Izrada 2D akcijske računalne igre u Unity-u

Letica, Roberto

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Roberto Letica

Izrada 2D akcijske računalne igre u Unity-u

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Roberto Letica

Matični broj: 45877/17–R

Studij: Informacijski sustavi

Izrada 2D akcijske računalne igre u Unity-u

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, srpanj 2021.

Roberto Letica

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom završnom radu prikazao sam kako razviti 2D akcijsku igricu za mobilne uređaje te kako tu igricu izdati na tržište te je pokušati monetizirati. Iako je igrica prvobitno trebala biti računalna, fokus je na kraju okrenut na pokušaj publiciranja igre kao mobilne igre. Igra se može igrat i ina računalima samo bi bilo potrebno isključiti mobilnu autentikaciju i staviti drugu vrstu autentiakcije tipa email-password. Korišteno je više alata za izradu ove mobilne igre, a primarni alat u kojem je rađena sama logika igrice je Unity, a Visual Studio je služio kao alat za pisanje koda u C#-u.

U ovom završnom radu biti će prikazana izrada igrice koja je krenula kao računalna igra te se na kraju preusmjerila ka mobilnoj igri sa ciljem da se ostvari dobit putem reklama i brojem korisnika koji bi igrali igricu. Ukratko će biti prikazano kako je tekao razvoj proizvoda te odakle sam došao do ideje za ovaj završni rad. Nadalje, spomenuti su alati koji su korišteni prilikom izrade, od alata za izradu ilustracija, alata za pisanje koda te alata za kreiranje vlastitih pjesama za igru. Osim toga, opisano je kako krenuti sa igrom u 2D svijetu te koje su komponente nužne ukoliko se igrica želi napraviti u tom svijetu. Zatim je opisan primjer korištenja NoSQL baze koju nudi Firebase te njihove autentikacije putem Google Play računa. Također na kraju je ukratko objašnjeno kako dodati reklame u igru i povezati ih sa Unity računom i sa igricom na Google Play konzoli kako bi igrica bila monetizirana te zarađivala na reklamam.

Ključne riječi: Unity, igra, 2D, android, Firebase, akcija, strategija

Sadržaj

1. Uvod

Moderno doba je pokrenuto informatičkom revolucijom, a sa tom revolucijom su zaživjele nove tehnologije te novi oblici komunikacije pa su s toga se razvile i mnoge računalne igre, mobilne igre te igre za različite konzole. Sa tom revolucijom dolazi nova era uređaja koja postaju dostupna korisnicima u bilo kojem trenutku, na bilo kojem mjestu, a također i taj korisnik je uvijek dostupan tom uređaju, a to je mobitel. Razvitkom mobilne tehnologije u zadnjih 20 godina razvija se i nova vrsta industrije, a to je mobilna gaming industrija. Danas gotovo svaka osoba posjeduje mobitel, te je gaming industrija uvidjela i iskoristila tu priliku da ponudi svoje proizvode što većem broju korisnika. Danas većina ljudi gleda kako da na što zanimljiviji način skrati svoje putovanje različitim transportima, kako da stvori oblik socijalizacije putem novih tehnologija ili jednostavno kako da skrati svoje vrijeme ili iskoristi svoj slobodan dan na vrstu zabave koja mu odgovara. Gaming industrija je tu vidjela svoje mjesto na tržištu u mobilnim igrama jer su danas korisnici mobilnih uređaja gotovo svakodnevno dostupni svojim mobitelima i pokušavaju svoje vrijeme iskoristiti provodeći se na njima. Stoga su različite tvrtke proizvele raznovrsne tipove igara sa jednim ciljem, a taj cilj je da se korisnik zadrži upravo na njihovoj mobilnoj igri. Stoga se na tržištu danas može naći igra za svakog korisnika, od najmlađeg doba pa i do onih starije dobi, tako da je konkurencija danas velika na tržištu mobilnih igara te se teško probiti na tržište kao solo developer sa vlastitom igrom. Iako je pokazano da postoje i iznimke gdje developeri uspiju i sa vlastitom igrom probiti se na tržište kraj tvrtki koje ulažu velike svote novaca u igre, ali za takav uspjeh je potrebna vrhunska ideja i izvrsna realizacija vlastite ideje.

