Razvoj interaktivnog personaliziranog sustava za učenje kroz igru

Lukačević, Josip

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 614

RAZVOJ INTERAKTIVNOG PERSONALIZIRANOG SUSTAVA ZA UČENJE KROZ IGRU

Josip Lukačević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 614

RAZVOJ INTERAKTIVNOG PERSONALIZIRANOG SUSTAVA ZA UČENJE KROZ IGRU

Josip Lukačević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 614

Pristupnik:	**Josip Lukačević (0036516431)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	doc. dr. sc. Tomislav Jagušt

Zadatak: **Razvoj interaktivnog personaliziranog sustava za učenje kroz igru**

Opis zadatka:

Učenje kroz igru integrira elemente zabave i interakcije kako bi potaknuo interes, angažman i motivaciju učenika, a samim time i bolje razumijevanje nastavnog gradiva. Kroz igru, učenici imaju priliku razvijati kreativnost, načine rješavanja problema, vježbati suradnju, tj. stvaraju dinamično okruženje koje potiče učenje. U sklopu diplomskog rada je potrebno istražiti utjecaj metoda personalizacije i igrifikacije na motivaciju za učenje te usporediti razumijevanje gradiva stečeno kroz igre s klasičnim metodama učenja. Nadalje, potrebno je razviti sustav za učenje kroz igru s ciljem poticanja interesa i motivacije učenika te olakšavanja usvajanja i ponavljanja gradiva. Kroz sustav koji će se razviti, nastavnici trebaju moći kreirati nekoliko različitih igara, od jednostavnih kvizova do interaktivnih obrazovnih igara u kojima sudjeluju 2 ili više korisnika. Učenici igrama pristupaju putem QR koda ili putem unosa šifre/PIN-a igre. Sustav treba pohranjivati i analizirati napredak učenika te na temelju rezultata personalizirati igre, npr. prilagođavajući težinu, stavljajući naglasak na područja i gradivo u kojem su učenici slabiji te dodavanjem objekata koji će se sakupljati kroz igru kako bi učenici mogli dobiti savjete ili npr. otključali sljedeći ili dodatni nivo i sl. Sustav je potrebno testirati te analizirati je li (i ako jest, u kojoj mjeri) personalizirani pristup učenju kroz igru efikasniji u usvajanju znanja u odnosu na tradicionalne metode učenja.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod 1

1. Učenje kroz igru 2

1.1. Igrifikacija u obrazovanju 2

1.2. Personalizirana igrifikacija 4

1.2.1. Personalizacija prema osobnosti igrača 5

1.2.2. Personalizacija prema vještinama igrača 7

2. Implementacija sustava 9

2.1. Android aplikacija 10

2.1.1. Kotlin 11

2.1.2. *Jetpack Compose* 21

2.1.3. Aplikacija *SmartQuest* 32

2.2. Poslužitelj 46

2.3. Baza podataka 48

3. Priprema istraživanja 50

3.1. Tehnike personalizacije 50

3.2. Dizajniranje priče i *Narrative atoms* 52

3.3. Kreiranje nelinearnih priča 55

4. Diskusija i analiza 61

Zaključak 66

Literatura 67

Sažetak 70

Summary 71

Skraćenice 72

Uvod

U današnjem obrazovnom sustavu, tehnologija igra ključnu ulogu u transformaciji načina na koji učenici percipiraju i usvajaju znanje. Jedan od najznačajnijih pristupa koji se sve više istražuje i primjenjuje je učenje kroz igru. Učenje kroz igru integrira elemente zabave i interakcije kako bi potaknulo interes, angažman i motivaciju učenika te im omogućilo bolje razumijevanje nastavnog gradiva. [1] Kroz igru, učenici ne samo da imaju priliku razvijati kreativnost, načine rješavanja problema i vježbati suradnju, već se i stvara dinamično okruženje koje potiče učenje na dubljem i sveobuhvatnijem nivou. [1]

Uvođenje elementa igre u učenje je obećavajuća metoda, no ne postoji univerzalna metoda koja će odgovarati svima, odnosno, neće se realizirati kod svakog učenika u jednakoj mjeri. Zbog tih nedostatka javlja se ideja personalizirane igrifikacije (*eng. gamification*), koja će ovisno o osobnosti učenika generirati prigodne elemente igre koji bi potaknuli motivaciju učenika na daljnje učenje.

U ovom diplomskom radu istražit će se utjecaj metoda personalizacije i igrifikacije na motivaciju za učenje, kao i razumijevanje gradiva stečenog kroz igre u usporedbi s klasičnim metodama učenja. Naglasak će biti na razvoju interaktivnog personaliziranog sustava za učenje kroz igru, s ciljem poticanja interesa i motivacije kod učenika te da olakša usvajanje i ponavljanje gradiva.

Sustav razvijen u sklopu ovog rada omogućit će nastavnicima kreiranje različitih igara, od jednostavnih kvizova do interaktivnih obrazovnih igara u kojima sudjeluju 2 ili više korisnika. Igre bi trebale omogućiti učenicima najbolji oblik igranja. Unutar igre, na početku svakog nivoa, nudit će se odabir lika (*eng. avatar*) s kojim će se nivo igrati. Svaki lik će imati različite tipove zadataka i priču unutar nivoa. Priča unutar nivoa i tip zadatka bit će bazirani na tipu osobnosti, a težina zadataka prilagođavat će se na osnovu prijašnjih rezultata i predznanju. Kroz provedbu ovog istraživanja, cilj je analizirati efikasnost personaliziranog pristupa učenju kroz igru u usvajanju znanja u odnosu na tradicionalne metode učenja.

1. Učenje kroz igru

Igre i tehnologija igara proširile su se van svojih granica što se očituje rastom industrije ozbiljnih igara te igara kao područja istraživanja. Iz tih istraživanja proizašao je pojam igrifikacija (*eng. gamification*). Pojam igrifikacije odnosi se na korištenje elemenata igre za poboljšavanje korisničkog iskustva i angažmana korisnika u uslugama izvan igara i aplikacija. [1] Ovaj pristup brzo je postao vrlo popularan u sferi digitalnog marketinga i prouzročio je intenzivne rasprave unutar profesionalne zajednice (2011. godine održan je prvi *summit* na temu igrifikacije [2]) kao i brojne igrificirane aplikacije koje uključuju bodove, značke, razine i ljestvice s rezultatima. Također, igrifikacija je zainteresirala istraživače, kao potencijalno sredstvo za stvaranje zanimljivih radnih mjesta te olakšavanje masovne suradnje. [3] Korištenje tehnologije nije strano ni u obrazovanju pa je tako igrifikacija našla primjenu i u obrazovnim ustanovama kao alat za poticanje kreativnosti i angažmana u razredima. [1]

1.1. Igrifikacija u obrazovanju

Tradicionalni pristup školstvu mnogim je učenicima dosadan i neučinkovit, stoga nastavnici nailaze na probleme s motivacijom i angažmanom. Korištenje obrazovnih igara kao alata za učenje obećavajući je pristup zbog njihove sposobnosti da jačaju znanje i važne vještine poput rješavanja problema, suradnje i komunikacije. [4]

Prema suvremenoj literaturi o igrifikaciji, uspješna primjena igrifikacije odnosi se na upotrebu elemenata igre kako bi se potaknule tri unutarnje potrebe za intrinzičnom motivacijom:

1. Povezanost: univerzalna potreba za interakcijom i povezivanjem s drugima.
2. Kompetencija: univerzalna potreba za učinkovitošću i svladavanjem problema u zadanom okruženju.
3. Autonomija: univerzalna potreba za kontrolom vlastitog života. [6]

Pregledom radova u posljednjem desetljeću, elementi igre mogu se podijeliti prema dvama kriterijima, a to su principi dizajna igrifikacije i mehanike igre. [4]

Neki od principa dizajna igrifikacije su: izazovi, personalizacija, otključavanje sadržaja, radnja/novi identiteti, integracija i ograničenje vremena, ali daleko najistraživaniji su: brza

povratna informacija, vidljiv status, sloboda izbora, sloboda za neuspjeh i društvena angažiranost. [4]

Primjeri primjene principa „sloboda izbora“ učenicima uključuju mogućnost odabira koji tip izazova žele riješiti: pisanje tradicionalnih eseja, sudjelovanje u grupnom ili individualnom projektu, pisanje akademskih radova, polaganje ispita ili završavanje umjetničkih zadataka. Ostali primjeri uključuju odabir specifičnih izazova, redoslijed i/ili brzinu završetaka izazova, odabir postavljanja ciljeva vještina ili način na koji su izazovi i njihovi tipovi vrednovani. [4]

Princip „sloboda za neuspjeh“ podrazumijeva odsustvo sankcija za loše izvršene zadatke i obično uključuje mogućnost da učenici pregledaju i ponovno predaju zadatke ili ponovno polaganje kvizova. Ovaj princip jedan je od najkontroverznijih za primjenu u tradicionalnoj učionici, ali ne postoje istraživanja koja su ispitala njegov učinak. [4]

Društvena angažiranost obuhvaća natjecanja na individualnoj i timskoj razini, sudjelovanje u grupnim obrazovnim aktivnostima i rad na timskim projektima, te suradnju i interakciju s drugim studentima. [4]

Pregledom postojeće literature može se ustvrditi kako su najistraživanije mehanike igre bodovi, značke i ljestvice s rezultatima, dok su istraživači pokazali nešto manji interes kod razina, virtualnih predmeta i avatara. [4]

Što se tiče upotrebe znački, u nekim slučajevima njihovo dodjeljivanje ne utječe na ocjenjivanje učenika, već je usmjereno na poticanje natjecateljske motivacije. Značke se dodjeljuju za razna postignuća, kao što su izazovi i sudjelovanje, učenje ili upravljanje vremenom. [4]

Razine su razmatrane u nekoliko vrsta poput igračke razine, razine igranja i razine igrača. Preporučuje se odabir razina tako da se u početku brzo stječu, ali tijekom vremena postaju sve teže. [4]

Primjer korištenja virtualne valute uključuje njeno trošenje na davanje savjeta za rješavanje slagalica, produljenje roka za zadatke, ponovno polaganje kvizova ili dobivanje pomoći kod određenih zadataka za domaću zadaću, produljenje roka bez kazne ili pak veću karticu za bilješke prilikom rješavanja testa. [4]

Istraživanja poput [5] pokazuju kako jednostavna primjena pojedinog vanjskog aspekta igrifikacije poput znački ili drugih ponavljajućih nagrada ne daje rezultate, već umjesto toga

igrifikacija mora uzeti u obzir motivaciju sudionika, ciljeve tečaja i dizajn igre. U stvari, sistemsko istraživanje mapiranja provedeno na razini angažiranosti i igrifikacije ukazuje na to da čak i do 40 % igrificiranih pristupa ne postiže značajne razlike u angažiranosti i motivaciji u usporedbi s neigrificiranim sustavima koji pružaju iste usluge. Uspješnu igrifikaciju u suradničkom učenju moguće je ostvariti u sustavima gdje suradnici mogu objaviti svoje kompetencije i usporediti rezultate s rezultatima svojih kolega. Isto tako, novija istraživanja pokušavaju istražiti i ostale principe dizajna igrifikacije, te je tako personalizacija u igrifikaciji postala predmet istraživanja u novijim radovima. [6]

1.2. Personalizirana igrifikacija

Istraživanja [1] su počela pretpostavljati da se pozitivni učinci igrifikacije mogu pojačati uzimajući u obzir osobne karakteristike korisnika. Naime, primijećeno je da ista igra može izazivati različite reakcije i posljedice kod različitih korisnika. Slično tome zaključuje se kako različite predispozicije i socijalne strukture donose personaliziran pojam zabave kod svakoga. Posljedica toga je nemogućnost dizajniranja univerzalno zabavne igre. Odnosno različiti korisnici interpretiraju iste igraće elemente na vrlo različite načine. Nadalje, pokazalo se da užitak koji proizlazi iz igre, korisnikova preferencija za određene igraće elemente, percipirana uvjerljivost igraćih elemenata i motivacija koja proizlazi iz igraćih elemenata sve utječu na korisnikovu osobnost i njegove osobne karakteristike. Ukratko, može se zaključiti da bi igrificirani sustavi trebali biti posebno prilagođeni različitim korisnicima kako bi igrifikacija ostvarila svoj puni potencijal. [6]

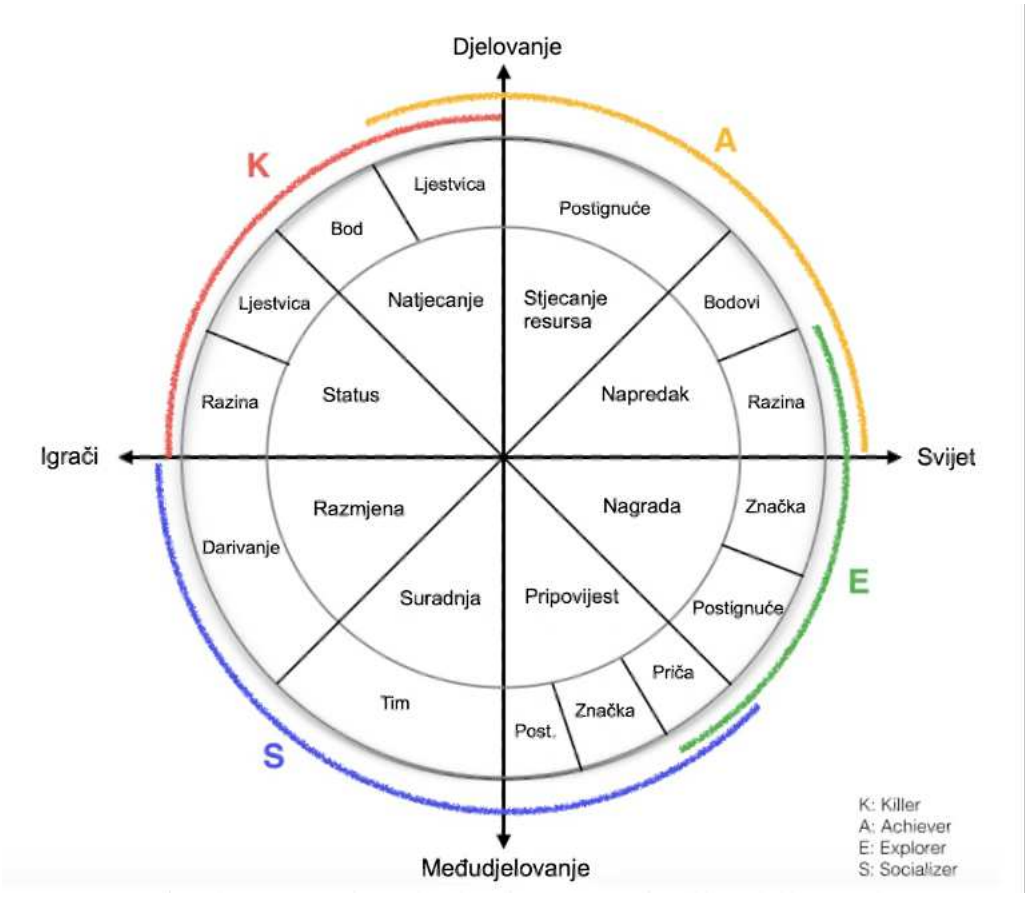
Uspjeh tehnika personalizacije već je dokazan u drugim digitalnim kontekstima. Personalizirana igrifikacija rezultira većim angažmanom ponašanja i emocija, istovremeno poboljšavajući samouvjerenost korisnika te njihovu percepciju o sustavu i jednostavnosti njegove upotrebe. Međutim, postojeća istraživanja koja proučavaju potencijal personaliziranih igrificiranih sustava ne proučavaju kako dizajnirati takve sustave [6]. Na primjer, nedavni rad o igrificiranom učenju [7] predstavlja sustav koji se prilagođava individualnom tempu svakog učenika, ali ne nudi personalizirani sadržaj. Ili nedavno istraživanje o igrificiranom učenju [8] koje ima personalizirani sadržaj, ali je sadržaj još uvijek ručno dodijeljen svakom učeniku. Iako početna istraživanja o personaliziranoj igrifikaciji dolaze na površinu, još uvijek postoji istraživačka praznina u procesima dizajniranja personaliziranih igrificiranih sustava. [6]

Druga djela [7][9] spominju prilagođene izazove, koji preuzimaju koncept igrifikacije i proširuju ih prema individualno prilagođenim zadacima koji se prilagođavaju osobnom iskustvu i znanju korisnika, na primjer korisnikovim sposobnostima i jezičnim vještinama. Prilagodba zadatka i izazova može uključivati i druge aspekte, poput kulturnog podrijetla korisnika. [6]

1.2.1. Personalizacija prema osobnosti igrača

Dizajn igrificiranih iskustava obično usvaja pristup „jedna veličina odgovara svima“, što može rezultirati neuspjehom jer se smatra da svi korisnici imaju isti profil. Drugi pristup može biti prilagodljiva igrifikacija, koja uzima u obzir da korisnici imaju različite motivacije tijekom njihove interakcije u igrificiranim sustavima. Prilagodljiva igrifikacija temelji se na klasifikaciji tipova igrača. [10] Prvi radovi o prilagodljivoj igrifikaciji [11] prikupili su informacije o korisničkim profilima prije igranja, a zatim su korisnicima predložili igraće elemente koji odgovaraju tim (statičkim) profilima. Međutim, ovaj početni igrački profil može biti netočno unesen od strane korisnika ili se može blago mijenjati tijekom vremena, stoga ponekad predloženi igraći elementi ne odgovaraju potpuno stvarnom profilu pa bi se trebalo uzeti u obzir više tipova igrača, a ne samo najzastupljeniji. [10]

Nekoliko istraživanja klasificiralo je igrače prema različitim tipovima. Studije su analizirale ponašanje igrača i razvrstale ih prema tim obilježjima. Prva studija [12] istražila je očekivanja igrača i proizvela prvi napor u kategorizaciji igrača, klasificirajući ih prema njihovim sklonostima za djelovanje/interakciju i orijentaciju. Ta klasifikacija (*Slika 1 Bartle klasifikacija tipova igrača [13]*) može se vizualizirati pomoću koordinatnog sustava, stavljajući „orijentiranost prema svijetu/igraču“ na x-os i „djelovanje/interakcija“ na y-os proizvodeći četiri kategorije: *killer*, *achiever*, *explorer* i *socializer*. [13] Slično tome, druga istraživanja [14] opisala su dodatne vrste igrača, poput *dominant, objectivist, humanist, inquisitive* i *creative*. Kasnija istraživanja [15] proširila su ovu klasifikaciju na devet različitih tipova igrača, uključujući *competitor, explorer, collector, achiever, joker, artist, storyteller, performer* i *director*. Unatoč tome, osnovna klasifikacija na četiri kategorije i dalje se smatra temeljnom za istraživanje u području igrifikacije. [13]



Slika 1 *Bartle* klasifikacija tipova igrača [13]

Prema klasifikaciji na četiri kategorije igrača, *killeri* pokušavaju dominirati nad drugim igračima djelovanjem na igrače u svojoj okolini. Ovaj tip igrača nije zainteresiran za uspješno izvršavanje zadataka ili postizanje visokih bodova, već nastoji postići bodove koji su dovoljni da dominiraju i pobijede druge. *Killeri* istražuju igru kako bi naučili nove načine kako naštetiti drugim igračima. Obično komuniciraju s drugima samo kako bi ih ponizili. [13]

Achieveri djeluju u svijetu i brinu se o zadacima u okolini kako bi pobijedili. Ovaj tip igrača određuje ciljeve i ulaže aktivne napore kako bi postigao te ciljeve i povećao svoje bodove koliko je god moguće. *Achieveri* se druže s drugim igračima samo kako bi saznali što oni znaju o osvajanju bodova. Mogućnost osvajanja bodova najviše potiče ovaj tip igrača da istražuje okruženje. *Achiever* brine samo o vlastitim bodovima i zadacima u okolini i obično nije zainteresiran za druge igrače niti je zainteresiran nanositi im štetu, osim eventualno u slučajevima kada su ti drugi igrači postigli visok rezultat ili kako bi spriječio druge igrače da osvoje nagradu. Tipovi igrača koji se mogu kategorizirati kao *achieveri* obično se hvale razinama koje su postigli i koliko brzo su ih postigli. [13]

Exploreri se integriraju sa svijetom. Žele istraživati okolinu i otkriti što više novih stvari. *Exploreri* obično traže greške i pomagače u igri. Osvajanje bodova je dosadna aktivnost koja je korisna samo za istraživanje sljedeće razine. Ovaj tip igrača želi se družiti s drugima samo ako će to dovesti do novog istraživanja i može poželjeti naštetiti drugima ako im se sprječava istraživanje u okolini. [13]

Posljednji tip, *socializeri*, komuniciraju s igračima u okolini i obično koriste komunikacijsku funkciju kako bi se družili. Ovaj tip igrača istražuje igru s ciljem otkrivanja o čemu drugi igrači pričaju te osjećaju potrebu za osvajanjem bodova kako bi mogli doseći nove zajednice. *Socializeri* obično štete samo onima koji povrijede njihove prijatelje. Drugim riječima, jedini cilj *socializera* je poticanje dobre komunikacije, upoznavanje novih ljudi i razvijanje dobrih prijateljstava. [13]

1.2.2. Personalizacija prema vještinama igrača

Osim promatranog aspekta personalizacije igrificiranog sustava za učenje prema osobnosti igrača, pojavljuje se i adaptivni pristup kao novi oblik učenja koji bi pružio personalizirano iskustvo učenicima prema njihovim preferencijama, trenutačnim vještinama i stilovima učenja. Ovaj pristup stavlja korisnika u središte pa prikuplja podatke, analizira i tumači razinu vještina i preferencije te primjenjuje potrebne prilagodbe sustavu kako bi odgovarao trenutačnoj sposobnosti učenika. [7]

Adaptivno učenje može se modelirati iz nekoliko perspektiva:

- Model domene: prilagodba kako će se sadržaj prezentirati i u kojem vremenu procesa učenja, uzimajući u obzir odnose između nastavnog materijala.
- Model učenika: usredotočenost korisnika, njegova interakcija sa sustavom, njegovi odgovori o osobnim karakteristikama te kontinuirano dinamičko prilagođavanje sustava.
- Model grupe: pronalaženje obrazaca učenja u grupama nakon analize pojedinačnih učenika.
- Model prilagodbe: apstrakcija logike prilagodbe sustava, metode procjene i radnji prilagodbe. [7]

Ovi modeli primjenjuju se u skladu sa sadržajem i ciljem softverskog sustava te njihova implementacija može znatno varirati kako bi odgovarala okruženju primjene. Postojeći adaptivni sustavi za učenje mogu implementirati jedan model ili njihovu kombinaciju, što

može rezultirati izrazito kompleksnim personaliziranim iskustvima za svakog pojedinog učenika. [7]

