Primjena proširene stvarnosti u razvoju računalne igre za platformu iOS

Medak, Zvonimir

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

Sveučilišni studij

PRIMJENA PROŠIRENE STVARNOSTI U RAZVOJU RAČUNALNE IGRE ZA PLATFORMU iOS

Diplomski rad

Zvonimir Medak

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 01.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Zvonimir Medak
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1144R, 13.10.2020.
OIB studenta:	46411366911
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Prof. dr. sc. Krešimir Nenadić
Naslov diplomskog rada:	Primjena proširene stvarnosti u razvoju računalne igre za platformu iOS
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba opisati izazove i pristupe u razvoju arkadnih računalnih igara s naglaskom na mogućnosti primjene proširene stvarnosti u računalnim igrama i osvrtom na postojeća slična rješenja. Također, treba analizirati i predložiti programsku arhitekturu računalne igre za iOS mobilne uređaje koja omogućuje početni zaslon, zaslon za definiranje postavki igre, prostor za igranje s mogućnostima upravljanja objektima prema pravilima igre, sustav obavijesti i rangiranja igrača, kao i ugradnju mogućnosti proširene stvarnosti u računalnu igru. Programsko rješenje treba ostvariti u odgovarajućim aktualnim programskim tehnologijama, jezicima i okvirima koji omogućuju nativni i hibridni razvoj. Ostvareno programsko rješenje
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum: 15.09.2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 18.09.2022.

Ime i prezime studenta:	Zvonimir Medak
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1144R, 13.10.2020.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena proširene stvarnosti u razvoju računalne igre za platformu iOS**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Mobilne računalne igre napredovale su u načinu igranja kojeg omogućavaju korisnicima. Jedan od značajnijih napredaka je proširena stvarnost (engl. Augmented Reality, AR) koja omogućuje korisnicima da u stvarnom svijetu postavljaju imaginarne predmete kojima mogu upravljati. Koncept proširene stvarnosti relativno je nov i još nije dosegao svoj potpuni potencijal. Jačanjem procesorske snage mobilnih uređaja omogućuje se brža obrada većih količina podataka. U ovom radu pokazat će se kako se može integrirati proširena stvarnost u arkadnim mobilnim računalnim igrama na *iOS* uređajima.

U diplomskom radu ostvarit će se programsko rješenje mobilne računalne igre koja uz pomoć kamere na uređaju dohvaća informacije o predmetima iz stvarnog prostora i obrađuje ih. Uz praćenje stvarnog prostora, prate se i imaginarni predmeti postavljeni kroz igru nad kojima je omogućeno korisničko upravljanje. Praćenjem objekata omogućuje se praćenje rezultata kojeg korisnik ostvari i praćenje završetka igre. Korisniku će se omogućiti promjena veličine prostora za igru i brzine igre u proširenoj stvarnosti. Metodologija izrade mobilne računalne igre oslanjat će se na reaktivno i asinkrono programiranje uz korištenje arhitekture VIPER. Arhitektura VIPER omogućuje intuitivni razvoj reaktivnog i asinkronog programiranja u aplikaciji zbog svojih strogo definiranih pravila komunikacije. Reaktivno programiranje omogućuje lakše praćenje događaja koji se pojave tijekom obrade informacija u proširenoj stvarnosti i pravilnu obradu dobivenih događaja u trenutcima kada događaj nastane. Asinkronim programiranjem postižu se bolje performanse mobilne računalne igre zbog iskorištenja više niti procesora za obradu podataka. Asinkrono programiranje je bitno u mobilnim računalnim igrama koje se oslanjaju na proširenu stvarnost, jer zahtijevaju veliku procesorsku snagu i obrađuju velike količine podataka.

U drugom poglavlju opisane su osnove proširene stvarnosti, pristupi u razvoju mobilnih računalnih igara, uklapanje proširene stvarnosti u mobilne računalne igre i trenutno stanje uz opis idejnog rješenja mobilne predložene računalne igre. U trećem poglavlju opisana su pravila, parametri, završetak, funkcionalni i nefunkcionalni zahtjevi te komponente mobilne računalne igre rada. U četvrtom poglavlju prikazan je način razvoja i programski pristup rješenja mobilne računalne igre prema potrebnim komponentama opisanima u trećem poglavlju uz opis arhitekture VIPER. Peto poglavlje prikazuje način korištenja i ispitivanje rada aplikacije uz analizu korisničkog iskustva s mogućim poboljšanjima s obzirom na rezultate analize.

2. STANJE U PODRUČJU PROŠIRENE STVARNOSTI I RAČUNALNIH IGARA

U ovom poglavlju detaljnije će se pregledati osnove, tipovi i primjena proširene stvarnosti, a zatim osnove i izazovi stvaranja mobilnih računalnih igara te trenutno stanje proširene stvarnosti u mobilnim računalnim igrama. Također, opisat će se idejno rješenje i očekivanja od mobilne računalne igre.

2.1. Proširena stvarnost

2.1.1. Osnovna načela proširene stvarnosti

Proširena stvarnost proširuje stvarni prostor dodavanjem imaginarnih predmeta. Prema [1], proširena stvarnost postavlja imaginarne predmete u stvarni prostor uzimajući u obzir položaj predmeta iz stvarnog svijeta da bi mogao pravilno prikazati položaje predmeta korisniku. Tijekom korištenja proširene stvarnosti korisnik vidi stvarni svijet, ali s dodacima koji ga upotpunjuju. Proširena stvarnost ima mogućnost poboljšanja uključenosti korisnika u okolinu i može poboljšati korisnikovu percepciju stvarnog svijeta. Prema [2], proširena stvarnost omogućuje ljudima prividnu prisutnost u stvarnom prostoru. S druge strane, prividna stvarnost pruža komunikaciju u prividnom prostoru s prividnim ljudima.

Prema [3], proširena stvarnost je poboljšana inačica stvarnog svijeta koja se postiže raznim digitalnim vizualnim elementima, zvukovima i ostalim osjetilnim podražajima koje možemo proizvesti tehnologijom. U posljednje vrijeme vidi se nagla zainteresiranost tvrtki za proširenu stvarnost, jer može prikupiti puno informacija o njihovim korisnicima. Navodi kako su neki od ranih povodnika proširene stvarnosti uspjeli napraviti svoje kataloge u proširenoj stvarnosti. Na takav način kupcima su omogućili lakši odabir stvari koje žele kupiti tako što im dozvole da ih odmah isprobaju u svom domu. Također, spominje se kako proširena stvarnost možda doživi uspjeh u drugim nosivim uređajima kao što su naočale, leće i drugi. Takvim pothvatom korisnici mogu imati cjelovit doživljaj proširene stvarnosti.

Proširena stvarnost najčešće se koristi u mobilnim uređajima dok se predviđala budućnost proširene stvarnosti u nosivim uređajima. Nosivi uređaji bili bi elegantnije rješenje za svakodnevne poslove gdje nemamo pristup mobilnom uređaju. Tehnologija je uvelike napredovala

tako da se proširena stvarnost danas čini normalnom pojavom, ali zahtijeva jaku procesorsku snagu za obradu podataka. Sve što kamera vidi su podaci koje uređaj treba obraditi na pravilan način.

2.1.2. Tipovi proširene stvarnosti

Prema [4], postoje dva tipa raspodjele proširene stvarnosti, a to su s ugrađenim predmetima i bez ugrađenih predmeta. Oba načina bit će objašnjena u daljnjem tekstu. Proširena stvarnost je napredna tehnologija koja obavlja nezamislivo mnogo stvari u pozadini da bi se pojavio samo jedan predmet u proširenoj stvarnosti.

Proširena stvarnost s ugrađenim predmetima tip je proširene stvarnosti koji pokušava na slici prepoznati predmete koji su unaprijed ugrađeni u aplikaciju ili uređaj. Koristi i točke koje postavlja u proširenu stvarnost da bi uređaj uspio identificirati poziciju i orijentaciju kamere. Kamere uređaja većinom šalju slike sivih tonova da smanje potrošnju resursa i pokušaju preko značajki objekata iz prostora pronaći objekt koji se podudara s njim, a da je već ugrađen u uređaj ili aplikaciju. U slučaju da to uspije, uređaj odmah zna koji je to objekt i uz pomoć prije spomenutih točaka za orijentaciju, uređaj prepoznaje kako je objekt okrenut. Ovakav tip proširene stvarnosti čest je kod aplikacija koje koriste strojno učenje za prepoznavanje točno određenih objekata u prostoru [4].

Proširena stvarnost bez ugrađenih predmeta tip je proširene stvarnosti koji mora prepoznavati predmete prateći uzorke, boje ili druge značajke objekata koje bi mogle pomoći da se prepozna. Primjer igre koja koristi takav tip proširene stvarnosti je *Pokémon Go* koja koristi GPS, brzinomjer i kompas na uređaju da bi orijentirala korisnika, a kad se upali proširena stvarnost onda prati okolinu i postavlja *Pokémona* unutar okoline korisnika [4].

Proširena stvarnost bez ugrađenih predmeta koristit će se i u predloženoj mobilnoj računalnoj igri u radu, jer se uređaj mora snaći u prostoru da bi pronašao pravo mjesto za postavljanje igraćeg prostora.

2.1.3. Proširena stvarnost u primjeni

Prema [5], budućnost mobilnih računalnih igara je nepredvidiva, ali smatra kako budućnost leži u prividnoj stvarnosti i 3D igrama koje bi i sama proširena stvarnost omogućila. Prema [2], jedan od novijih trendova koji se pojavljuju u mobilnim računalnim igrama na platformi *iOS* je proširena stvarnost koju naziva i novom vrstom prividne stvarnosti. Za proširenu stvarnost smatra se da će u budućnosti poboljšati i olakšati život. Jedan od ranih povodnika proširene stvarnosti bila je *IKEA*.

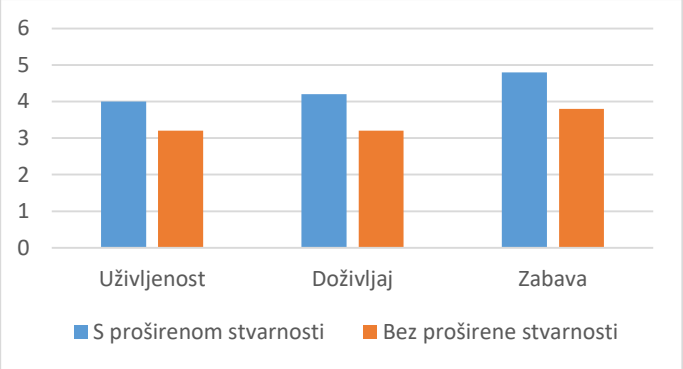
IKEA je, prema [6], uspjela napraviti aplikaciju u kojoj omogućuje korisnicima odabir bilo koje stvari iz svog kataloga te postavljanje u stvarni prostor. Tako je korisnicima omogućeno da se lakše odluče i da imaju uvid u izgled prostorije nakon što kupe određenu stvar.

Prema [7], provedeno je istraživanje da bi se vidio utjecaj proširene i prividne stvarnosti na užitak igre *Clash Tanks*. Cilj istraživanja bio je ispitati mogu li se proširena i prividna stvarnost pojaviti u istoj igri i na koji način bi to utjecalo na cjelokupno korisničko iskustvo. Spominju da većinom korisnici tijekom igranja mogu uživati samo u proširenoj stvarnosti ili samo u prividnoj stvarnosti, a da je kombinirana pojava još neispitana. Igra koristi prividnu stvarnost da bi korisnika postavila u vozačevu kabinu, a proširenu stvarnost koriste da bi prikazali kontrole za upravljanje robotom koji se nalazi pored njih u stvarnom svijetu. Prividnom stvarnošću promijenili bi cjelokupnu okolinu koja se nalazi oko robota što bi omogućilo korisnicima da se više užive u samo iskustvo igranja. Proširena stvarnost koristi se da bi promijenila izgled neprijateljskog robota i da bi se prikazali životni bodovi i štit. Također, oštećenja proizvedena oružjem i efekti koji bi se proizveli oružjem prikazani su u proširenoj stvarnosti. Smatra se da bi se prividnom stvarnošću uspjelo povećati zainteresiranost korisnika, užitak igranja i načine upravljanja, dok proširena stvarnost daje dodatne mogućnosti raznih efekata i poboljšanja izgleda stvarnih predmeta. Na slici 2.1 vidljiv je udio ispitanika koji su se prije eksperimenta susreli s proširenom stvarnošću i udio onih koji nisu [7].

Slika 2.1. Prikaz udjela ispitanika koji su koristili i onih koji nisu koristi proširenu stvarnost [7]

Ispitanicima je zadano nekoliko zadataka koji su uključivali korištenje i ne korištenje prividne i proširene stvarnosti da bi se dobili pravilni rezultati usporedbe. Ispitanici koji su imali prijašnjeg iskustva s korištenjem prividne stvarnosti u igrama, obavili su svoje zadatke brže nego ispitanici koji nisu dok su morali koristiti prividnu stvarnost. Bez obzira na iskustvo u korištenju proširene stvarnosti, rezultati uživljenosti, iskustva i zabave bili su pozitivni. Ispitanici su izjavili kako su se

mogli više uživjeti u igru kad je bila uključena proširena stvarnost te kako im je to povisilo razinu užitka igranja igre. Također, ispitanicima se povećala razina zabave što se moglo očekivati uz rezultate koji su postignuti za razinu užitka tijekom igranja igre. Grafički prikaz rezultata ispitanika vidljiv je na slici 2.2 [7].



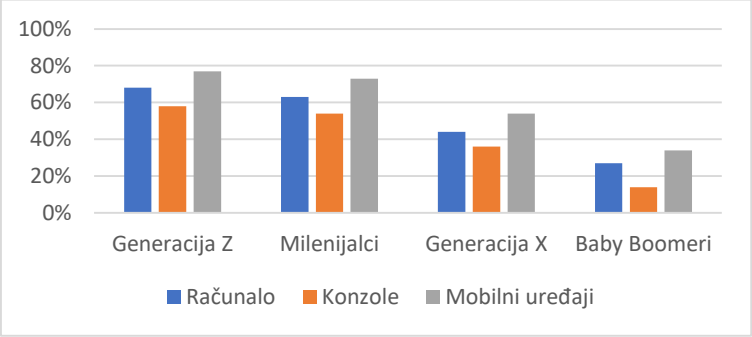
Slika 2.2. Prikaz rezultata ispitanika nakon igranja igre s proširenom i bez proširene stvarnosti [7]

Uz navedene dobre strane, postojala je i jedna loša gdje je jedan od ispitanika izjavio kako mu iskustvo tijekom korištenja prividne stvarnosti nije bilo dobro. Takvi slučajevi su učestali kod osoba koje prvi put koriste prividnu stvarnost jer može izazvati vrtoglavicu, mučninu i nelagodu. Prividna stvarnost prebaci osobu u drugačiji svijet dok se treba kretati u stvarnom što može loše utjecati na osobe koji je prvi put koriste [7].

2.2. Mobilne računalne igre

Prema [8], video igre su način izražavanja svojih ideja. Takvim načinom osobe mlađe dobi mogu, uz izražavanje svojih misli, učiti i stvarati igre. Provedenim se anketama 58% profesionalnih programera koji stvaraju video igre izjasnilo da je programiranje glavni dio njihovog posla, ali da je najveća zabava i užitak stvaranje igara u kojim će drugi uživati. U industriji razvoja računalnih igara poanta je stvaranja igre kojom će se programeri ponositi, a korisnici uživati. Stvaranje računalnih igara zahtijeva od programera da kritički razmišlja o svojim idejama koje želi pretvoriti u igru. Nakon svakog odrađenog dijela u razvoju programske podrške dolazi vrijeme testiranja i popravaka što dovodi do želje za poboljšanjem i stvaranjem najboljeg mogućeg proizvoda. Tijekom programiranja bitno je znati da je svaki „neuspjeh“ zapravo prilika da se nešto novo nauči i da je bitna upornost i želja za proizvodnjom najboljeg mogućeg proizvoda.

Mlađe generacije odrastaju u svijetu gdje ih tehnologija i informacije zasipaju sa svih strana pa je i veća šansa da će se upustiti u igranje mobilnih računalnih igara prije nego starije generacije. Prema [9], istraživanje pokazuje da svaki osmi *Gen Z* ili *Millennial* korisnik igra video igre oko sedam sati tjedno, dok se iz *Baby Boom* generacije izjasnilo 42% ispitanika da igraju video igre oko dva i pol sata tjedno. Mlađe generacije provode više vremena od starijih u svijetu video igara jer to smatraju načinom života. Kad ne igraju igre onda gledaju video sadržaj o video igrama, ali su i aktivniji u zajednicama koje su nastale pod utjecajem igara što im pruža društveno zadovoljstvo. U društvenom smislu, igre ih mogu povezivati tako da započinju razgovore s osobama kojim je zajednička tema video igra. Video igre mogu utjecati na mlađe generacije tako da se zanimaju za čitanje i razvoj video igara. Jedna od lošijih stvari je što mlađe generacije imaju veću šansu potrošiti novac na video igre jer više vremena provode igrajući igre. Najveći udio igranja video igara događa se na mobilnim uređajima, ali mlađe generacije najviše mijenjaju platforme na kojim igraju. Kod mlađih generacija izraženo je mijenjanje platformi jer se na scenu probija *cross-platform* igranje igara koji omogućuje igranje iste igre na više različitih platformi. Rezultat popularnosti mobilnih uređaja kao platforma za igranje igara je očekivan jer su osobe najizloženije mobilnim uređajima. Korisnicima je najlakše nakratko odigrati video igru na mobilnom uređaju dok je za druge platforme potrebno više opreme. Rezultat istraživanja udjela platformi za igranje po generacijama koji uzima u obzir Generaciju Z (engl. Gen Z), Milenijalci (engl. Millenials), Generacija X (engl. Gen X) i generacija *Baby Bommersa* vidljiv je na slici 2.3 [9].