Kao primjer vlastite ideje te pokušaj realizacije te ideje u ovom radu opisana je igrica za mobilne uređaje Android sustava, a igrica spada pod više žanrova. Žanrovi pod koje bi ju smjestio su: Strateška igra, Logička igra i Akcijska igra. Ovakvi tipovi igara su mi najzanimljiviji stoga sam odlučio sam razviti igricu tog tipa koju bih i ja sam igrao. Igra sadrži 2 svijeta, a svaki taj svijet sadrži 24 razine. Cilj igre je svladati svaku razinu koristeći 4 pomoćna objekta, a svaki objekt ima vlastitu cijenu, te je potrebno sa što manjom cijenom preći svaku razinu. Također je implementirana i ljestvica najboljih igrača tako da uvijek postoji doza kompetitivnosti te da se igrači potrude smisliti što bolji način za prijeći svaki razinu. No kako igrica ne bi bila jednostavna napravljene su i različite zamke koje korisnika sprječavaju u prelaženju razine te sam ih dodao kako igrica ne bi razvila dozu monotonije.

Motivaciju za ovaj završni rad sam dobio u tome što sam imao želju da napravim nešto svoje što ću moći plasirati na tržište, da naučim nove tehnologije sa kojima još nisam bio upoznat, poput Unity-a, Firebase-a, Inkscape-a i ostalih programa korištenih za razvoj ove igrice te da dobijem iskustvo u razvoju mobilne igre od beta verzije pa sve do finalne verzije.

2. Metode i tehnike rada

Metode i tehnike rada korištene u ovom radu su Unity, Microsoft Visual studio, Firebase, Google Play servisi, FL Studio te Inkscape. Unity i Visual studio su alati korišteni za development mobilne igre te je primarni jezik za razvoj igrice bio C#, FL Studio je korišten za razvoj vlastite glazbe koja je kasnije dodana u igricu, Inkscape je korišten kao alat za razvoj različitih ilustracija, a Firebase je korišten za autentikaciju korisnika te je korištena NoSQL baza podataka za spremanje podataka o korisnicima, odnosno o tome do koje su razine došli, koliko života imaju, novaca i tih sličnih podataka.

Kao primarnu pripremu za završni radio koristio sam platformu Udemy na kojoj su različiti kursovi za učenje, te sam odlučio proći 20-satni kurs C#-a i izrade mobilnih igara u Unity-u gdje je primjenjivan C#. Osim te web platforme koristio sam i YouTube kao izvor informacija u slučaju kada bih imao neke probleme kod razvoja određenih stavki unutar igrice. Također je pomogao predmet „Programsko Inženjerstvo“ gdje sam stekao nove vještine u programiranju te sam usavršio svoje prethodno znanje. Nadalje kako do sad nisam bio upoznat sa NoSQL bazama podataka, imao sam dosta problema sa web platformom Firebase-om gdje sam uložio puno truda te pogledao dosta kursova kako bih sve napravio da funkcionira i kako bi igrica bila povezana od same login autentikacije putem Google play računa pa do spremanja korisničkih podataka u bazu podataka. Za alat Inkscape sam također koristio YouTube kao izvor podataka te sam prošao jedan kurs kako bih stekao osnovno znanje za sami program, te sam za dodavanje Google play servisa također prolazio video tutorijale na YouTube-u.

3. Opis alata korištenih za izradu igre

Glavni alati korišteni za izradu ove mobilne igre su Unity, Microsoft Visual Studio te Firebase. Uz te glavne alate korišten je C# kao programski jezik za izradu skripti unutar Visual Studia. Pomoćni alati koji su korišteni su: Inkscape, FL Studio te Google play servisi. U idućim potpoglavljima opisati ću ukratko pomoćne alate te ću nešto opširnije opisati glavne alate korištene za izradu završnog rada.

3.1. FL Studio, Inkscape i Google play servisi

FL Studio je programski alat koji sam koristio za razvoj vlastite glazbe unutar igrice. S obzirom da mi je cilj ovog završnog rada bio plasirati igru na tržište odlučio sam izraditi vlastitu glazbu putem ovog programa kako igrica ne bi dobila [1]copyright strike u slučaju da sam koristio tuđe pjesme. Uz FL Studio koristio sam i programski alat Inkscape koji mi je služio za dizajniranje nekih ilustracija unutar igrice te za uljepšavanje već postojećih ilustracija. Od Google play servisa sam koristio servise za ljestvice igrača sa najboljim rezultatima za svaku razinu te sam login korisnika vršio putem njegovog Google play računa te ukoliko korisnik nema Google play račun ne može igrati igricu, a s time je stavljen fokus na to da je igrica isključivo za mobitele sa Android sustavom.