Kao primjer primjene adaptivne igrifikacije [16] uzeli su postojeći sustav *„My Math Academy“*. To je *game-based,* prilagodljiv personalizirani sustav učenja osmišljen kako bi pomogao djeci osnovnoškolske dobi poboljšati razumijevanje osnovnih matematičkih pojmova i operacija. Kada učenici prvi put koriste *„My Math Academy“*, prolaze preddijagnostičke testove unutar igre koji određuju njihovo prethodno znanje i smještaju ih u igre prilagođene njihovoj razini vještine. Igre započinju nastavnom aktivnosti koja pruža pregled igre, scenarija problema i audio upute o matematičkom sadržaju potrebnom za uspješno završavanje zadataka. Nakon završetka nastavne aktivnosti, učenik prelazi na drugu aktivnost unutar igre, koja uključuje više razina, lako, srednje, teško i „*the boss*“ razinu. Učenik pokazuje kako je ovladao cilj učenja prelaženjem *„the boss“* razine unutar te igre. Nadalje, svaka razina uključuje četiri do šest krugova, odnosno pitanja, a na temelju performansi svakog učenika, adaptivni algoritmi u stvarnom vremenu određuju koju će igru, i na kojoj težini, preporučiti. Unutar svake aktivnosti, prilagodljivost dinamički funkcionira kako bi modificirala nastavu i pružila potporu, povratnu informaciju i sadržaj koji će voditi učenike kroz svako pitanje. [16]

Također, [7] u svom istraživanju razmatraju i opcije ponavljanja istih pitanja na različitim razinama. U svrhu učinkovitijeg ocjenjivanja učenikovog znanja, svaka lekcija je podijeljena na elemente znanja (*eng*. *knowledge items KI*) koji predstavljaju bilo koji važan koncept sadržan u toj određenoj lekciji. Svako pitanje za određenu lekciju ima povezan jedan *KI.* Nakon što učenik odgovori na pitanja u lekciji, imat će rezultat za svaki *KI* unutar te lekcije, a taj rezultat se mijenja prema odgovorima koje učenik ponudi na pitanja koja su povezana s tim *KI*. Netočan odgovor smanjit će učenikov rezultat za taj *KI* i sljedeće pitanje povezano s tim *KI* prilagodit će se novoj težini. [7]

Opisani sustav omogućuje mogućnost ponovne upotrebe pitanja na različite načine i za pružanje povratnih informacija. Ako učenik pogrešno odgovori na pitanje, ono će se ponovno primijeniti kasnije s korigiranom težinom. Cilj ovog pristupa je potaknuti korisnika da nastavi vježbati te da ne reagira negativno jer je napravio pogrešku. Pogreške se tretiraju kao prihvatljiv događaj i nagrađuju se bodovima iskustva kako bi pomogli korisniku da se ne osjeća obeshrabreno zbog njih. [7]

2. Implementacija sustava

U sklopu rada implementiran je i sustav za kreiranje i igranje personaliziranih obrazovnih igara. Kroz sustav predavači mogu kreirati kreativne igre koje će pobuditi motivaciju kod igrača za vježbanjem nastavnog sadržaja. Kod kreiranja igre predavač ima slobodu oblikovati igru tako da je napravi zanimljivom svima jer se jedna igra može igrati na više načina te igrači mogu personalizirati put učenja svojim preferencijama. Igre se prilagođavaju igračima prateći njihov rezultat. Točnim rješavanjem zadataka, igrač će dobivati teže zadatke. Ako zadatci u jednom trenu postanu preteški i igrač nekoliko puta krivo odgovori na pitanja, tada će se težina zadatka prilagoditi i spustiti za jednu razinu. Također, sustav prati koje nastavne cjeline su igraču bile problematične te mu nudi opciju ponavljanja tih cjelina.

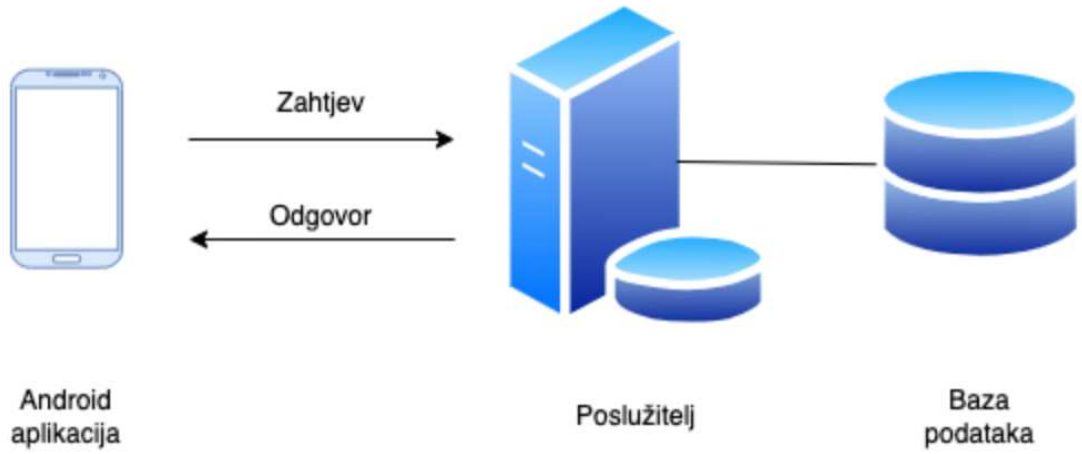
Igrači mogu igrati kreirane igre. Kreirane igre su organizirane u nelinearne priče koje igračima nude više opcija za proći razinu. Igre se mogu igrati neograničen broj puta. Tako igrači mogu istraživati sve moguće ishode igre ili ponavljati gradivo dok ga ne savladaju. Ako ne želi igrati cijelu igru, igrač može ponovno igrati jednu po jednu razinu iz postojećih igara za nastavnu cjelinu u kojoj je imao slabije rezultate. Osim toga, igrači mogu pratiti ljestvicu od top tri igrača i pokušati biti bolji od ostalih.

Sustav se sastoji od tri dijela, mobilne aplikacije za uređaje s operacijskim sustavom Android, poslužitelja razvijenog u razvojnom okruženju *Spring Boot* i *PostgreSQL* baze podataka. Navedene komponente su organizirane u arhitekturu klijent-poslužitelj što je učestali arhitekturni model u razvoju softverskih sustava, posebno onih koji zahtijevaju mrežnu komunikaciju.

Klijent-poslužitelj model je distribuirana aplikacijska struktura koja dijeli zadatke ili radna opterećenja između pružatelja resursa ili usluga, nazvanih poslužitelji, i tražitelja usluga, koji se zovu klijenti. U klijent-poslužitelj arhitekturi, kada klijentsko računalo putem interneta pošalje zahtjev za podacima poslužitelju, poslužitelj prihvaća traženi zahtjev i isporučuje tražene podatke klijentu. Klijenti ne dijele nijedan od svojih resursa. Primjeri klijent-poslužitelj modela su e-pošta, *World Wide Web*, itd. [17]

Klijent je komponenta koja inicira zahtjeve i koristi usluge pružene od strane poslužitelja. U ovom slučaju to su mobilni uređaji koji šalju zahtjeve putem *HTTP* protokola prema centralnom poslužitelju. Poslužitelj je komponenta koja prima zahtjeve od klijenata,

obrađuje ih i vraća odgovore. Poslužitelji obično pružaju usluge pohrane podataka, autentifikacije, poslovne logike i slično.



Slika 2 Prikaz modela klijent-poslužitelj

Uz klijente i poslužitelja, treća komponenta sustava je baza podataka. Baza podataka služi za trajnu pohranu podataka kao što su igračevi rezultati za pojedine nastavne cjeline, ukupni bodovi ili igre koje su predavači kreirali. Međutim, bazi podataka klijenti ne mogu izravno pristupiti. Za klijente, taj posao odrađuje poslužitelj u ulozi posrednika. Poslužitelj kontrolira i upravlja pristupom bazi podataka, osiguravajući sigurnost, integritet i učinkovitost u manipulaciji podacima.

2.1. Android aplikacija

U sklopu ovog rada razvijena je mobilna aplikacija za uređaje s operacijskim sustavom Android. Ova aplikacija jedna je od ključnih komponenata razvijenog sustava za učenje kroz igru. Aplikacija ima dvije osnovne funkcionalnosti, kreiranje personaliziranih obrazovnih igara te igranje kreiranih igara. Funkcionalnost stvaranja obrazovnih igara prvenstveno je namijenjena profesorima i nastavnicima, ali sustav ne postavlja nikakve restrikcije na to tko se smije, a tko ne smije okušati u kreiranju igre. Druga glavna funkcionalnost aplikacije je igranje igara i ta je funkcionalnost namijenjena učenicima ili nastavnicima koji žele testirati svoju igru.

Igre se igranju tako da igrač upiše šifru igre ili skenira QR kod koji je nastavnik dobio kada je kreirao igru. Nakon upisivanja šifre, igra se dohvaća s poslužitelja te je igrač slobodan istraživati što ta igra nudi. Igre su podijeljene u razine i igrač sakuplja bodove rješavajući

zadatke unutar razine. Kada igrač riješi sve razine u igri, sakupljeni bodovi se nadodaju prijašnjim bodovima. Tako igrač može pratiti koliko je bodova ukupno ostvario te pokušati biti bolji od ostalih. Koliko bodova mu nedostaje do najboljeg, igrač uvijek može provjeriti na početnom ekranu igre.

Aplikacija je razvijena u programskom jeziku Kotlin. Za razvoj korisničkog sučelja korišten je moderan razvojni alat *Jetpack Compose* koji se zasniva na deklarativnom pristupu programiranju. Za povezivanje korisničkog sučelja i poslovne logike aplikacije, primijenjen je arhitekturni obrazac *MVVM* (*Model-View-ViewModel*). U daljnjem tekstu će se detaljnije opisati sve komponente aplikacije i korištene tehnologije.

2.1.1. Kotlin

Kotlin je moderan, ali zreo programski jezik dizajniran da poboljša efikasnost i rad programera. Sažet je, siguran, interoperabilan s Javom i drugim jezicima te pruža mnoge načine za ponovnu upotrebu koda između različitih platformi za produktivno programiranje. [18]

Na Google I/O 2019 (godišnja Googleova konferencija za programere [19]), najavljeno je da će razvoj za Android sve više biti usmjeren na Kotlin, čemu je Google ostao dosljedan. Kotlin je izražajan i sažet programski jezik koji smanjuje uobičajene pogreške u kodu i lako se integrira u postojeće aplikacije. [20]

Kotlin se pojavio kao alternativa za Javu jer se prilikom prevođenja koda generira *bytecode* koji je kompatibilan s Javinim *bytecodeom*, ali tu nije kraj mogućim primjenama Kotlina. Tako se Kotlin koristi ne samo za razvoj mobilnih aplikacija za Android, nego i za razvoj *front-enda*, *back-enda*, *data science* i za *cross-platform* mobilni razvoj. [21]

Najvažnija stvar koja Kotlin čini atraktivnim je njegov dizajn koji je više *user-friendly* nego Java, čime se olakšava pisanje koda, smanjuju se greške i povećava se produktivnost. Dok je Java uglavnom ograničena na objektno-orijentirano programiranje, Kotlin također nudi značajke funkcionalnog programiranja. [21]

Neke od glavnih prednosti Kotlina su dobra čitljivost, sigurnost od *null* vrijednosti, *extension* funkcije, korutine te jednostavno pristupanje atributima klase zbog čega nema potrebe za pisanjem *getter* i *setter* funkcija. [21]

Kotlin uvodi mnoge novosti u odnosu na Javu, a one najbitnije koje su korištene u razvoju ovog sustava opisane su dalje u tekstu.

2.1.1.1 Korutine

Korutina (*engl. coroutines*) je primjer suspenzivne (*engl. suspendable*) obrade. Konceptualno je slična niti, u smislu da preuzima blok koda za izvršavanje koji radi paralelno s ostatkom koda. Međutim, korutina nije vezana ni za jednu određenu nit. Može obustaviti svoje izvršavanje u jednoj niti i nastaviti u drugoj. Korutine se mogu smatrati laganim nitima, ali postoji niz važnih razlika koje njihovu stvarnu upotrebu čine vrlo različitima od niti. [22]

Korutine se mogu stvoriti jednostavno pozivom funkcije *launch*. Funkcija *launch* pokreće novu korutinu paralelno s ostatkom koda, koji se nastavlja izvršavati nezavisno. Funkcija *launch* dostupna je jedino u dosegu korutine koji se može definirati s pomoću *buildera* poput *coroutineScope.* Navedeni *builder* stvara doseg korutine, odnosno *scope*, koji premošćuje „svijet“ bez korutina i spaja ga sa „svijetom“ običnih funkcija. Stvoreni *scope* trajat će sve dok sva pokrenuta „djeca“ ne završe svoj posao. [22]

Unutar korutina se, uz standardne funkcije, koriste posebne funkcije koje se označavaju s ključnom riječi *suspend*. Suspenzivne funkcije su drugačije od običnih funkcija po tome što imaju mogućnost koristiti druge suspenzivne funkcije kako bi privremeno zaustavile izvršavanje korutine. Suspenzija korutine ne blokira temeljnu nit, već omogućava drugim korutinama da se izvršavaju i koriste temeljnu nit za svoj kod. [22]

```
is HomeEvent.OnLoadGameClicked -> {

    viewModelScope.launch {
        collectFor(
            {
                gameService.getGame(event.gameId)
            }, { result ->
                currentGameHelper.setCurrentGame(result)
                _uiEvent.emit(UiEvent.NavigateToNextScreen(Graph.LEVELS_LIST))
            }, {
                _uiEvent.emit(UiEvent.ShowErrorWhileFetchingGameAlert)
            }
        )
    }

    analyticsHelper.logAnalytics(GAME_LOADED, additionalData: "gameId:${event.gameId},")
}
```

Slika 3 Prikaz poziva korutine

Na *Slika 3 Prikaz poziva korutine* vidi se prikaz poziva korutine u dosegu *ViewModela* i poziva *suspend* funkcije *collectFor* koja obavlja mrežni poziv te nakon što primi rezultat, nastavlja s obradom dobivenog rezultata.

2.1.1.2 *Sealed* klase i sučelja

Sealed klase i sučelja omogućuju kontrolirano nasljeđivanje hijerarhijskih klasa. Sve izravne potklase *sealed* klase poznate su u vrijeme prevođenja. Druge potklase ne mogu se pojaviti izvan modula i paketa unutar kojih je *sealed* klasa definirana. Ista logika vrijedi za *sealed* sučelja i njihove implementacije: kada je modul sa *sealed* sučeljem preveden, nove implementacije se ne mogu stvoriti. [23]

Kada se *sealed* klase i sučelja kombiniraju s *when* izrazom (objašnjeno niže u tekstu), može se obuhvatiti ponašanje svih mogućih potklasa i osigurati da se ne stvaraju nove potklase koje bi negativno utjecale na kod. [23]

Sealed klase najbolje se koriste u scenarijima kada:

- Želi se ograničeno nasljeđivanje klasa: postoji unaprijed definiran, konačan skup potklasa koje nasljeđuju klasu, a sve su poznate u vrijeme prevođenja.
- Potreban je *type-safe* dizajn: kada su sigurnost i uzorkovanje (*pattern matching*) ključni u projektu. Posebno za upravljanje stanjima ili rukovanje složenom uvjetnom logikom.
- Rad sa zatvorenim API-ima: potreban je robustan i održiv javni API za biblioteke koje osiguravaju da ih treće strane koriste na predviđen način. [23]

```
jlukacevic
sealed class UiEvent {

    jlukacevic
    data class NavigateToNextScreen(val route: String) : UiEvent()

    jlukacevic
    object ShowErrorWhileFetchingGameAlert : UiEvent()

}
```

Slika 4 Deklaracija *sealed* klase i njenih potklasa

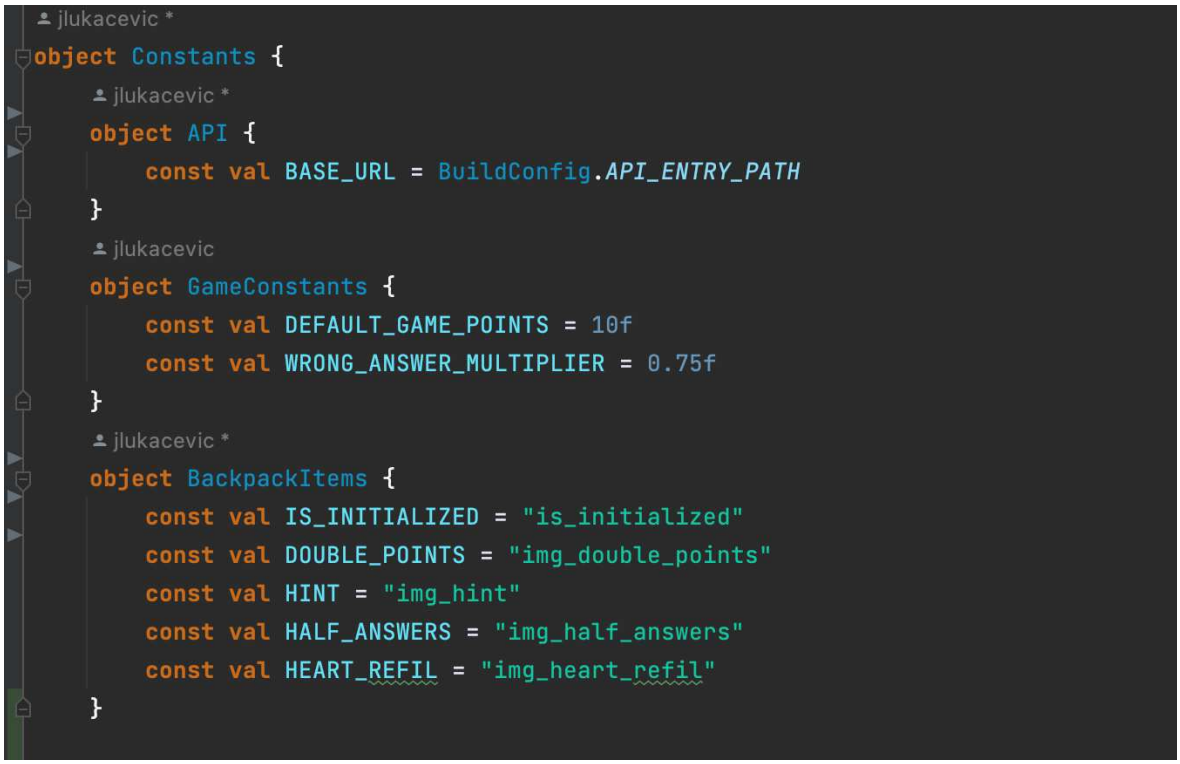
Na *Slika 4 Deklaracija sealed klase i njenih potklasa* vidi se primjer deklaracije *sealed* klase *UiEvent* koja definira sve moguće *evente* s kojima *ViewModel* može komunicirati s korisničkim sučeljem (detaljnije objašnjeno niže u tekstu).

2.1.1.3 *Object*

Ključna riječ *object* u Kotlinu ima dvije funkcije. Prva uloga ključne riječi *object* je za kreiranje objekata anonimne klase, odnosno klasa koje nisu eksplicitno deklarirane s deklaracijom klase. Takve klase su korisne za jednokratnu upotrebu. Može ih se definirati od nule, naslijediti iz postojećih klasa ili implementirati sučelja. Instance anonimnih klasa također se nazivaju anonimnim objektima jer su definirane izrazom, a ne imenom. [24]

Druga funkcija ključne riječi *object* je kreiranje objekata po oblikovnom obrascu *singleton*. *Singleton* osigurava da klasa ima samo jednu instancu, odnosno objekt. Za deklariranje *singleton* objekta potrebno je navesti ključnu riječ *object* koju prati ime klase. Na ovaj način se u kotlinu na jednostavan i efikasan način može implementirati *singleton* obrazac. *Singleton* se često naziva i anti-obrazac [25], ali kada ga je ovako lako definirati, ponekad može biti od velike pomoći. Kako bi referencirali tako deklarirani objekt, potrebno je izravno navesti ime koje mu je pridruženo. [24]

U implementiranom sustavu takvi objekti su iskorišteni za dvije važne funkcionalnosti. Prva funkcionalnost je grupiranje svih konstanti koje se koriste u projektu (*Slika 5 Deklariranje singleton objekata*). Druga funkcionalnost *singleton* objekta je praćenje svih relevantnih podataka o trenutnoj igri koja se igra i definiranje funkcija koje će izmjenjivati te podatke.



Slika 5 Deklariranje *singleton* objekata

2.1.1.4 *Data* klase

Data klase u Kotlinu se uglavnom koriste za pohranu podataka. Za svaku *data* klasu, prevodilac automatski generira dodatne članske funkcije koje omogućuju ispis instance u čitljivom obliku, usporedbu instanci, kopiranje instanci i slično. Za definiranje *data* klase potrebno je dodati ključnu riječ *data* ispred deklaracije klase. [26]

Kako bi generirani kod imao smisla, *data* klase moraju imati barem jedan parametar u primarnom konstruktoru, svi parametri primarnog konstruktora moraju biti označeni s *val* ili *var* (ključne riječi u Kotlinu za deklariranje nepromjenjivih, odnosno promjenjivih varijabli) te ne mogu biti apstraktne, otvorene, *sealed* ili unutarnje (*inner*). Jedna od najkorisnijih generiranih funkcija je funkcija *copy*. Funkcija za kopiranje objekta, omogućujući promjenu nekih od atributa dok ostale zadržava nepromijenjene. [26]

Slika 6 Deklaracija *data* klase

Na *Slika 6 Deklaracija data klase* prikazan je primjer deklaracije *data* klase koja koristi još jedan novitet u Kotlinu u odnosu na Javu, a to je dodjeljivanje zadanih vrijednosti parametrima funkcije ili konstruktora. Ovako deklarirana klasa može se instancirati praznim konstruktorom, ali pošto su zadane predefinirane vrijednosti za sve parametre konstruktora, oni će poprimiti te vrijednosti.