Slika 2.3. Prikaz udjela platformi za igranje po generacijama [9]

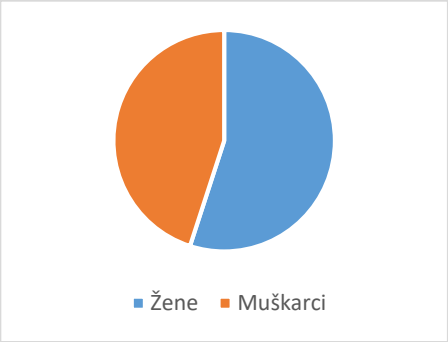
2.2.1. Stanje mobilnih računalnih igara

Mobilne računalne igre su igre koje je moguće igrati na mobilnim uređajima. Opisuje ih prenosivost i mogućnost igranja bilo kad i bilo gdje.

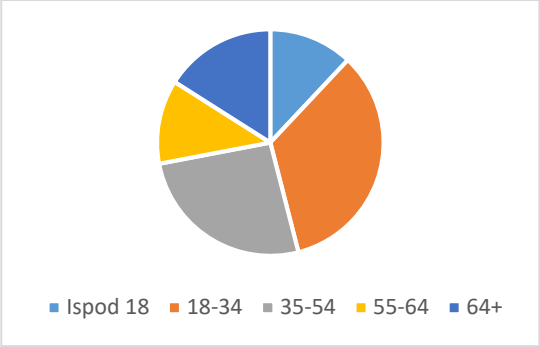
Prema [10], mobilne igre dobivaju sve veću popularnost, a u vrijeme pisanja članka dosegnula je brojku od 2.7 milijardi igrača u cijelome svijetu koji u prosjeku igraju šest i pol sati tjedno mobilne računalne igre. Ovisno o generacijama postoje različiti razlozi igranja mobilnih igara, a najčešći su opuštanje, natjecanje, društveni aspekti i odmor. Također, igrači će puno lakše potrošiti novac na mobilnu računalnu igru nego na igre na drugim platformama. Količina novca potrošena u prvoj polovici 2021. godine bila je 44.7 milijardi dolara. U istraživanju su pratili zaradu po platformama App Store i Google Play gdje je App Store nadmašio Google Play za 7.3 milijarde dolara [11].

Prema [11], kineska organizacija *Tencent* imala je igru *Honor of Kings* koja im je donijela najveću zaradu u prvoj polovici 2021. godine od 1.5 milijardi dolara, a druga igra im je bila lokalizirana inačica popularne igre *PUBG Mobile* koja im je donosila zaradu od 7.4 milijuna dolara dnevno. Brojke o zaradama su isključivo s Google Playa i App Storea, a zarada je potencijalno i veća zbog ostalih platformi koje se nisu uzele u obzir pri istraživanju.

Prema [12], na slici 2.4 vidljiva je podjela prema spolu mobilnih igrača. Vidljivo je kako je veći broj osoba ženskog spola koje igraju mobilne računalne igre. Na slici 2.5 prikazane su dobne skupine ispitanika koji igraju mobilne računalne igre. Vidljivo je kako dob ne utječe na prisutnost interesa za mobilne igre. Dob je podijeljena u pet skupina u kojim niti jedna ne pada ispod 10%. Također, vidljivo je kako najučestalija skupina igrača je od punoljetne do srednje dobi.



Slika 2.4. Spolna podjela ispitanika koji igraju mobilne računalne igre [12]



Slika 2.5. Dobna podjela ispitanika koju igraju mobilne računalne igre [12]

Slično kao i na slici 2.4, prema [13], 64% žena više preferira mobilne računalne igre nego druge platforme, dok je kod muškaraca zainteresiranost nešto manja, točnije svega 38%.

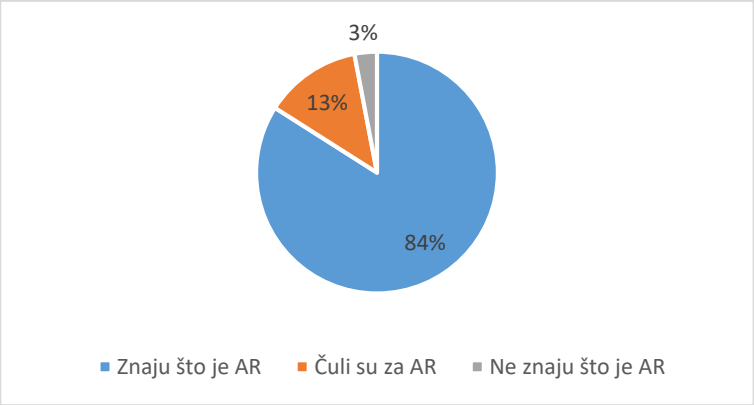
2.2.2. Prenosivost mobilnih računalnih igara i izazovi

Prema [14], razvoj mobilnih igara većinom se smatra kao manja zadaća naspram razvoja računalnih igara za druge platforme koje imaju puno više resursa koje mogu potrošiti na poboljšanje raznih aspekta igre. Iz razloga što mobilni uređaji nemaju toliko resursa potrebno je pripaziti na prenosivost igara da bi se mogle pokretati na raznim inačicama uređaja. Prenosivost je jedna od glavnih faza razvoja mobilnih računalnih igara i na nju se stavlja najveći naglasak kako ne bi došlo do velikih pogrešaka koje znaju donijeti velike novčane i vremenske gubitke. Uz različite mogućnosti uređaja programeri moraju biti pažljivi tijekom razvoja prenosivosti mobilnih računalnih igara. Kao što je već navedeno, prenosivost je najbitnija faza razvoja mobilne računalne igre pa su učestali i izazovi koje je potrebno nadići. Postoje i tvrtke koje se specifično bave prenosivošću mobilnih računalnih igara. Neki od izazova prenosivosti su različite veličine zaslona, rezolucija, zvuk, memorija, operacijski sustav, različiti operateri, lokalizacija, različiti zakoni u državama, itd. Neki od načina rješavanja problema prenosivosti je izbacivanje nekoliko inačica aplikacije da bi bila optimizirana za točno određenu konfiguraciju uređaja ili za točno određeno tržište. Da bi prenosivost mobilnih računalnih igara bila što bolje definirana potrebno je da je igra proširiva, lako održiva, prilagodljiva ovisno o uređaju, itd.

2.3. Proširena stvarnost u mobilnim računalnim igrama

Prema [15], navodi se kako proširena stvarnost ima veliki potencijal za poboljšanje korisničkog iskustva u mobilnim igrama. Pretpostavka je bila kako će do 2022. godine 34% cijelog tržišta proširene i izmiješane stvarnosti biti u video igrama. Provedeno je istraživanje nad grupom

ispitanika koji su igrali mobilne računalne igre koje je moguće igrati u proširenoj stvarnosti i bez nje. Postavljali su im pitanja s glavnim konceptima kao što su je li im igra okupirala pažnju osjetila i misli, je li korisnik osjetio da može iskoristiti svoje sposobnosti, je li mu igra uspjela u potpunosti okupirati pažnju, je li igra imala pozitivan utjecaj, je li igra imala negativan utjecaj, je li se korisnik osjećao frustrirano ili napeto, je li igra bila izazovna, je li korisnik imao pozitivni utjecaj nakon igranja igre, je li korisnik imao negativan utjecaj nakon igranja igre, je li mu bilo teško vratiti se u stvarni svijet i je li umoran nakon igranja. Pitanja su postavljena kroz dva upitnika od kojih se jedan provodi tijekom igranja, a drugi nakon igranja. Na slici 2.6 vidljiv je udio ispitanika koji je poznavao što je proširena stvarnost prije početka istraživanja, udio ispitanika koji su čuli za proširenu stvarnost i udio ispitanika koji nikad nisu čuli za proširenu stvarnost. Dobna skupina ispitanika obuhvaćala je osobe od 18 do 39 godina što čini podatke o poznavanju proširene stvarnosti očekivanim da prevladavaju osobe koje znaju što je proširena stvarnost [15].



Slika 2.6. Prikaz udjela ispitanika o poznavanju proširene stvarnosti (engl. AR) [15]

Nakon usporedbe prve igre uz uključenu i isključenu proširenu stvarnost dobiveni rezultati pokazali su kako im je proširena stvarnost više okupirala pažnju i bila izazovnija, a da su pozitivni i negativni utjecaji bili jednaki. Područje u kojem je prevladala igra bez proširene stvarnosti bilo je osjećaj iskorištenja korisničkih mogućnosti. Što je očekivano jer proširena stvarnost može imati i druge utjecaje na ispitanike koji se ne usredotoče u potpunosti na igru ili u slučaju da igra sporije reagira. Nakon svih obavljenih upitnika grupirani su podaci u pet različitih grupa ovisno o načinu korištenja proširene stvarnosti, a to su postavljeni u prostor, postavljeno na pod, potrebno hodati i orijentirati se, potrebno pomicati objekte u proširenoj stvarnosti te potrebno kretanje i pomicanje objekata u proširenoj stvarnosti [15].

U tablici 2.1 prikazani su rezultati istraživanja razlika korisničkog iskustva između igre s proširenom stvarnošću i igre bez proširene stvarnosti. Istraživanje iz [15], odnosi se na više načina postavljanja predmeta u prostor proširene stvarnosti, ali u tablici 2.1 prikazani su samo rezultati koji se odnose na predmete koji se postavljaju na tlo. Način postavljanja predmeta na tlo korišten je i u mobilnoj računalnoj igri rada. Negativne vrijednosti odnose se na negativan doživljaj proširene stvarnosti, a pozitivne na pozitivan. Prema rezultatima iz tablice, vidljivo je kako proširena stvarnost pozitivno utječe na većinu promatranih aspekata ispitanika. Proširena stvarnost ima potencijal podići korisničko iskustvo u mobilnim računalnim igrama na novu razinu, ali je potrebno pametno je implementirati kako rješenje ne bi bilo prekomplicirano za korištenje[15].

Tablica 2.1. Pregled rezultata ispitanika o osjećajima nakon igranja igre u proširenoj stvarnosti

Udubljenje osjetila	23.0%
Kompetencije	4.8%
Pozitivni utjecaj	22.2%
Negativan utjecaj	-27.8%
Protok igre	24.7%
Tenzije	11.5%
Zahtjevnost	25.9%
Pozitivno sveukupno iskustvo	18.6%
Negativno sveukupno iskustvo	-27.3%
Povratak u stvarnost	-4.5%
Umor	-26.9%

Prema [16], namjera je bila napraviti mobilnu računalnu igru koja bi služila u edukacijske svrhe, a kao primarnu tehnologiju koristili bi proširenu stvarnost. Kao temu igre uzeli su edukacije djece i tinejdžera kako biti odgovoran vlasnik kućnog ljubimca. Napravljena su dva ljubimca u obliku psa i mačke o kojim su napisali njihove životne potrebe. Upute oko obveza koje dobiju s ljubimcima pokazuju se tijekom igranja igre u obliku oblačića iznad ljubimca. Nakon igranja igre moguće je proći kroz ispit s pitanjima u vezi naučenog iz mobilne računalne igre. Trenutno igra više ne postoji na App Storeu, ali u radu se navodi da je bila na App Storeu i da je nekolicina korisnika izjavila da im je aplikacija zanimljiva. Ovakvo istraživanje govori kako proširena stvarnost ima veliku moć približiti stvarne događaje na zabavan i edukativan način.

2.4. Računalne mobilne igre u proširenoj stvarnosti

Mobilne računalne igre opisane u ovom potpoglavlju moguće je preuzeti na platformama App Store ili Google Play Store. Jedina iznimka je što ne postoji inačica mobilne računalne igre *Stack AR* za Android nego samo inačica bez proširene stvarnosti.

2.4.1. Pokémon Go

Računalna mobilna igra *Pokémon Go* napravljena je od strane *Niantic Inc*. *Nianticu Pokémon Go* nije prva igra u kojoj se koristi proširena stvarnost, kasnije će se spomenuti *Ingress Prime*. *Niantic* je tvrtka koja se među prvima bacila u vode proširene stvarnosti i uspješni su u tome. *Pokémon Go* je njihova najuspješnija mobilna računalna igra kojoj je tema serijal *Pokémon* i hvatanje *Pokémona* po cijelome svijetu. Igra omogućuje korisnicima, bez obzira gdje se nalazili u svijetu, traženje i hvatanje *Pokémona*. *Pokémon Go* omogućuje korištenje igre i uređajima koji nemaju mogućnost proširene stvarnosti ili je samo ne žele koristiti.

Igra ima mnoštvo složenih zaslona kao što su *Login*, *Home*, *Pokedex*, *Quests* i zaslon sa svim uhvaćenim *Pokémonima*. Glavni zaslon igre je *Home* gdje se korisniku omogućuje navigacija na sve ostale zaslone i igranje igre. Prati se korisnikova lokacija i na karti se prikazuju *Pokémoni* koje korisnik može hvatati. Proširena stvarnost se očituje u igri tek nakon što korisnik želi uhvatiti *Pokémona*, tada mu se daje mogućnost uključivanja proširene stvarnosti da bi se *Pokémon* prikazao u stvarnom prostoru i da bi ga korisnik pronašao i uhvatio. Također, *Pokémon Go* omogućuje korisniku da i uhvaćene *Pokémone* stvori u proširenoj stvarnosti i postavi ih negdje u stvarnom svijetu da bi ih mogao uslikati.

2.4.2. The Witcher: Monster Slayer

The Witcher: Monster Slayer računalna mobilna igra nastala je od strane *Spokko sp. z.o.o*., a svoju inspiraciju našla je u jako popularnoj računalnoj igri *The Witcher 3* koju je napravio *CD Projekt Red*. Glavna pozadina *The Witcher* igara, bilo to mobilnih ili drugih, su knjige *Witcher* serijala koje je napisao *Andrzej Sapkowski*. Zbog velike popularnosti *Witcher* serijala i proširene stvarnosti na mobilnim uređajima, htjeli su se proširiti i na mobilne uređaje s *The Witcher: Monster Slayer* mobilnom igrom koja nije doživjela uspjeh koliki se očekivao. Trenutna ocjena na Google Play trgovini je 3.0/5.0, iako na App Storeu je 4.3/5.0, ali ima samo 24 glasa.

The Witcher: Monster Slayer sastoji se od nekoliko zaslona kao što su *Login*, *Home*, *Inventory*, *Character*, *Quests* i *Settings*. Igra omogućuje korisniku da odmah nakon registracije prođe vodič

kroz igru i započne s početnim zadacima. Glavni zaslon na kojem korisnik provodi najviše vremena je *Home*. Na *Homeu* se nalazi karta sa stvarnim ulicama na kojoj se očituje korisnikova trenutna lokacija u svijetu. Korisnik je vještac kojem je cilj postati što jači i sakupiti što više trofeja različitih zvijeri na koje naiđe tijekom istraživanja svijeta. Kretanjem kroz svijet korisnik može naići na nove zadatke, a i čudovišta s kojim se može boriti. *The Witcher: Monster Slayer* igra proširenu stvarnost primjenjuje u borbama protiv čudovišta. Korisnik može upaliti proširenu stvarnost i boriti se u stvarnom svijetu s čudovištima koristeći sve u svojoj moći da pobijedi zvijer te osvoji novi trofej. Igru je moguće igrati i bez proširene stvarnosti, ali se korištenjem proširene stvarnosti dobiva veća razina stvarnosti tijekom uzimanja novih zadataka i borbe sa zvijerima.

2.4.3. Stack AR

Stack AR je arkadna računalna mobilna igra koju je razvio *Ketchapp*. *Ketchapp* je poznat po stvaranju arkadnih mobilnih igara koje su korisnici hvalili kao što su *Mr Gun* ili *Knife Hit*. *Stack AR* je jednostavna arkadna igra u kojoj se omogućuje korisniku da gradi toranj od blokova koji mu dolijeću sa strane. Igra korisniku omogućuje da uključi i način igranja u proširenoj stvarnosti gdje se prvotno skenira horizontalna površina da bi se korisniku pokazalo gdje može postaviti početni blok. Nakon što korisnik odabere početno mjesto igre, igra započinje i korisnik mora što preciznije postavljati blokove jedan na drugi. U slučaju da korisnik ne uspije postaviti blok točno na iznad prijašnjeg, dio bloka otpada i sljedeći blok je jednake veličine kao zadnji postavljeni blok. Blokovi se smanjuju sve dok korisnik u potpunosti ne promaši prijašnji blok i tada igra završava te se korisniku ispisuju bodovi koje može podijeliti.