3.2. Firebase

Firebase je platforma razvijena na infrastrukturi od Google-a, a razvijen je kao [2]BaaS platforma. Nudi developerima različite alate i servise koji im olakšavaju razvoj aplikacija i igara. Servisi koje sam koristio u izradi završnog rada su „*Realtime Database*“ servis i „*Authentication“* servis. Realtime Database servis nudi spajanje igre putem WebSocket-a umjesto putem HTTP-a, a spajanje putem WebSocket-a je puno brže nego putem HTTP-a. Svi podaci koji se šalju putem WebSocket-a se sinkroniziraju toliko brzo koliko korisnikov Internet može podnijeti. Čim korisnik spremi neke podatke, podatci se ažuriraju istog trena te svi povezani korisnici dobivaju iste podatke gotovo u istom trenutku. Također, Realtime Database je NoSQL baza podataka koja sprema podatke u obliku JSON dokumenta. Firebase podržava mnogo različitih vrsta autentikacija, bilo to putem Facebook-a, putem Google play računa, Twittera ili ostalih društvenih mreža. Također svaka Firebase autentikacija se sprema direktnu u Firebase bazu podataka pa je olakšan pristup podacima [1]. Autentikaciju unutar

[1] Opomena koja se dobiva u slučaju korištenja bilo kakvog sadržaja za koji nemamo dopuštenje da koristimo.
[2] Backend-as-a-Service

igre sam vršio putem Google play računa te sam tako izbjegao autentikaciju putem društvenih mreža, kako bi igrica bila samo za Android uređaje.

3.3. Microsoft Visual Studio

Microsoft visual studio je programski alat stvoren od firme Microsoft koji se koristi za različite tipove razvoja programa, od razvoja računalnih programa, različitih web aplikacija i web usluga te za razvoj mobilnih aplikacija. Sadrži mnoge alate, kompajlere i mnoge druge značajke koje uvelike olakšavaju programiranje te razvoj programa. Kao programski alat postoji preko 20 godina krenuvši od prve verzije koja se zvala Visual Studio 97 pa sve do današnje verzije Microsoft Visual Studio 2019. Visual Studio [3]IDE kao program pomaže programerima pri pisanju i uređivanju koda. Korisničko sučelje koje se nalazi u samom programu se koristi pri razvoju programa, pisanju skripti, uklanjanju pogrešaka te uređivanju samog koda. Također pruža usluge uređivača koda IntelliSense koji pomaže programeru pri pisanju koda i refaktoriranju koda, a osim uređivača koda postoji i integrirani alat za ispravljanje pogrešaka koji ujedno radi na source-level razini i machine-level razini. Neki od ostalih ugrađenih alata su dizajner za izgradnju GUI aplikacija, web dizajner, dizajner klasa i mnogi drugi. Osim već ugrađenih alata, Visual Studio IDE nudi i mnogo različitih ekstenzija na svojoj tržnici koje korisnik može skinuti sebi po potrebi [2].

U završnom radu sam ga koristio kao alat u kojem sam pisao C# skripte. Koristio sam ga za pisanje svih skripti unutar samo igrice, a neke od skripti koje sam pisao u Visual Studiu su skripte za zamke, za glavnog igrača unutar igrice odnosno loptu, za pokretanje različitih animacija u određenom trenutku te za ostale skripte koje su bile potrebne kako bi mobilna igra funkcionirala.