2.1.1.5 *Null safety*

Sigurnost od *null* vrijednosti, odnosno *null safety,* spomenuli smo kao jednu od glavnih prednosti Kotlina naspram Jave. U Kotlinu, sustav tipova podataka razlikuje reference koje mogu sadržavati *null* vrijednost i one koje ne mogu. Na primjer, obična varijabla tipa *String* ne može sadržavati *null* vrijednost. Kako bi dopustili *null* vrijednosti, potrebno je varijablu deklarirati kao *nullable* dodavanjem znaka „?“ nakon definiranja tipa varijable, npr. *String?*. [27]

Za pristupanje vrijednosti u varijablama koje su označene kao *nullable* uvijek prvo treba provjeriti je li njihova vrijednost *null*. Za takve slučajeve Kotlin je razvio posebnu vrstu poziva vrijednosti koji se naziva *safe call*. *Safe call* se obavlja dodavanjem znaka „?“ nakon imena varijable. Takvi pozivi vrlo su korisni kod ulančanih poziva. [27]

Za obavljanje određenih operacija samo na vrijednostima koje nisu *null, safe call* može se koristiti s Kotlinovom *scope* funkcijom *let* (Kotlin nudi nekoliko *scope* funkcija koje obavljaju blok koda u kontekstu objekta nad kojim su pozvane. [28] Takav poziv će izvršiti blok koda unutar *let* funkcije, ako objekt nad kojim je funkcija pozvana nije *null*. [27]

Kada se *safe call* nalazi na lijevoj strani dodijele vrijednosti i jedan od primatelja vrijednost je *null*, izračun vrijednosti desne strane dodijele vrijednosti se preskače. Ako se *safe call* pojavi na desnoj strani dodijele vrijednosti, a primatelj vrijednosti nije definiran kao *nullable* tip, treba definirati vrijednost koju će on poprimiti ako *safe call* vrati *null* vrijednost. Za

takve slučajeve, u Kotlinu se koristi *Elvis* operator „?:“. Desno od *Elvis* operatora potrebno je definirati vrijednost koja će se primjenjivati, ako je izraz lijevo od operatora *null*. [27]

Kotlin još nudi i *safe cast* operaciju koja se koristi za pretvorbu tipova. *Safe cast* se obavlja pozivom ključne riječi *as* i dodavanjem znaka „?“. Tako se pretvorba tipova neće provesti ako objekt nije željenog tipa. [27]

```
newLevelState.newLevel.knowledgeItem?.let {
    Text(text = it.name)
} ?: kotlin.run {
    Text(text = "Odaberi nastavnu lekciju")
}
```

Slika 7 Primjer korištenja *safe call* i *Elvis* operatora

Na *Slika 7 Primjer korištenja safe call i Elvis operatora* prikazan je primjer korištenja *safe call*-a sa *scope* funkcijom *let.* Objekt *knowledgeItem* je označen kao *nullable* te ako nije *null* dohvaća se vrijednost njegovog atributa *name*, u suprotnom se izvršava izraz desno od *Elvis* operatora, u ovom slučaju to je još jedna *scope* funkcija *run*.

2.1.1.6 *When* izraz

When izraz definira uvjetni izraz s više grana. Sličan je izrazu *switch* u Javi ili C-u. *When* se može koristiti kao izraz ili kao naredba. Ako se koristi kao izraz, vrijednost prve odgovarajuće grane postaje vrijednost cjelokupnog izraza. Ako se koristi kao naredba, vrijednosti pojedinih grana se zanemaruju. Kao i kod *if* naredbe, svaka grana može biti blok koda, a njegova vrijednost je vrijednost zadnjeg izraza u bloku. Ako se *when* koristi kao izraz, grana *else* je obavezna, osim ako prevodilac može dokazati da su svi mogući slučajevi pokriveni uvjetima grana, na primjer s unosima podtipova *sealed* klasa. [29]

```
uiEvent.collectLatest { event ->
    when (event) {
        is HomeViewModel.UiEvent.NavigateToNextScreen -> {
            navController.navigate(route = event.route)
        }

        is HomeViewModel.UiEvent.ShowErrorWhileFetchingGameAlert -> {
            showErrorWhileFetchingGameAlert = true
        }
    }
}
```

Slika 8 Prikaz korištenja *when* naredbe

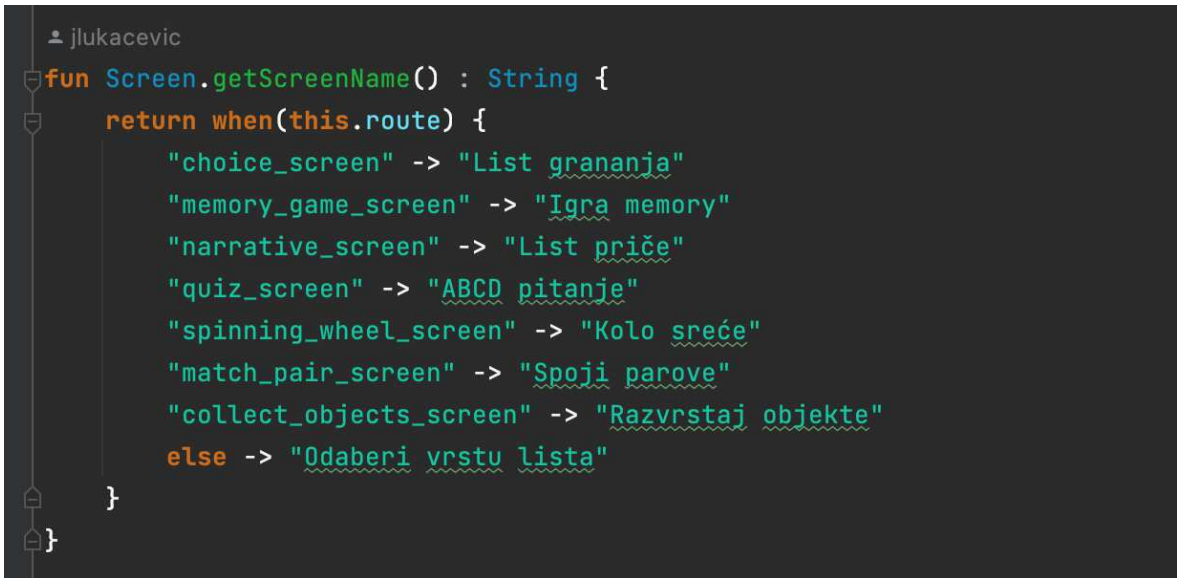
Na *Slika 8 Prikaz korištenja when naredbe* prikazan je primjer korištenja *when* naredbe kombinirane sa *sealed* klasom *UiEvent* (*Slika 4 Deklaracija sealed klase i njenih potklasa*) koja definira sve moguće *evente* koje *ViewModel* može poslati korisničkom sučelju. Zavisno o tipu primljenog *eventa,* izvršavaju se drugačije radnje na korisničkom sučelju.

2.1.1.7 *Extension* funkcije

Kotlin omogućuje proširenje klase ili sučelja novim funkcionalnostima bez potrebe za nasljeđivanjem iz klase ili korištenjem obrazaca kao što je Dekorator. To se postiže putem posebnih deklaracija nazvanih ekstenzije. [30]

Na primjer, mogu se napisati nove funkcije za klasu ili sučelje iz biblioteke treće strane koja se ne može mijenjati. Takve funkcije mogu se pozivati na uobičajen način, kao da su metode izvorne klase. Ovaj mehanizam naziva se *extension* funkcija. Postoje i *extension* svojstva koja omogućuju definiranje novih svojstava za postojeće klase. [30]

Za deklariranje ovakvih funkcija, potrebno je dodati tip prijamnika kao prefiks imenu funkcije. Prijamnik je tip koji se proširuje. Ekstenzije zapravo ne mijenjaju klase koje proširuju. Definiranjem ekstenzija ne unose se novi članovi u klasu, već se samo omogućuje pozivanje novih funkcija s notacijom točke na varijablama tog tipa. [30]



Slika 9 Deklariranje *extension* funkcije

Na *Slika 9 Deklariranje extension funkcije* vidi se prikaz deklariranja *extension* funkcije koja za objekte tipa *Screen*, na osnovu vrijednosti njihove varijable *route* vraća ljudski čitljiv naziv.

2.1.1.8 *Flow*

Jedna od vrlo korisnih službenih biblioteka za Kotlin su asinkroni tokovi koji sekvencionalno emitiraju vrijednosti i završavaju normalno ili s iznimkom. Takvi tokovi podataka su definirani sučeljem *Flow*. [31]

Sučelje *Flow* nudi i mnoge funkcije za transformaciju podataka u toku kao što su *map, filter, take, zip*… Ove funkcije se primjenjuju na prethodni tok ili tokove i vraćaju novi tok na koji se mogu primijeniti dodatni operatori. Ove funkcije ne izvršavaju nikakav kod i same nisu *suspend* funkcije. One samo postavljaju lanac operacija za buduće izvršenje i brzo se vraćaju. Ovo svojstvo naziva se hladni tok. [31]

Tokovi definiraju i nekoliko završnih funkcija poput *collect, single, reduce, toList*... Ove funkcije su *suspend* funkcije i one pokreću prikupljanje toka. Završne funkcije se primjenjuju na prethodni tok i aktiviraju izvršenje svih operacija. [31]

Tok podataka može se kreirati s nekoliko *builder* funkcija kao na primjer *flow* funkcija. Kako bi postojali podaci koji će se obrađivati i prikupljati potrebno ih je emitirati korištenjem funkcije *emit*. [31]

```
private val _uiEvent = MutableSharedFlow<UiEvent>()
val uiEvent = _uiEvent.asSharedFlow()
```

Slika 10 Deklariranje varijable tipa *MutableSharedFlow*

```
is HomeEvent.OnNavigateToLevelClicked -> {
    launch {
        _uiEvent.emit(
            UiEvent.NavigateToNextScreen(
                route = currentGameHelper.currentScreen?.screen?.route ?: Graph.HOME
            )
        )
    }
}
```

Slika 11 Emitiranje *eventa* putem toka podataka

```
LaunchedEffect(key1 = true) {
    uiEvent.collectLatest { event ->
        when (event) {
            is HomeViewModel.UiEvent.NavigateToNextScreen -> {
                navController.navigate(route = event.route)
            }

            is HomeViewModel.UiEvent.ShowErrorWhileFetchingGameAlert -> {
                showErrorWhileFetchingGameAlert = true
            }
        }
    }
}
```

Slika 12 Promatranje rezultata toka podataka

Na *Slika 10, Slika 11, Slika 12* prikazan je proces emitiranja *eventa* putem toka podataka i prikupljanja tih evenata. Varijabla *_uiEvent* je tipa *MutableSharedFlow* koji implementira sučelje *Flow.* Nadalje, varijabla *_uiEvent* unutar korutine emitira *event* tipa *UiEvent* (*Slika 4 Deklaracija sealed klase i njenih potklasa*). Tok podataka definiran varijablom *_uiEvent* predaje se korisničkom sučelju te se podaci iz tog toka prikupljaju pozivom funkcije *collectLatest* nad tim tokom. Tako prikupljeni podaci iz toka će uvijek skupljati posljednji podatak koji je emitiran, te ako se prijašnji podatak još obrađuje, njegova obrada se otkazuje.

2.1.2. *Jetpack Compose*

Jetpack Compose je moderni deklarativni UI (*user interface*) alat za Android. *Compose* olakšava pisanje i održavanje korisničkog sučelja aplikacije pružajući deklarativni API koji omogućuje prikazivanje korisničkog sučelja aplikacije bez imperativnog mijenjanja prikaza na *front-endu*. [32]

U imperativnom pristupu, hijerarhija prikaza u Androidu predstavljena je kao stablo UI *widgeta*. Kako se stanje aplikacije mijenja zbog stvari poput interakcija korisnika, hijerarhiju korisničkog sučelja potrebno je ažurirati kako bi prikazivala trenutne podatke. Korisničko sučelja mijenja se kretanjem kroz stablo koristeći funkcije poput *findViewById()* i mijenjanje čvorova pozivanjem metoda poput *button.setText(String)*, *container.addChild(View)* ili *img.setImageBitmap(Bitmap)*. Ove metode mijenjaju unutarnje stanje *widgeta*. [32]

Ručno manipuliranje prikazima povećava vjerojatnost grešaka. Na primjer ako se neki podatak zaboraviti ažurirati na jednom od mjesta na kojem se prikazuje ili ako se stvori nelegalno stanje u kojemu se pokušava postaviti vrijednost čvora koji je upravo uklonjen iz korisničkog sučelja. [32]

Tijekom posljednjih nekoliko godina, cijela industrija počela se prebacivati na deklarativni model korisničkog sučelja, što uvelike pojednostavljuje izradu i ažuriranje korisničkog sučelja. Tehnika funkcionira tako da konceptualno regenerira cijeli zaslon ispočetka, a zatim primjenjuje samo potrebne promjene. Ovaj pristup izbjegava složenost ručnog ažuriranja hijerarhije prikaza sa stanjem. [32]

Jedan od izazova kod regeneriranja cijelog zaslona je vremenski trošak, računalna snaga i potrošnja baterije. Da bi se smanjio ovaj trošak, *Compose* inteligentno bira koje dijelove korisničkog sučelja treba ponovno nacrtati u bilo kojem trenutku. Zbog toga postoje određene implikacije na način na koji je potrebno dizajnirati UI komponente kako bi one ispravno prikazivale sva ažuriranja podataka. [32]

Kod mnogih imperativnih objektno orijentiranih UI alata, korisničko sučelje se inicijalizira instanciranjem stabla *widgeta.* Ti *widgeti* bi nastajali od *XML layout* datoteka. *Widget* održava svoje unutarnje stanje i izlaže *getter* i *setter* metode s pomoću kojih se može promijeniti njegovo stanje. [32]

U deklarativnom pristupu *Composea*, *widgeti* su relativno bez stanja i ne izlažu *setter* ili *getter* funkcije jer *widgeti* nisu izloženi kao objekti. Korisničko sučelje ažurira se

pozivanjem iste *Composable* funkcije s različitim argumentima. Ovakvim pristupom olakšano je upravljanje stanjem putem nekog od arhitekturnih obrazaca poput *ViewModela*. Tada su *Composable* komponente odgovorne za pretvaranje trenutnog stanja korisničkog sučelja svaki put kada se promatrani podaci ažuriraju. [32]

Kada dođe do promjene stanja podataka koje *Composable* komponenta promatra, ona se ponovno iscrtava na korisničkom sučelju s novim podacima. Ovaj proces ponovnog pozivanja *Composable* funkcije naziva se rekompozicija. Rekompozicija se obavlja samo nad elementima koji su ovisili o promijenjenom stanju, ostali dijelovi se preskaču kako bi se uštedjelo na vremenu i resursima. [32]

Composable funkcije se razlikuju od običnih funkcija u Kotlinu po tome što imanju anotaciju *@Composable* koja signalizira *Compose* prevodiocu da ta funkcija treba biti pretvorena u komponentu korisničkog sučelja. [32]

```
jlukacevic *
@Composable
fun ChoiceContainer(modifier: Modifier = Modifier, onClick: () -> Unit, imageRes: String) {
    Box(
        modifier
            .clickable {
                onClick()
            },
        contentAlignment = Alignment.Center
    ) {
        Image(
            painter = rememberAsyncImagePainter( model: "${Constants.API.BASE_URL}games/images/${imageRes}"),
            contentDescription = ""
        )
    }
}
```

Slika 13 Primjer *Composable* funkcije

Na *Slika 13 Primjer Composable funkcije* vidi se jednostavna *Composable* funkcija koja za prikazivanje slike koristi dvije gotove *Composable* komponente *Box* i *Image*.

Jetpack Compose nudi mnoštvo gotovih komponenti koje se mogu koristiti za stvaranje korisničkog sučelja. Neke od najvažnijih takvih komponenti su komponente koje se brinu za raspored drugih komponenata na korisničkom sučelju. Pa tako za vertikalni raspored komponenata postoji gotova komponenta *Column*. Ova komponenta će svoje *child* komponente rasporediti na korisničko sučelje redom, jednu ispod druge. Za horizontalni raspored komponenata postoji gotova komponenta *Row*. *Row* će svoje *child* komponente rasporediti na korisničkom sučelju redom, s lijeva na desno. Treća os po kojoj se komponente mogu raspoređivati može se zamisliti kao stog na koji se komponente slažu „u dubinu“.

Gotova komponenta *Box* će svoje *child* komponente slagati jednu iznad druge tako da komponenta koja je navedena kasnije bude prikazana iznad one koja je navedena prije nje. Navedene komponente se mogu ugnježđivati na bilo koji način što omogućuje izgradnju vrlo kompleksnih i atraktivnih korisničkih sučelja.

Osim navedenih komponenti postoje i *lazy* verzije ovih komponenata. Ove komponente su korisne za prikazivanje velikih lista podataka. Tako se za vertikalni prikaz liste komponenata mogu koristiti komponente *LazyColumn* i *LazyVerticalGrid.* Komponente koje će svoje *child* komponente raspoređivati horizontalno, s lijeva na desno su *LazyRow* i *LazyHorizontalGrid.* Ove komponente su izrazito korisne jer štede na resursima aplikacije i iscrtavaju samo komponente koje su trenutno vidljive na zaslonu korisničkog sučelja. Također, ove komponente su po *defaultu scollabilne* kada to treba, dok na primjer, običnom *Columnu* treba dodati mogućnost *scrolla* ako je ona potrebna. U suprotnom, neke komponente možda neće biti vidljive na korisničkom sučelju.

Lazy komponente su nastale kao zamjena za *RecyclerView* koji je u klasičnom pristupu izgradnje korisničkog sučelja korišten za prikazivanje liste podataka. *RecyclerView* je *XML widget* koji se dodavao u druge *layout* komponente i zahtijevao je dodatnu *XML layout* datoteku koja bi definirala izgled jednog *itema* u listi. Osim što je za implementaciju *RecyclerView* potrebno imati dvije *XML* datoteka, potrebno je također implementirati i adapter koji bi vodio brigu o tome kako će se podaci iz liste podataka povezati s definiranim izgledom *itema* i kako će *item* reagirati na korisničke interakcije.

Kako bi doskočili ovom problemu, *Composable* komponente za prikazivanja liste podataka defiliraju nekoliko *overload* funkcija *item* i *items*. Ove funkcije nisu *Composable* funkcije, ali primaju *Composable* sadržaj. Funkcije *item* i *items* su funkcije omotači za *Composable* funkcije koje definiraju izgled jednog *itema.*

```
LazyColumn(
    modifier = Modifier
        .fillMaxSize()
        .padding(paddingValues)
        .padding(PaddingValues(horizontal = MaterialTheme.spacing.extraLarge)),
    contentPadding = PaddingValues(vertical = MaterialTheme.spacing.extraLarge),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Top
) {
    if (levels.isEmpty()) {
        item {
            Text(text = "Lista levela je prazna")
        }
    }
    items(count = levels.size) { index ->
        LevelRow(index = index, isPlayed = playedLevels.contains(levels[index])) {
            onEvent(LevelsScreenEvent.OnLevelClicked(index = index))
        }
    }
    item {
        Button(
            enabled = playedLevels.containsAll(levels),
            onClick = {
                onEvent(LevelsScreenEvent.OnLevelCompleted)
            }) {
            Text(text = "Završi razinu")
        }
    }
}
```

Slika 14 Primjer korištenja komponente *LazyColumn*

Na Slika 14 Primjer korištenja komponente *LazyColumn* je prikazano kako je komponenta *LazyColumn* iskorištena za prikazivanje liste razina u jednoj igri. Funkcija *items* kao prvi argument prima broj razina koje se nalaze u listi razina koje treba prikazati na korisničkom sučelju, a kao drugi argument prima *Composable* sadržaj koji će definirati izgled jednog *itema.* Osim liste razina, na korisničkom sučelju će se prikazati i tekst *„Lista levela je prazna"* ako lista razina nema nijednog člana te će se prikazati i gumb koji služi za spremanje rezultata ostvarenih u igri.

Kod kreiranja korisničkog sučelja, osim komponenata za pozicioniranje komponenti, jako korisna komponenta je komponenta *Scaffold*. *Scaffold* je temeljna struktura koja pruža standardiziranu platformu za složena korisnička sučelja. Ona drži različite dijelove korisničkog sučelja, poput aplikacijskih traka i plutajućih akcijskih gumba, dajući aplikacijama koherentan izgled i dojam. [33]

Compose je deklarativan i stoga je jedini način za ažuriranje korisničkog sučelja pozivanje iste *Composable* komponente s novim argumentima. Ti argumenti predstavljaju stanje korisničkog sučelja. Svaki put kada se stanje ažurira, dolazi do rekompozicije. [34]

Composable funkcije mogu koristiti *remember API* za pohranu objekta u memoriju. Izračunata vrijednost će se pohraniti i neće se mijenjati pod utjecajem rekompozicije. [34]

Stanje se u *Jetpack Composeu* može stvoriti pozivom funkcije *mutableStateOf.* Ova funkcija stvara promjenjivo stanje tipa *MutableState<T>* koje se može promatrati i koje je integrirano s *Compose runtime-om.* Ovaj tip sadrži samo jednu varijablu naziva *value* koje je zadanog tipa *T*. Svaka promjena vrijednosti varijable *value* će zakazati rekompoziciju svih *Composable* funkcija koje čitaju vrijednost atributa *value*. [34]

Stanje se u *Composable* funkcijama može deklarirati na tri načina:

- `val mutableState = remember { mutableStateOf(default) }`
- `var value by remember { mutableStateOf(default) }`
- `val (value, setValue) = remember { mutableStateOf(default) }` [34]

Sljedeće pitanje koje se nameće je gdje čuvati stanje, s obzirom na to da se složena korisnička sučelja mogu sastojati od više *Composable* funkcija koje će čitati to stanje. Odgovor je naravno, „ovisi“. Gdje čuvati stanje ovisi o tome je li stanje potrebno *UI* logici ili poslovnoj logici aplikacije. Najbolje prakse za čuvanje stanja su:

- Stanje treba čuvati u kod najnižeg zajedničkog pretka od svih *Composable* komponenti koje čitaju ili mijenjaju stanje.
- Stanje treba držati najbliže mjestu gdje se koristi.
- Od vlasnika stanja, onima koji ga koriste treba izložiti nepromjenjivo stanje i *evente* za izmjenu stanja.