2.4.4. Angry Birds AR: Isle of Pigs

Angry Birds AR: Isle of Pigs, kao i njegovi prethodnici, napravljen je od strane *Rovio Entertainmenta*. *Angry Birds* stariji je serijal igara na mobilnim uređajima koji je imao mnoštvo nastavaka i u nekim trenucima bio jako popularan. *Rovio Entertainment* pokušao je doživjeti staru slavu *Angry Birds* serijala uvođenjem proširene stvarnosti u njihov najpoznatiji serijal igara. Igra je napravljena tako da korisnik može odabrati razinu koju želi igrati i postaviti je na pravilnu površinu unutar stvarnog svijeta. Nakon što korisnik pronađe pravilnu površinu na koju se može postaviti odabrana razina, vrijeme je za igranje poznate igre u proširenoj stvarnosti. Korisnik se može ograničeno kretati kroz prostor da bi pronašao najbolju poziciju za pogoditi i da bi što uspješnije prošao razinu. *Angry Birds AR: Isle of Pigs* nije doživjela uspjeh kao neki od njenih prethodnika, ali uvela je funkcionalnost proširene stvarnosti u dobro poznat serijal *Angry Birds*.

2.4.5. Kings of Pool

Kings of Pool je napravljena od strane *Uken Inc.,* a glavna tema je igranje biljara. Računalna mobilna igra korisniku omogućuje da se prijavi da bi mogao započeti s igrom. Korisniku se omogućuje kupovanje ili osvajanje različitih štapova za igranje koji variraju po jačini. Igra omogućuje korisniku igranje na različitim lokacijama u svijetu za različitu svotu vrijednosti koja se skuplja u igri. Glavna razlika što čini *Kings of Pool* drugačijim od drugih je što su omogućili korisnicima koji imaju mogućnost korištenja proširene stvarnosti, da postave biljarski stol bilo gdje u stvarnom prostoru te da tako mogu igrati kao i bez proširene stvarnosti.

2.5. Idejno rješenje predložene mobilne računalne igre

Predložena mobilna računalna igra treba sadržavati mogućnosti odabira početka igre na početnom zaslonu. Korisnik treba imati mogućnost promjene parametara igre da bi igru mogao otežati ili olakšati na zaslonu s postavkama igre. Prije samoga početka igre, korisnika treba obavijestiti o pravilima igre da bi znao koji je cilj igre i kada igra završava. Također, korisnika treba obavijestiti i o mogućnosti promjene parametara igre. Tijekom igranja igra treba pratiti stvarne predmete u prostoru da bi se pravilno mogao postaviti prostor za igru. Uz stvarne predmete, igra treba pratiti imaginarne predmete koje postavi u prostor da bi se očitao kraj igre i pratili bodovi koje korisnik postigne tijekom jedne igre. Kad korisnička igra završi, potrebno je upitati korisnika želi li pohraniti rezultat. U slučaju da se korisnik odluči na pohranu rezultata, potrebno je pohraniti rezultat i podatke o korisniku u bazu podataka.

Mobilna računalna igra predložena u radu, s obzirom na opisane mobilne računalne igre u potpoglavlju 2.4, u prvoj inačici neće imati mogućnost igre bez proširene stvarnosti. S druge strane, ima sličnosti s većinom spomenutih mobilnih računalnih igara gdje je potrebno učitavanje horizontalne površine da bi se dodali imaginarni objekti u stvarni prostor. Također, kao i većina spomenutih mobilnih računalnih igara, koristi tip proširene stvarnosti bez ugrađenih predmeta.

3. PRIJEDLOG MODELA MOBILNE RAČUNALNE IGRE

U ovom poglavlju opisat će se pravila, parametri i načini praćenja završetka mobilne računalne igre *Snake.* Nadalje, opisat će se funkcionalni i nefunkcionalni zahtjevi mobilne računalne igre, te dati prijedlog modela mobilne računalne igre kroz komponente mobilne računalne igre.

3.1. Opis mobilne računalne igre Snake

3.1.1. Pravila igre

Računalna mobilna igra *Snake* poštuje glavna pravila prijašnjih igara koje zahtijevaju od igrača da kontrolira zmiju unutar okvira igre. Okvir igre je prostor u kojem se zmija može kretati. Korisnik kontrolira zmiju tako što povlači po zaslonu lijevo, desno, gore i dolje. Potrebno je znati kako će skretanje zmije u jednu od strana biti uračunato ovisno o poziciji kamere i zmije. U slučaju da korisnik pokuša izaći izvan predviđenog prostora, igra završava, jer se zmija sudarila sa zidom. Također, postoji još jedan način kako igra može završiti, a to je tako što zmija glavom dodirne svoje tijelo. Glavni cilj igre je postići što veći rezultat bez da se igrač sudari sa zidom ili da se glava zmije sudari s drugim dijelom zmije. Korisnik svoj rezultat povećava tako što skuplja novčiće koji se pojavljuju unutar prostora predviđenog za igru. U svakom trenutku na prostoru za igru može se nalaziti samo jedan novčić, a vrijednost novčića ovisi o parametrima igre koji se mogu mijenjati prije početka igre. Parametri se odnose na veličinu prostora unutar kojeg se zmija može kretati, što je manji prostor veći broj bodova, i brzinu zmiju, što je brža zmija to je veći broj bodova.

3.1.2. Načini praćenja završetka igre

Završetak igre definiran je pravilima koja su navedena u potpoglavlju 3.1.1. Predložena mobilna računalna igra rada treba pratiti kretanje zmije kroz prostor, njenu poziciju i poziciju zida. Što se tiče praćenja predmeta u prostoru tu znatno pomaže *ARKit* koji će biti objašnjen u sljedećem poglavlju. Omogućuje praćenje predmeta i ima mogućnost otkrivanja dodira predmeta u proširenoj stvarnosti. Da bi mobilna računalna igra rada prepoznala koji su se predmeti dodirnuli u proširenoj stvarnosti potrebno ih je pravilno definirati inače ne bi moglo biti diferencirano je li pokupljen novčić ili dotaknut zid. *ARKit* ima mogućnost dodatnog definiranja objekata kojim se u svakom trenutku zna koja dva objekta su se dodirnula. U slučaju završetka igre, prate se glava zmije, tijelo zmije i zid. Čim glava zmije dotakne zid ili glava zmije dodirne dio tijela, igra završava.

3.1.3. Parametri igre

Parametri koji utječu na igru su veličina prostora za igranje i brzina zmije. Pri prvom pokretanju računalne mobilne igre parametri su postavljeni na srednje vrijednosti. U slučaju da ih korisnik želi mijenjati, imat će tu mogućnost prije početka igre na zaslonu s postavkama igre. Parametri utječu na broj bodova koje će korisnik ostvariti pri svakom prikupljanju novčića, a računa se prema izrazu 3.1.

$$vrijednost\ novčića = brzina\ zmije * veličina\ prostora\ za\ igru \tag{3-1}$$

Parametar koji se odnosi na veličinu prostora za igranje, kao što i samo ime govori, utječe na veličinu prostora unutar kojeg korisnik može upravljati zmijom. Što je prostor manji, korisnik dobiva više bodova za svaki novčić koji pokupi. Manjim prostorom korisniku se smanjuje prostor za pogreške jer pri većim veličinama zmije potrebno je pravilno kretanje kako se korisnik ne bi zablokirao tijelom zmije i time bi u jednom trenutku glavom zmije dodirnuo tijelo zmije. Postoje tri razine veličine prostora za igru, a to su mali, srednji i veliki. Kao što je ranije spomenuto, veličina prostora za igru utječe na količinu bodova pa se tako i bodovi kreću tri, dva i jedan za spomenute veličine prostora za igru.

Parametar koji se odnosi na brzinu zmije utječe na brzinu kretanja zmije u prostoru. Što je brzina veća, korisnik dobiva više bodova za svaki novčić koji pokupi. Većom brzinom korisnik, kao i kod manjeg prostora za igru, ima manje prostora za pogreške jer se povećanjem brzine smanjuje vrijeme za reakciju. Postoje tri razine brzine zmije kao i kod veličine prostora za igranje. Korisnik može odabrati spori, srednji i brzi način igre. Želimo nagraditi korisnike koji se odluče za veće brzine zmije pa prema navedenim načinima igre vrijednosti se kreću jedan, dva i šest.

Vidljivo je iz spomenutoga da će korisnici koji se odluče za brži način igre na manjem prostoru za igru dobivati znatno više bodova za prikupljene novčiće nego korisnici koji se odluče za sporiju igru ili manji prostor. Iz spomenutog izraza po kojem se računa vrijednost novčića, vidljivo je da rast vrijednosti novčića nije linearan, jer će korisnicima pri većim brzinama i manjim prostorom za igru biti znatno teže.

3.2. Funkcionalni i nefunkcionalni zahtjevi

3.2.1. Funkcionalni zahtjevi

Prema [17], funkcionalni zahtjevi su opisi funkcija koje aplikacija mora omogućavati korisnicima. Ti opisi funkcija sastoje se od ulaza, ponašanja aplikacije i izlaza, da bi se stvorila što detaljnija slika što točno aplikacija treba sadržavati.

Funkcionalni zahtjevi su obvezni. U agilnim modelima razvoja programske podrške nazivaju se korisničke priče (engl. user story) koje sadržavaju što korisnik mora napraviti da bi došao u određeno stanje, što može napraviti i što očekuje da će se dogoditi. Kada programeri imaju dostupne funkcionalne zahtjeve, tada mogu dati pravilne vremenske procjene.

Funkcionalni zahtjevi mobilne računalne igre ovog rada mogu se podijeliti na tri dijela: prije početka igre, igranje igre i završetak igre. Prije početka igre na početnom zaslonu treba omogućiti korisniku početak igre, prikaz zaslona ljestvice najboljih rezultata i odlazak na zaslon s postavkama igre. Na zaslonu s postavkama igre treba omogućiti promjenu brzine igre i veličine područja igranja te pregled pravila igre. Dio funkcionalnih zahtjeva igranja igre treba omogućiti zaslon za igranje arkadne igre *Snake* u proširenoj stvarnosti. Igra će pratiti stvarne i imaginarne predmete u prostoru te pratiti međusobne dodire predmeta u prostoru. Funkcionalni zahtjevi završetka igre su zaslon za upit korisnika želi li pohraniti rezultat, pohrana korisničkih rezultata i pregled pohranjenih korisničkih rezultata.

3.2.2. Nefunkcionalni zahtjevi

Prema [18], nefunkcionalni zahtjevi su zahtjevi koje sustav mora zadovoljiti. Neki od njih su brzina, sigurnost, pouzdanost, prenosivost, kompatibilnost, održivost, performanse i lakoća korištenja.

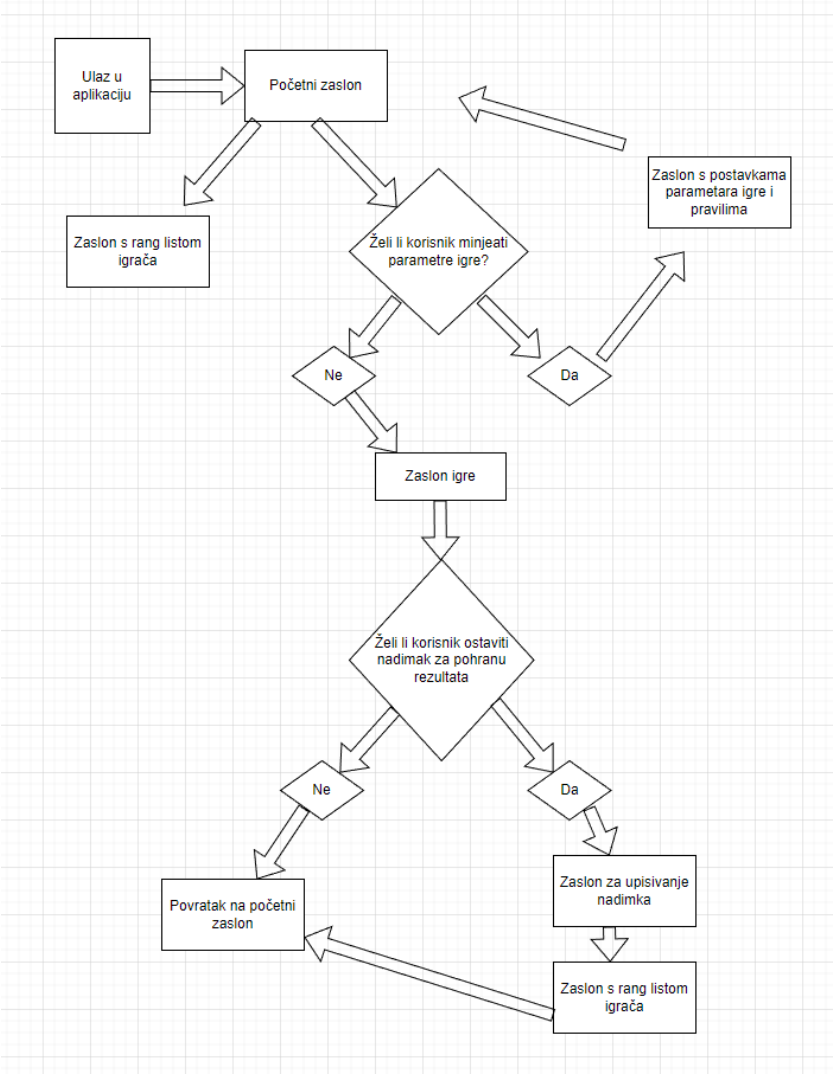
Prema [19], nefunkcionalni zahtjevi su zahtjevi koji određuju kvalitetu aplikacije. Navodi prednosti nefunkcionalnih zahtjeva u poštovanju legalne strane razvoja programske podrške, kvalitete, performansi, sigurnosti, lakoće korištenja, ekonomičnosti i korisničkog iskustva.

U slučaju mobilne računalne igre rada, želi se održati opterećenje RAM-a ispod 500 MB da bi se i na starijim uređajima mobilna računalna igra mogla koristiti. Potrebno je omogućiti kompatibilnost inačica *iOS* sustava od *iOS 13* do trenutno najnovijeg *iOS 15*. Također, potrebno je pisati čitljiv i razumljiv kod poštivanjem načela arhitekture VIPER i principa SOLID kako bi se

omogućilo lako održavanje i nadogradnja. Pri razvoju mobilne računalne igre potrebno je koristiti asinkrono programiranje da bi performanse bile što bolje.

3.3. Komponente mobilne računalne igre

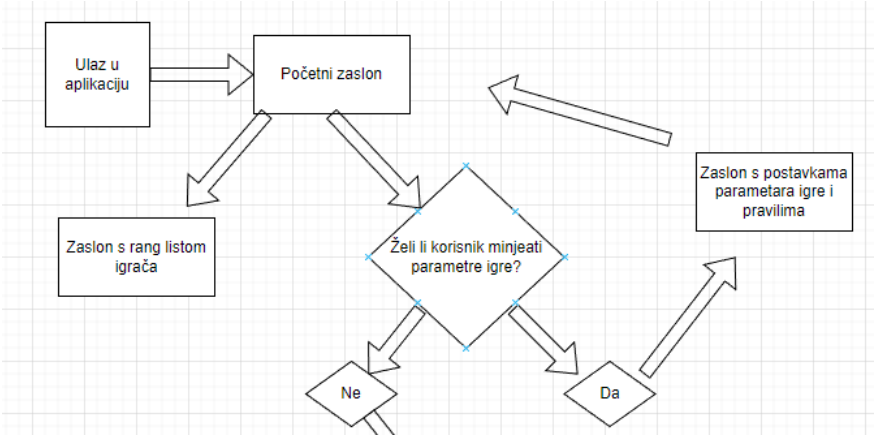
Na slici 3.1 prikazan je dijagram toka komponenti mobilne računalne igre. Opisane su komponente od ulaska u aplikaciju do završetka igre. Pri ulasku u aplikaciju korisnik može otići na zaslon s ljestvicom najboljih rezultata, zaslon s postavkama ili zaslon za igranje igre. Nakon pokretanja igre i završavanja na jedan od načina, korisniku se daje mogućnost pohrane postignutog rezultata. U slučaju da korisnik pristane na pohranu rezultata prikazuje se zaslon ljestvice najboljih rezultata, a povratkom natrag prikazuje se početni zaslon. Sve komponente i korisnički slučajevi opisat će se detaljnije u daljnjem tekstu.



Slika 3.1. Dijagram toka korištenja komponenti mobilne računalne igre

3.3.1. Početne komponente mobilne računalne igre

Pojednostavljenjem dijagrama na slici 3.1 dobije se nekoliko manjih smislenih cjelina. Prva cjelina je početni zaslon s postavljanjem parametara igre i rang lista igrača. Ako se izdvoji cjelina iz dijagrama na slici 3.1 dobije se smanjeni dijagram toka prikazan na slici 3.2.