3.4. Unity

Unity je još jedan u nizu primarnih alata korištenih pri izrada završnog rada. To je program koji se koristi pri izradi 2D/3D igara. Prednost Unity-a je što nije vezan za izradu igara samo za jednu platformu nego nudi mogućnosti izrade igara za različite platforme, od igara za mobilne igre, za konzole te za računalne igre. Uz to što nudi mogućnost izrade igara za različite platforme, posjeduje i različite ugrađene značajke koje uvelike olakšavaju programiranje bilo 2D ili 3D svijeta. Neke od ugrađenih značajki su fizika, 3D renderiranje, detektiranje kolizija, animiranje likova, animiranje korisničkog sučelja, dizajniranje korisničkog sučelja i mnoge druge značajke. Uz sve ove značajke Unity ima svoju tržnicu poput Visual Studija na kojoj

[3] Integrated development environment – Integrirano razvojno okruženje

korisnik može kupovati ilustracije potrebne za igricu, dizajn korisničkog sučelja, glazbu koju će koristiti u igrici, pozadine na razinama, materijale koje će stavljati na svoje objekte te mnoge druge stvari koje programeru trebaju [3].

Sve ove značajke mnogo pomažu početnicima koji se žele baviti programiranjem igrica, a nemaju još širok spektar znanja da bi sami mogli napraviti svoj game engine. Upravo zbog toga danas je Unity jako raširen među studentima te uz svoju jednostavnost nudi i YouTube tutorijale koji pomažu početnicima da isprogramiraju svoje prve igrice te steknu osnovno znanje o korištenju programa. Upravo zbog tih razloga sam odlučio Unity koristiti kao game engine za izradu završnog rada.

3.4.1. Unity komponente i značajke

U ovom poglavlju su opisane Unity komponente koje su korištene pri izradi igrice. Kako je igrica rađena u 2D svijetu opisivane su samo komponente 2D svijeta koje su korištene. Iz navedenog poglavlja paralelno je i objašnjena moć Unity game engine-a te koliko kao program olakšava razvijanje igrica korisnicima.

3.4.1.1. 2D svijet

Kada se kreira novi projekt korisnik u početku može odabrati da li želi raditi u 2D svijetu ili 3D. Bez obzira koji svijet korisnik odabrao na kraju opet unutar editora kada se program otvori može odabrati željeni svijet [4].

Kako je igrica rađena u 2D svijetu kada se otvori projekt odabere se 2D svijet i u pogledu na sceni (eng. *Scene view*) postavi se 2D pogled. Taj pogled nam prikazuje kako scena trenutačno izgleda te ukoliko se starta dobit ćemo prikaz koji pokriva glavna kamera (eng. *Main camera*), a to je glavna kamera koja snima određeni dio svijeta te ga prikazuje igraču [5].

Ako se želi dodati neki objekt unutar 2D svijeta može se kreirati putem desnog klika i odabirom željenog 2D objekta. Najčešće dodavani objekti u mojoj igrici su ilustracije(eng. *Sprite)*. Svaka ilustracija koja je bila dodana unutar projekta morala je biti 2D slika postavljana pod komponentom Sprite. Ukoliko je nekom objektu dodavana slika stvara se komponenta za renderiranje ilustracija(eng. *Sprite Renderer)* te se putem te komponente ilustracija prikazivala na željenom objektu [6].

3.4.1.2. Rigidbody 2D i Collider 2D

Komponente „Rigidbody 2D“ i „Collider 2D“ su najviše koristile pri izradi samog određivanja načina kako će se razine odvijati te kako će objekti djelovati prilikom sudara te kako će fizika utjecati na njih.

Komponenta „Rigidbody 2D“ se dodaje na objekt kada želimo da taj objekt bude pod kontrolom fizike. Posebnost kod ove komponente je da kada se doda objektu taj objekt se može kretati samo po xy osima, dok po z osi ne može. Neke od vrijednosti kojima se može manipulirati unutar ove komponente su masa objekta (eng. *Mass)*, različite vrste otpora poput linearnog otpora (eng. *Linear drag)* i kutnog otpor (eng. *Angular drag)*. Uz te vrijednosti također se može utjecati i na jačinu gravitacije unutar svake razine modificirajući vrijednost varijable gravitacijske skale (eng. *Gravity scale)*. Uz sve navedene varijable može se utjecati i na opciju tip tijela (eng. *Body Type*) koja određuje kako će se objekt ponašati pod djelovanjem različitih sila [7].