Najniži zajednički predak također ne mora niti biti *Composable* komponenta. Na primjer, kada se stanje podiže u *ViewModel* jer je uključena poslovna logika. [35]

Dakle ako se radi o jednostavnom stanju koje ne čita više *Composable* komponenata, to stanje se može držati izravno u komponenti koja to stanje čita. Kada je stanje kompleksnije, ono se može izdvojiti u posebne *state holder* klase koje će se brinuti o tom stanju. Ove klase pružaju praktične funkcije prilikom pozivanja *Composable* funkcija kako bi se izbjeglo pisanje kompleksne *UI* logike. *State holder* klase se stvaraju i pamte u kompoziciji. Budući

da slijede životni ciklus *Composable* komponente, mogu prihvatiti tipove koje pruža *Compose* biblioteka, kao što su *rememberNavController* ili *rememberLazyListState*. [35]

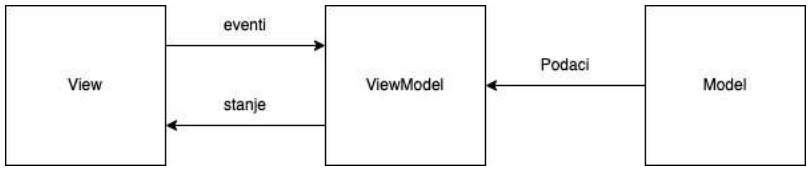
Kada je za promjenu stanja potrebno uključiti i poslovnu logiku aplikacije tu je i potrebno stanje izdvojiti u komponentu koja ima pristup poslovnom sloju aplikacije. Tu ulogu najčešće ima *ViewModel*. *ViewModel-i* nisu pohranjeni kao dio kompozicije. Njih pruža razvojni okvir i oni su povezani s *ViewModelStoreOwnerom*, koji može biti *Activity, Fragment,* navigacijski graf ili odredište navigacijskog grafa. Tada je *ViewModel* izvor istine i najniži zajednički predak za stanje korisničkog sučelja. [35]

2.1.2.1 *MVVM* s *Jetpack Composeom*

MVVM odnosno *Model*, *View, ViewModel* je arhitekturni obrazac koji služi za odvajanje *UI* komponenta od poslovne logike aplikacije. Osim toga, poslovna logika se odvaja od poziva prema bazi podataka ili mrežnih poziva. [36]

U *MVVM* obrascu *child* komponente nemaju izravne reference na *parent* komponentu nego koriste *Observable* tipove podataka. Kao što i ime obrasca sugerira, on se sastoji od tri komponente:

- *Model* - vvaj sloj sadrži podatke. *Model* ne može komunicirati izravno s prikazom (*View).* Umjesto toga, podaci se izlažu *ViewModelu* putem tipova koji se trebaju promatrati.
- *View* - ovaj sloj služi za obavještavanje *ViewModela* o korisničkim akcijama. *View* promatra *ViewModel* i ažurira se sukladno promjenama bez posjedovanja vlastite logike aplikacije. Njegova je funkcija čisto prezentacijska.
- *ViewModel* - Ovaj sloj djeluje kao most između *Modela* i *View*a. Odgovoran je za preoblikovanje podataka iz *Modela* u format pogodan za prikaz. *ViewModel* pruža podatkovne tokove *View*u i koristi *hooks* ili *callbackove* za dinamičko ažuriranje prikaza. Također, preuzima podatke od *Modela* kada je to potrebno, čime osigurava neovisnost *View*a o *Modelu*. [36]



Slika 15 Graf *MVVM* arhitekture

Na *Slika 15 Graf MVVM arhitekture* je prikazana skica grafa *MVVM* arhitekture i tokovi podataka između komponenata. *View* promatra stanje koje *ViewModel* izlaže. Kada dođe do promjene stanja, dolazi do rekompozicije *UI* komponenata koje čitaju to stanje. *View* također prati i korisnikove interakcije i prenosi ih *ViewModelu* u obliku *eventa*. Model pruža podatke *ViewModelu* te ih on obrađuje i prilagođava obliku pogodnom za prikaz na korisničkom sučelju.

Zato što *Composable* komponente primaju stanje i izlažu *evente*, obrazac jednosmjernog protoka podataka dobro se uklapa u *Jetpack Compose*. Jednosmjerni protok podataka je oblikovni obrazac, gdje stanje teče prema dolje, a događaji prema gore. Sljedeći jednosmjerni protok podataka, mogu se odvojiti komponente koje prikazuju stanje u korisničkom sučelju od dijelova aplikacije koji pohranjuju i mijenjaju stanje. [37]

Petlja ažuriranja korisničkog sučelja za aplikaciju koja koristi jednosmjerni protok podataka izgleda ovako:

- *Event* - dio korisničkog sučelja generira događaj i prosljeđuje ga prema gore, kao što je klik na gumb koji se prosljeđuje *ViewModelu* za obradu.
- Ažuriranje stanja - rukovoditelj događaja može promijeniti stanje.
- Prikaz stanja - držatelj stanja prosljeđuje stanje prema dolje, a korisničko sučelje ga prikazuje. [37]

```
KnowledgeItemPickerItem(
    onClick = {
        onEvent(HomeEvent.OnKnowledgeItemClicked(item.name))
    },
    knowledgeItem = item
)
```

Slika 16 Slanje *eventa ViewModelu*

```
is HomeEvent.OnKnowledgeItemClicked -> {
    viewModelScope.launch {

        _homeState.value = _homeState.value.copy(isLoading = true)

        collectFor({
            userService.getRandomLevel(event.knowledgeItemName)
        }, { result ->
            currentGameHelper.setCurrentGame(result)
            _homeState.value = _homeState.value.copy(isLoading = false)
            _uiEvent.emit(UiEvent.NavigateToNextScreen(Graph.LEVELS_LIST))
        }, {
            _homeState.value = _homeState.value.copy(isLoading = false)
            _uiEvent.emit(UiEvent.ShowErrorWhileFetchingGameAlert)
        })

    }

    analyticsHelper.logAnalytics(
        ON_REVISE_KNOWLEDGE_ITEM_CLICKED,
        additionalData: "knowledge_item:${event.knowledgeItemName}"
    )
}
```

Slika 17 Obrada *eventa* u *ViewModelu*

```
if (state.isLoading) {
    Box(
        Modifier
            .fillMaxSize()
            .background(
                alpha = 0.3f,
                brush = Brush.horizontalGradient(
                    listOf(
                        Color.LightGray,
                        Color.LightGray
                    )
                )
            )
        ,
        contentAlignment = Alignment.Center
    ) {
        CircularProgressIndicator()
    }
}
```

Slika 18 Prikazivanje trake učitavanja

Na *Slika 16 Slanje eventa ViewModelu* prikazano je kako *UI* komponenta *KnowledgeItemPickerItem* na korisnikov klik poziva funkciju *onEvent* s odgovarajućim argumentima. Funkcija *onEvent* je deklarirana u *ViewModelu* i to je jedina javno izložena funkcija. Ova funkcija je predana korisničkom sučelju kako bi se omogućila komunikacija prema *ViewModelu*. Funkcija *onEvent* prima objekte tipa *HomeEvent*. Ovo je *sealed* klasa kako bi prije kompiliranja bio određen skup *evenata* koje korisničko sučelje može emitirati.

Na *Slika 17 Obrada eventa u ViewModelu* prikazan je dio *onEvent* funkcije koji je zadužen za obradu *eventa* koji je emitiran na *Slika 16 Slanje eventa ViewModelu*. Obrada ovog *eventa* sastoji se od ažuriranja stanja korisničkog sučelja postavljenjem varijable koja sugerira treba li se prikazati traka za učitavanje. Nakon toga *ViewModel* pokreće asinkroni mrežni poziv. Kada *ViewModel* dobije rezultat mrežnog poziva ponovno ažurira stanje korisničkog sučelja i javlja mu da više ne treba prikazivati traku za učitavanje i zatim, ovisno o tome je li poziv bio uspješan ili ne, emitira *event* za navigaciju na sljedeći ekran ili *event* za prikaz pogreške prilikom dohvata podataka. Ovi *eventi* se obrađuju u *Composable* komponentama.

Na *Slika 18 Prikazivanje trake učitavanja* prikazan je dio korisničkog sučelja koji, ovisno o trenutnom stanju, prikazuje traku učitavanja. Ova traka korisniku signalizira da se njegov klik ili akcija obrađuje u pozadini te da će uskoro dobiti rezultate obrade.

2.1.2.2 Navigacija

Kada gradimo sve kompleksnije aplikacije javlja se potreba za kreiranjem više različitih dijelova sadržaja unutar aplikacije. U *Jetpack Composeu* je navigacija između takvih različitih dijelova sadržaja izričito pojednostavljena zbog *jetpack* biblioteke za navigaciju. Navigacija iz standardnih *jetpack* biblioteka pruža i podršku za aplikacije pisane u *Jetpack Composeu*. [38]

Navigacijska komponenta se sastoji od tri glavna dijela:

1. Navigacijski graf - ovo je resurs koji prikuplja sve podatke vezane uz navigaciju na jednom mjestu. To uključuje sve lokacije u aplikaciji, koje se nazivaju odredištima, kao i moguće putove koje korisnik može slijediti kroz aplikaciju. Može ga se zamisliti kao veliku knjigu koja sadrži sve lokacije u aplikaciji i načine na koje se između njih može kretati.
2. *NavHost* - ovo je jedinstveni *Composable* element koji se možete uključiti u elemente korisničkog sučelja. Prikazuje različita odredišta iz navigacijskog grafa. *NavHost*

povezuje *NavController* s navigacijskim grafom koji specificira odredišne *Composable* elemente između kojih se može navigirati. Kako se korisnik kreće između *Composable* elemenata, sadržaj NavHosta automatski prolazi kroz rekompoziciju. Svako odredište u navigacijskom grafu povezano je s rutom.

3. *NavController – NavController* je centralni API za navigacijsku komponentu. Ova komponenta zadržava stanje i prati stog povratka *Composable* komponenata koje čine zaslone u aplikaciji. [38]

```
@Composable
fun RootNavigationGraph(
    navController: NavHostController
) {
    NavHost(
        navController = navController,
        route = Graph.ROOT,
        startDestination = Graph.WELCOME
    ) {

        composable(route = Graph.WELCOME) {
            val viewModel: WelcomeScreenViewModel = hiltViewModel()
            WelcomeScreen(
                navController = navController,
                onEvent = { viewModel.onEvent(it) },
                viewModel.uiEvent
            )
        }
```

Slika 19 Prikaz navigacije u aplikaciji

Na *Slika 19 Prikaz navigacije u aplikaciji* prikazana je korijenska *Composable* komponenta *RootNavigationGraph* koja kao argument prima *NavHostController*. U komponenti *RootNavigationGraph* deklariran je *NavHost* koji sadrži sve *Composeable* komponente koje čine zaslon aplikacije na svojim jedinstvenim rutama. Pojedinom zaslonu se predaje objekt *navController* koji dalje brine o navigaciji između zaslona. Kada je potrebno navigirati korisnika na sljedeći prikaz zaslona, poziva se funkcija *navControllera* i predaje mu se ruta na koju je potrebno navigirati korisnika. Ako je korisnika potrebno navigirati na prethodno prikazani zaslon, dovoljno je pozvati funkciju *navControllera* da trenutnu rutu makne s vrha stoga te će se tako automatski navigirati na prethodnu aktivnu rutu.

```
LaunchedEffect(key1 = true) {
    uiEvent.collectLatest { event ->
        when (event) {
            is LevelsScreenViewModel.UiEvent.NavigateToNextScreen -> {
                navController.navigate(route = event.route)
            }

            is LevelsScreenViewModel.UiEvent.CloseLevelsScreen -> {
                navController.popBackStack(Graph.HOME, inclusive = false)
            }
        }
    }
}
```

Slika 20 Funkcije *navControllera*

Na *Slika 20 Funkcije navControllera* prikazan je primjer korištenja funkcije *navigate* koja prima rutu do *Composable* komponente na koju treba navigirati korisnika. Ako se pokuša navigirati do rute koja nije definirana u navigacijskom grafu, dolazi do iznimke. Na *Slika 20 Funkcije navControllera* također je prikazan i slučaj korištenja funkcije *popBackStack*. Ova funkcija, kada je pozvana bez argumenta, će ukloniti trenutnu rutu sa stoga zaslona i navigirati korisnika na prethodnu aktivnu rutu. U ovom slučaju funkcija je pozvana s dva argumenta, jedan je ruta do koje treba isprazniti stog, a drugi je zastavica koja govori treba li *navController* i tu rutu maknuti sa stoga.

2.1.2.3 *Hilt*

Hilt je biblioteka za ubrizgavanje ovisnosti za Android koja smanjuje opsežan kod potreban za ručno ubrizgavanje ovisnosti u projektu. Ručno ubrizgavanje ovisnosti zahtjeva da se ručno konstruiraju sve klasu i njezine ovisnosti te korištenja kontejnera za ponovno korištenje i upravljanje ovisnostima. *Hilt* pruža standardni način korištenja ubrizgavanja ovisnosti u aplikaciji pružanjem spremnika za svaku Android klasu u projektu i automatskim upravljanjem njihovim životnim ciklusima. *Hilt* je izgrađen na osnovu popularne biblioteke za ubrizgavanje ovisnosti *Dagger* kako bi se iskoristile prednosti točnosti u vrijeme prevođenja, performansi u vrijeme izvođenja, skalabilnosti i podrške za Android Studio koje pruža *Dagger*. [39]

Sve aplikacije koje koriste *Hilt* moraju sadržavati klasu *application* koja je označena s *@HiltAndroidApp. @HiltAndroidApp* pokreće generiranje *Hiltovog* koda, uključujući osnovnu klasu za aplikaciju koja služi kao spremnik ovisnosti na razini aplikacije. [39]

Slika 21 *Application* klasa

Na *Slika 21 Application klasa* prikazana je klasa *Application* koja služi isključivo kao ulazna točka za ubrizgavanje ovisnosti.

Hilt je preporučeno rješenje za ubrizgavanje ovisnosti u Android aplikacijama i radi besprijekorno s *Composeom*. *Hilt* može konstruirati i *ViewModel* klase kojima je potrebno dodati anotaciju *@HiltViewModel.* Kako bi se dobila instanca klase *ViewModela*, treba koristiti funkciju *hiltViewModel*. [40]

Na *Slika 19 Prikaz navigacije u aplikaciji* može se vidjeti primjer korištenja ubrizgavanja *ViewModel* klase s pomoću funkcije *hiltViewModel*. Instanca potrebnog *ViewModela* se ubrizgava u *Composable* komponentu, a ono u čemu dolazi do izražaja korisnost biblioteke *Hilt*, također se ubrizgavaju i potrebne ovisnosti u konstruktor klase *ViewModel.* Potrebne ovisnosti za kreiranje tog *ViewModela*, mogu se vidjeti na *Slika 22 Ubrizgavanje ovisnosti u konstruktor ViewModela*

Slika 22 Ubrizgavanje ovisnosti u konstruktor *ViewModela*

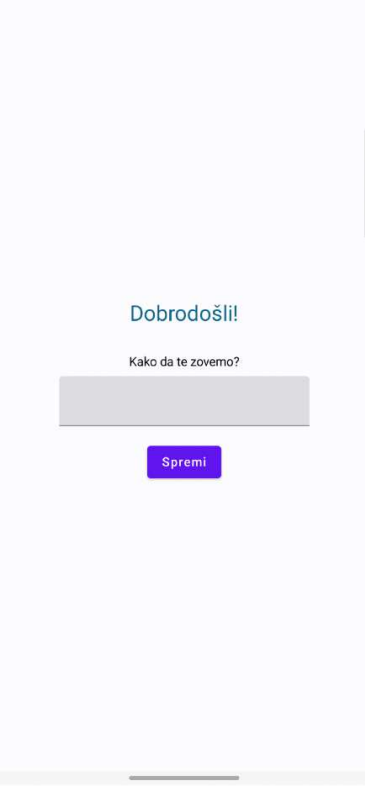
2.1.3. Aplikacija *SmartQuest*

U sklopu ovog rada izrađena je mobilna aplikacija za uređaje s operacijskim sustavom Android pod nazivom *SmartQuest*. Aplikacija je razvijena s ciljem testiranja kako personalizirani pristup igrifikaciji utječe na ishode učenja. *SmartQuest* sadrži igre koje su podijeljene na razine, a razine dolaze u dva oblika:

- Razine s pričom
- Razine koje sadrže samo zadatke

Razine s pričom temelje se na nelinearnim pričama koje korisnicima nude više načina prolaska kroz igru, omogućujući im da sami biraju svoj put kroz obrazovni sadržaj. Ovaj pristup omogućuje personalizirano iskustvo učenja, prilagođavajući se interesima i preferencijama korisnika. S druge strane, razine koje sadrže samo zadatke, fokusiraju se na rješavanje konkretnih izazova i kvizova, kao što su „ABCD“ pitalice, *memory* igre i sparivanje parova, čime se testira poznavanje nastavnog sadržaja.

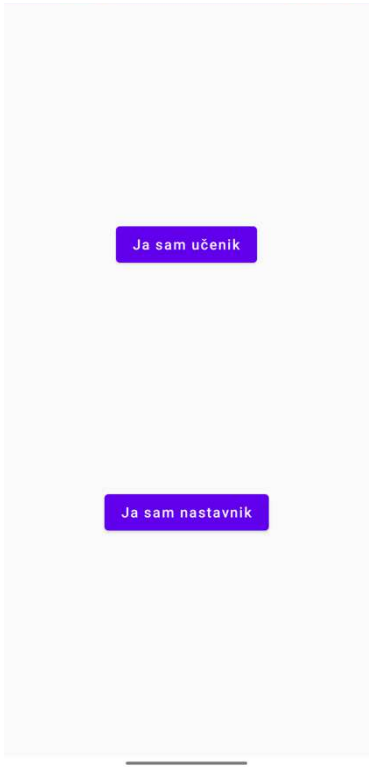
Prilikom prvog ulaska u aplikaciju, korisnik se treba „registrirati“. Ovdje nije korišten nikakav sustav autentifikacije jer nema osjetljivih podataka i pretpostavka je kako će svaki korisnik koristit svoju instancu aplikacije, instaliranu na vlastitom uređaju. Zbog navedenih pretpostavki, prilikom registracije korisnika se traži da upiše samo ime s kojim će ga se oslovljavati. Ime koje korisnik navede koristi se samo na ljestvici najboljih igrača.



Slika 23 Registracija korisnika

Nakon registracije, i svaki sljedeći put kada korisnik uđe u aplikaciju, prikazuje se početni zaslon aplikacije. Na početnom zaslonu aplikacije, korisnik bira koju funkcionalnost aplikacije želi koristiti. Ako korisnik želi igrati kreirane obrazovne igre, treba stisnuti gumb

s naslovom „Ja sam učenik“. U suprotnom, ako korisnik želi kreirati novu igru, potrebno je stisnuti gumb s naslovom „Ja sam nastavnik“.



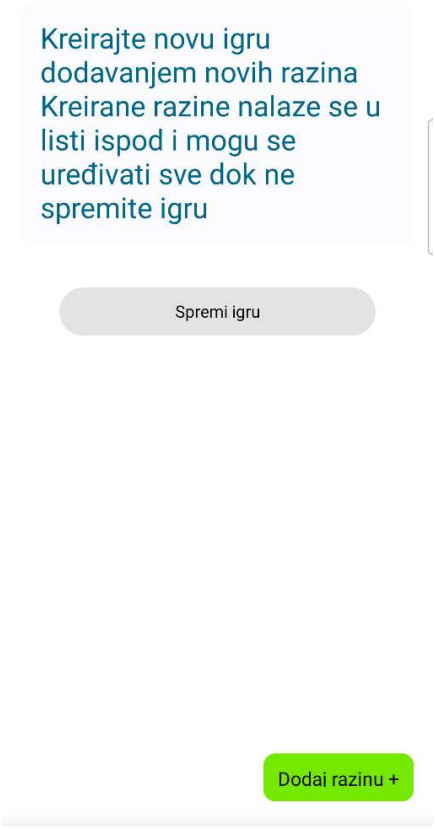
Slika 24 Odabir funkcionalnosti

Klikom na gumb s naslovom „Ja sam nastavnik“ otvara se zaslon za kreiranje nove obrazovne igre. Svaka igra se sastoji od jedne ili više razina pa je tako prvi korak u kreiranju nove igre, dodavanje razine u igru. Jednom kada korisnik otvori zaslon za kreiranje igara, na njemu može vidjeti naslov koji mu govori kako se nalazi u dijelu aplikacije namijenjenom za stvaranje nove igre. Ispod naslova nalazi se i lista razina koje su dodane u novu igru. Kako je korisnik tek ušao na ovaj zaslon, lista razina je prazna. Ispod liste razina, nalazi se gumb za spremanje igre s naslovom „Spremi igru“. Kada je lista razina prazna, ovaj gumb je onemogućen.

Posljednja stvar koju korisnik može vidjeti na ovom ekranu je zeleni gumb s naslovom „Dodaj novu razinu +“. Ovaj gumb nalazi se u donjem desnom kutu ekrana. Ovo je tzv. *floating action button (FAB)* što bi u slobodnom prijevodu značilo lebdeći ili plutajući akcijski gumb. Posebnost ovog tipa gumba je što je on uvijek vidljiv na zaslonu i „lebdi“ iznad ostalog sadržaja.

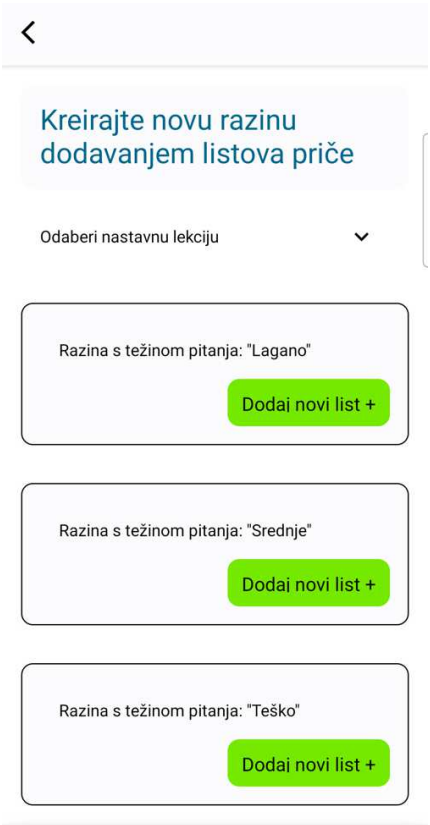
Ovaj tip gumba vrlo je učestala pojava u nativnim aplikacijama za Android uređaje i najčešća primjena mu je dodavanje novog elementa ili kreiranje sadržaja. Ovaj tip gumba može se

naći npr. u aplikacijama za upravljanje elektroničkom poštom gdje služi za kreiranje nove poruke ili u aplikaciji za bilješke, u kojoj služi za stvaranje nove bilješke. Tako se ovaj gumb, na zaslonu za kreiranje igre, koristi za dodavanje nove razine u igru.