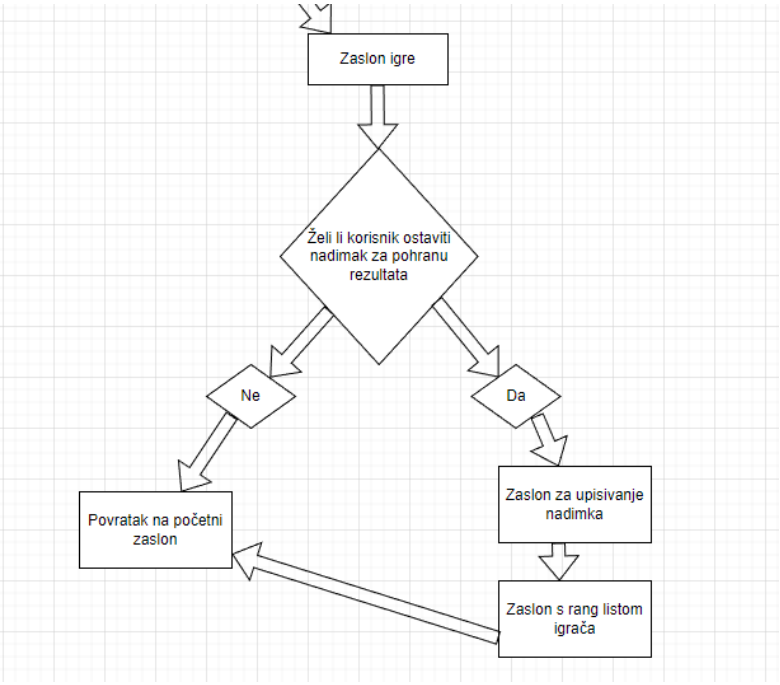


Slika 3.2. Dijagram toka aktivnosti postavljanja parametara igre

Nakon ulaska u mobilnu računalnu igru, korisnika dočeka početni zaslon s nekoliko mogućnosti. Korisnik može pogledati rang listu prijašnjih igrača te njihove rezultate. U slučaju da je lista prazna, dočekat će ga poruka da još nema rezultata. Korisnik prije početka igre, na zaslonu s postavkama igre, može promijeniti neke od parametara igre kao što su veličina igraće površine i brzina zmije što će utjecati na količinu bodova koju dobije tijekom igre. U slučaju da korisnik želi započeti igru, prikazat će mu se zaslon na kojem se spominje da može promijeniti neke od parametara igre na kojem može odabrati da ili da ne želi. U slučaju da korisnik odabere odgovor „Da“, vodi se na zaslon s postavkama igre, a u slučaju da korisnik odabere „Ne“ vodi se na zaslon igre. Uz navedene opcije, korisnik može odabrati i opciju „Ne pokazuj više“ kako mu se ne bi prikazivao zaslon s upitom svaki put kad želi pokrenuti novu igru.

3.3.2. Komponente početka i završetka igre

Daljnjim pojednostavljivanjem dijagrama sa slike 3.1 na dio komponenti koje se tiču početka i završetka igre dobiva se dijagram prikazan na slici 3.3.

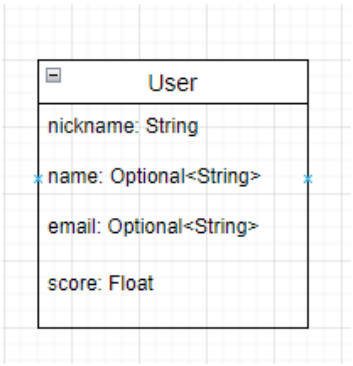


Slika 3.3. Dijagram toka igranja i kraja igre

Nakon što je korisnik pokrenuo igru sa željenim parametrima, potrebno je pronaći horizontalnu površinu na koju se može postaviti područje za igru. Kad se postavi područje za igru, kreće odbrojavanje nakon kojeg korisnik mora upravljati zmijom po pravilima igre koja će biti opisana u sljedećem poglavlju da bi što uspješnije igrao igru. Nakon što korisnik završi igru, pojavit će se zaslon na kojem može odabrati želi li sačuvati rezultat u bazu podataka ili ne. U slučaju da korisnik odabere da ne želi, vraća se na početni zaslon. Ako korisnik odabere da želi sačuvati rezultat u bazu podataka, prikazuje mu se zaslon za upisivanje nadimka nakon kojeg se prikazuje rang lista svih igrača u bazi podataka. Nakon što korisnik završi s rang listom vraća se na početni zaslon.

3.3.3. Podaci u bazi podataka

Baza podataka čuvat će podatke samo o korisnicima koji su izjavili želju da se njihov rezultat pohrani u bazu podataka. Podaci koji će se čuvati o korisnicima su nadimak, ime, e-pošta i ostvareni rezultat. Ime i e-pošta su neobvezni parametri koje korisnik ne mora unijeti da bi se pohranio rezultat. Model korisnika koji se čuva je jednostavan te neće trošiti puno resursa pri dohvaćanju podataka. Model korisnika vidljiv je na slici 3.4.



Slika 3.4. Model korisnika u bazi podataka

Pohranjivanje rezultata odvijat će se na platformi *Firebase* preko koje će svi korisnici moći pristupiti jednakim rezultatima u bilo koje vrijeme. Pohranjivat će se modeli podataka kakvi su opisani u drugom poglavlju da bi se mogli pratiti svi postignuti rezultati korisnika.

U slučaju podataka koji se vežu za uređaj, kao što je odabir brzine zmije, veličine prostora igranja i treba li se prikazivati zaslon s informacijama prije početka igre, koristit će se *UserDefaults* što je lokalna baza koju je omogućio *Apple*.

4. PROGRAMSKO RJEŠENJE RAČUNALNE IGRE SNAKE

U ovom poglavlju opisat će se korištena okruženja, alati i tehnologije za izradu mobilne računalne igre. Naglasak je na programskom kodu i programskim rješenjima klijentske i poslužiteljske strane te opisu korištene arhitekture VIPER.

4.1. Korišteni alati i tehnologije pri izradi računalne igre

4.1.1. Xcode

Prema [20], *Xcode* je razvijen od strane Applea kako bi stvorili razvojno okruženje za sve svoje platforme. Trenutno je dostupan samo za *macOS* sustave, a omogućuje razvijanje aplikacija za *iOS*, *iPadOS*, *watchOS* i *tvOS* sustave. Kako je ograničen samo na *macOS* sustave potrebno je imati Appleovu opremu da bi se mogle razvijati aplikacije. *Xcode* omogućuje pisanje u mnoštvu jezika kao što su *C#, Python, Objective-C, Swift* i drugi. Za razvoj aplikacija na *iOS* uređajima najčešće se koristi objektno-orijentirani programski jezik Swift. Tijekom razvoja aplikacija, *Xcode* korisniku omogućuje vizualno slaganje zaslona, programatsko slaganje zaslona, automatsko dovršavanje funkcija, daje prijedloge s obzirom na napisano i druga pomagala.

Za instaliranje aplikacija na mobilne uređaje, potrebno je imati razvojni certifikat koji omogućuje samo lokalne instalacije, a za postavljanje aplikacija na App Store ili TestFlight, potrebno je imati certifikate veće razine dopuštenja koji se naplaćuju. Razvojni certifikat moguće je dobiti samom registracijom Apple ID-jem. Također, moguće je koristiti i simulatore koji dolaze s *Xcodeom* da bi testirali svoje aplikacije pri čemu nisu potrebni certifikati. Simulatori samo simuliraju *iOS* verziju na odabranoj verziji uređaja dok koriste memoriju i ostale resurse s *macOS* uređaja zbog čega su u većini slučajeva brži od Android emulatora. Tijekom testiranja moguće je vidjeti potrošnju resursa na uređaju ili simulatoru te postoje napredni alati koji omogućavaju provjeru *memory leakova*, *retain cycleova*, *pregled vrijednosti UI* (engl. user interface) *elemenata trenutno prikazanih na zaslonu* i druga pomagala.

Posljednja inačica *Xcodea* je 14 koja je izašla na WWDC22 u lipnju 2022. godine i omogućuje razvoj za *iOS 16* uređaje koji bi trebali izaći u rujnu 2022. godine.

4.1.2. Swift

Prema [21], *Swift* je jedinstveni programski jezik koji omogućuje razvoj programske podrške za mobilne uređaje, računala i poslužitelje. Brz je, siguran i *open-source*. *Apple* ga razvija, ali uzima

i prijedloge korisnika koji ga koriste. Konstantno se razvija i dodaju se nove stvari da bi se korisnicima skratilo vrijeme i olakšalo stvaranje aplikacija. Neke su prednosti *Swifta* stvaranje svih varijabli prije nego što se koriste, *optional* vrijednosti se uvijek moraju riješiti na neki način te se memorija alocira i oslobađa automatski. Uz ostale, prednost *Swifta* je što je *high-level* objektno-orijentirani jezik, ali i dalje pruža brzine *low-level* programskih jezika.

Prema [22], *Swift* je stvoren da bi zamijenio *Objective-C* kao primarni jezik stvaranja aplikacija za *Appleove* uređaje. Moguće ga je koristiti u istom projektu gdje se koristi i *Objective-C* što omogućuje postupan prelazak starijih *Objective-C* projekata na *Swift*. Navodi kako je *Swift* uspio obaviti neke zadaće brže od *Jave*, ali svakako je brži od svog prethodnika *Objective-C*-a. *Swift* je izašao 2014. godine što ga čini relativno mladim programskim jezikom i s velikim očekivanjima u prostoru povećanja brzine kao i mogućnosti razvoja aplikacija za još više platformi.

4.1.3. Combine

Prema [23], *Combine* je deklarativni način za obradu podataka kroz vrijeme. Takve vrijednosti mogu se dohvaćati u pozadinskim procesima pa se obrada treba odraditi kad se dobiju vrijednosti nakon završetka procesa, a mogu biti i procesi koji se obavljaju na glavnoj niti. *Combine* kao sučelje omogućuje korisnicima praćenje vrijednosti kroz *Publishere* koje se mogu mijenjati tijekom vremena, a omogućuje im i *Subscribere* koji primaju te vrijednosti i rade s njima što korisnik želi. Ovakav način programiranja naziva se reaktivno programiranje gdje se reagira na promjene vrijednosti unutar sustava i koristi se u sustavima gdje se očekuju promjene tijekom vremena. U većini slučajeva koristi se reaktivno programiranje, ali treba biti pažljiv da se ne koristi tamo gdje nije potrebno jer se dodatno komplicira stanje programskog koda.

4.1.4. Asinkrono programiranje

Prema [24], asinkronim programiranjem korisniku se omogućuje da radi što želi u aplikaciji dok se podaci dohvaćaju ili stvaraju u pozadini. Korisniku se prikazuje jedan sadržaj s kojim može raditi za štogod je sadržaj namijenjen dok mu se drugi dio sadržaja učitava u pozadini. Nakon što se sadržaj učita, prikazuje se korisniku. Također, spominje kako asinkrono programiranje omogućuje pokretanje više neovisnih procesa odjednom bez čekanja da jedan od njih završi. Time se u većoj mjeri iskorištavaju resursi uređaja da bi korisničko iskustvo bilo bolje i brže.

U inačici *Swifta* 5.5 izašao je i način asinkronog programiranja *async/await* koji omogućuje programerima da pišu kao da se kod izvodi liniju po liniju, ali je asinkron. Prema [25], moguće je

skroz ukloniti *closureove* u kojim smo obavljali funkcionalnosti s vrijednostima koje smo iščekivali jer se uz sučelje *async/await* može jednostavno spremiti iščekivana vrijednost u varijablu, a program se neće dalje izvoditi dok se ta vrijednost ne dobije.

Glavne inačice asinkronog programiranja u *Swiftu* su *RxSwift, Combine, DispatchQueue, async/await, NotificationCenter.* Iz navedenog vidljivo je kako asinkrono programiranje nije nov koncept i da postoje razni načini kako se može koristiti na *iOS* platformi. U slučaju mobilne računalne igre rada za postizanje asinkronosti korišten je primarno *Combine,* ali na nekim mjestima i *async/await* da bi se prikazao primjer korištenja i *async/await* paradigme asinkronog programiranja.

4.1.5. ARKit

Kao što je već objašnjeno u drugom poglavlju, proširena stvarnost je način postavljanja imaginarnih objekata u stvarni svijet koji omogućuje korisnicima dodatnu interakciju sa stvarnošću. *ARKit* je način implementacije proširene stvarnosti na platformi *iOS* kojeg je moguće koristiti od *iOS 11*. Jedna od nativnih aplikacija koja koristi *ARKit* je *Metar* kojim je moguće kamerom izmjeriti udaljenost između dvije točke u prostoru.

Prema [26], *ARKit* koristi kameru iPhonea kako bi dobio informacije iz okoline gdje pokušava otkriti zidove i podlogu kako bi otkrio geometriju prostora. Prati vertikalne i horizontalne površine te kretanje uz pomoć žiroskopa, brzinomjera i kompasa. *ARKit* dopušta dodavanje imaginarnih predmeta u stvarni svijet i omogućuje interakciju s istim. Također, čar *ARKita* je praćenje objekata u prostoru i kad iPhone ne gleda u objekt nego se korisnik može kretati, a objekt je i dalje na svom prvotnom mjestu. *ARKit* je moguće koristiti i na više uređaja koji su povezani mrežom tako da sve osobe koje su povezane na mrežu vide iste objekte. Kao što je i prije spomenuto, za realiziranje proširene stvarnosti potrebna je jaka procesorska moć da bi se sve informacije uspjele na vrijeme obraditi pa je potrebno i to imati na umu.

Prema [27], *ARKit* odrađuje puno teškog posla u pozadini koji se tiče prepoznavanja slika, kamere, prepoznavanja objekata i pozicioniranja objekata u prostoru kako bi programerima olakšali razvoj aplikacija. *ARKit* tijekom praćenja stvarnog svijeta pokušava zapamtiti što više glavnih značajki u stvarnom svijetu kako bi znao prepoznati udaljenosti i poziciju osobe i objekata u prostoru. Takvim praćenjem *ARKit* generira „mapu svijeta“ u kojem se korisnik nalazi. Također, *ARKit* pamti oblike

iz okoline te ako se uključi odsjaj na objektima može u odrazu prikazati objekte iz okoline koji se tamo nalaze.

4.1.6. Firebase

Prema [28], *Firebase* je platforma koja olakšava razvoj aplikacija. Omogućuje jednostavno stvaranje projekta za platforme poput *iOS-a*, *Androida*, *Weba*, *Unityja*, *Fluttera*. Putem *Firebasea* moguće je pratiti analitiku, stvoriti poslužiteljska računala za čuvanje korisničkih podataka, pratiti performance aplikacije, izdavanje inačica aplikacije, pratiti neželjene ispade aplikacije i druge stavke.

Firebase je alat koji programerima omogućava servise koje bi inače sami morali implementirati. Tim načinom programeri se mogu fokusirati da bi napravili najbolju moguću aplikaciju. Svi podaci se prenose na *cloud* gdje ih *Firebase* čuva na sigurnome. Iako je *Firebase* lagan za koristiti i dalje se u nekim slučajevima željeno ponašanje ne može postići s osnovnim paketom *Firebasea*, ali zato *Firebase* omogućuje backend developerima da ga mogu i doraditi [29].

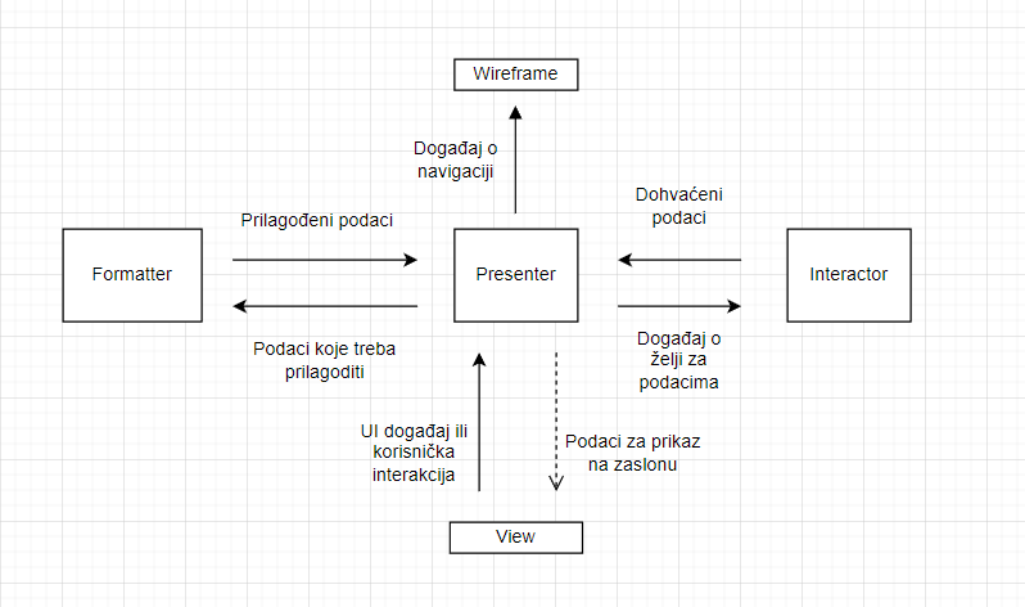
4.2. Programska arhitektura VIPER

4.2.1. Opis arhitekture VIPER

VIPER je predložak pisanja programskog koda i grupiranja datoteka unutar mapa radi lakšeg pronalaženja potrebnog dijela koda. *VIPER* postoji u nekoliko varijacija, ali svaka se arhitektura *VIPER* sastoji od pet glavnih dijelova *View, Interactor, Presenter, Entity, Router*. U varijaciji koja će biti opisana i korištena u radu slijede se isti principi s drugačijim nazivima, ali spomenut će se i dodatna varijacija koja u kompleksnijim datotekama razdvaja kod na smislenije cjeline.