Komponenta „Collider 2D“ određuje oblik 2D objekta u svrhu fizičkih sudara. Ta komponenta se ne vidi kada se igrica igra te se može uređivati tek kada se odabere mogućnost „Edit Collider“. Kako se komponenta ne vidi cilj joj je da bude istog oblika kakvog je i objekt na kojem se nalazi kako bi što stvarnije predočavala sudare između objekata. Tipovi komponente koji se mogu koristiti uz Rigidbody 2D su [8]:

- „Circle Collider 2D“ – korišten kod objekata okruglog oblika poput lopti i novčića u igrici
- „Box Collider 2D“ – korišten kod objekata kvadratnog oblika poput kutije u igrici
- „Polygon Collider 2D“ – korišten kod objekata nepravilnog oblika poput nepravilnih objekata pomoću kojih su kreirani leveli
- „Edge Collider 2D“ – nije korišten
- „Capsule Collider 2D“ korišten kod objekata koji su služili kao zamke s vratima i kod trampolina
- „Composite Collider 2D“ – nije korišten

3.4.2. Animator i Animator Controller

Iduće bitne komponente kod razvoja igrica su „Animator“ i „Animator Controller“. „Animator“ je komponenta koju dodajemo objektu, tada taj objekt ima dodan animator koji traži referencu na „Animator Controller“ putem kojeg se dodaje željena animacija objektu [9].

„Animator Controller“ omogućava dodavanje i uređivanje redoslijeda animacijskih klipova putem animacijskih translacija unutar editora za animacije. Pretežito je očekivano da će svaki objekt imati više animacija te da će se te animacije izvršavati ovisno o situaciji do koje dođe, a to se sve sređuje putem translacija koje se onda aktiviraju putem koda u skriptama [10].

3.4.3. Ostale komponente

Osim navedenih komponenti koje su korištene, koristile su se i druge komponente. Za zvuk unutar igrice služila je komponenta „Audio Source“ na kojoj se dodaje pjesma određene duljine te se različitim opcijama može modificirati ta pjesma. Najznačajnije opcije su jačina zvuka (eng. *Volume)* kojom se određuje koliko će glasno pjesma svirati, zatim opcija „Loop“ kojom određujemo ukoliko želimo da pjesma uvijek iznova svira kada dođe do kraja [11].

Iduća bitna komponenta je „Script“ komponenta putem kojem se dodaju skripte na objekte te pomoću tih skripti određujemo što želimo da se događa sa tim objektom u određenom trenutku. Unutar skripti se upravlja sa kolizijama ako želimo da se nešto dogodi u tom trenutku, pokreću se animacije ako korisnik pritisne nešto ili se neki događaj dogodi u igrici, pokreće se zvuk ili zaustavlja ovisno o tome što se treba dogoditi i mnoge druge stvari se uređuju putem skripti.

4. Ideja i opis igre

U idućem poglavlju opisat ću kako sam došao do ideje za igru, te ću opisati cilj igre te žanrove igre pod koje spada. Kako se igrice pretežito razlikuju po žanrovima i grupiraju po načinu igranja odnosno po tome koji sadržaj igra nosi tako sam i ja ovu igricu svrstao u akcijsku, logičku i stratešku igru iako doza akcije nije prevelika u igrici.

4.1. Žanrovi igre

Akcijske igre (eng. *Action games)* su opisane kao igre sa kontinuiranom napetošću te je fokus postavljen na različite kretnje, borbe, koordinacije i reakcijsko vrijeme. Kao žanr se razvija sredinom 70-ih godina prošlog stoljeća te se u početcima slabo borila na tržištu kraj igrica sportskog i arkadnog žanra. Preokret se dogodio krajem 70-ih godina sa igricom „*Space Invaders“* koja je doživjela veliki uspjeh te industrija igrica postaje dominirana akcijskim žanrom . Nakon dominacije u industriji igrica, akcijske igrice su se održale sve do danas sa različitim tipovima igara. Danas postoje različite vrste akcijskih igara te su neke od njih jedno od najigranijih igrica na svijetu, a neke od poznatijih su igre ratne tematike poput serijala „*Call Of Duty*“, „*Battlefield*“ i „*Fortnite“* [12].

Strateške igre (eng. *Strategy games)* su igre koje potiču igrača na razmišljanje te da sa smišljenim načinom prijeđe određenu razinu ili zapreku. Postoje različite vrste strateških igara poput turn-based i real-time strateških igara. Real-time strateške igre se igraju tako da više igrača u isto vrijeme igraju istu igricu jedni protiv drugih. Jedne od poznatijih igara tog tipa su „*Starcraft“, „Rise of Nations“* i „*Age of Empires“.* Nasuprot tih igara su turn-based strateške igre gdje korisnici igraju u isto vrijeme, ali moraju čekati da završi protivnikov potez kako bi mogli

odigrati vlastiti potez. Većina strateških igara ima svoje korijene u društvenim strateškim igrama [13].