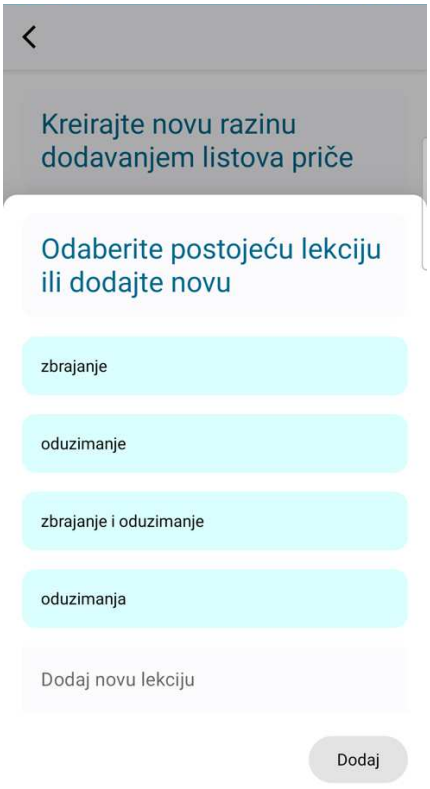


Slika 25 Zaslon za kreiranje nove igre

Kada korisnik pritisne gumb za dodavanje nove razine, prikazuje mu se sučelje za stvaranje nove razine. Jedna razina mora sadržavati svoju nastavnu lekciju i mora sadržavati tri vrste priče za tri težine jer svaka razina dolazi u tri različite težine. Težine su označene kao „Lagano“, „Srednje“ i „Teško“. Za svaku težinu razine potrebno je dodati jedan ili više „listova“ priče.



Slika 26 Kreiranje nove razine



Slika 27 Odabir nastavne lekcije

Nastavna lekcija se odabire pritiskom na tekst „Odaberi nastavnu lekciju“ nakon čega se preko donjeg dijela zaslona pojavljuje novi dio zaslona koji se prikazuje iznad osnovnog sadržaja. Ovakav tip prikaza sadržaja na zaslonu naziva se *Bottom Sheet*. *Bottom Sheet* elementi sve su zastupljeniji u modernim aplikacijama i uglavnom se koriste za prikazivanje detalja o nekom objektu na zaslonu ili kao zamjena za zastarjele *dropdown* izbornike.

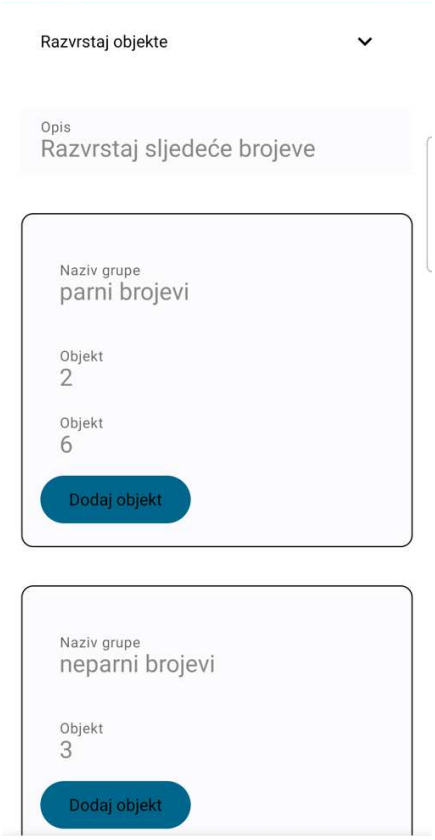
Otvaranjem *Bottom Sheeta* korisniku se nude opcije odabira jedne od postojećih nastavnih lekcija ili dodavanje nove lekcije. Nakon odabira željene lekcije, korisnik može krenuti dodavati svoje listove u priču. Pritiskom na gumb s naslovom „Dodaj novi list +“ korisniku se otvara sučelje za stvaranje listova priče. Postoji sedam vrsta listova koje korisnik može dodati. Svaki tip lista priče ima drugačiju ulogu u stvaranju priče. Ti tipovi su:

- List grananja: na ovom listu korisnik može dodati dva do četiri odabira. Svaki odabir predstavlja novi *storyline* koji igrač može odabrati. Svaki odabir treba imati svoj naziv, opis i sliku koja će prikazivati taj odabir. Nakon lista grananja potrebno je dodati barem jedan sljedeći list za svaki mogući odabir.
- Igra *memory*: na ovom listu korisnik dodaje parove kartica za igru *memory*. Igra *memory* se sastoji od parnog broja kartica koje imaju prednju i stražnju stranu. Stražnje strane kartica izgledaju jednako te igra počinje tako da su sve kartice okrenute stražnjom stranom prema gore. Cilj igre je spariti dvije kartice čije prednje strane čine jedan par. Jedna kartica može biti dio samo jednog para i igraču bi trebalo biti intuitivno prepoznati koje dvije kartice čine par. Igra se igra tako da igrač okreće dvije kartice prednjom stranom prema gore i ako te kartice čine par, one ostaju okrenute prednjom stranom prema gore, u suprotnom kartice je potrebno vratiti u početnu poziciju, stražnjom stranom prema gore. Igra je gotova kada su sve kartice uparene. Prilikom kreiranja lista, korisnik može dodati opis igre koji bi na primjer, pomogao igraču u sparivanju kartica ili kako bi igru bolje uklopio u priču.
- List priče: ovaj list je osnovni list za stvaranje priče, na kojem korisnik dodaje tekst priče, sliku koja će se prikazivati kao pozadina zaslona i lik pripovjedača koji „priča“ tekst kojeg je korisnik dodao.
- ABCD pitanje: na ovom listu korisnik dodaje pitanje s četiri ponuđena odgovora. Korisnik mora označiti koji je od ponuđenih odgovora točan odgovor.
- Kolo sreće: na ovom listu korisnik stvara kolo sreće s nagradama. U igri postoje četiri nagrade koje igrači mogu koristiti na listovima priče u kojima se provjerava njihovo znanje. Nagrade mogu iskoristiti kako bi ostvarili veći broj bodova ili kako

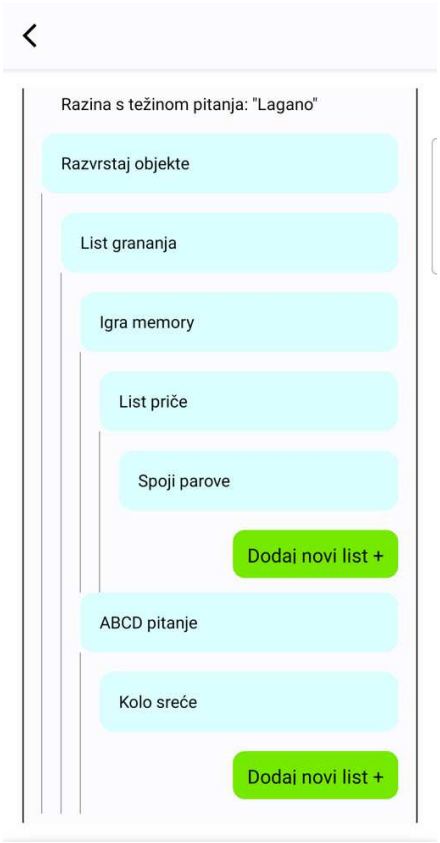
bi dobili pomoć prilikom odgovaranja na pitanja. Prilikom kreiranja ovog lista, korisnik može dodati veći broj jedne vrste nagrade kako bi povećao vjerojatnost za njeno dobivanje, odnosno smanjio vjerojatnost za dobivanje drugih nagrada.

- Spoji parove: na ovom listu korisnik kreira kartice koje čine parove. Cilj igre je spariti sve kartice.
- Razvrstaj objekte: na ovom listu korisnik dodaje jednu ili više „košara“ u koje treba staviti određene objekte. Prilikom kreiranja lista, korisnik dodaje objekte u njihove košare. Prilikom igranja ovog lista, košare će se nalaziti na dnu zaslona, a objekti na vrhu. Cilj igre je dovući objekte u njihove odgovarajuće košare. Korisnik također može dodati objekte koji nemaju svoju košaru. Igra završava kada su svi objekti koji imaju dodijeljenu košaru pridruženi odgovarajućoj košari.

Sadržaj sučelja za kreiranje lista priče se dinamički izmjenjuje ovisno o odabranom tipu lista koji se želi dodati. Jednom kada su dodani svi potrebni elementi za odabrani tip lista priče, gumb s naslovom „Spremi list“ postaje omogućen. Pritiskom na taj gumb, list priče se dodaje u razinu, na razinu težine s koje je bilo otvoreno sučelje za kreiranje lista.



Slika 28 Primjer kreiranja lista priče "Razvrstaj objekte"



Slika 29 Dodavanje listova priče u razinu

Nakon dodavanja listova priče u razinu, pritiskom na karticu lista otvara se sučelje za uređivanje već dodanog lista priče. Također, posljednji dodani list priče se može izbrisati povlačenjem s desna na lijevo. Jednom kada je razina dovršena i dodani su svi potrebni elementi potrebno je pritisnuti gumb „Spremi razinu“ što će dodati razinu u listu razina te se može krenuti na dodavanje novih razina ili spremanje igre. Nakon što je igra spremljena kreira se QR kod s kojim se može pristupiti igri, ali i broj koji predstavlja pin pod kojim se igra može dohvatiti. Za generiranje QR koda korištena je javno dostupna biblioteka treće strane pod nazivom *Lightspark*.

Slika 30 Spremanje igre

Odabirom opcije „Ja sam učenik“ otvara se početno sučelje za igranje kreiranih obrazovnih igara. Ovo sučelje ima nekoliko mogućnosti. Glavna funkcionalnost je dohvaćanje obrazovnih igara. Igre se mogu dohvatit skeniranjem QR koda ili upisivanjem pin koda.

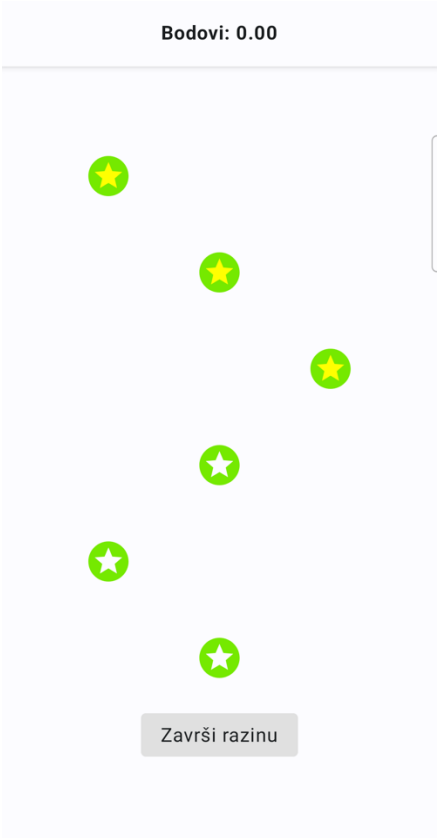
Za dohvaćanje igre putem QR koda potrebno je pritisnuti zeleni gumb u gornjem desnom dijelu zaslona s ikonom koja simbolizira QR kod. Pritiskom ovog gumba otvara se kamera uređaja. Za otvaranje i korištenje kamere uređaja, aplikacija mora dobiti eksplicitno dopuštenje od korisnika. Tek kada korisnik dopusti aplikaciji da koristi resurse uređaja, može se otvoriti okno kamere kako bi se skenirao QR kod. Jednom kada korisnik dodijeli dopuštenje aplikaciji za korištenje kamere, ne mora ponovno pitati za dopuštenje dok se to dopuštenje ne ukloni. Za prepoznavanje QR koda korištena je javno dostupna Google-ova biblioteka pod nazivom *ZXing.* Ova biblioteka prepoznaje QR kod, analizira ga i vraća rezultat. Rezultat QR koda će biti brojevni kod pod kojim se igra može dohvatiti, te će se automatski pokrenuti dohvaćanje igre pod tim kodom.

Za dohvaćanje igre putem koda, potrebno je upisati brojevni kod pod kojim je igra spremljena te pritisnuti gumb „Dohvati igru“.

Početno sučelje za igranje igara ima još dvije komponente. Prva komponenta je odjeljak koji prikazuje igračeve nastavne lekcije s najslabijim rezultatom. Prikazuju se tri nastavne lekcije, odnosno manje ako igrač nije igrao razine s drugim nastavnim lekcijama. Pritiskom na karticu nastavne lekcije dohvaća se nasumično izabrana razina s odabranom nastavnom lekcijom. Na ovaj način igrač može ponavljati gradivo nastavne lekcije u kojoj ima lošije rezultate.

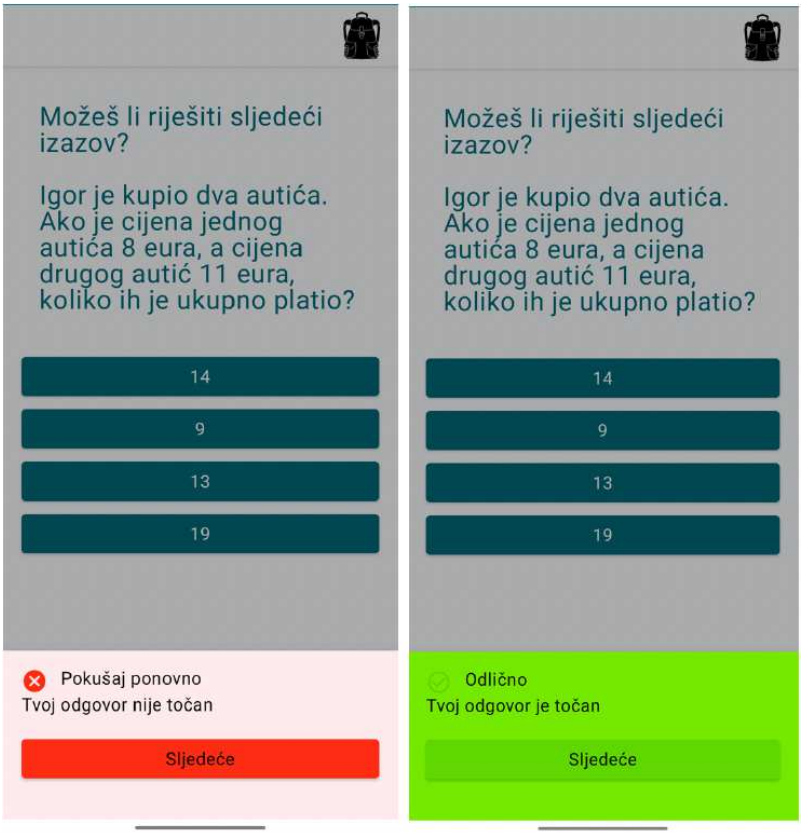
Posljednja komponenta sučelja za igranje igara je tablica rezultata najboljih igrača koja prikazuje ukupne bodove i imena tri najbolja igrača. Na ovaj način potiče se natjecateljski duh među igračima. Onima koji se nalaze na tablici može se pobuditi motivacija za igranje igara kako bi ostali na tablici najboljih igrača, dok igrači koji ne vide svoje ime na tablici, mogu biti motivirani željom za sustizanjem onih ispred sebe. Svoje trenutne bodove igrač uvijek može vidjeti na vrhu ovog sučelja.

Nakon dohvaćanja igre na jedan od opisanih načina ili putem odabira nastavne lekcije za ponavljanje, otvara se sučelje za biranje razina u igri. Jedna razina označena je ikonom zvjezdice. Na početku sve razine označene su bijelom zvjezdicom, a nakon što je pojedina razina odigrana, ona je označena žutom zvjezdicom i ta razina se više ne može igrati. Na vrhu zaslona igrač može vidjeti koliko je do sada ostvario bodova u ovoj igri. Jednom kada su sve razine odigrane, igrač može završiti igru pritiskom na gumb „Završi razinu“. Pritiskom na ovaj gumb, igrač se ponovno vraća na početno sučelje za igranje igara, a ostvareni bodovi se dodaju ukupnom broju bodova. Također, ažuriraju se nastavne lekcije s najslabijim rezultatima, ali i tablica s rezultatima najboljih igrača.



Slika 31 Prikaz liste razina

Jedna razina se sastoji od jednog ili više listova priče. Odgovaranjem na pitanja, igrač sakuplja bodove. Bodovi se ostvaruju na listovima na kojima se provjerava znanje. To su listovi: Igra *memory*, ABCD pitanje, Spoji parove i Razvrstaj objekte. Bodovanje u svim igrama funkcionira na jednak način. Na početku igrač ima mogućnost ostvariti maksimalno 10 bodova. Korisnik osvaja maksimalan broj bodova ako iz prvog pokušaja točno odgovori na pitanje, odnosno riješi igru bez pogreške. Svakom pogreškom koju igrač napravi, maksimalan broj bodova se smanjuje. Za svaku pogrešku koju igrač napravi, trenutna vrijednost maksimalnih mogućih bodova se množi s faktorom 0,75. Ako igrač ponudi dva kriva odgovora, a zatim točan odgovor bodovi će se računati tako da se početnih 10 bodova pomnože s 0,75 što postavlja vrijednost maksimalnih mogućih bodova na 7,5, zatim će se pomnožiti 7,5 bodova s 0,75 što postavlja vrijednost maksimalnih mogućih bodova na 5,625 te konačno, točnim odgovorom igrač osvaja 5,625 bodova. Igra se na svakom listu završava točnim odgovorom, odnosno dok igrač ne ponudi točan odgovor, ne može krenuti na sljedeći list priče.



Slika 32 Prikaz stanja zaslona nakon netočnog i nakon točnog odgovora

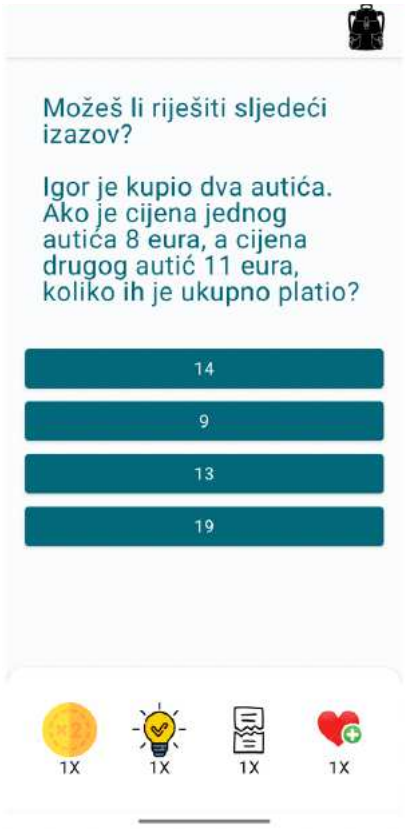
Prilikom rješavanja jedne razine, igrač sakuplja bodove koji se na kraju razine zbrajaju i dijele s brojem maksimalnih mogućih ostvarenih bodova kako bi se dobio rezultat riješenosti razine. Ovisno o rezultatu riješenosti razine, igrači se razvrstavaju u težinske skupine za nastavnu cjelinu koja je pridružena toj razini. Postoje tri težine zadataka, „Lagano“, „Srednje“ i „Teško“ koje su stvorene prilikom kreiranja razine. Ovisno o tome u kojoj težinskoj razini se igrač nalazi, dobivat će verzije razine te težine.

Ako igrač riješi razinu s rezultatom većim od 0.8, odnosno postotkom riješenosti većim od 80 %, igrač će biti smješten u sljedeću težinsku kategoriju, ako takva postoji. Odnosno, ako je igrač trenutno na razini „Srednje“ za nastavnu lekciju „Zbrajanje“, taj igrač će dobivati zadatke iz razine „Teško“ kada bude igrao sljedeću razinu s pridruženom nastavnom lekcijom „Zbrajanje“.

Nadalje ako igrač ima rezultat razine manji od 0,5 odnosno manje od 50 % riješenosti, taj igrač će biti smješten u razinu niže, ako takva postoji. Ako je rezultat riješenosti između 0,5 i 0,8 igrač će ostati u trenutnoj težinskoj kategoriji. Također, igrač ostaje u istoj težinskoj kategoriji ako se već nalazi na maksimalnoj težini zadataka, a rezultat razine bude veći od 0,8 ili ako se nalazi na minimalnoj težini zadataka, a rezultat razine bude manji od 0.5.

Prilikom igranja zadataka s listova priče na kojima se provjerava znanje, igrači imaju pristup svojoj „torbi za *hintove*“. U njoj se nalaze predmeti koje igrači mogu osvojiti na listi „Kolo sreće“ i koji im mogu pomoći pri rješavanju zadataka ili kako bi povećali broj osvojeni bodova. Ta četiri predmeta su:

1. Dvostruki bodovi na trenutnom pitanju
2. Spoji jedan par / ukloni jedan odgovor
3. Spoji pola parova / makni pola odgovora
4. Zanemari jedan pogrešan odgovor



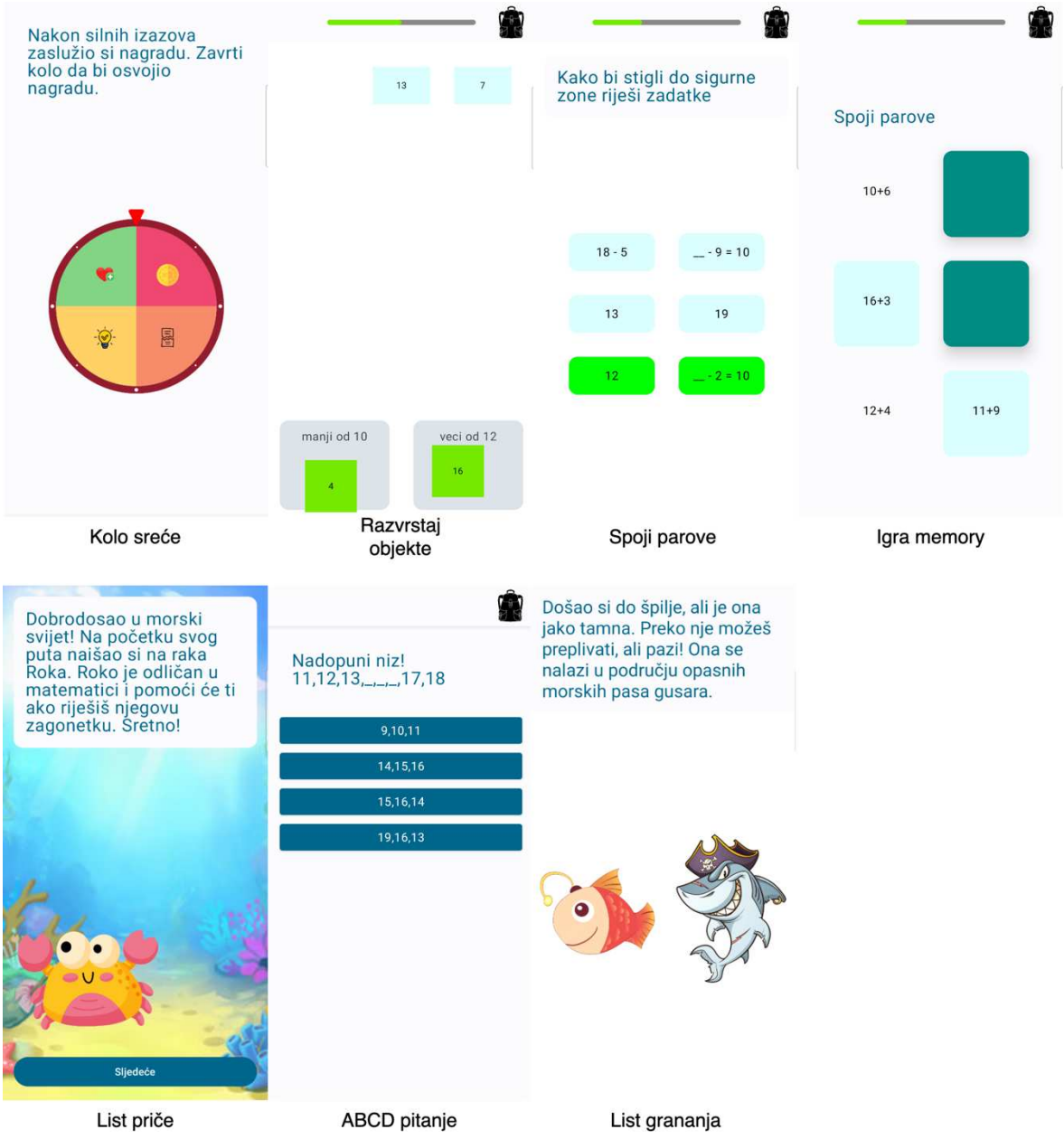
Slika 33 Prikaz sadržaja torbe s *hintovima*

Predmet za udvostručavanje bodova množi trenutnu vrijednost maksimalnog mogućeg broja bodova s 2, neovisno o tome je li se on smanjio zbog prijašnjih netočnih odgovora. Također, ovaj predmet utječe i na moguć ukupni broj bodova u razini te se povećavaju i ti bodovi, kako bi se osiguralo da rezultat razine ne bude veći od 1.

