and Usability: web, mobile, and product design*, sv. 4., str. 651-659, 2013.

[2] T.Rodde, “21% of Users Abandon an App After One Use” , 2018., http://info.localytics.com/blog/21-percent-of-users-abandon-apps-after-one-use, pristup ostvaren 7.10.2020.

[3] A. Miniukovich, A. De Angeli. (2014). „Visual Impressions of Mobile App Interfaces“, *NordiCHI*, sv. 1., str. 31-40, Helsinki, Finska 2014.

[4] M. Kosa, P. Spronck „What Tabletop Players Think about Augmented Tabletop Games: A Content Analysis“, *International Conference on the Foundations of Digital Games*, sv.6, str. 2-7, Malmö, Švedska, 2018.

[5] J. Wallace, J. Pape, Y.L.B. Chang, P. Mcclelland, T.C., Graham, S. Scott, M. Hancock, „Exploring automation in digital tabletop board game“, *CSCW '12 Computer Supported Cooperative Work*, sv.1, str. 231-234, Seattle, WA, USA, 2012.

[6] W.B Goh, W. Shou, J. Tan, G.T. Jackson Lum. „Interaction design patterns for multi-touch tabletop collaborative games“, *CHI ’12 Extended Abstracts on Human Factors in Computing Systems (CHI EA ’12). Association for Computing Machinery*, sv. 1., str. 141–150., New York, NY, SAD, 2012.

[7] K. Emmerich, M. Masuch. „Game Metrics for Evaluating Social In-game Behavior and Interaction in Multiplayer Games*“, Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology (ACE ’16). Association for Computing Machinery* sv. 20, str. 1-8, New York, NY, SAD, 2016.

[8] C.Ciligot, “10 Best Practices to Enhance Your Mobile App User Experience”, 2020., https://clearbridgemobile.com/best-practices-to-enhance-your-mobile-app-user-experience/, pristup ostvaren 7.10.2020.

[9] N.Babich, “A Comprehensive Guide To Mobile App Design”, 2018., https://www.smashingmagazine.com/2018/02/comprehensive-guide-to-mobile-app-design/, pristup ostvaren 7.10.2020.

[10] A.Jaiswal, “How To Master User Experience (Ux) In Mobile Apps?”, 2019., https://www.ecommerce-nation.com/master-user-experience-ux-mobile-apps/, pristup ostvaren 7.10.2020.

[11] F.Spillers,“What is Mobile User Experience (UX) Design?”, 2020., https://www.interaction-design.org/literature/topics/mobile-ux-design, pristup ostvaren 7.10.2020.

[12] N.Babich, ”10 Do’s and Dont’s of Mobile App Design“, 2018., https://xd.adobe.com/ideas/principles/app-design/10-dos-donts-mobile-app-design/, pristup ostvaren 7.10.2020.

[13] A.Meehan, „Dungeons & Dragons 5E“, 2019., https://www.dicebreaker.com/games/dungeons-and-dragons-5e#:~:text=Dungeons%20%26%20Dragons%205E%20is%20the,Editions, pristup ostvaren 25.9.2020.

[14] F.Adarsh, „Android Studio 4.0“, 2020, https://android-developers.googleblog.com/2020/05/android-studio-4.html, pristup ostvaren 25.9.2020.

SAŽETAK

Mobilna aplikacija razvijena u ovom završnom radu za cilj ima pomoći pri igranju društvene igre Dungeons and Dragons. Korištenjem aplikacije korisniku se uvelike ubrzava postupak kreiranja lika, te početak igre. Uz kreiranje lika, korisniku je omogućena dodatna interakcija s likom i bacanje kockica koje društvena igra zahtijeva. Aplikacija je razvijena u okolini Android Studio koristeći programski jezik Java. Stvorena baza podataka korisniku omogućuje spremanje kreiranog lika za kasnije korištenje, te kontrolu nad posjedovanom opremom. Kako bi aplikacija bila što korisnija, razmatran je stupanj do kojeg korisnika treba ograničavati pravilima igre, a gdje korisniku treba dati slobodu izbora. Također, posebna pažnja posvećena je dizajnu i funkcionalnosti aplikacije. Analizom razvijene aplikacije kroz ispitivanje slučajeva korištenja, može se reći da aplikacija ostvaruje zadane ciljeve, te da uistinu služi kao dobro pomagalo pri igranju društvene igre, odnosno da poboljšava korisničko iskustvo.

Ključne riječi: društvena igra Dungeons and Dragons, korisničko iskustvo, kreiranje likova, mobilna Android aplikacija.

ABSTRACT

Mobile Android application for enhancing user experience in playing a strategic board game

Mobile application developed in purpose of this final paper serves as an assistant to a board game Dungeons and Dragons. User is greatly faster in creating a character and starting with the game. With character creation, there's an added option of interacting with the character and rolling the dice required by the game. Application is developed in Android Studio environment using Java programming language. Created database enables the user to save the created character for later use and control its personal equipment. To make the application even more useful, the degree of which the user is limited by the rules of the game and user's freedom of choice are considered. Finally, special attention is paid to the design and the functionallity of the application. By analyzing the application through its use cases, it is established that it fulfills its purpose in being a great assistant when playing the game, meaning that it does enhance user experience.

Key words: tabletop game Dungeons and Dragons, user experience, character creation, mobile Android application.

ŽIVOTOPIS

Jakov Prpić rođen je 28.10.1998. u Slavonskom Brodu gdje je pohađao OŠ „Đuro Pilar“ te kasnije matematički smjer gimnazije „Matija Mesić“. Trenutno je treća godina preddiplomskog studija Računarstva na FERIT-u u Osijeku. Koristio je i upoznat je s programskim jezicima C, C++, C# i Java. Osim na fakultetu, koristio je Udacity programe kako bi dodatno naučio koristiti razvojnu okolinu Android Studio za razvijanje mobilnih aplikacija. Osim razvoja Android aplikacija, ima interes za razvoj računalnih igara te je prošao kroz početne programe Unity-a. U svakoj ozbiljnoj situaciji daje sve od sebe te teško odustaje od neriješenog problema. Iako je kompetitivan, odlikuje u timskim situacijama te se lakoćom prilagođuje novim okolinama.

PRILOZI

Prilog 1. Završni rad u docx formatu

Prilog 2. Završni rad u pdf formatu

Prilog 3. Programski kod mobilne aplikacije