Prema [30], *VIPER* rješava problem velikih *View Controllera* i omogućava veliku razinu testabilnosti uz razdvajanje koda u smislene cjeline. Svaka komponenta arhitekture VIPER ima točno određenu zadaću koju obavlja. *View* sadrži samo logiku vezanu uz UI, a podatke prima od strane *Presentera* te javlja *Presenteru* ako dođe do korisničke interakcije s UI-jem. *Interactor* sadrži logiku za dohvaćanje podataka, bilo to s API-ja ili lokalne podatke, a referenca na *Interactor* čuva se unutar *Presentera*. *Presneter* je srž arhitekture VIPER, jer povezuje sve komponente i sadržava reference na sve komponente, a obavlja svu logiku vezanu uz obradu događaja koji stižu s *Viewa* ili *Interactora*. *Entity* su modeli koji većinom sadrže samo informacije i koji se dohvaćaju u *Interactoru*. Posljednja komponenta standardne varijacije arhitekture VIPER je *Router* koji obavlja svu logiku vezanu uz navigaciju.

U slučaju mobilne računalne igre rada *Router* je preimenovan u *Wireframe*, ali postoji i jedna dodatna komponenta koja se može koristiti, a naziva se *Formatter*. *Formatter* služi kao dodatna komponenta kod kompleksnih zaslona gdje postoji nekolicina načina kako prikazati zaslon i sadržava logiku oko formatiranja podataka koje mu proslijedi *Presenter* u podatke koji su poznati *Viewu*. Potpuni način komunikacije vidljiv je na slici 4.1 [30].



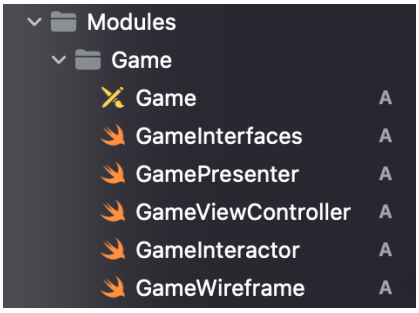
Slika 4.1. Prikaz komunikacije komponenti arhitekture VIPER [30]

Iz slike 4.1 vidljivo je da *Presenter* sadrži reference na ostale komponente, ali sadržava slabu referencu na *View* objekt, a to je potrebno jer bi inače *View* objekt imao jaku referencu na *Presenter* i *Presenter* bi imao jaku referencu na *View* što bi značilo da modul nikad ne bi izašao iz memorije.

Prema [31], slabe reference su samo pokazivači na objekte koji ne čuvaju objekt od oslobađanja iz memorije, dok jake reference ne dopuštaju da se objekt oslobodi iz memorije. Slabe reference potrebno je koristiti na mjestima gdje potencijalno dva objekta mogu imati jake reference jedan na drugog, što bi se dogodilo u slučaju kada bi u arhitekturi VIPER *Presenter* imao jaku referencu na *View*.

4.2.2. Primjena arhitekture VIPER

Arhitektura *VIPER* omogućava pregledniji i čitljiviji kod, ali i organiziranu strukturu datoteka unutar cijelog projekta. Mobilna računalna igra ovog rada pisana je s primjenom arhitekture koda *VIPER*, a na slici 4.2 vidljiva je struktura datoteka unutar jednog modula. Jedna komponenta aplikacije naziva se modulom.



Slika 4.2. Prikaz strukture datoteka unutar Game modula koristeći arhitekturu VIPER

Programski kod koji je vezan uz UI nalazi se u datoteci *GameViewController* unutar koje se prati korisnička interakcija sa zaslonom i šalju se u *GamePresenter*, ali se i na zaslon postavljaju podaci koji su dobiveni od strane *GamePresentera*. *GamePresenter* obrađuje događaje koji dolaze iz *GameViewControllera* te u suradnji s *GameInteractorom* priprema podatke koji su potrebi *GameViewControlleru* kako bi se prikazao pravilan sadržaj na zaslonu. *GameWireframe* ima dužnost navigacije nakon što mu *GamePresenter* javi da treba odvesti na drugi zaslon. Postoje dvije datoteke koje se nisu spominjale u prijašnjem potpoglavlju. Game datoteka je *Storyboard*. *Storyboard* su datoteke koje sadrže zaslon koji želimo prikazati s određenim *ViewControllerom*. *GameInterfaces* datoteka sadrži protokole putem kojih komuniciraju prije spomenute klase. Komuniciraju putem protokola da bi se stvarna implementacija sakrila od klase koja poziva neku od funkcionalnosti druge, ali i da bi se sakrile ostale funkcije za koje pozivajuća klasa ne treba znati [32].

4.3. Opis programskog rješenja klijentske strane

4.3.1. Programsko rješenje početnog zaslona

Početni zaslon služi kao spona između više zaslona, sam po sebi ne sadrži kompleksnu logiku. Sadržava pohranjivanje i dohvaćanje podataka o tome treba li prikazati zaslon s informacijama o mogućnosti promjene parametara na zaslonu s postavkama igre. Na slici 4.3 vidljiv je način kako funkcionira promjena stanja i treba li prikazati zaslon s informacijama. *DidSelect* parametar poziva metodu koju sadrži *InteractorInterface update(shouldShowSettingsInformation:).* Metoda prima jedan parametar koji je tipa *Bool* i označava je li pritisnut gumb ili nije te se suprotna vrijednost potom šalje metodi.

```
let didSelectCheckboxHandler: (Bool) -> Void = { [unowned interactor] in
    interactor.update(shouldShowSettingsInformation: !$0)
}
let checkboxConfigurator = CheckboxConfigurator(text: "Don't show this again", didSelect:
    didSelectCheckboxHandler)
```

Slika 4.3. Prikaz događaja nakon pritiska na gumb da se zaslon s informacijama ne prikazuje

Interactor sadržava instancu *UserStoragea* koju nasljeđuje protokol na slici 4.4 i tako Interactor može mijenjati vrijednost parametra *shouldShowSettingsInformation* svaki put kad se stisne na gumb. Protokol sadržava vrijednosti *gameSpeed* i *arenaSize* što su parametri igre koji utječu na brzinu zmije i veličinu prostora za igru

```
protocol DefaultsStorageInterface: AnyObject {
    var gameSpeed: GameSpeed { get set }
    var arenaSize: ArenaSize { get set }
    var shouldShowSettingsInformation: Bool { get set }
}
```

Slika 4.4. Prikaz protokola DefaultsStorageInterface

U slučaju kada korisnik stisne gumb za igranje, poziva se dohvaćanje trenutne vrijednosti varijable *shouldShowSettingsInformation* i odlučuje se treba li korisnika prikazati zaslon igre ili mu prikazati zaslon s informacijama. U slučaju da se prikazuje zaslon s informacijama, dodaje se vibracija kako bi se poboljšalo korisničko iskustvo jer se prikazao neočekivani zaslon što je vidljivo na slici 4.5.

```
func handle(playAction: Signal<Void>) {
    let navigationHandler: () -> Void = { [unowned self] in
        guard interactor.shouldShowSettingsInformation else {
            wireframe.navigateToGame()
            return
        }
        let input = makeSettingsInformationInput()
        UISelectionFeedbackGenerator().prepareAndGenerateFeedback()
        wireframe.navigateToSettingsInformation(input: input)
    }
    playAction
        .sink(receiveValue: navigationHandler)
        .store(in: &cancellables)
}
```

Slika 4.5. Odlučivanje što treba prikazati korisniku nakon pritiska na gumb za početak

Zaslon *PageSheet* koji sadrži informacije o postavljanju parametara na zaslonu s postavkama igre je zaslon koji će se koristiti i na zaslonu s postavkama. Moguće ga je koristiti gdje god je potrebno,

ali zna raditi samo s jednim tipom podatka koji je nazvan *ComponentConfigurable* i vidljiv na slici 4.6.

```
/// A common protocol for components that can be loaded through their configurators.

/// The configurator which implements the protocol needs to supply a concrete view for which it's used.
protocol ComponentConfigurable {
    var view: UIView { get }
}
```

Slika 4.6. Prikaz protokola ComponentConfigurable za stvaranje viewova

Na slici 4.6 vidljivo je da protokol sadržava samo jedan *property* nazvan *view* što omogućuje da kroz *Presenter* pošaljemo *Configurator* koji će naslijediti *ComponentConfigurable*. *PageSheetViewController* zna koji *view* treba prikazati u bilo kojem trenutku s obzirom na *ComponentConfigurable* koji dobije. Pretvorbu objekta koji nasljeđuje *ComponentConfigurable* u *UIView* koji možemo vidjeti na slici 4.7.

```
func handle(configurators: [ComponentConfigurable]) {
    let componentConfigurationHandler: (ComponentConfigurable) -> Void = { [unowned self] configurator in
        let view = configurator.view
        contentStackView.addArrangedSubview(view)
    }
    configurators.forEach(componentConfigurationHandler)
    var height: CGFloat = 0
    configurators.forEach { component in
        height += component.view.frame.height
    }
    let containerSize = containerView.systemLayoutSizeFitting(UIView.layoutFittingExpandedSize)
    // Needed for the custom transition, which uses the property to determine the height of the
        controller
    preferredContentSize = CGSize(width: containerSize.width, height: containerSize.height + height)
}
```

Slika 4.7. Prikaz stvaranja komponenti unutar PageSheetViewControllera

Na slici 4.7 vidljiv je izračun visine sadržaja unutar *ViewControllera* koji se pohranjuje u varijablu *preferredContentSize*, a razlog za to je prilagođeni način prikazivanja *ViewControllera*, koji je nativno dostupan tek od inačice iOS 15+, a rad podržava inačice iOS 13+. *PageSheetPresenter* i *PageSheetInteractor* ne sadržavaju nikakvu logiku nego samo prosljeđuju *ComponentConfigurable* objekte koji se pošalju kroz *PageSheetWireframe*.

4.3.2. Programsko rješenje zaslona postavki igre

Unutar postavki korisniku se omogućava promjena parametara igre koji su opisani u potpoglavlju 3.1. Da bi se čitali i mijenjali podaci iz lokalne baze potrebno je imati referencu na objekt koji nasljeđuje protokol sa slike 4.4 unutar *Interactora*. Nakon dolaska na zaslon potrebno je dohvatiti podatke o brzini zmije i veličini prostora za igru te pomoću kojih se postavljaju početne vrijednosti

u *CurrentValueRelaye. CurrentValueRelayi* koriste se u *PickerViewovima* koji omogućuju odabir drugih vrijednosti parametara što je vidljivo na slici 4.8 [32].

```
let selectedGameSpeedRelay = CurrentValueRelay<SelectedSetting>(handle(initialGameSpeed:
    interactor.gameSpeed))
let selectedArenaSizeRelay = CurrentValueRelay<SelectedSetting>(handle(initialArenaSize:
    interactor.arenaSize))
```

Slika 4.8. Postavljanje početnih vrijednosti u CurrentValueRelaye za odabir postavki igre [32]

PickerView moguće je koristiti na bilo kojem mjestu za biranje brzine zmije i veličine prostora za igranje. Odabir početne vrijednosti vidljiv je na slici 4.9 gdje se prolazi kroz sve moguće vrijednosti parametara i dodjeljuje im se vrijednost gumba.

```
func handle(initialGameSpeed: GameSpeed) -> SelectedSetting {
    switch initialGameSpeed {
    case .slow:
        return .first
    case .normal:
        return .second
    case .fast:
        return .third
    }
}

func handle(initialArenaSize: ArenaSize) -> SelectedSetting {
    switch initialArenaSize {
    case .small:
        return .first
    case .medium:
        return .second
    case .large:
        return .third
    }
}
```

Slika 4.9. Postavljanje početnih vrijednosti prema parametrima igre

PickerView zna raditi s vrijednošću gumba, odnosno *SelectedSetting* objektom. Funkcija *handle(selectedButton:selectedButtonChangeRelay: isInitialSetup:)* omogućuje preko parametara poznavanje koji je gumb trenutno odabran. Nakon što obavi promjenu boje i animaciju gumba u novo stanje, šalje vrijednost novog pritisnutog gumba kroz *selectedButtonChangeRelay* što je vidljivo na slici 4.10.

```
func handle(
    themeChangeUsing selectedButton: SelectedButton,
    selectedButtonChangeRelay: CurrentValueRelay<SelectedButton>,
    isInitialSetup: Bool = false
) {
    switch selectedButton {
    case .first:
        firstButton.change(theme: .primary, shouldAnimate: true)
        secondButton.change(theme: .secondary)
        thirdButton.change(theme: .secondary)
    case .second:
        firstButton.change(theme: .secondary)
        secondButton.change(theme: .primary, shouldAnimate: true)
        thirdButton.change(theme: .secondary)
    case .third:
        firstButton.change(theme: .secondary)
        secondButton.change(theme: .secondary)
        thirdButton.change(theme: .primary, shouldAnimate: true)
    }
    guard !isInitialSetup else { return }
    selectedButtonRelay.accept(selectedButton)
}
```

Slika 4.10. Odabir koji gumb treba prikazati kao odabran i slanje vrijednosti kroz Relay

Unutar *SettingsPresentera* prate se vrijednosti *selectedButtonChangeRelaya* i pretvaraju se vrijednosti iz *SelectedSettinga* u objekt parametra koji je promijenjen. Nakon promjene parametra, u *Interactoru* poziva se metoda koja osvježava vrijednost promijenjenog parametra. Na slici 4.11 prikazan je primjer funkcije *handle(gameSpeed:)* koja osluškuje promjene parametra brzine zmije, ali na isti način radi i funkcija *handle(arenaSize:)* koja osluškuje promjenu parametra veličine prostora za igru.

```
func handle(gameSpeed: Driver<SelectedSetting>) {
    let updateGameSpeedHandler: (SelectedSetting) -> Void = { [unowned interactor] setting in
        let gameSpeed: GameSpeed
        switch setting {
        case .first:
            gameSpeed = .slow
        case .second:
            gameSpeed = .normal
        case .third:
            gameSpeed = .fast
        }
        interactor.update(gameSpeed: gameSpeed)
    }
    gameSpeed
        .sink(receiveValue: updateGameSpeedHandler)
        .store(in: &cancellables)
}
```

Slika 4.11. Pretvaranje odabranog gumba u brzinu zmije

Uz navedene mogućnosti zaslona s postavkama igre, moguće je provjeriti pravila igre gdje se ponovno koristi *PageSheet* modul. *PageSheet* prikazuje se na jednak način kao i na početnom zaslonu samo s drugim sadržajem.

4.3.3. Programsko rješenje zaslona ljestvice najboljih rezultata

Zaslon ljestvice s najboljim rezultatima omogućuje dohvaćanje najboljih rezultata iz *Firebase* baze podataka. Dolaskom na zaslon ljestvice s najboljim rezultatima poziva se funkcija *makeSections()* koja preko *Interactora* pokušava dohvatiti najbolje rezultate. Ako su rezultati prazni, šalje se prazan niz prema *HighscoresViewControlleru*, a ako postoje, transformiraju se u objekte s kojim *ViewController* zna raditi. Poziv dohvaćanja rezultata i stvaranje objekata vidljiv je na slici 4.12.

```
func makeSections() -> Driver<[TableSectionItem]> {
    let userItemBuilder: ([UserModel]) -> [TableSectionItem] = { userData in
        guard !userData.isEmpty else { return [] }
        let items = userData.map(HighscoreTableCellItem.init)
        return [BlankTableSection(items: items)]
    }
    return interactor
        .getHighscores()
        .map(userItemBuilder)
        .handleLoadingAndError(with: wireframe)
        .asDriverOnErrorComplete()
}
```

Slika 4.12. Dohvaćanje i stvaranje niza objekata za prikazivanje na HighscoresViewControlleru

HighscoresInteractor sadrži objekt koji nasljeđuje protokol *HighscoreServing*. Objekt koji nasljeđuje protokol *HighscoreServing* mora imati implementirane funkcije kojima može dohvatiti rezultate iz *FirebaseFirestore* baze. Protokol služi kao fasada za skrivanje stvarne implementacije i prikazan je na slici 4.13.

```
protocol HighscoreServing {
    func save(userModel: UserModel) -> AnyPublisher<Void, Error>
    func getHighscores() -> AnyPublisher<[UserModel], Error>
}
```

Slika 4.13. Prikaz HighscoreServing protokola koji omogućuje dohvaćanje i spremanje podataka

Dohvaćanje najboljih rezultata prikazano je na slici 4.14. Korišten je način *async/await* kako bi se asinkrono odvijalo dohvaćanje podataka. Metodom *.createAsync* kod se pretvorio u *Combine Publisher* što omogućuje daljnje korištenje Combinea nad *async/awaitom*. Prvo, potrebno je dohvatiti pravilnu zbirku s *FirebaseFirestorea* koja se nalazi pod nazivom “players” gdje se dohvaćaju svi dokumenti i nakon toga pretvaraju u *UserModele*.

```swift
func getHighscores() -> AnyPublisher<[UserModel], Error>  {
    let userModelBuilder: (QueryDocumentSnapshot) -> UserModel = { document in
        guard let userModel = try? UserModel(dictionary: document.data()) else {
            return .empty
        }
        return userModel
    }
    return AnyPublisher.createAsync { [unowned self] in
        return try await withCheckedThrowingContinuation({ continuation in
            let query = firestore
                .collection(Constants.HighscoreService.players)
                .order(by: Constants.HighscoreService.score, descending: true)
            query.getDocuments { data, error in
                guard error.isNil, let data = data else {
                    if let error = error {
                        continuation.resume(throwing: error)
                    } else {
                        continuation.resume(returning: [])
                    }
                    return
                }
                let userData = data
                    .documents
                    .map(userModelBuilder)
                continuation.resume(returning: userData)
            }
        })
    }
}
```