Logičke igre (eng. *Puzzle games)* su igre koje glavni fokus stavljaju na misaone vještine umjesto na reflekse. U njima se traži od igrača da na misaoni način riješi level ili problem koji ga je zadesio. Većina tih igara ne prati priču u igrici te rijetko kada postoje nekakvi likovi ili zlikovci kao u drugim igricama, zbog toga najčešće igrice imaju ljestvicu sa najboljim rezultatima. Jedna od poznatijih igrica tog tipa je „*Tetris“*, ali također većina igrica uključuje elemente ovog tipa poput „*God of War“ i „Prince of Persia: The Sands of Time“* [14].

Ukratko bi se opisalo kako igrica sadrži elemente akcijske igre što se tiče kretnje glavnog igrača odnosno lopte te zamke koje zaustavljaju tu loptu. Od elemenata strategije sadrži poticaj igrača na razmišljanje kako na što efikasniji način riješiti svaku razinu, da pri tome bude što manji rezultat kako bi taj rezultat ujedno bio i što bolji. Kao elemente logičkog dijela sadrži također misaoni dio gdje igrač treba koristeći logiku preći svaku razinu te ujedno na kraju postoji ljestvica sa najboljim rezultatima kako bi se održala kompetitivnost među igračima.

4.2. Opis igre, cilja igre i ideje za igru

Kako bi opisao igru i sami cilj igre potrebno je prvo objasniti ideju za igru. Kako sam oduvijek igrao igrice takvog tipa gdje je igraču potrebno koristeći logiku i strategiju da prijeđe razinu, odlučio sam smisliti i napraviti nešto tog tipa. Ovo je prva ideja koju sam dobio jer nisam našao nešto slično na tržištu igara stoga sam mislio da bi igrica mogla doživjeti uspjeh pa sam ju odlučio ujedno i publicirati i staviti reklame unutar igrice. Dobivši ideju za igru razvio sam plan kako bi se trebala igrati, koji će biti cilj igre i što će igricu kontinuirano činiti zanimljivom.

Igra ima 2 svijeta u kojem svaki svijet sadrži 24 razine te da bi korisnik mogao ići dalje mora prijeći razinu na kojoj se nalazi. Svjetovi su podijeljeni na ljetni svijet i na zimski svijet te su tako zamke prilagođene njihovim razinama. Igrač koji igra igru je stvoren na početak svake razine te ima ponuđene objekte sa kojima treba riješiti razinu. Objekti koji su u ponudi da bi igrač riješio razinu su trampolin, kutija, ventilator i magnet. Osim objekata za rješavanje razina, svaki svijet nosi svoje zamke. Zamke u ljetnom svijetu sa kojima igrač se susreće su vrata koja se otvaraju kada igrač dodirne prođe kroz zastavicu koja je boje od tih vrata, voda, pomični objekti na mapi, kamenje koje odbacuju igrača izvan mape, te zamka koja odbacuje korisnika u zrak te može u nekim slučajevima služiti korisno, a u nekim može smetati igrača pri rješavanju razine. Zamke u zimskom svijetu su ledenice koje ukoliko igrač dodirne odmah gubi, pomični objekti na mapi, snjegović koji ima radijus u kojem privlači igrača te mu tako ne dopušta da se kreće dalje po mapi, ledeni čekić koji odbacuje igrača ukoliko ga igrač dotakne, ledene pločice koje u trenutku kada ih igrač dotakne nakon 1 sekundu nestaju, led koji ubrzava

igrača dok se kreće po njemu i posljednja zamka su grude snijega koje padaju i pokušavaju udarit igrača.