Predmet koji spaja jedan par / uklanja jedan odgovor i predmet koji spaja pola parova / uklanja pola odgovora su slični i djeluju na jednak način. Kod igre *memory* i igre spajanja parova, uparit će jednu, odnosno pola kartica. Kod ABCD pitanja uklonit će jedan, odnosno dva odgovora od ponuđenih, a da to nije točan odgovor. Naposljetku, kod igre u kojoj je

potrebno razvrstati objekte, ispod objekta će se pojaviti naziv odgovarajuće košare u koju taj objekt treba razvrstati.

Posljednji predmet je predmet koji zanemaruje jedan pogrešan odgovor. Taj predmet se može iskoristiti tek nakon što je igrač ponudio barem jedan pogrešan odgovor. Kada se iskoristi ovaj predmet, maksimalan mogući broj bodova će se pomnožiti s 1/0,75 što će vratiti vrijednost maksimalnih mogućih bodova na razinu prije pogrešnog odgovora.



Slika 34 Prikaz primjera sedam mogućih listova priče

2.2. Poslužitelj

U sklopu razvijenog sustava bilo je potrebno razviti i centralni poslužitelj s kojim će mobilna aplikacija komunicirati. Mobilna aplikacija mora komunicirati s poslužiteljem kako bi mogla dohvatiti ili pohraniti obrazovne igre, pratiti igračev napredak za pojedine nastavne lekcije, ažurirati korisnikove ukupne bodove te dohvaćati tablicu rezultata najboljih igrača.

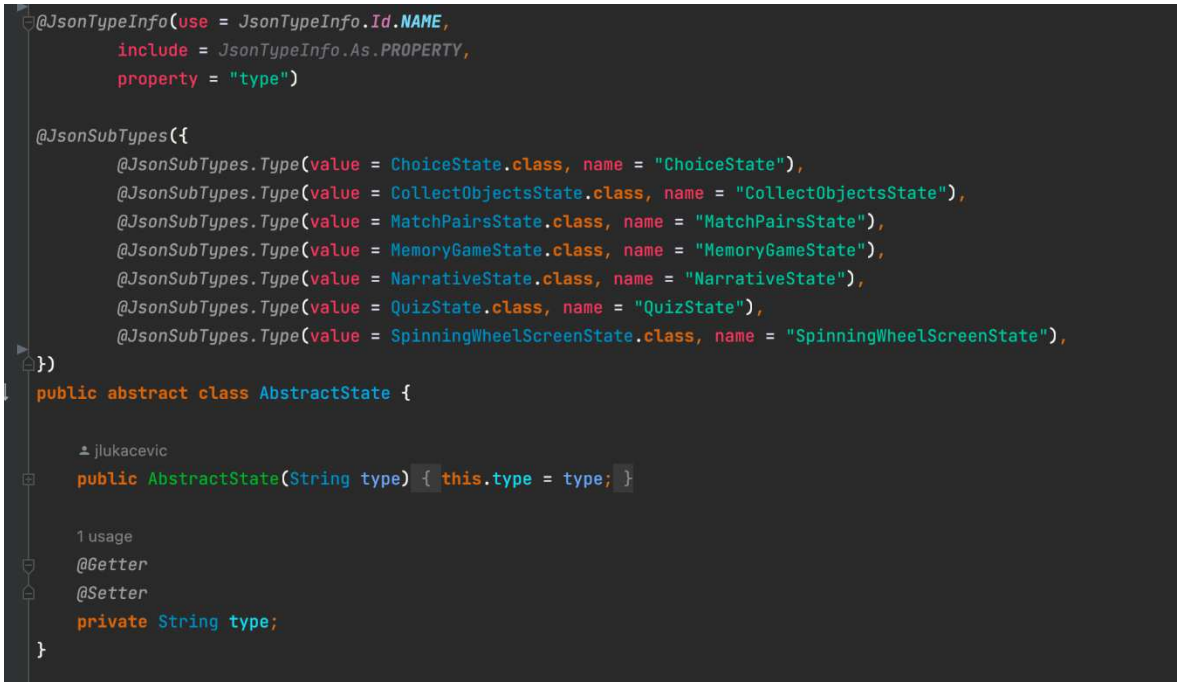
Razvijen je jednostavan poslužitelj u programskom jeziku Java, u razvojnom okuženju *Spring Boot*. *Spring Boot* je poznata tehnologija, koja se zadržala kao jedan od čestih alata za izradu poslužiteljskih aplikacija.

Poslužiteljska aplikacija je organizirana u troslojnu arhitekturu koja dijeli aplikaciju na sloj *Controllera,* sloj *Servisa* i *DAO* sloj. Kontroleri su zaduženi za obradu *HTTP* zahtjeva. U sklopu ovog sustava, kontroleri zaprimaju zahtjeve koje šalju klijentske mobilne aplikacije, pozivaju odgovarajuće servise i vraćaju odgovore klijentu. Servisi sadrže poslovnu logiku aplikacije. U ovom slučaju to je uglavnom pretvaranje *DTO (Data Transfer Object)* objekata u *Entity* objekte i obrnuto, te prosljeđivanje *Entity* objekata repozitorijima koji će se pobrinuti za pohranu podataka. Ti repozitoriji čine treći, *DAO* (*Data Access Object*) sloj aplikacije. Ovaj sloj je odgovoran za pristup podacima i komunikaciju s bazom podataka, ali *Spring Boot* ovo olakšava korištenjem *Spring Data JPA* biblioteke s pomoću koje možemo jednostavno definirati repozitorije koji automatski pružaju osnovne *CRUD* (*Create, Read, Update, Delete*) operacije.

Dvije osnovne funkcionalnosti ovog poslužitelja su pohranjivanje i dohvaćanje obrazovnih igara. Uloga poslužitelja je pretvaranje tipova podataka između dviju strana s kojima komunicira. Poslužitelj s jedne strane komunicira s mobilnom aplikacijom. Mobilna aplikacija komunicira s objektima prikazanim *JSON* datotekama. S druge strane poslužitelj komunicira s relacijskom bazom podataka koja svoje podatke organizira na drugačiji način te je objekte koje koristi mobilna aplikacija potrebno prilagoditi pravilima pohrane podataka u relacijskoj bazi podataka. *Spring Boot* ovdje uvelike olakšava posao svojim anotacijama iz *Spring Data JPA* biblioteke, ali neki objekti i dalje zahtijevaju poseban pristup pri mapiranju.

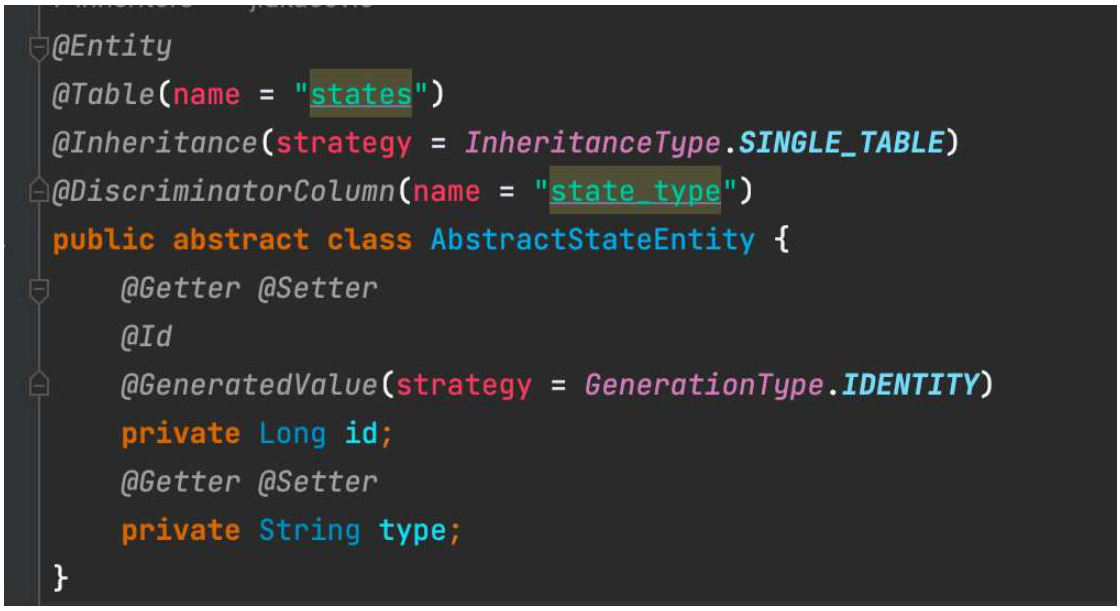
U aplikacijama koje su kreirane objektno-orijentiranim pristupom, često se koristi princip nasljeđivanja kako bi se objekti različitih klasa koji dijele istu baznu klasu mogli tretirati na jednak način. Tako se u mobilnoj aplikaciji različite igre tretiraju na jednak način jer dijele istu baznu klasu, a pojedinosti vezane za konkretne igre, definirane su u njihovim izvedenim

klasama. Kako bi se takvi objekti spremili u bazu podataka, potrebno je navesti koji sve oblici bazne klase mogu postojati te kako će se moći odrediti o kojem izvedenom tipu se radi.



Slika 35 Prikaz definiranja polimorfnih stanja

Na *Slika 35 Prikaz definiranja polimorfnih stanja* prikazano je kako su s pomoću anotacija definirana stanja različitih listova priče. Svako stanje je definirano klasom koja nasljeđuje klasu *AbstractState*. Preko ove klase, sva stanja moraju imati atribut *type* prema kojemu se određuje o kojem izvedenom tipu se radi. Jednom kada je apstraktnom tipu dodijeljen izvedeni tip, može se krenuti s mapiranjem objekata u *Entity* objekte. Svi *Entity* objekti također moraju nasljeđivati jednu apstraktnu klasu u kojoj će se definirati tip nasljeđivanja.



Slika 36 Prikaz apstraktne *Entity* klase

Na *Slika 36 Prikaz apstraktne Entity klase* prikazana je apstraktna klasa *AbstractStateEntity* koju nasljeđuju sve konkretne *Entity* klase koje definiraju stanje. Anotacijom *@Inheritance* određuje se kako će se sva izvedena stanja spremati u jednu relacijsku tablicu podataka, a kada se ta stanja budu dohvaćala iz baze podataka, automatski će se dohvatiti samo objekti izvedenog stanja.

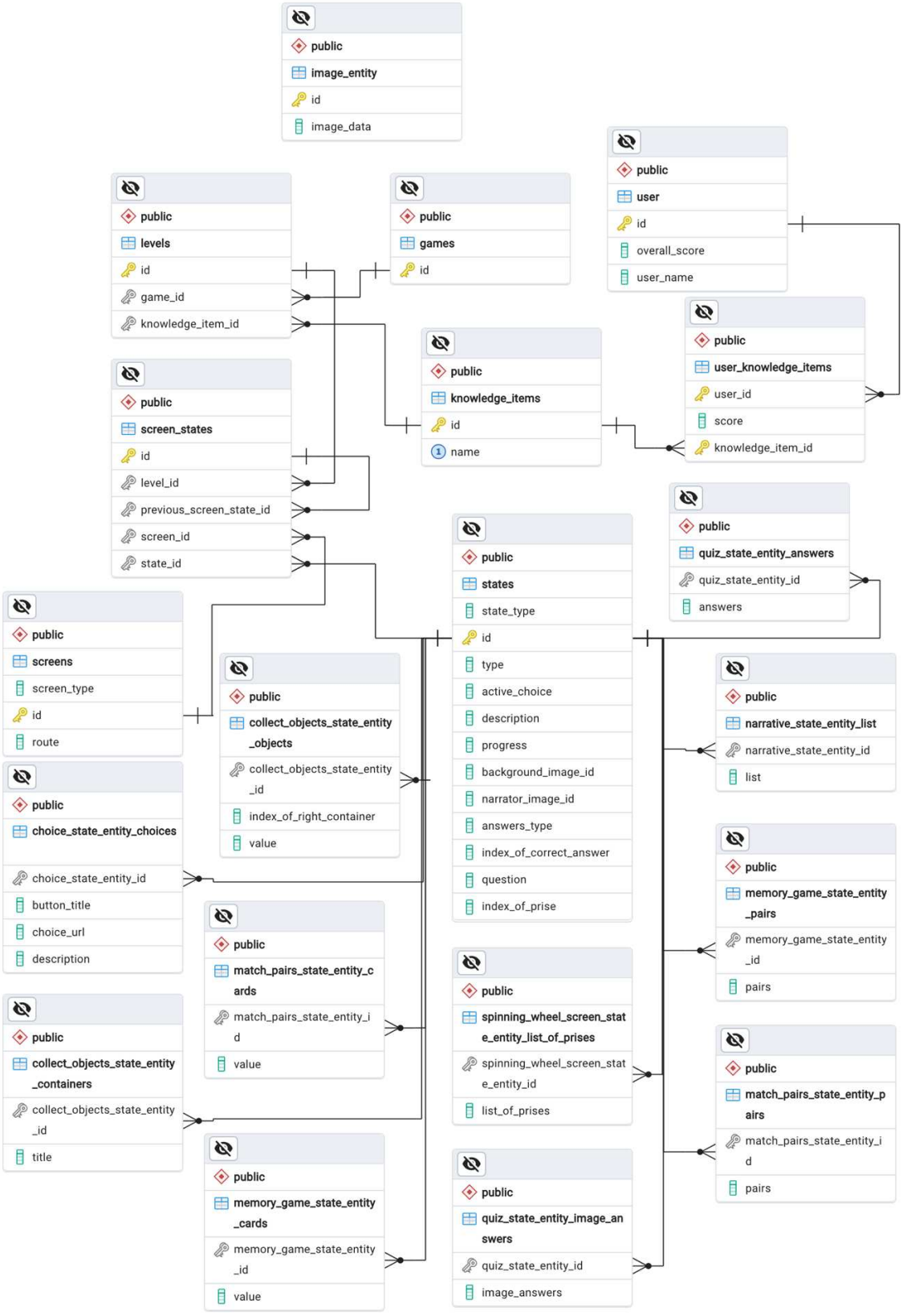
Osim pohrane i dohvaćanja igara, potrebno je i pohraniti podatke o korisnicima i informacije o njihovom napretku za određenu nastavnu lekciju ili njihove ukupne bodove. U ovom slučaju mapiranje objekata je jednostavno. Dodatno, nakon svake odigrane razine, ažurira se lista nastavnih lekcija kojima je igrač pristupio te njegova razina znanja za tu nastavnu cjelinu. Kada se igrač prvi puta susreće se nekom nastavnom cjelinom, kreće od najlakših zadataka. Također, kada igrač završi neku od igara, ažuriraju se njegovi ukupni bodovi.

2.3. Baza podataka

Za potrebe ovog rada korištena je relacijska baza podataka *PostgreSQL*. Na *Slika 37 Shema baze podataka* je prikazana shema baze podataka. Na shemi se može vidjeti kako mnoštvo manjih tablica ima referencu na tablicu *state* koja predstavlja sedam različitih stanja za pojedine listove priče.

Također, može se vidjeti kako tablica *image_entity* nije povezana ni s jednom drugom tablicom. U ovoj tablici se čuvaju slike za pojedine listove priče. Slike nisu izravno povezane s pripadajućom igrom jer bi tako dohvaćanje slike usporilo dohvat cijele igre. Na ovaj način

dohvat igre kratko traje, a slike se dohvaćaju jedna po jedna kada ih je potrebno prikazati na zaslonu.



Slika 37 Shema baze podataka

3. Priprema istraživanja

Za provedbu istraživanja osmišljeno je nekoliko obrazovnih igara. Sustav je implementiran tako da korisnici svojim odabirima mogu mijenjati putanju učenja, odnosno personalizirati tok igre. Kako bi povećali motivaciju većeg broja korisnika, potrebno je igre dizajnirati tako da svaki tip osobnosti može pronaći put koji će baš njemu biti zanimljiv i motivirati ga da dovrši igru.

3.1. Tehnike personalizacije

Tehnike personalizacije usmjerene na korisnika koriste se za povezivanje elemenata igre s različitim korisničkim profilima. Neke se fokusiraju na specifične karakteristike korisnika poput motivacije, osobina, tipova igrača i vrsta interakcije, dok druge kombiniraju više karakteristika kako bi odredile obrazovne aktivnosti i elemente igre. Postoje razne taksonomije koje se koriste za identifikaciju tipa igrača putem upitnika, a neke od najčešćih su *Bartle*, *BrainHex* i *HEXAD*. Rješavanjem upitnika korisnik dobiva rezultate u kojem postotku pripada kojem tipu igrača. Odluka o konačnom tipu igrača obično se temelji na prevladavajućem postotku ili kombinaciji istih. Određivanje tipa igrača omogućuje prilagodbu elemenata igre prema dobivenom tipu. [10]

Osnovni koncept za adaptivnu igrifikaciju je model tipa igrača, koji klasificira koje vrste elementa igre maksimalno povećavaju motivaciju korisnika. Jedan od takvih modela je *HEXAD*. *HEXAD* definira šest tipova igrača:

1. Disruptor: motivira ga mogućnost mijenjanja sustava.
2. Free spirit: motivira ga mogućnost slobodnog istraživanja sustava.
3. Achiever: motivira ga mogućnost osvajanja izazova i otključavanja skrivenog sadržaja.
4. Player: motivira ga sama igra.
5. Philanthropist: motivira ga dijeljenje dobara i pomaganje drugim igračima.
6. Socializer: motiviraju ga socijalne veze. [10]

Također, u modele za određivanje tipa igrača u razmatranje se može uzeti tip igrača koji se ne vole igrati i na njih bi igrifikacija djelovala kontraproduktivno. [10]

Analizom korelacije *HEXAD* tipova igrača i raznih elemenata dizajna igre [10] došli su do skupa od 14 elemenata igre koji pokrivaju cijeli spektar tipova igrača. Taj skup od 14 elemenata je odabran tako da su za svaki tip igrača odabrana barem dva elementa igre i to tako da je jedan element povezan za samu prirodu tipa igrača i specifičan je samo za njega, dok je drugi element biran tako da se smatra komplementarnim ili dodatnim za drugi tip igrača, ako postoji. [10]

Lista od 14 izdvojenih elemenata igre:

1. Razvojni alat: omogućuje korisniku kreiranje određenih mehanika igrifikacije poput znački, izazova i skrivenog sadržaja koji je potrebno otključati.
2. Izazov: igrač mora savladati izazov, kao što je dosezanje određenog nivoa i rješavanje problema u određenom vremenu.
3. Uskršnje jaje: mehanizam koji se sastoji od slike koja, kada se pritisne pet puta zaredom, omogućuje pristup mini-igri itd.
4. Otključavanje: kada igrač savlada određeni izazov, otključava se skriveni sadržaj, koji može biti poruka, mini-igra itd.
5. Značka: dodjeljuje se igraču kada uspije dovršiti težak zadatak.
6. Razina: pokazuje napredak korisnika u dovršavanju zadataka, podijeljenog na razine.
7. Bod: igrač dobiva bodove, iskustvo, virtualni novac itd.
8. Ljestvica: prikazuje poredak rezultata.
9. Otvaranje darova: igrač otvara darove koje je dobio.
10. Lutrija: igra na sreću (rulet) koja omogućuje igračima povećanje svojih rezultata.
11. Društvena mreža: mala društvena mreža koja omogućuje igračima da kreiraju profil, dodaju prijatelje i pregledavaju njihove profile.
12. Društveni status: zbirka poredaka igrača temeljenih na njihovim rezultatima, posebno onima vezanim za društvene interakcije, poput broja pratitelja, posjetitelja itd.
13. Dijeljenje znanja: igrač šalje poruke pomoći svima u grupi.
14. Dar: igrač šalje darove drugim korisnicima. [10]

Analiza je rezultirala tablicom *Tablica 1* u kojoj je prikazana distribucija elemenata igre prema njihovom primarnom tipu igrača i dodatnim tipovima igrača. [10]

Tablica 1 Distribucija elemenata igre prema tipu igrača [10]

Tip igrača	Element igre	Dodatni tipovi igrača
Disruptor	Razvojni alat Izazovi	Free Spirit Player, Achiever, Free Spirit
Free Spirit	Otključavanje sadržaja Uskršnje jaje	- Player
Achiever	Značka Razina napretka	Player Player
Player	Lutrija Ljestvica Otvaranje nagrada Bodovi	- - - -
Socializer	Društvena mreža Društveni status	Free Spirit
Philanthropist	Dijeljenje znanja Darivanje	- -

Neki od navedenih elemenata igre su implementirani u samu mehaniku sustava kao što su razine, bodovi, ljestvica najboljih igrača i lutrija na kojoj se mogu osvojiti nagrade u obliku predmeta koji igračima mogu udvostručiti bodove ili im pomoći u rješavanju zadataka. Ostale elemente igre pokušat ćemo ostvariti temom igre, odnosno kreiranjem pozadinske priče koja će igrače voditi kroz edukativne izazove.