Slika 4.14. Dohvaćanja podataka s FirebaseFirestorea unutar HighscoreServicea

U slučaju da *HighscoresViewController* dobije prazan niz, postavlja se zaslon koji prikazuje da trenutno nema rezultata pohranjenih u bazi podataka. U slučaju da niz nije prazan, prikazuju se ćelije s podacima o korisnicima i njihovim rezultatima, a implementacije je vidljiva na slici 4.15 [33].

```swift
func handle(sections: Driver<[TableSectionItem]>) {
    let placeholderHandler: ([TableSectionItem]?) -> Void = { [unowned self] in
        highscorePlaceholderView.isHiddenAnimated = !($0?.isEmpty ?? true)
    }
    sections
        .compactMap { $0 }
        .handleEvents(receiveOutput: placeholderHandler)
        .assignWeakified(on: dataSource, at: \.sections)
        .store(in: &cancellables)
}
```

Slika 4.15. Prikazivanja zaslona bez rezultata i skrivanja istog [33]

4.3.4. Programsko rješenje zaslona na kojem se igra

Pri dolasku na zaslon na kojem se igra, korisnika se zahtijeva dozvola korištenja kamere jer *Apple* zahtijeva korisnikovo dopuštenje za korištenje kamere. Nakon što korisnik dozvoli korištenje kamere, započinje učitavanje prostora. Korisnik pomiče kameru kako bi učitao što više prostora, a kad uređaj prepozna sa sigurnošću da se radi o horizontalnoj podlozi, postavlja prostor za igranje. Nakon što se korisnik odluči za određeno mjesto na horizontalnoj podlozi, dodirom po zaslonu

započinje igru. Otkrivanje horizontalnih površina odvija se u *GameViewControlleru.* Učitavanje horizontalne površine kroz ARKit vidljivo je na slici 4.16.

```
func handle(willRenderScene: Signal<SCNScene>) -> Signal<ARHitTestResult?> {
    let hitTestResultBuilder: (SCNScene) -> ARHitTestResult? = { [unowned self] scene in
        let hit = sceneView.hitTest(view.center, types: .existingPlane)
        sceneView.removeAllChildNodes()
        guard ((hit.first?.anchor as? ARPlaneAnchor) != nil), let firstResult = hit.first else { return nil }
        return firstResult
    }
    return willRenderScene
        .map(hitTestResultBuilder)
        .asSignal()
}
```

Slika 4.16. Prikaz učitavanja horizontalne površine kroz ARKit

Pri svakom učitavanju provjerava se dodiruje li središnja točka zaslona horizontalnu površinu jer želimo pomicanje prividnog prostora za igru na horizontalnoj površini kad korisnik pomiče kameru. Unutar *GamePresentera* nalazi se poziv funkcije iz *Interactora* za stvaranje prostora za igranje koja je vidljiva na slici 4.17. U ovom trenutku, prostor za igranje služi samo kao prikaz gdje će se nalaziti prostor [34].

```
func handle(willRenderScene: Signal<ARHitTestResult?>, startGameAction: Signal<Void>) -> Driver<SCNNode?> {
    let arenaNodeBuilder: (ARHitTestResult?) -> SCNNode? = { [unowned self] hitResult in
        guard let hitResult = hitResult else { return nil }
        let position = hitResult.worldTransform.columns.3
        return interactor.makeArena(with: SCNVector3(position.x, position.y, position.z), isStart: true, time: nil)
    }
    return willRenderScene
        .prefix(untilOutputFrom: startGameAction)
        .map(arenaNodeBuilder)
        .asDriver()
}
```

Slika 4.17. Stvaranje prostora za igru [34]

Prostor za igru se stacionira tek nakon što korisnik dodirne zaslon dok je prostor za igru prikazan na zaslonu što je vidljivo na slici 4.18. Odvija se tako da instanca *Timera* odbrojava od tri do nula i nakon toga stvara se prostor za igru s postavljenim brojilom bodova.

```
func handle(
    startGameAction: Signal<Void>,
    arenaNode: Driver<SCNNode?>,
    gameStartedRelay: PassthroughRelay<Void>
) -> Driver<SCNNode?> {
    let timer = makeStartGameTimer()

    let arenaCountdownBuilder: (Int, SCNNode) -> SCNNode = { [unowned self] time, arenaNode in
        guard time > 0 else {
            return interactor.makeGameStartedNode(using: arenaNode.position)
        }
        return interactor.makeArena(with: arenaNode.position, isStart: false, time: time)
    }

    return startGameAction
        .prefix(untilOutputFrom: gameStartedRelay)
        .map(to: timer)
        .switchToLatest()
        .withLatestFrom(arenaNode.compactMap { ($0) }) { ($0, $1) }
        .handleEvents(receiveOutput: { _ in gameStartedRelay.accept(()) })
        .map(arenaCountdownBuilder)
        .asDriver()
}
```

Slika 4.18. Stvaranje stacioniranog prostora za igranje uz odbrojavanje

Stvaranje čvorova koji se postavljaju kao objekti u prošireni prostor nalazi se unutar klase *GameManager*. *GameManager* ima zadaću stvaranja glavnog čvora u koji se postavljaju svi ostali čvorovi, a to je čvor prostora za igru. Hijerarhija je vidljiva na slici 4.19 gdje se postavljaju čvorovi zmije i novčića kao podčvorovi čvora prostora za igru. Također, *GameManager* je zadužen za držanje referenci na postavljene objekte u prostoru kako bi se iz *GamePresentera* moglo pristupiti i učiniti s njima štogod je potrebno.

```
func makeGameStartedNode(using arenaNodePosition: SCNVector3) -> SCNNode {
    let arenaCenterNode = makeArenaWithWalls(for: arenaNodePosition)
    guard let snakeNode = makeSnakeNode(position: arenaNodePosition),
          let foodNode = makeFoodNode()
    else { return SCNNode() }

    arenaCenterNode.addChildNode(snakeNode)
    arenaCenterNode.addChildNode(foodNode)
    updateScoreNode(currentScore: 0)

    return arenaCenterNode
}
```

Slika 4.19. Postavljanje početnih čvorova potrebnih za igru

Čvorovi prostora za igru napravljeni su kao statične točke koje služe samo kao prepreka da zmija ne može izaći izvan prostora. *SnakeNode* je čvor koji predstavlja zmiju kojom korisnik upravlja. *SnakeNode* sadrži funkcije za skretanje, kretanje i rast, a sastoji se od više *SnakeSegmentNodeova.*

Svaki *SnakeSegmentNode* je čvor koji predstavlja glavu ili tijelo zmije. Na slici 4.20 prikazane su funkcije koje sadrži *SnakeNode.*

```
public func turnRight() {
    let value = (snake.moveDirection.rawValue - 1 + 4) % 4
    guard let direction = Direction(rawValue: value) else { return }
    snake.moveDirection = direction
    rotateHeadNode(direction: .right)
}

public func move() {
    guard let snakeHead = snake.headPosition else { return }
    var newBody = [SIMD2<Float32>(x: snakeHead.x + snake.moveDirection.float32.x, y: snakeHead.y + snake.moveDirection.float32.y)]
    lastBodySegment = snake.body.removeLast()
    newBody.append(contentsOf: snake.body)
    snake.body = newBody
    updateNodes()
}

public func grow() {
    guard let lastBodySegment = lastBodySegment, let lastNode = bodyNodes.last else { return }
    snake.body += [lastBodySegment]
    let newNode = createNewNode(position: lastBodySegment, lastNodeDirection: lastNode.direction)
    bodyNodes += [newNode]
    addChildNode(newNode)
    updateNodes()
}
```

Slika 4.20. Funkcionalnosti SnakeNodea

Potrebno je dodati da postoji i funkcija *turnLeft()* koja funkcionira na isti način kao i *turnRight()* samo drugačije računa promjenu smjera. Funkcija *move()* zapravo pomiče dijelove tijela zmije tako da joj govori sljedeću poziciju na kojoj se treba nalaziti i briše zadnju poziciju na kojoj se više ne nalazi. Posljednja funkcija, grow(), kao i što samo ime naznačuje, povećava veličinu zmije kad korisnik pokupi novčić. Postavlja novi čvor na mjesto posljednjega čvora zmije kako bi se dobio izgled kao da se novi čvor pojavio na kraju zmije. *UpdateNodes()* funkcija poziva se nakon povećanja ili pomicanja zmije, a ona zapravo postavlja naredbu za animaciju pokreta zmije u prostoru što je vidljivo na slici 4.21.

```
func updateNodes() {
    bodyNodes = bodyNodes.sorted()
    bodyNodes
        .enumerated()
        .forEach { index, node in
            let postition = snake.body[index]
            node.position = SCNVector3(Float(postition.x), snake.arenaPosition.y, Float(postition.y))
            let moveAction = SCNAction.move(to: SCNVector3(Float(postition.x), snake.arenaPosition.y, Float(postition.y)), duration: 0.1)
            node.runAction(moveAction)
        }
}
```

Slika 4.21. Kretanje zmije

GameManager zadužen je i za stvaranje čvorova hrane, a to se odvija u funkciji *makeFoodNode()* koja vraća opcionalni čvor hrane. Prvotno, potrebno je uzeti veličinu prostora za igranje koji je pohranjen u lokalnu bazu da bi se znao prostor u koji se trebaju postaviti čvorovi hrane. Računanje

pozicije na koju se može postaviti čvor hrane vidljiva je na slici 4.22. Potrebno je dohvatiti nasumičnu vrijednost unutar prostora za igru koja se ne nalazi na udaljenosti od zmije od pet centimetara. Petlja se ponavlja dok se ne dobije pozicija hrane koja ima ispravne parametre.

```
func makeFoodNode() -> FoodNode? {
    guard let arenaNode = arenaNode, let snakeNode = snakeNode else { return nil }
    let arenaWallDistanceFromCenter = userStorage.arenaSize.wallDistance - 0.05
    var foodNode: FoodNode
    repeat {
        let x = Float.random(in: (arenaNode.position.x - arenaWallDistanceFromCenter) ... (arenaNode.position.x +
            arenaWallDistanceFromCenter))
        let z = Float.random(in: (arenaNode.position.z - arenaWallDistanceFromCenter) ... (arenaNode.position.z +
            arenaWallDistanceFromCenter))
        foodNode = FoodNode()
        foodNode.position = SCNVector3(x, arenaNode.position.y, z)
    } while snakeNode.bodyNodes.contains(where: { segmentNode -> Bool in
        return (segmentNode.worldPosition.x - 0.05 ... segmentNode.worldPosition.x + 0.05).contains(foodNode.position.x)
        && (segmentNode.worldPosition.z - 0.05 ... segmentNode.worldPosition.z + 0.05).contains(foodNode.position.z)
    })
    foodNode.runAction(SCNAction.rotation(time: 5))
    self.foodNode = foodNode
    return foodNode
}
```

Slika 4.22. Stvaranje čvorova hrane

Jedna od najbitnijih komponenti igre su i sudari objekata u proširenom prostoru, a oni se osluškuju u *GamePresenteru* koji na svaki sudar poziva funkciju za provjeru koji se sudar dogodio. Funkcija *handle(didBeginContact:foodNodeRelay:gameOverRelay:)* unutar sebe poziva funkcije koje provjeravaju koji se objekt s kojim sudario, a to se odvija provjerom *categoryBitMaska.* Funkcija je vidljiva na slici 4.23.

```
func handle(
    didBeginContact: Signal<SCNPhysicsContact>,
    foodNodeRelay: PassthroughRelay<FoodNode?>,
    gameOverRelay: PassthroughRelay<Void>
) {
    didBeginContact
        .removeDuplicates()
        .throttle(for: .seconds(0.1), scheduler: RunLoop.main, latest: false)
        .sink(receiveValue: { [unowned self] in handle(contact: $0, foodNodeRelay: foodNodeRelay, gameOverRelay: gameOverRelay) })
        .store(in: &cancellables)
}
```

Slika 4.23. Pozivanje funkcije za provjeru sudara nakon što se dogodio

Svakom objektu koji želimo prepoznati u sudarima moramo postaviti *categoryBitMask* te maske onih objekata s kojim želimo provjeravati. U slučaju ovog rada, potrebne su provjere sudara zmije i hrane, glave zmije i tijela zmije te zmije i zida [35]. Na slici 4.24 prikazane su provjere.

```
func handle(didSnakeMakeContactWithFoodUsing contact: SCNPhysicsContact) -> Bool {
    return contact.nodeA.physicsBody?.categoryBitMask == ContactCategory.food.rawValue
        || contact.nodeB.physicsBody?.categoryBitMask == ContactCategory.food.rawValue
}

func handle(didSnakeMakeContactWithBodyUsing contact: SCNPhysicsContact) -> Bool {
    return (contact.nodeA.physicsBody?.categoryBitMask == ContactCategory.snakeHead.rawValue
        && contact.nodeB.physicsBody?.categoryBitMask == ContactCategory.snake.rawValue)
        || (contact.nodeA.physicsBody?.categoryBitMask == ContactCategory.snake.rawValue
            && contact.nodeB.physicsBody?.categoryBitMask == ContactCategory.snakeHead.rawValue)
}

func handle(didSnakeMakeContactWithArenaUsing contact: SCNPhysicsContact) -> Bool {
    return contact.nodeA.physicsBody?.categoryBitMask == ContactCategory.arena.rawValue
        || contact.nodeB.physicsBody?.categoryBitMask == ContactCategory.arena.rawValue
}
```

Slika 4.24. Provjera objekata koji su se sudarili

4.3.5. Programsko rješenje pohrane rezultata

Mogućnost pohrane podataka omogućuje se korisniku nakon što završi igru. Korisniku se prikazuje zaslon *UserFormViewController* koji mu omogućuje unos nužnih i neobveznih podataka koje želi pohraniti u bazu podataka. Pri dolasku na zaslon, korisniku je onemogućen pritisak na gumb za pohranu podataka dok ne unese nadimak koji je obvezan. Način promjene stanja gumba za pohranu podataka vidljiv je na slici 4.25.

```
func handle(isSaveActionEnabledUsing userModel: Driver<UserModel>) -> Driver<Bool> {
    userModel
        .map { $0.nickname.isNotBlank }
        .asDriver()
}
```

Slika 4.25. Omogućavanje gumba za pohranu podataka

Nakon što je korisniku omogućena pohrana podataka, pritiskom na gumb za pohranu odvija se pohrana podataka preko *UserFormInteractora* koji u sebi sadržava instancu *HighscoreServinga*. *HighscoreServing* protokol omogućuje pohranu i čitanje podataka iz *FirebaseFirestoragea* kao što je to ranije navedeno. Funkcija *handle(saveAction:userModel:)* prima dodire na gumb za pohranu podataka i nakon dodira poziva funkciju za pohranu podataka koja se nalazi u *Interactoru.* Prikaz događaja prilikom pohrane podataka vidljiv je na slici 4.26.

```
func handle(saveAction: Signal<Void>, userModel: Driver<UserModel>) {
    let saveUserResultRequestHandler: (UserModel) -> Driver<Void> = { [unowned self] in
        interactor
            .save(userModel: $0)
            .handleLoadingAndError(with: wireframe)
            .asDriverOnErrorComplete()
    }
    saveAction
        .withLatestFrom(userModel)
        .flatMap(saveUserResultRequestHandler)
        .sink(receiveValue: { [unowned wireframe] in wireframe.navigateToResult(userAction: .done) })
        .store(in: &cancellables)
}
```

Slika 4.26. Pritiska gumba za pohranu podataka

Na slici 4.27 prikazana je pohrana podataka o postignutom rezultatu koja se nalazi unutar *HighscoreService* klase. Također, kao i kod čitanja podataka, korišten je *async/await* da bi se asinkrono odvijale radnje i odmah je pretvoreno u *Combine Publisher* kako bi se zadržala reaktivna načela koja su korištena u radu. Prvotno, potrebno je napraviti *JSON* objekt od *UserModela* da bi instanca *FirebaseFirestorea* znala kako pohraniti podatke, a nakon toga provjerimo postoji li ikakva greška i ovisno o tome vratimo rezultat ili grešku.

```
func save(userModel: UserModel) -> AnyPublisher<Void, Error> {
    AnyPublisher.createAsync { [unowned self] in
        try await withCheckedThrowingContinuation { continuation in
            guard let data = userModel.JSONObject else {
                continuation.resume(throwing: SnakeError.default)
                return
            }

            let completionHandler: (Error?) -> Void = { error in
                guard let error = error else {
                    continuation.resume(returning: ())
                    return
                }
                continuation.resume(throwing: error)
            }
            firestore
                .collection(Constants.HighscoreService.players)
                .document()
                .setData(data, completion: completionHandler)
        }
    }
}
```

Slika 4.27. Pohrana rezultata unutar HighscoresServicea

4.3.6. Programsko rješenje pohrane podataka

Podaci se pohranjuju u dvije različite baze podataka, jedan je lokalna, a druga je udaljena o kojima će biti više govora u potpoglavlju 4.4. Podaci koji se pohranjuju u udaljenu bazu pohranjuju se modelom *UserModel* preko *HighscoresServicea* kao što je opisano u ranijim potpoglavljima. Pristupna točka za *FirebaseFirestore* je *HighscoreService*, a za lokalnu bazu je *UserStorage.* Kako

bi se lakše implementirala pohrana u lokalnu bazu napravljen je *property omotač UserDefault* koji je vidljiv na slici 4.28.