Kako bi igrica dobila smisao ima svoj cilj. Na svakoj razini igrač može postići određen rezultat, svaka razina ima svoj rezultat za 3 zvijezde, za 2 zvijezde i za 1 zvijezdu. Do rezultata se dolazi na način da objekti uz koje igrač rješava razine imaju svoju vrijednost. Trampolin ima vrijednost 5, kutija 2, ventilator 3 i magnet 1. Igrač mora pomoću strategije i logike riješiti svaku razinu na način da postigne što manji rezultat odnosno da uz što manje objekata riješi svaku razinu. Kako bi igrica bila napeta na kraju se mogu pogledati i ljestvice sa rezultatima te se tako igrači mogu i natjecati.

5. Izrada igre

U narednom poglavlju je opisana izrada igrice od beta verzije pa do finalne verzije. Uz opis izrade igre biti će opisani i alati koji su korišteni kako su se koristili i primjenjivali, od Unity alata koji su služili za izradu igre, zatim Visual Studio alata koji su pomagali kod pisanja skripti, Firebase alata bez kojih ne bi bilo autentikacije ni baze podataka te Google Play servisa bez kojeg ne bi bilo ljestvice sa najboljim rezultatima, a na kraju će biti i opisano kako je igrica publicirana na Google Play Store-u te će biti prikaz koliko se zaradi novaca od određenog broja reklama koji igrači pogledaju.

5.1. Početak izrade

Početak izrade je tekao tako da je napravljen novi projekt u 2D svijetu te je kreirana prva scena u Unity-u koja će predstavljati prvu razinu. Kreiravši prvu scenu, postavljena je pozadina te je određen prikaz kamere koji će biti na svim razinama. Za kameru je postavljena „*Orthograpic“* projekcija te je zatim postavljena veličina kamere točno onoliko da pokriva pozadinu kako bi ograničio svaku razinu pomoću prikaza kamere [5]. Kada je riješen pogled na scenu, dodan je 2D objekt koji će predstavljati glavnog lika.

5.1.1. Glavni lik

Za glavni lik je odabran 2D objekt oblika kružnice koji bi predstavljao loptu od nekog sporta. Nakon što je dodan objekt bilo je potrebno glavnog lika povezati sa fizikom 2D svijeta. Kako bi lopta bila povezana sa fizikom 2D svijeta dodane je komponenta „Rigidbody 2D“. Kako bi bilo što manje posla oko fizike 2D svijeta Unity engine je omogućio varijablu „Body Type“ da se postavi na vrijednost „Dynamic“ i da se potvrdni okvir označi kod varijable „Simulated“. Dodavši te vrijednosti fiziku je preuzeo Unity engine te je samo još bilo potrebno staviti odgovarajuće vrijednosti za varijable „Mass“ i za varijablu „Gravity Scale“. Postavivši vrijednosti tih varijabli na 1 i pokrenuvši igricu, lopta se ponašala kao što bi trebala u stvarnom