3.2. Dizajniranje priče i *Narrative atoms*

U igrifikaciji često nastojimo utjecati na ponašanje ili ga promijeniti, a poznavanje priče svake osobe može nam pomoći da shvatimo kako ih najbolje uključiti i motivirati. Kreiranjem priče za svakog korisnika ili nešto kraćeg narativa, može biti posebno korisno u

fazi dizajniranja sustava. Korisnike je potrebno provesti kroz priču, po mogućnosti onu koja se mijenja na temelju njihovih odluka i načina na koji žele proći kroz nju. Na ovaj način priča stvara osjećaj svrhe koji će puno više utjecati na korisnika nego kada bi mu se samo dodijelili bodovi za obavljeni zadatak. [41]

Za daljinu analizu stvaranja narativa i priča, potrebno je definirati pojam *Narrative atoms*. *Narrative atoms,* odnosno atomi narativa su male jedinice narativa priče koje mogu, unutar konteksta cjelokupnog narativa, postojati same za sebe. [41] U standardnim linearnim pričama, svaki atom bi bio smješten po redu, jedan iza drugoga pa time nije nužno da mogu biti samostalni i jasni sami za sebe. Međutim, u mnogim igrama narativ se mijenja na nelinearan način. Da bi to funkcioniralo svaki narativni atom mora biti sposoban stajati samostalno bez potrebe da ga drugi atom podržava. [41]

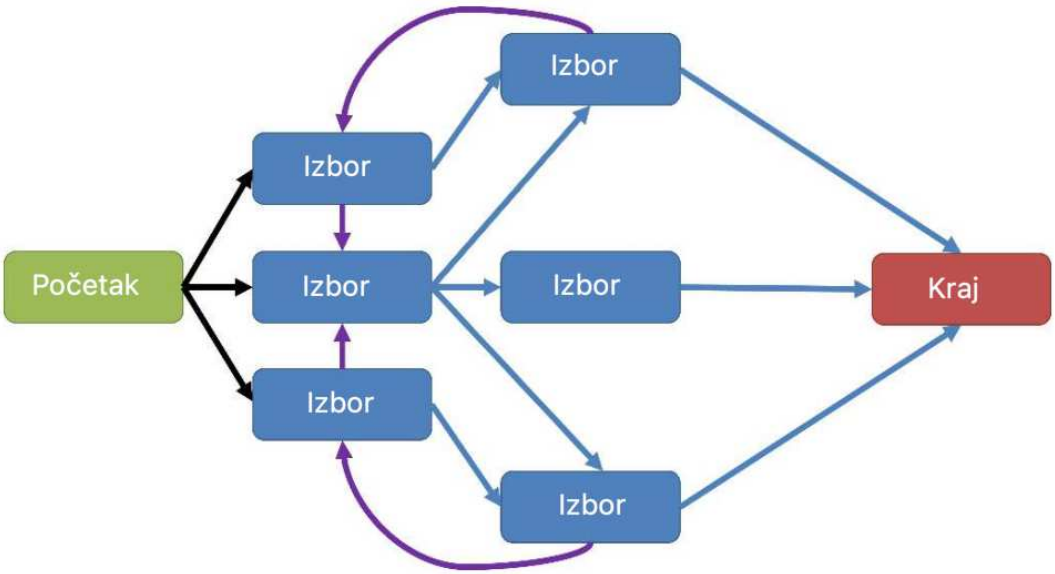
Osnovna forma priče sastoji se od tri dijela, početak, sredinu i kraj. Također, svaki atom narativa bi trebao imati svoj početak, sredinu i kraj. Tako mogu samostalno stajati ako postoji potreba za to, kao na primjer u pričama u kojima postoje grananja narativa. Za početak, nelinearni narativ mora imati početnu točku koja može biti jedinstvena ili se može razlikovati za svakog lika u priči. Nakon toga potrebno je kreirati nekoliko dijelova sredine narativa, neke će igrači vidjeti, a neke možda neće kada budu prvi put prolazili kroz igru. Naposljetku, može postojati više mjesta na kojima će igra završiti. Budući da će igrač moći navigirati kroz priču na više načina, potrebno je odrediti kako će svaki izbor koji igrač napravi utjecati na ishod njihove priče. [41]

Kod kreiranja igrificiranog sustava ili sustava temeljenog na igri, treba pokušati stvoriti izbore koji imaju smisao. Izbori bi trebali mijenjati ishode iskustva, ali čak i ako samo djeluju kao da imaju smisla, to može biti dovoljno. U sustavima koji se temelje na igri potrebno je igračima dozvoliti da odaberu svoj put kojim će igrati igru. [41]

Ljudi vole imati osjećaj da njihov odabir ima značenje, a također vole misliti da imaju stvarno pravo izbora. Autonomija je jedan od ključnih motivatora, pogotovo kod tipova igrača poput *Free Spirit* tipa, ali to nije jedini tip igrača koji je motiviran autonomijom. Kad igrač ima osjećaj da nema slobodu kretanja i biranja svoje sudbine, osjeća se ograničeno i ne povezuje se s iskustvom koje može steći pričom. Iz tog razloga potrebno je omogućiti igračima da rješavaju probleme na više načina. U narativima treba omogućiti odabir kako će odgovarati na pitanja ili kojim će putem sljedeće ići u narativu. [41]

Kombiniranjem narativnih atoma i smislenih izbora, započeli smo istraživati arhitekturu narativnih izbora, gdje svaki izbor radi stvarnu razliku ili se barem čini tako. U odnosu na tradicionalne priče, u kojima se prati linearan slijed događaja, kod igara igrač ima puno više mogućnosti. Igre pak mogu odvesti u drugu krajnost gdje igrač ima potpunu slobodu jer ne postoji stvarni kraj igre, kao što je to na primjer u igri *Minecraft*. U ovom primjeru izbori i dalje postoje, ali oni nisu vođeni pričom te je igraču dovoljno dodijeliti alate kojima će stvarati svoje izbore. [41]

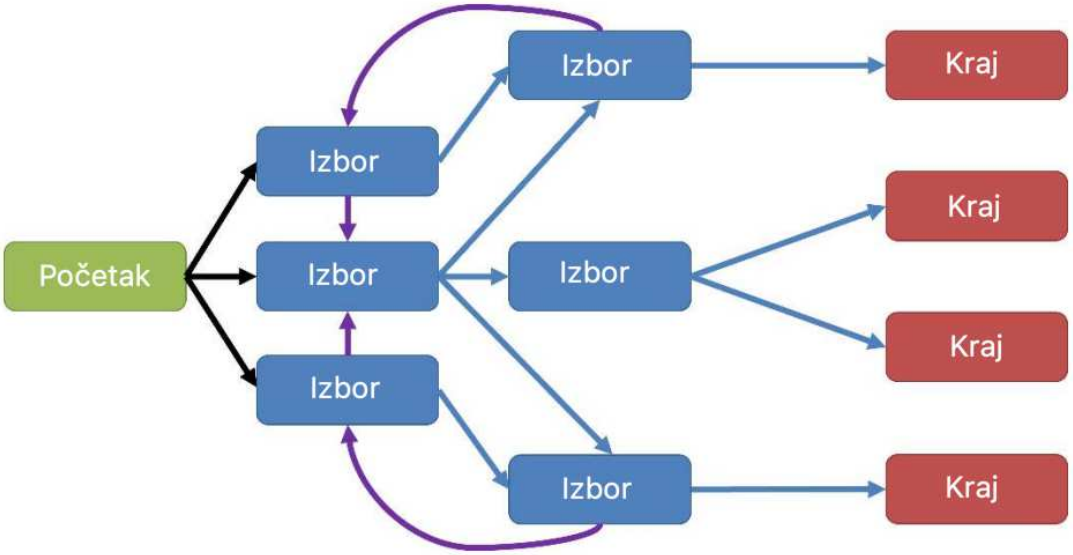
Jedna opcija arhitekture narativa je dati igračima „lažne“ opcije izbora. Igrač dolazi do grananja putova i može birati želi li ići lijevo ili desno, ali zapravo oba puta na kraju vode do istog završetka. Putevi se mogu razlikovati po događajima koje će igrači iskusiti, ali završni cilj je isti. Ova arhitektura je validan odabir ako se može napraviti da izbori djeluju značajno i da imaju važnost za igru. [41]



Slika 38 Prikaz kompleksnog narativnog atoma s lažnim izborom [41]

Alternativa arhitekturi lažnog odabira je arhitektura stvarnog odabira gdje igračevi izbori utječu na to kako će se igra igrati. Jednostavan primjer ove arhitekture bi bio kada igrač dođe

do grananja puteva, koji god put da odabere, odvest će ga do potpuno različitog ishoda. Izbor u ovom slučaju ima utjecaj na ostatak igre. [41]



Slika 39 Prikaz kompleksnog narativnog atoma sa stvarnim izborom [41]

Uz arhitekturu narativa, važno je kako će se to sve složiti u pravu priču. Ovdje se lako primjenjuje model pričanja priča s pet dijelova (poziv, izazov, transformacija, preokret, rješenje) koji se primjenjuje u većini kratkih pripovijetki poput sapunica. Ovaj model drži priče jednostavnima gdje svaka epizoda može biti kratka samostojeća priča, dok još uvijek postoji napredak likova i napredak zapleta koji može voditi u sljedeću epizodu. Tako gledatelji koji ranije nisu gledali sapunicu mogu lako pratiti priču, dok oni koji je gledaju godinama mogu uživati na dubljoj razini. [41]

3.3. Kreiranje nelinearnih priča

Kako bismo testirali razvijeni sustav potrebno je kroz nastavničko sučelje kreirati nekoliko igara. Kod stvaranja razine, priče se mogu razlikovati od težine do težine, ali ipak bi bilo poželjno kada bi listovi priče i listovi grananja bili jednaki na svim težinama, a da samo listovi u kojima se provjerava znanje budu prilagođeni težini na kojoj se razina igra.

Sustav je implementiran tako da su određeni elementi igre uključeni u svaku razinu i svaku igru. Tako se bodovi skupljaju za svaku razinu te se nakon riješene igre nadodaju ukupnom igračevom rezultatu, ako igrač skupi dovoljno bodova, moći će vidjeti svoje ime na ljestvici najboljih igrača. Unutar razina, igrači mogu naići na elemente koji mjere njihov napredak na

trenutnom pitanju, kada to ima smisla, npr. kod igre memory ili kod igre u kojoj se spajaju parovi. Nadalje, unutar igre mogu se osvajati predmeti koji pomažu igračima u daljnjem odgovaranju na pitanja ili predmeti koji će im uvećati sakupljene bodove, a takvi predmeti se osvajaju na kolu sreće.

Navedeni elementi bi trebali pobuditi motivaciju za rješavanjem zadataka i prelaska igre kod tipova igrača *Player, Achiever* i *Free Spirit*. Implementacija sustava ograničava elemente koji bi motivirali igrače tipa *Disruptor, Socializer* i *Philanthropist* jer njih motivira mogućnost utjecaja na sustav i druge igrače, a to je u ovom slučaju moguće samo izbijanjem na ljestvicu najboljih igrača. Kako bi utjecali na motivaciju što većeg broja igrača, potrebno je kroz dobro osmišljenu priču pokušati motivirati i one igrače kojima implementirani elementi igre možda ne bi bili zanimljivi sami po sebi.

Priče u igrama ćemo dizajnirati po principu narativnih atoma s arhitekturom lažnog izbora. Igrači će za vrijeme igranja igre moći donositi odluke koje će se odnositi na to kojim putem će ići, ali na kraju, svi putevi će završavati na otprilike jednake načine. Arhitektura lažnog izbora je prikladna u ovoj situaciji jer će samo oni znatiželjni igrači ponovno igrati istu igru ili oni koji žele sakupiti što više bodova prolaskom iste igre. Nadalje, odabrana arhitektura će biti odgovarajuća i u obrazovnom pogledu jer bi očekivani ishodi učenja trebali biti jednaki za svakog igrača, ma kojim god oni putem prolazili kroz igru.

Upravo zbog jednakih očekivanih ishoda učenja, priče u igri treba promišljeno razgranati. Potrebno je obratiti pažnju da omjer listova priče u kojima se provjerava znanje bude otprilike jednak na svim mogućim putevima priče. Iznimka mogu biti opcionalni putevi koji služe kao sporedna misija u igri gdje se onda može dodati više zadataka koji će se rješavati i prigodno nagraditi.

Također, dizajn sustava potiče to da narativi u razinama budu samostalne cjeline i da se igrači mogu unutar igre prebacivati iz jedne priče u drugu. Tako igračima dajemo dodatan izbor kojim putem žele nastaviti. Neće imati osjećaj da su nešto propustili jer nisu pratili jednu priču od početka i lakše će prilagoditi igru svojem tipu igrača i onome što ih zanima.

Analizirajmo sada jednu takvu nelinearnu priču u kojoj ćemo imati „za svakoga po nešto“. Kao primjer uzet ćemo jednu razinu tako kreirane igre. Popratit ćemo kako se ta igra grana i zašto.

Na početku igre, igraču se nudi opcija s kojim likom želi igrati odabranu razinu u igri. Likovi su odabrani na način kako bi pokušali pobuditi motivaciju kod igrača s različitim tipovima

osobnosti. Odabirom jednog od likova, igrači dobivaju kratki opis lika i uvid u to što bi se moglo raditi na toj razini. Tako bi bilo za očekivati da će igrači s tipom osobnosti *philanthropist* ili *socializer* odabrati lika kod kojega se spominje prijateljstvo, pomaganje drugima i zajedničke avanture. Dok će lika kod kojeg se spominju izazovi, avanture u nepoznato i donošenje odluka odabrati igrači s tipom osobnosti *achiever*, *player* i *free spirit*. Za igrače s tipom osobnosti *disruptor* preostaje opcija igranje igre gdje mogu riješiti samo zadatke ako nisu zainteresirani za sudjelovanje u igri.

Nakon odabira lika s kojim žele igrati igru, sljedeći izbor koji će igrači imati je kako će se priča u razini odvijati. Svaka priča ima nekoliko mogućih puteva. Ti putevi bi trebali biti suprotni ako se to uklapa u priču ili barem nuditi dovoljno različite opcije. Tako na primjer kada jedan put vodi priču u smjeru traženja pomoći od likova u igri, drugi put bi trebao nuditi igraču da njegov lik sam, vlastitom snalažljivošću i sposobnošću pređe situaciju u kojoj su se zatekli. Kada su opcije nastavka puta suprotne one će biti zanimljive igračima sa suprotnim tipovima osobnosti.

Na *Slika 40 Odabir lika s kojim će se igrati razina* se vide ponuđene opcije likova s kojima se razina može igrati. Prikazana razina se može igrati s „Dupin Dino“ ili „Ninja ratnik Aiko“, treća opcija je rješavanje zadataka bez nelinearne priče.

„Dupin Dino“ je opisan kao lik koji istražuje morske dubine s pomoću prijatelja koje će sresti putem, treba pronaći svoje jato dupina.

„Dupin Dino je neustrašivi istraživač dubina. Postani dupin Dino i pridruži se morskim prijateljima i prati ih u zabavnim vodenim avanturama. No, zbog svojih mnogobrojnih pustolovina izgubio si svoje jato. Neka tvoja sljedeća avantura bude pronalazak izgubljenog jata, ali ne zaboravi zabaviti se!“

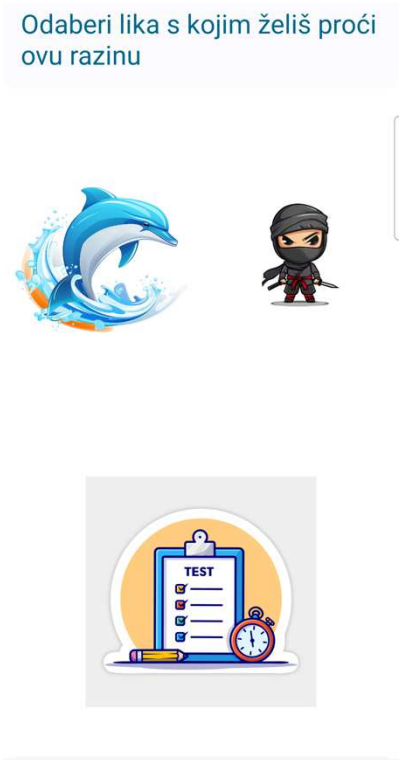
Ovakvim opisom, priča o „Dupinu Dini“ bi trebala zainteresirati igrače s tipom osobnosti *socializer* i pobuditi njihovu društvenost. Također, zbog priče o avanturama i istraživanju morskih dubina, ova priča bi mogla zainteresirati igrače čiji tip osobnosti je *free spirit*.

„Ninja ratnik Aiko“ je opisan kao mladi sanjar koji teži tome da postane najveći ninja ratnik u svome selu, a kako bi to postao morat će proći mnoge izazove, ali će putem i stvarati nova prijateljstva.

„Mladi ninja ratnik Aiko sanja o tome da postane najveći ninja ratnik u svom selu. Pridruži se Aiko na njegovom putu prema školi ninja ratnika, gdje će naučiti drevne

tehnike i steći nove prijatelje. No, put nije lak - pun je izazova i opasnosti. Tvoja odluka će oblikovati Aikovu sudbinu.“

Ovakvim opisom, priča o „Ninja ratniku Aiko“ bi trebala zainteresirati igrače s tipom osobnosti *achiever* i *player* zbog opisanih izazova, ali čak i igrače s tipom osobnosti *disruptor* jer ih možda zainteresira mogućnost odlučivanja nad njegovom sudbinom.



Slika 40 Odabir lika s kojim će se igrati razina

Pogledajmo sada kako se dalje grana priča za „Dupina Dinu“. Nakon uvodne priče i kviza („ABCD“ pitalice), priča dolazi do grananja. Na grananju igrač ima dvije opcije, potražiti pomoć od druge morske životinje ili upustiti se u opasnu avanturu na kojoj bi lik sam riješio problem koji se pred njime nalazi.

„Došao si do špilje, ali je ona jako tamna. Preko nje možeš preplivati, ali pazi! Ona se nalazi u području opasnih morskih pasa gusara.“

Kao prva opcija nudi se da protagonist potraži pomoć od prijateljske morske životinje koja se nalazi u blizini.

„Na ulazu u špilju nalazi se riba Rea koja izgleda prijateljski i s pomoću svoje svjetiljke te može provesti kroz tamnu špilju.“

Kao druga opcija nudi se odlazak u „opasnu“ avanturu.

„Zaobiđi špilju i iskušaj sreću s morskim psima“

Slika 41 Grananje priče u razini

Ovdje se razina nastavlja s dvije nove linije priče, ali u obje linije dolazi do preokreta u priči. U prvoj liniji (traženje pomoći od druge prijateljske morske životinje), protagonist upada u zasjedu koju mu priređuje životinja koja ga vodi kroz špilju. U drugoj liniji („opasna“ avantura), ono što je trebala biti „opasna“ avantura razrješuje se bez sukoba i opasnosti.

„Rea te vodi tunelima mračne špilje i već se nazire svijetlo na drugom kraju špilje, ali u jednom trenutku svjetlo se gasi i ponovno je sve mračno.“

„Plivaš preko špilje i pratiš njene zavojite oblike. U jednom trenutku iznad tebe se pojavljuju 3 velika morska psa. Morski psi su veliki i izgledaju opasno, ali ti si neustrašiv i ne bojiš ih se. Morski psi su vidjeli tvoju hrabrost jer nisi bježao i odlučili su te pustiti dalje ako uspiješ riješiti njihovu zagonetku.“

Ovdje igrač ponovno ima dvije opcije kako nastaviti priču. Kao prva opcija nudi se „borba“ s likom koji ga napada, a druga opcija je izbjegavanje sukoba.

„Rea te pokušava napasti iznova i iznova, ali ostao si nepokolebljiv u svom izazovu, i pouzdao se u svoju briljantnu primjenu matematike.“

„Plivaj što brže prema izlazu.“

Koji god put igrač odabrao, mora riješiti matematički zadatak kako bi razriješio situaciju u kojoj se zatekao. Nakon uspješno riješenog zadatak, ponovno se sve priče ujedinjuju u jednu zajedničku točku. Ovako je stvorena zajednička točka za nastavak priče u drugim razinama.

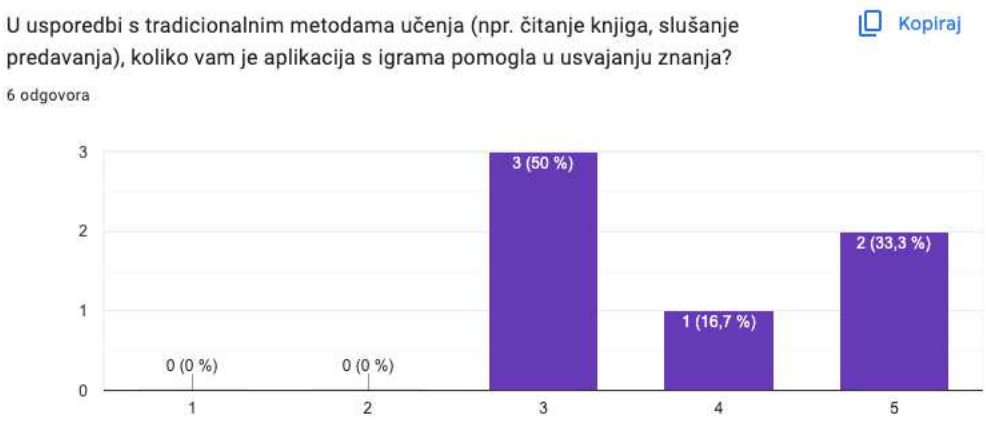
Slika 42 Završetak razine

4. Diskusija i analiza

Istraživanje je provedeno na šestero učenika u dobi od 7 godina. U sklopu istraživanja, učenici su rješavali kvizove u tri kreirane igre te su nakon odigranih igara odgovarali na anketu. Anketa se sastojala od deset pitanja kroz koja se učenike pitalo kako je na njihovu motivaciju utjecao personalizirani i adaptivni pristup igrifikaciji te da usporede koliko im je ovakav način učenja koristan u odnosu na tradicionalne metode učenja poput čitanja knjige, slušanja predavanja i slično. Zbog broja ispitanih sudionika rezultati možda nisu reprezentativni, ali i iz njih se naziru određeni zaključci.

Slika 43 Prikaz utjecaja personaliziranih igara na motivaciju

Na *Slika 43 Prikaz utjecaja personaliziranih igara na motivaciju* prikazani su odgovori na pitanje o tome kako je personalizirani pristup igrifikaciji utjecao na njihovu motivaciju. Utjecaj su mogli ocijeniti ocjenama od 1 do 5 s time da je ocjena 1 značila „uopće nije motivirajući“, a ocjena 5 je značila „vrlo motivirajući“. Iz rezultata je vidljivo kako je većini personalizacija pozitivno djelovala na motivaciju gdje je 66,7 % ispitanika dalo ocjenu 4 ili 5 dok je kod ostatka manje utjecalo na motivaciju. Nitko nije ocijenio da ovakav pristup uopće nije motivirajući.