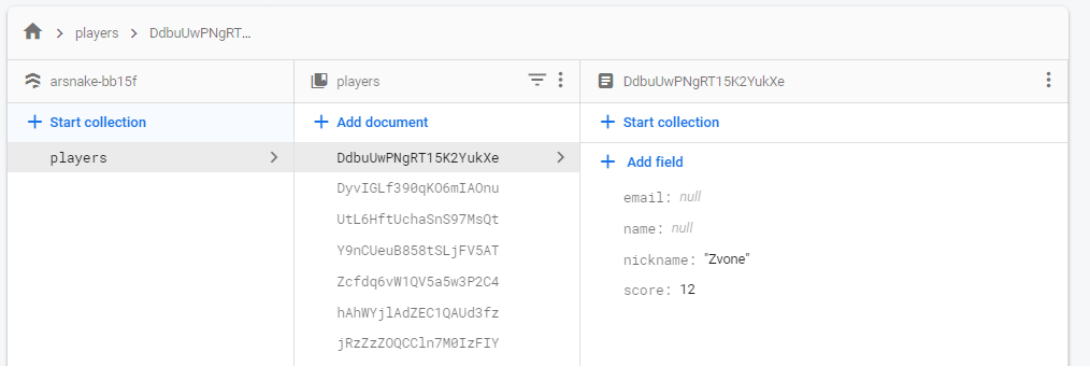
```
@propertyWrapper
final class UserDefault<Value> where Value: Codable {
    let key: UserStorage.DefaultsKeys
    let defaultValue: Value
    let container: UserDefaults

    init(_ key: UserStorage.DefaultsKeys, defaultValue: Value, userDefaults: UserDefaults = .standard) {
        self.key = key
        self.defaultValue = defaultValue
        self.container = userDefaults
    }

    var wrappedValue: Value {
        get { container.value(ofType: Value.self, forKey: key) ?? defaultValue }
        set { container.set(value: newValue, forKey: key) }
    }
}
```

Slika 4.28. Property omotač UserDefault

Omotačem *Property* dobiva se mogućnost dodavanja *annotationa* iznad vrijednosti koji će obraditi vrijednost i pohraniti je u lokalnu bazu podataka.

4.4. Opis programskog rješenja poslužiteljske strane

Za rješavanje problema pohrane podataka korištena je platforma *Firebase* koja ima mogućnost korištenja besplatne pohrane podataka. Korištena je *Firestore Database* funkcionalnost koja olakšava način pretraživanja, postavljanja, dohvaćanja i čitanja podataka. Struktura podataka o rezultatima korisnika objašnjena je u trećem poglavlju ovog rada, a na slici 4.29 vidljiv je način pohranjivanja podataka po zbirkama i dokumentima.



Slika 4.29. Prikaz pohranjenih podataka na platformi Firebase

Zbirke predstavljaju jednu pristupnu točku s grupom datoteka. U slučaju ovog rada, svaka datoteka je korištena kao zaseban rezultat koji se pohranjivao pod automatski generiranim ID-jem od strane

Firebasea. Pri odabiru nekog od dokumenata vidljivi su podaci koji su pohranjeni u tom dokumentu.

U slučaju lokalnog pohranjivanja podataka, kao što je spomenuto ranije, korišten je *UserDefaults* koji je *Appleov* API za pohranu podataka na uređaj. U slučaju da se aplikacija obriše s uređaja nestaju i pohranjeni podaci. *UserDefaults* bez ikakvih dodataka omogućuje pohranu primarnih podataka iz tog razloga brzina zmije i veličina prostora za igranje su pohranjeni kao *String* tipovi podataka, a za prikaz zaslona s informacijama prije pokretanja igre je odabran tip podatka *Bool.* Pohranom podataka u lokalnu bazu dobije se na brzini dohvaćanja podataka i na tome što nije potrebna internetska mreža. Na slici 4.30 prikazan je i način korištenja *property* omotača *UserDefault* koji je omogućio čitljivije spremanje podataka u lokalnu bazu

```
final class UserStorage {

    // MARK: - Singleton -

    static let instance: DefaultsStorageInterface = UserStorage()
    private init() { }

    // MARK: - Properties -

    // MARK: - User defaults storage

    @UserDefault(.gameSpeed, defaultValue: Constants.GameSpeed.normal)
    var gameSpeedValue: String

    @UserDefault(.arenaSize, defaultValue: Constants.ArenaSize.medium)
    var arenaSizeValue: String

    @UserDefault(.shouldShowSettingsInformation, defaultValue: true)
    var shouldShowSettingsInformation: Bool
}
```

Slika 4.30. Prikaz podataka pohranjenih u lokalnu bazu UserDefaults

5. KORIŠTENJE I ISPITIVANJE MOBILNE RAČUNALNE IGRE

U ovom poglavlju bit će opisani načini korištenja mobilne računalne igre uz provjeru jesu li funkcionalnost pravilno implementirane. Također, opisan je utjecaj arhitekture VIPER, analiza korisničkog iskustva i moguća poboljšanja na temelju provedene analize korisničkog iskustva.

5.1. Načini korištenja mobilne računalne igre

Ostvarena mobilna računalna igra može se podijeliti na četiri cjeline, prije početka igre, pregled najboljih rezultata, igra i završetak igre.

Prva cjelina, prije početka igre, sastoji se od dva zaslona, početnog zaslona i zaslona s postavkama. Na početnom zaslonu korisnik može otići na zaslon s najboljim rezultatima, započeti igru i otići na zaslon s postavkama. U slučaju da korisnik prvi put ili da još nije odabrao opciju da se zaslon s informacijama ne prikazuje, svaki puta kada odabere gumb za početak igre prikazuje se zaslon s informacijama. Na zaslonu s informacijama korisnika se obavještava kako može promijeniti parametre igre na zaslonu s postavkama i može odmah s prikazanog zaslona otići na zaslon s postavkama, a može ignorirati poruku i nastaviti dalje na igru. Također, korisnik može s početnog zaslona otići na zaslon s postavkama gdje odabire parametre igre. Parametri igre su, kao što je ranije navedeno, brzina zmije i veličina prostora za igru. Parametri utječu na količinu bodova koja se dobije pri sakupljanju novčića tijekom igre. U slučaju da korisnik nije upoznat s pravilima igre i načinom igranja, može pritisnuti gumb za pravila igre koji mu prikazuje zaslon s ispisanim pravilima. Ako se korisnik odlučio na odlazak na zaslon s najboljim rezultatima tako je došao do druge cjeline.

Druga cjelina sastoji se od jednog zaslona s dva stanja. Prvo stanje je stanje prazne baze podataka kad se prikazuje zaslon s natpisom da ne postoje pohranjeni rezultati, a drugo stanje je stanje kad postoje podaci u bazi i prikazuju se ćelije u listi s podacima.

Treća cjelina je početak igre. Kad korisnik započne igru mora pronaći horizontalnu površinu u prostoru gdje želi postaviti prostor za igru. Nakon pronalaska površine, korisnik dodirom po zaslonu postavlja prostor za igru i započinje ju. Cilj korisnika tijekom igre je skupiti što više novčića prije nego što igra završi po pravilima koja su navedena u ranijim poglavljima. Nakon što korisnik završi igru, dolazi do četvrte cjeline završetak igre.

Završetak igre sastoji se od jednog zaslona na kojem korisnik može, ali ne mora, ostaviti podatke u obliku nadimka, imena i e-pošte, Kako bi pohranio rezultat i da bi rezultat bio vidljiv na listi najboljih rezultata korisnik mora pohraniti rezultat. Ako korisnik odluči pohraniti rezultat, nakon pohrane korisnika se odvodi na zaslon s najboljim rezultatima. U slučaju da se korisnik ipak odluči ne pohraniti rezultat, odvodi se natrag na početni zaslon.

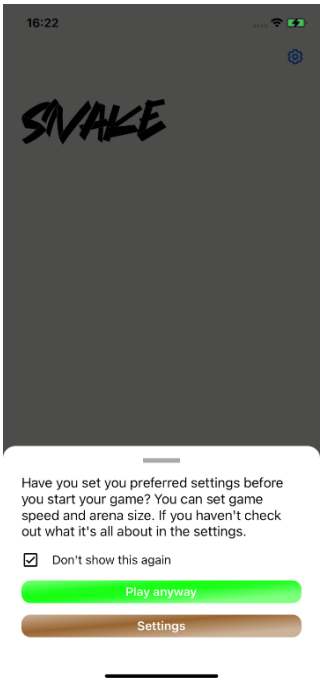
5.2. Ispitivanje rada mobilne računalne igre

5.2.1. Ispitivanje rada početnog zaslona

Ulaskom u aplikaciju otvoren je početni zaslon s tri mogućnosti, otvaranjem liste najboljih rezultata, početkom igre i otvaranjem zaslona s postavkama igre. Početni zaslon prikazan je na slici 5.1 iz čega je vidljivo da postoje mogućnosti odlaska na zaslon s postavkama, početak igre i na zaslon s ljestvicom najboljih rezultata.

Slika 5.1. Prikaz početnog zaslona

Odabirom gumba “Play” prikazuje se zaslon preko dijela zaslona na kojem je moguće pokrenuti igru, odabrati da se ne prikazuje više i otići na zaslon s postavkama. Odabirom da se ne prikazuje više i ponovnim pritiskom na gumb “Play” zaslon se nije prikazao nego je započela igra. Zaslon s informacijama prikazan je na slici 5.2.

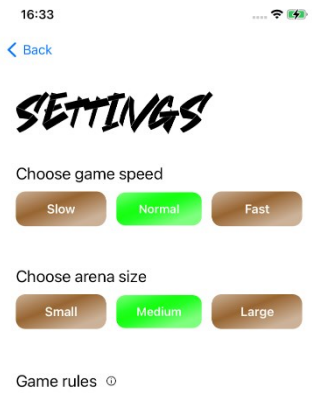


Slika 5.2. Prikaz zaslona s informacijama na početnom zaslonu

Nakon prvotnog prikazivanja i odabira da se zaslon s informacijama više ne prikazuje dokazalo se očekivano ponašanje mobilne računalne igre.

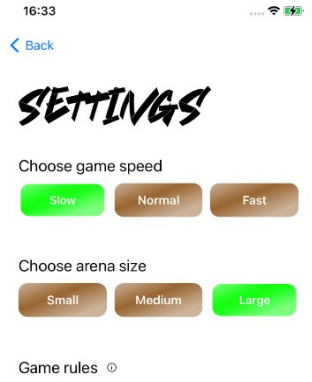
5.2.2. Ispitivanje rada zaslona s postavkama

Ulaskom na zaslon s postavkama igre, koji je prikazan na slici 5.3, odabrani parametri igre su srednji parametri, ali korisnik ima mogućnost odabrati druge parametre. Kako je navedeno u četvrtom poglavlju, početne vrijednosti su srednji parametri što znači da je ispitivanje opisalo očekivano ponašanje.



Slika 5.3. Prikaz zaslona s postavkama igre

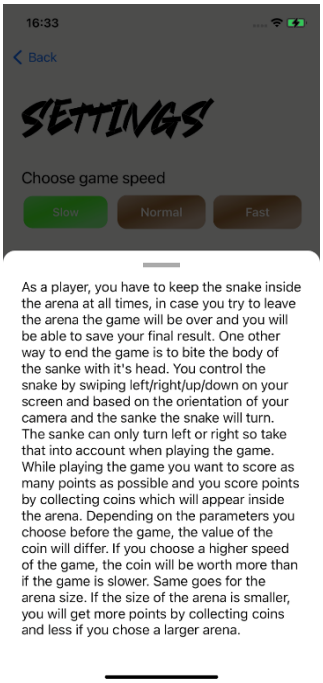
Promjenom brzine zmije na „sporo“ i veličine područja za igranje na „veliko“ te je izlaskom i ulaskom na zaslon ispitano pravilno funkcioniranje lokalne podataka i zaslona s postavkama igre. Zaslon nakon promjene parametara vidljiv je na slici 5.4.



Slika 5.4. Prikaz zaslona s postavkama nakon promjene parametara

Kako je zaslon nakon ponovnog povratka imao odabrane vrijednosti nakon promjene parametara možemo zaključiti da pohrana i dohvaćanje pohranjenih parametara radi kako je i očekivano.

Ako se korisnik želi prisjetiti pravila igre, pritiskom na gumb pored “Game rules” prikazuje mu se zaslon s pravilima koji je vidljiv na slici 5.5.

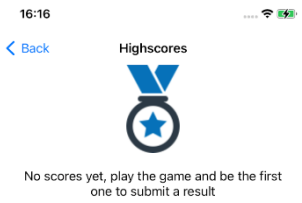


Slika 5.5. Prikaz zaslona s pravilima i načinima igre

Kao što je vidljivo na slici 5.5 zaslon s pravilima igre se prikazao i ispisana su pravila što je očekivano ponašanje.

5.2.3. Ispitivanje rada ljestvice najboljih rezultata

U slučaju da nema pohranjenih rezultata u bazi podataka *FirebaseFirestore*, prikazuje se zaslon na kojem se opisuje kako nema rezultata, a vidljiv je na slici 5.6.



Slika 5.6. Prikaz zaslona ljestvice najboljih rezultata bez rezultata

Ako korisnik odigra igru i pohrani rezultat, prikazuje mu se zaslon na kojem se nalaze i rezultati, a zaslon je prikazan na slici 5.7.

Slika 5.7. Prikaz zaslona ljestvice najboljih rezultata

Kao što je vidljivo sa slika 5.6 i 5.7, prikazane su različite informacije na zaslonu ovisno o postojanju pohranjenih podataka u bazi što je očekivano ponašanje mobilne računalne igre.

5.2.4. Ispitivanje rada zaslona igre

Tijekom ispitivanja igre potrebno je provjeriti postavlja li se prostor za igru i dodirom na zaslon stacioniranje istog. Također, potrebno je provjeriti i brzinu kretanja zmije te veličinu prostora za igru za različite odabrane parametre na zaslonu s postavkama. Zaslon nakon pravilnog postavljanja igre prikazan je na slici 5.8.

Slika 5.8. Prikaz zaslona igre nakon što je korisnik započeo igru

Tijekom igranja potrebno je provjeriti povećava li se zmija nakon svakog pokupljenog novčića. Također, potrebno je provjeriti računanje bodova nakon svakog pokupljenog novčića. Svaki novčić ima jednaku vrijednost, ovisno o odabranom parametru igre. Potrebno je provjeriti i imaju li novčići različite vrijednosti za druge parametre igre. Zaslon nakon nekoliko pokupljenih novčića prikazan je na slici 5.9.

Slika 5.9. Prikaz igre nakon nekoliko pokupljenih novčića

Kako je prikazano na slici 5.8, početno stanje igre prikazuje zaslon postavljenog prostora za igru s rezultatom 0.0 i malom zmijom. Na slici 5.9 vidljivo je kako su se nakon nekoliko pokupljenih novčića povećali zmija i rezultat.

5.2.5. Ispitivanje rada završetka igre

Potrebno je provjeriti rade li stanja završetka igre, a to su kad glava zmije dodirne tijelo zmije ili kad zmija dodirne zid prostora za igru. U slučaju kad glava zmije dodirne tijelo zmije ili kad zmija dodirne zid prostora za igru korisnika se odvodi na zaslon na kojem može pohraniti rezultat. Zaslon za pohranu podataka ima onemogućen gumb za pohranu podataka dok se ne unesu potrebni parametri. U slučaju da korisnik odabere opciju „No thank you“ vraća se na početni zaslon. Zaslon za pohranu podataka s onemogućenim gumbom vidljiv je na slici 5.10.

Slika 5.10. Zaslon za unos podataka o rezultatu

Dodavanjem obveznog parametra nadimka, gumb se osposobljava i odvodi se korisnika na ljestvicu najboljih rezultata, a zaslon je vidljiv na slici 5.11.

Slika 5.11. Zaslon s omogućenim gumbom za pohranu podataka

Prema slikama 5.10 i 5.11 vidljivo je kako ponašanje onemogućavanja i omogućavanja gumba za pohranu podataka radi očekivano nakon upisivanja nadimka.

5.3. Osvrt na arhitekturu VIPER

Arhitektura VIPER omogućila je pisanje čitljivijeg i bolje strukturiranog koda i datoteka. Arhitektura VIPER omogućuje odvajanje koda u mnoštvo manjih cjelina koje je lako testirati i koje poštuju načela SOLID. Omogućuje lakše stvaranje komponenata koje se mogu koristiti na više mjesta. Također, velika prednost arhitekture VIPER je što se većina klasa skriva iza protokola koji otkrivaju informacije koje su potrebne prema vanjskim klasama. Takav pristup omogućuje enkapsuliranje informacija unutar klasa kako vanjske klase ne bi mogle utjecati na informacije koje ih se ne tiču. VIPER omogućuje lakši pronalazak klasa koje se mogu vezati pod iste protokole da bi se i pravi tipovi klasa sakrili iza protokola.

Arhitektura VIPER omogućuje reaktivno i asinkrono programiranje zbog strogo definiranih pravila komunikacije komponenti arhitekture. U svakom trenutku se zna gdje se trebaju osluškivati događaji u aplikaciji kako bi se pravilno obradili. Uz pravilnu implementaciju arhitekture VIPER može se postići vrlo čitljiva, čista aplikacija koju je moguće u budućnost nadograditi bez puno poteškoća.

5.4. Analiza korisničkog iskustva

Mobilna računalna igra uzela je u obzir neke od temelja korisničkog iskustva da bi priuštila što bolje korisničko iskustvo kad je budu koristili stvarni korisnici. Najbolji način za provjeru korisničkog iskustva jest korištenje od strane stvarnih korisnika. Aplikacija je testirana od strane pet osoba da bi se pokrilo više dobnih granica. Osobe su imale 11, 24, 29, 47 i 60 godina. Testiranje je trajalo tri sata da bi se dobio prvi dojam proširene stvarnosti u mobilnoj računalnoj igri. Dvije osobe od pet testiranih, osobe starije životne dobi, nisu se do sada susrele s proširenom stvarnosti, a samo jedna od njih u slobodno vrijeme igra mobilne računalne igre.

Osoba od 11 godina navela je kako je mobilna računalna igra zabavna, ali joj nedostaje više načina igre ili potencijalno različitih prostora za igru. Također, pohvalila je kako je mobilna računalna igra laka za koristiti te da joj nije trebalo dugo da se navikne na način igre.

Osobe od 24 i 29 godina navode slične prednosti i mane kao osoba od 11 godina, da bi potencijalno drugi prostori igre omogućili još zanimljivijim igrama kako korisnik ne bi znao što ga može dočekati prije početka igre. Uz navedeno, pohvalile su vibracije mobitela i animacije gumbova.

Osobe od 47 i 60 godina izjasnile su se na sličan način. Navode kako im je ideja igre zanimljiva, ali kako ne uspijevaju pratiti zmiju i upravljati njome dok u nekim slučajevima trebaju pomicati kameru. Pohvalile su kako su pravila i način igre dobro objašnjeni. Također, pohvalile su i mogućnosti promjene parametara igre te kako ih je to zadržalo duže u igri jer su mogle smanjiti brzinu i povećati veličinu prostora za igru dok se nisu malo priviknule na igru. Konačne ocjene korisničkog iskustva pojedinaca vidljive su u tablici 5.1.

Tablica 5.1. Prikaz konačnih ocjena korisničkog iskustva pojedinaca

Godine	Proširena stvarnost je poboljšanje iskustva	Ocjena
11	Da	5/5
24	Da	4/5
29	Da	4/5
47	Da	4/5
60	Da	3/5

Iz rezultata tablice vidljivo je kako je prosječna ocjena korisnika 4/5 i kako je proširena stvarnost pozitivno utjecala na poboljšanje korisničkog iskustva.

5.5. Budući koraci za poboljšanje mobilne računalne igre

U svrhu poboljšanja mobilne računalne igre idući koraci bili bi poboljšanje učitavanja horizontalnih ploha jer u nekim slučajevima pri niskoj razini svjetlosti se sporije učita prostor za igru. Također, potrebno je uzeti u obzir i osvrte korisnika iz potpoglavlja 5.4 te ubaciti dodatne prostore za igru koji možda ne izgledaju kao klasični prostori za igru starijih inačica igre *Snake*. Uz stavke vezane za korisničko iskustvo, potrebno je dodatno istražiti je li moguće smanjiti potrošnju radne memorije mobilne računalne igre dodatnom optimizacijom koda.

6. ZAKLJUČAK

Cilj ovog diplomskog rada bio je istražiti proširenu stvarnost, mobilne računalne igre i utjecaj korištenja proširene stvarnosti u mobilnim računalnim igrama. Pregledom trenutnog stanja spomenutih područja uvidjeli su se načini implementacije proširene stvarnosti na platformi *iOS* u obliku mobilne računalne igre. Također, da bi mobilna računalna igra imala što aktualnije načine prikazivanja sadržaja korisnicima, pregledane su trenutno popularne mobilne računalne igre s proširenom stvarnošću. Prije početka razvoja mobilne računalne igre, predstavljeno je idejno rješenje mobilne računalne igre uz opise pravila arkadne igre *Snake* koja se odnose i na mobilnu računalnu igru rada. Uz navedeno, postavljeni su funkcionalni i nefunkcionalni zahtjevi i prema njima su napravljeni dijagrami toka koji prikazuju načine interakcije korisnika s aplikacijom. Nakon implementacije i detaljnoga opisa prema komponentama modela mobilne računalne igre, ispitano je i korisničko iskustvo.

Analiza rezultata ispitivanja pokazuje da je igra uspjela zainteresirati razne dobne generacije unosom proširene stvarnosti u arkadnu igru *Snake*. Također, rezultati ispitivanja pokazali su da je proširena stvarnost zanimljiv dodatak osobama mlađe dobi zbog puno veće uključenosti mobilnih računalnih igara u njihove živote. Proširena stvarnost pokazala se kao osvježenje stare arkadne igre *Snake* te je pozitivno utjecala na korisnike u većini slučajeva jer omogućuje da se više užive u igranje kao što je bio slučaj i tijekom istraživanja kod igara koje su nudile mogućnost igranja s proširenom stvarnosti i igranja bez proširene stvarnosti. Razvoj programske podrške u arhitekturi VIPER pokazao se vrlo dobar. Arhitektura VIPER zahtijeva podjele koda u razne datoteke i time omogućuje vrlo čist i čitljiv kod, a uz sve to omogućuje i lako testiranje cijelih komponenti aplikacije. Arhitektura VIPER lako se uklapa u reaktivno i asinkrono programiranje, jer postoje točno određena pravila koja komponenta s kojom smije komunicirati. Reaktivno programiranje za sobom povlači i asinkrono programiranje koje se pokazalo kao dobar dodatak aplikaciji, jer proširena stvarnost zahtijeva veliku procesorsku snagu. Asinkronim programiranjem postignuta je brža obrada informacija na više niti procesora što iskustvo igranja čini boljim nego da se sve odvija na glavnoj niti.

LITERATURA

[1] C.W. M. Leao, J. P. Lima, V. Teichrieb, E. S. Albuquerque, J. Kelner, *Demo – Altered reality: augmenting and diminishing reality in real time*, 2011 IEEE Virtual Reality Conference, str. 259 – 260, Signapore, 2011., doi: 10.1109/VR.2011.5759477

[2] H. Kundariya, *The future of mobile game development for iOS platform*, My Story, 2019., https://yourstory.com/mystory/aa6fb4afd6-the-future-of-mobile-g/amp, pristupljeno: 17.6.2022.

[3] A. Hayes, *Augmented Reality*, Investopedia, 2020., https://www.investopedia.com/terms/a/augmented-reality.asp, pristupljeno 17.6. 2022.

[4] The Franklin Institute, The Science of Augmented Reality?, The Franklin Institute, https://www.fi.edu/science-of-augmented-reality, pristupljeno 17.6.2022.

[5] Redbytes Software, *Mobile Game Development – It's past, Present & Future,* 2017., Medium, 2017., https://medium.com/@RedbytesSoftwar/mobile-game-development-its-past-present-future-a8ce77b5eabd, pristupljeno: 17.6.2022.

[6] I. Lunden, *IKEA Place, the retailer's first ARKit app, creates lifelike pictures of furniture in your home,* TechCrunch, 2017., https://techcrunch.com/2017/09/12/ikea-place-the-retailers-first-arkit-app-creates-lifelike-pictures-of-furniture-in-your-home/, pristupljeno: 28.8.2022.

[7] S. Ranade, M. Zhang, M. Al-Sada, J. Urbani, T. Nakajima, *Clash Tanks: An Investigation of Virtual And Augmented Reality Gaming Experience*, 2017 Tenth International Conference on Mobile Computing and Ubiquitous, Japan, 2017., doi: 10.23919/ICMU.2017.8330112

[8] L. Castaneda, M. Sidhu, *Beyond Programming: The Power of Making Games*, The Journal, 2015., https://thejournal.com/articles/2015/02/18/beyond-programming-the-power-of-making-games.aspx, pristupljeno: 18.6.2022.

[9] J. Weustink, *80% of Gen Z and Millennial Consumers Play Games*, New ZOO, 2021., https://newzoo.com/insights/articles/consumer-data-gen-z-millennials-baby-boomer-gen-x-engagement-games-esports-metaverse/, pristupljeno 19.6.2022.

[10] T. Cohen, *The number one reason consumers are playing mobile games,* Fyber, 2021., https://blog.fyber.com/reason-consumers-playing-mobile-games/, pristupljeno 18.6.2022.

[11] S. Chan, *Global App Spending Approached $65 Billion in the First Half of 2021, Up More Than 24% Year-Over-Year*, Sensor Tower, 2021., https://sensortower.com/blog/app-revenue-and-downloads-1h-2021, pristupljeno 18.6.2022.

[12] eMarketer, *Demographic Profile of US Mobile Gaming App Users, Jan 2021*, eMarketer, 2021., https://www.emarketer.com/chart/246982/demographic-profile-of-us-mobile-gaming-app-users-jan-2021-of-total-each-group, pristupljeno: 18.6.2022.

[13] Google Play, *Change The Game*, Google, 2019., https://services.google.com/fh/files/misc/changethegame_white_paper.pdf, pristupljeno 18.6.2022.

[14] T.P. Camara, R. B. V. Lima, R. F. Guimaraes, A. L. G. Damasceno, V. R. Alves, P. H. Macedo, G. L. Ramalho, *Massive Mobile Games Porting: Meantime Study Case,* 2006.,

https://www.cin.ufpe.br/~sbgames/proceedings/files/Massive%20Mobile%20Porting.pdf, pristupljeno: 20.2.2022.

[15] M. Braun, S. Beuck, M. Wolfel, *How does Augmented Reality Improve the play Experience in Current Augmented Reality Enhanced Smartphone Games?, 2019.* 2019 International Conference on Cyberwords, str. 407 – 410, Japan, 2019., doi: 10.1109/CW.2019.00079

[16] T. S. Lan, *Learning Through Augmented reality mobile game application*, 2013 IEE 63rd Annual Conference International Coundil for Education Media, str. 1 - 5, Singapore, 2013., doi: 10.1109/CICEM.2013.6820151

[17] M. Martin, *What is a Funtional Requirement in Software Engineering? Specification, Types, Examples,* Guru 99, updated 2022., https://www.guru99.com/functional-requirement-specification-example.html, pristupljeno 30.7.2022.

[18] Alexsoft, *Non-functional Requirements: Examples, Types, How to Approach,* Alexsoft, updated 2022., https://www.altexsoft.com/blog/non-functional-requirements/, pristupljeno: 30.7.2022.

[19] AshmeetSaggu, *Non-functional Requirements in Software Engineering, 2022.,* https://www.geeksforgeeks.org/non-functional-requirements-in-software-engineering/, pristupljeno: 28.8.2022.

[20] Apple, Xcode 14, Apple, https://developer.apple.com/xcode/, pristupljeno 18.6.2022.

[21] Swift, About Swift, Swift, https://docs.swift.org/swift-book/, pristupljeno 18.6.2022.

[22] L. Fairweather, *Switft Programming, More Than Just an 'Apple Language', Medium,* 2020., https://medium.com/swlh/swift-programming-more-than-just-an-apple-language-f9333e0cf30b, pristupljeno 18.6.2022.

[23] Apple, Combine, Apple, https://developer.apple.com/documentation/combine, pristupljeno 18.6.2022.

[24] J. Johnson, *Asynchronous Programming: A Beginner's Guide*, BMC Blogs, 2020., https://www.bmc.com/blogs/asynchronous-programming/, pristupljeno 18.6.2022.

[25] P. Hudson, *Async await*, Hacking With Swift https://www.hackingwithswift.com/swift/5.5/async-await, pristupljeno 18.6.2022.

[26] A. Williams, *What is Apple ARKit: Everything you need to know about iPhone AR*, Ars Technica, 2018., https://www.trustedreviews.com/explainer/what-is-apple-arkit-3286676, pristupljeno 19.6.2022.

[27] S. Axon, *How ARKit 2 works, and why Apple is so foused on AR,* 2018., https://arstechnica.com/gadgets/2018/06/arkit-2-why-apple-keeps-pushing-ar-and-how-it-works-in-ios-12/, pristupljeno 19.6.2022.

[28] Google, Firebase, Google, https://firebase.google.com/, pristupljeno 18.6.2022.

[29] D. Stevenson, *What is Firebase? The complete story, abridged., Medium,* 2018., https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0, pristupljeno 18.6.2022.

[30] Infinum, *iOS-VIPER-Xcode-Templates*, Github, updated 2021., https://github.com/infinum/iOS-VIPER-Xcode-Templates, pristupljeno 18.6.2022.

[31] H. Matos, *„Weak, Strong, Unowned, Oh My!" – A guide to references in Swift*, Krakendev, 2015., https://krakendev.io/blog/weak-and-unowned-references-in-swift, pristupljeno 18.6.2022.

[32] CombineCommunity, *CombineExt, Github,* updated 2022., https://github.com/CombineCommunity/CombineExt, pristupljeno: 21.2.2022.

[33] CombineCommunity, *CombineCocoa, Github,* updated 2022., https://github.com/CombineCommunity/CombineCocoa, pristupljeno 21.2.2022.

[34] Apple, *ARKit, Apple, https://developer.apple.com/documentation/arkit/*, pristupljeno: 21.2.2022.

[35] R. Percival, R. Slim, J. Slim, *The Complete ARKit Course – Build 11 Augmented Reality Apps,* updated 2020., https://www.udemy.com/course/ios-augmented-reality-the-complete-course-on-arkit/, pristupljeno: 20.4.2022.

SAŽETAK

U diplomskom radu istražena je proširena stvarnost, mobilne računalne igre, povezanost proširene stvarnosti i mobilnih računalnih igara te trenutno stanje korištenja proširene stvarnosti u mobilnim računalnim igrama. Također, opisani su izazovi i pristupi pri razvoju mobilne računalne igre u proširenoj stvarnosti. Programski ostvarena mobilna *iOS* računalna igra omogućuje primjenu proširene stvarnosti, stoga korisnici imaju mogućnost igrati poznatu arkadnu igru *Snake* u proširenoj stvarnosti. Rješenje mobilne računalne igre pisano je prema predlošku arhitekture VIPER, a metodologija izrade oslanjala se na reaktivno i asinkrono programiranje. Također, opisan je utjecaj arhitekture VIPER pri razvoju mobilne računalne igre. Mobilna računalna igra *Snake* omogućuje korisnicima upravljanje objektima u proširenoj stvarnosti prema pravilima igre, rangiranje igrača prema ostvarenim rezultatima i pohranu rezultata. Ispitivanje rada aplikacije pokazalo je njeno očekivano ponašanje, a iz istraživanja korisničkog iskustva saznaje se da je utjecaj proširene stvarnosti na korisnike pozitivan.

Ključne riječi: asinkrono programiranje, iOS, korisničko iskustvo, mobilna računalna igra, programska arhitektura VIPER, proširena stvarnost.

ABSTRACT

The aim of this master's thesis was to explore augmenter reality, mobile games, connection between augmented reality and mobile games, and the current state of using augmented reality in mobile games. Furthermore, this thesis describes challenges and approaches in developing mobile games in augmented reality. The software-realized mobile iOS game enables the application of augmented reality, so users have the ability to play the famous arcade game Snake in augmented reality. The mobile game solution was written according to the VIPER architecture template, and the development methodology was based on reactive and asynchronous programming. Additionally, the thesis describes the influence of the VIPER architecture in the development of a mobile game. The mobile game Snake allows users to manage objects in augmented reality according to the rules of the game, rank players according to the achieved results and store the results. Testing the operation of the application showed its expected behavior, and from the user experience research it was found that the impact of augmented reality on users is positive.

Key words: asynchronous programming, augmented reality, iOS, mobile game, user experience, VIPER architecture.

ŽIVOTOPIS

Zvonimir Medak rođen je 25.8.1998. u Slavonskom Brodu. Osnovnu i srednju školu pohađao je u svom rodnom gradu. Trenutno je na drugoj godini diplomskog studija Programsko inženjerstvo na FERIT Osijeku. Ima iskustva s nekolicinom programskih jezika kao što su C, C++, C#, Objective-C Java, Kotlin i Swift. Prije dvije godine završio je iOS akademiju organiziranu od strane tvrtke Factory.hr gdje je stekao znanja iOS-a i Swifta, dok se prije akademije bavio Android programiranjem. Po završetku iOS akademije radio je više od godinu dana na komercijalnim projektima u tvrtki Factory.hr, a potom se zaposlio u tvrtki Infinumu gdje trenutno radi na komercijalnim projektima i stječe znanja u razvoju iOS aplikacija.

PRILOZI

Prilog 1. Diplomski rad u docx formatu

Prilog 2. Diplomski rad u pdf formatu

Prilog 3. Programski kod mobilne računalne igre Snake