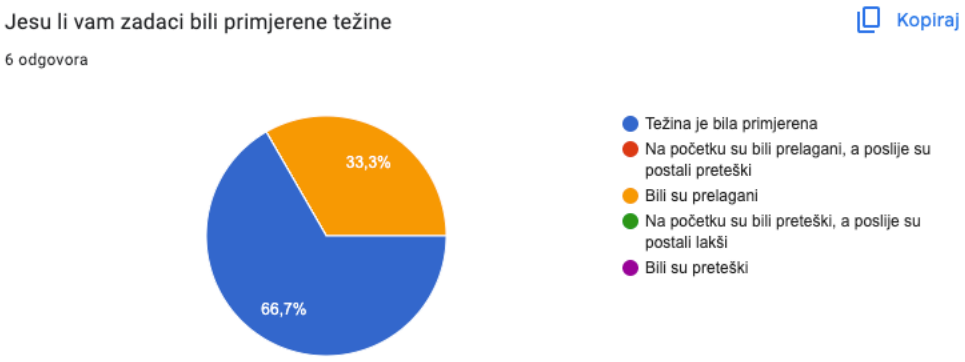


Slika 44 Prikaz usporedbe igrificirane metode učenja s tradicionalnim metodama

Na *Slika 44 Prikaz usporedbe igrificirane metode učenja s tradicionalnim metodama* prikazani su odgovori na pitanje usporedbe igrificiranog pristupa učenju s tradicionalnim metodama. Usporedbu su mogli dati u obliku ocjena od 1 do 5 s time da je ocjena 1 značila „aplikacija uopće nije korisna“, a ocjena 5 je značila „aplikacija je jako korisna“. Iz grafa je vidljivo kako je 50 % ispitanika dalo neutralnu ocjenu 3 dok je ostalih 50 % dalo ocjene 4 ili 5.

Slika 45 Prikaz usporedbe razina s pričom i razina sa zadacima

Na *Slika 45 Prikaz usporedbe razina s pričom i razina sa zadacima* prikazani su rezultati odgovora na pitanje u kojem je bilo potrebno odgovoriti sviđaju li im se više razine s pričom ili razine koje su imale samo zadatke odabirom jedne do dviju ponuđenih opcija. Opciju „Razine s pričom“ odabralo je 66,7 % ispitanih dok je opciju „Razine sa zadacima“ odabralo 33,3 % ispitanih.



Slika 46 Prikaz ocjene težine zadataka

Na *Slika 46 Prikaz ocjene težine zadataka* prikazani su odgovori na pitanje u kojem su trebali ocijeniti težinu zadataka u igri. Ponuđene opcije su bile: „Težina je bila primjerena“, „Na početku su bili prelagani, a poslije su postali preteški“, „Bili su prelagani“, „Na početku su bili preteški, a poslije su postali lakši“ i „Bili su preteški“. Opciju „Težina je bila primjerena“ je odabralo 66,7 % ispitanih dok je opciju „Bili su prelagani“ odabralo 33,3 % ispitanih. Ostale opcije nitko nije odabrao.

Ostala pitanja odnosila su se na poboljšanje korisničkog iskustva igranja aplikacije. Ispitano je koliko je lagano bilo koristiti aplikaciju, koja igra je bila najzanimljivija, što bi dodali, što im se najviše svidjelo, što im se nije svidjelo i na kraju da ocijene cjelokupni dojam na aplikaciju.

Iz prikupljenih podataka o utjecaju personaliziranog pristupa prolasku kroz igru i podataka o tome kako se većini ispitanika više sviđaju zadatci s pričom možemo zaključiti kako personalizirani pristup pozitivno utječe na motivaciju učenika.

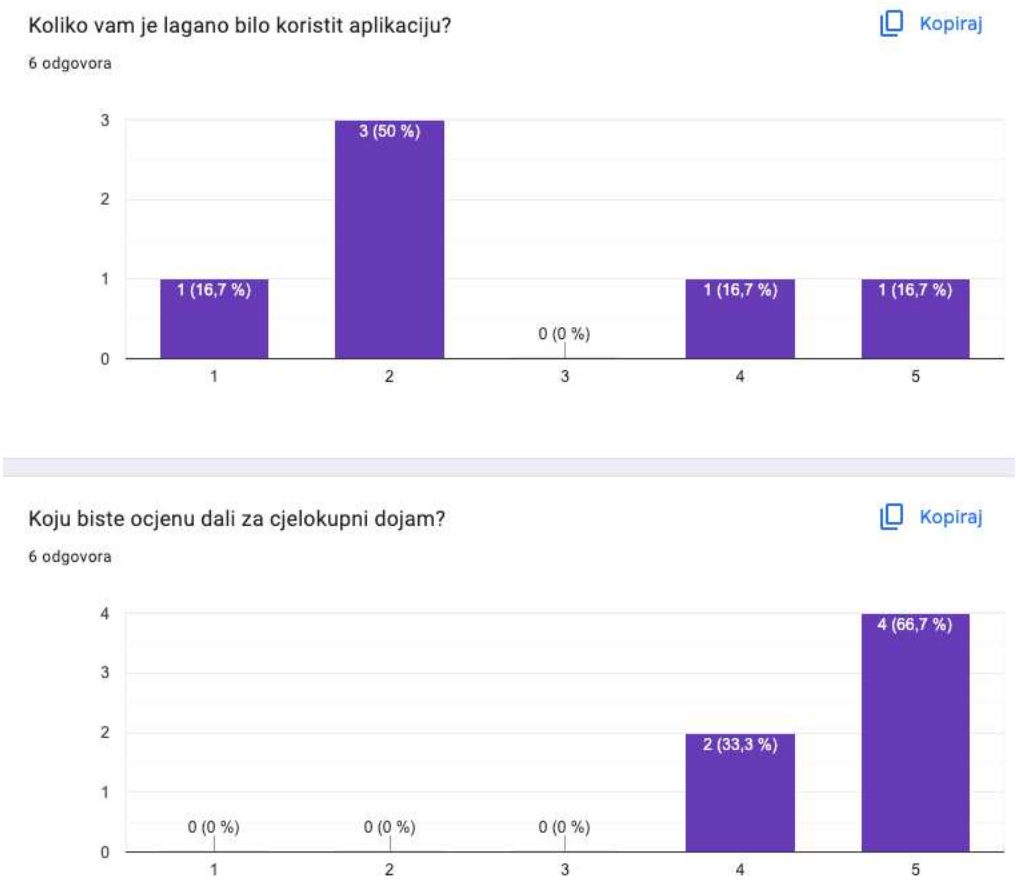
Prilikom dizajna sustava predviđeno je kako će se sudionicima s drugačijim tipom osobnosti ipak više svidjeti samo rješavanje zadataka. Zbog igrača s tim tipom osobnosti je omogućeno preskakanje priče i rješavanje samo zadataka. Analizom pojedinačnih odgovora na anketu uočeno je kako su ispitanici koji su rekli da im se više sviđaju razine u kojima se rješavaju samo zadaci, dali nižu ocjenu na pitanje koliko je personalizirani prolazak kroz priču utjecao na njihovu motivaciju u odnosu na one koji su rekli da im se više sviđaju razine s pričom. Iz ovoga možemo zaključiti kako je preskakanje priče uspješan oblik personalizacije sustava za te tipove igrača.

Analizom odgovora na pitanje kojemu je trebalo usporediti korisnost aplikacije u odnosu na tradicionalne metode učenja, može se vidjeti kako je polovina sudionika ipak suzdržana oko

toga je li ovakav sustav koristan. Druga polovina smatra kako je ovakav sustav koristan za svladavanje nastavnog sadržaja. Kako bi se poboljšao dojam o tome koliko je ovakav sustav koristan za svladavanje nastavnog sadržaja možda bi bilo korisno kada bi postojao dio sustava u kojemu bi učenici mogli na primjer pročitati definicije pojmova kao što to trenutno mogu raditi u svojim udžbenicima, a personalizirane igre s adaptivnim pitanjima bi onda mogle poslužiti za uvježbavanje gradiva ili ponavljanje prije ispita kao svojevrsna radna bilježnica.

Na pitanje „Jesu li vam zadaci bili primjerene težine“ većina ljudi je odgovorila kako su zadaci bili primjerene težine, ali čak trećini ispitanika, zadaci su bili prelagani. Iz ovoga možemo zaključiti kako i ovakav jednostavan algoritam za prilagodbu težine zadataka može biti koristan, ali možda bi bio korisniji kada bi se korisnicima na početku igre davali zadatci iz razine „Srednje“. Tako bi smanjili broj koraka do odgovarajuće težine zadataka jer oni kojima je primjerena težina „Teško“ bi morali riješiti samo jednu razinu s rezultatom iznad 80 % kako bi došli do zadataka primjerene težine. Isto vrijedi i za one kojima je primjerena težina zadataka „Lagano“, rješavanjem jedne razine ispod 50 % došli bi do primjerene težine.

Čest odgovor na pitanje „Što vam se nije svidjelo“ su primjedbe na količinu teksta u zadacima i na to kako neki tekst zadatka možda nije razumljiv djeci ovog uzrasta. Ovo je realan problem i njegovom rješenju bi se moglo pristupiti suradnjom s profesorima koji predaju tim uzrastima. Oni bi mogli imati bolji osjećaj za to koja količina teksta će biti primjerena kojem uzrastu, a da se pritom ne izgubi smisao nelinearnih priča i kako bi se korisnici tijekom igre ipak mogli poistovjetiti s likovima u priči.



Slika 47 Prikaz zadovoljstva korisnika aplikacijom

Na *Slika 47 Prikaz zadovoljstva korisnika aplikacijom* su prikazani odgovori na pitanja o tome koliko je lagano bilo koristiti aplikaciju i ocjena za cjelokupni dojam sustava. Na pitanje o tome koliko je bilo lagano koristiti aplikaciju sudionici su odgovarali ocjenama od 1 do 5 gdje je 1 značilo „Aplikacija je jednostavna i lako se koristi“ dok je 5 značilo „Aplikacija je zbunjujuća i jako ju je teško koristiti.“. Ispitanicima je uglavnom bilo lako koristiti se aplikacijom te ih je 66,7 % dalo ocjenu 1 ili 2. Kako bi se aplikacija dodatno pojednostavila možda bi bilo korisno na početku aplikacije dodati pokaznu igru kroz koju bi se korisnika upoznalo s elementima igre i načinom kako se igra može igrati. Također, analizom aktivnosti korisnika u aplikaciji zaključeno je kako nijedan ispitanik nije iskoristio niti jedan *hint* te bi trebalo istražiti jesu li zadatci bili jednostavni pa nije bilo potrebe za njima ili korisnici nisu bili upoznati s tom mogućnošću iako je ta opcija opisana u uputama koje su ispitanici dobili za pomoć pri instalaciji mobilne aplikacije.

Na pitanje da ocijene cjelokupni dojam ispitanici su odgovarali s ocjenama od 1 do 5 gdje je 1 značilo „vrlo loše“, a 5 „odlično“. Svi ispitanici su odgovorili s ocjenama 4 ili 5.

Zaključak

Ovaj rad istražuje utjecaj personalizirane igrifikacije na motivaciju i ishode učenja kod učenika kroz implementaciju interaktivnog sustava za učenje kroz igru. Tehnologija i igre pokazale su se moćnim alatima za poboljšanje edukativnih procesa, a integracija personaliziranih pristupa unutar igrifikacije predstavlja korak naprijed u prilagođavanju nastavnih metoda individualnim potrebama i preferencijama učenika.

U sklopu rada razvijen je sustav za kreiranje i igranje personaliziranih edukativnih igara. Igre se kreiraju i igraju putem mobilne aplikacije za uređaje s operacijskim sustavom android koja je razvijena za potrebe ovog sustava. Igre se temelje na nelinearnim pričama. Tijekom igranja igre učenici mogu birati način na koji žele proći kroz igru što stvara personalizirano iskustvo. Ovakvim pristupom utječe se na motivaciju igrača jer igrači različitih tipova osobnosti mogu prolaziti igru na različite načine. Također igrači kojima se ne sviđaju igre mogu rješavati samo zadatke, a preskočiti priču. Sustav prati napredak igrača za pojedine nastavne lekcije i na osnovu prijašnjih odgovora igraču dodjeljuje zadatke primjerene težine.

Također, u sklopu rada provedeno je i istraživanje kako bi se testirao utjecaj personaliziranih igara na motivaciju učenika i kako bi se usporedila učinkovitost ovakvog sustava u odnosu na tradicionalne metode učenja. Analizom rezultata istraživanja utvrđeno je kako mogućnost biranja vlastitog načina prolaska kroz igre ima pozitivan učinak na motivaciju učenika, ali isto tako kako mogućnost preskakanja priče i rješavanje zadataka ima bolji utjecaj kod drugih tipova osobnosti. Usporedbom ovakvog načina učenja s tradicionalnim načinima učenja utvrđeno je kako ovakav sustav može biti koristan, ali polovina ispitanika je ipak suzdržana o razini korisnosti. Također, možda bi bilo potrebno poboljšati algoritam sustava za dodjeljivanje zadataka primjerene težine jer korisnici koji su sposobni rješavati teže zadatke moraju prvo proći nekoliko razina zadataka koji će biti prelagani za njihovu razinu znanja.

Literatura

[1] Kalinauskas, M. (2014). Gamification in fostering creativity. *Socialinės Technologijos*, *4*(01), 62-75.

[2] Goad. L. Gamification Summit 2011: Big Themes from the Inaugural Event. ZDNet, 24. siječnja 2011. https://www.zdnet.com/article/gamification-summit-2011-big-themes-from-the-inaugural-event/; pristupano 27. lipnja 2024.

[3] Deterding, S., Sicart, M., Nacke, L., O'Hara, K., & Dixon, D. (2011). Gamification. using game-design elements in non-gaming contexts. In *CHI'11 extended abstracts on human factors in computing systems* (pp. 2425-2428).

[4] Dicheva, D., Dichev, C., Agre, G., & Angelova, G. (2015). Gamification in education: A systematic mapping study. *Journal of educational technology & society*, *18*(3), 75-88.

[5] Seaborn K, Fels DI (2015) Gamification in theory and action: A survey. International Journal of Human-Computer Studies 74:14–31. https://doi.org/10.1016/j.ijhcs.2014.09.006

[6] Knutas, A., Van Roy, R., Hynninen, T., Granato, M., Kasurinen, J., & Ikonen, J. (2019). A process for designing algorithm-based personalized gamification. *Multimedia Tools and Applications*, *78*, 13593-13612.

[7] Jianu, E. M., & Vasilateanu, A. (2017, October). Designing of an e-learning system using adaptivity and gamification. In *2017 IEEE international systems engineering symposium (ISSE)* (pp. 1-4). IEEE.

[8] MORA, Alberto, et al. Effect of personalized gameful design on student engagement. In: *2018 IEEE global engineering education conference (EDUCON)*. IEEE, 2018. p. 1925-1933.

[9] JABBOUR, Joe, et al. Challenges in producing tailored internet patient education materials. *International Journal of Radiation Oncology, Biology, Physics*, 2017, 97.4: 866-867.

[10] Rodríguez, I., Puig, A., & Rodríguez, À. (2022). Towards adaptive gamification: A method using dynamic player profile and a case study. *Applied Sciences*, *12*(1), 486.

[11] LAVOUÉ, Elise, et al. Adaptive gamification for learning environments. *IEEE Transactions on Learning Technologies*, 2018, 12.1: 16-28.

[12] BARTLE, Richard. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research*, 1996, 1.1: 19.

[13] Kocadere, S. A., & Çağlar, Ş. (2018). Gamification from player type perspective: A case study. *Journal of Educational Technology & Society*, *21*(3), 12-22.

[14] FERRO, Lauren S.; WALZ, Steffen P.; GREUTER, Stefan. Towards personalised, gamified systems: an investigation into game design, personality and player typologies. In: *Proceedings of the 9th Australasian conference on interactive entertainment: Matters of life and death*. 2013. p. 1-6.

[15] FULLERTON, Tracy. *Game design workshop: a play centric approach to creating innovative games*. CRC press, 2008.

[16] Bang, H. J., Li, L., & Flynn, K. (2023). Efficacy of an adaptive game-based math learning app to support personalized learning and improve early elementary school students' learning. *Early Childhood Education Journal*, *51*(4), 717-732.

[17] GeeksforGeeks. Client-Server Model. GeeksforGeeks. 19. travnja 2024. https://www.geeksforgeeks.org/client-server-model/; pristupano 10. lipnja 2024.

[18] Kotlin – Getting started with Kotlin 8. svibnja 2024. https://kotlinlang.org/docs/getting-started.html; pristupano 10. lipnja 2024.

[19] Wikipedia – Google I/O 28. svibnja 2024. https://en.wikipedia.org/wiki/Google_I/O; pristupano 10. lipnja 2024.

[20] Developer Android – Android's Kotlin-first approach 5. travnja 2024. https://developer.android.com/kotlin/first; pristupano 10. lipnja 2024.

[21] Oleschuk, L., The Main Java Competitor. Why Is Kotlin Still Less Popular than Java Despite All Its Advantages?, CodeGym, 14. ožujka 2023. https://codegym.cc/groups/posts/1071-the-main-java-competitor-why-is-kotlin-stillless-popular-than-java-despite-all-its-advantages; pristupano 10. lipnja 2024.

[22] Kotlin - Coroutines basics, 12. lipnja 2024. https://kotlinlang.org/docs/coroutines-basics.html; pristupano 13. lipnja 2024.

[23] Kotlin – Sealed classes and interfaces, 11. travnja 2024. https://kotlinlang.org/docs/sealed-classes.html; pristupano 13. lipnja 2024.

[24] Kotlin – Object expressions and declarations, 15. travnja 2024. https://kotlinlang.org/docs/object-declarations.html; pristupano 13. lipnja 2024.

[25] "AIA SG Techblog. Gaonkar S. 15. kolovoza 2021. Why singleton pattern is considered as anti-design pattern. https://medium.com/aia-sg-techblog/why-singleton-pattern-is-considered-as-anti-design-pattern-c81dd8b7e757" pristupano 13. lipnja 2024.

[26] Kotlin – Data classes 30. siječnja 2024. https://kotlinlang.org/docs/data-classes.html; pristupano 13. lipnja 2024.

[27] Kotlin – Null safety 11. rujna 2023. https://kotlinlang.org/docs/null-safety.html; pristupano 13. lipnja 2024.

[28] Kotlin – Scope functions 23. listopada 2023. https://kotlinlang.org/docs/scope-functions.html; pristupano 13. lipnja 2024.

[29] Kotlin – Conditions and loops 26. ožujka 2024. https://kotlinlang.org/docs/control-flow.html#when-expression; pristupano 13. lipnja 2024.

[30] Kotlin – Extensions 11. rujna 2023. https://kotlinlang.org/docs/extensions.html; pristupano 13. lipnja 2024.

[31] Kotlin – Flow https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-flow/; pristupano 13. lipnja 2024.

[32] Developers – Thinking in Compose 13. lipnja 2024. https://developer.android.com/develop/ui/compose/mental-model; pristupano 14. lipnja 2024.

[33] Developers – Scaffold 13. lipnja 2024. https://developer.android.com/develop/ui/compose/components/scaffold; pristupano 15. lipnja 2024.

[34] Developers – State and Jetpack Compose 13. lipnja 2024. https://developer.android.com/develop/ui/compose/state; pristupano 16. lipnja 2024.

[35] Developers – Where to hoist state 25. ožujka 2024. https://developer.android.com/develop/ui/compose/state-hoisting; pristupano 18. lipnja 2024.

[36] Gupta, S., MVVM with Jetpack Compose, Medium, 17. prosinca 2023. https://medium.com/@shashank.gupta006/mvvm-with-jetpack-compose-03cbd8c211c1; pristupano 18. lipnja 2024.

[37] Developers – Architecting your Compose UI, 25. ožujka 2024. https://developer.android.com/develop/ui/compose/architecture; pristupano 18. lipnja 2024.

[38] Vasava, K. Navigation in Jetpack compose. Full guide Beginner to Advanced., Medium, 19. rujna 2023. https://medium.com/@KaushalVasava/navigation-in-jetpack-compose-full-guide-beginner-to-advanced-950c1133740; pristupano 19. lipnja 2024.

[39] Developers – Dependency injection with Hilt, 18. lipnja 2024. https://developer.android.com/training/dependency-injection/hilt-android; pristupano 19. lipnja 2024.

[40] Developers – Compose and other libraries, 18. lipnja 2024. https://developer.android.com/develop/ui/compose/libraries#hilt; pristupano 19. lipnja 2024.

[41] MARCZEWSKI, Andrzej. Even Ninja Monkeys like to play. *London: Blurb Inc*, 2015, 1.1: 28.

Sažetak

Ključne riječi: igrifikacija, personalizirana igrifikacija, adaptivna igrifikacija, Android aplikacija

U ovom radu istražuju se utjecaji personalizirane igrifikacije na motivaciju i učinkovitost učenja. Naglasak je na razvoju mobilne aplikacije koja omogućava personalizirano iskustvo učenja kroz igru, prilagođeno individualnim potrebama i preferencijama učenika. Kroz sustav se mogu kreirati različite vrste igara, od kvizova do kompleksnih obrazovnih igara, koje se prilagođavaju osobnosti i razini znanja korisnika. Igre se temelje na nelinearnim pričama gdje učenici imaju slobodu izbora kojim putem žele prolaziti kroz igru, što stvara personalizirano iskustvo i djeluje na motivaciju igrača s različitim tipovima osobnosti. Sustav prati igračev napredak za pojedine nastavne lekcije te na osnovu prethodnih odgovora prilagođava težinu zadataka. Cilj je utvrditi efikasnost personaliziranog pristupa učenju kroz igru u usvajanju znanja u odnosu na tradicionalne metode učenja.

Summary

Keywords: gamification, personalized gamification, adaptive gamification, Android application

In this research, it is explored the impact of personalized gamification on motivation and effectiveness of learning. The focus is on the development of a mobile application that provides a personalized learning experience through gaming according to the individual needs and preferences of each student. The system enables the creation of different game types, such as simple quizzes or complex educational games. The games adapt to the user's personality and level of knowledge. They are based on non-linear narratives in which students can choose which path they want to take, creating a personalized experience that influences the motivation of players with different personality types. The system tracks the player's progress for each lesson and adjusts task difficulty based on previous choices. The goal is to determine the efficiency of personalized gamified learning approaches in knowledge acquisition compared to traditional teaching methods.

Skraćenice

HTTP – Hypertext Transfer Protocol

API – Application Programming Interface

MVVM – Model-View-ViewModel

JSON – JavaScript Object Notation

CRUD – Create, Read, Update, Delete,

DTO – Dana Transfer Object

DAO – Data Access Object

JPA – Jakarta Persistence API

XML – Extensible Markup Language

UI – User Interface