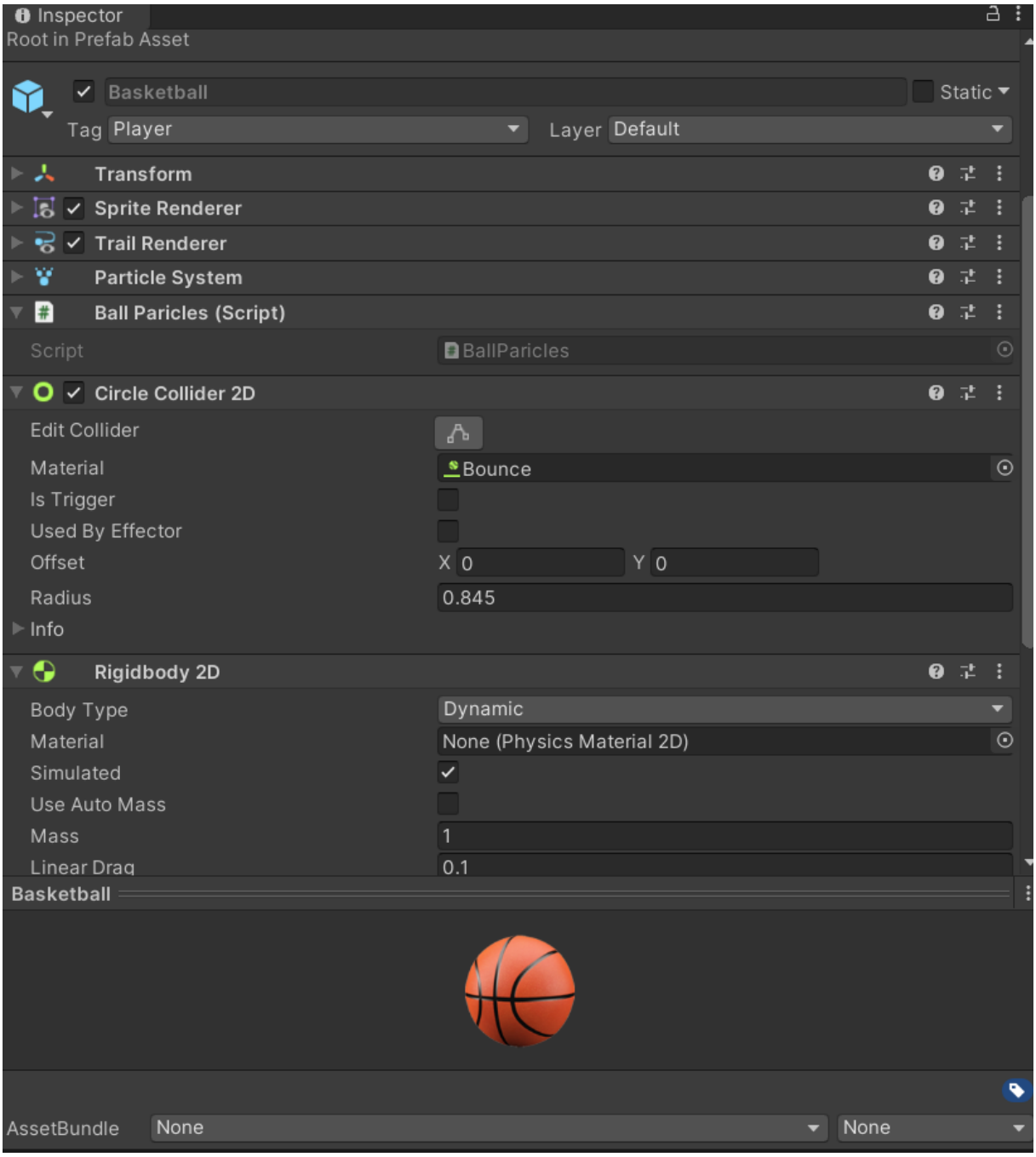
svijetu [7]. No lopta je još uvijek prolazila kroz ostale 2D objekte koji su bili dodani u scenu stoga je dodana i komponenta „Circle Collider 2D“. Nakon što je dodana ta komponenta lopta je se zaustavljala kada se sudarala sa drugim 2D objektima no lopta nije odskakala kako bi trebala, stoga je bilo potrebno pod varijablu „Material“ u sudaraču dodati materijal koji bi u slučaju bilo kakve kolizije davao lopti frikciju i poskočnost [8]. Nakon što su dodane te dvije komponente lopta se ponašala u 2D svijetu u skladu sa fizikom 2D svijeta. Za kraj je bilo potrebno urediti loptu te su dodane komponente „Trail Renderer“, „Particle System“ te skripta „Ball Particles“. „Trail Renderer“ je ukrasna komponenta koja služi tomu da lopta ostavlja trag iza sebe ovisno o tome kuda ide [31]. Komponenta „Particle System“ i skripta služe tomu kada se lopta sudari sa određenim objektom da bude prikaz ukrasnog oblačića [23]. U nastavku je prikazan kod što se dogodi kada se lopta sudari sa određenim objektom te kada će se prikazivati ukrasni oblačići.

```
private ParticleSystem particle;

    private void Awake()
    {
        particle = GetComponent<ParticleSystem>();


    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
      if (collision.gameObject.tag == "Smetala" || collision.gameObject.tag
      == "Block")
        {
            particle.Play();
        }
      }
```

U kodu se koristi metoda OnCollisionEnter2D(Collision2D collision) koja upravlja sa kolizijama između objekata [15]. Kao što je prikazano kada se lopta sudari sa objektima koji također imaju na sebi komponentu Collider2D u tom slučaju će se startat „Particle System“ na lopti.

Nakon što je lopta uređena tako da funkcionira u 2D svijetu i da je podložna zakonima fizike, za kraj je bilo potrebno dodati ilustracije te komponentu putem koje bi dodali ilustraciju na loptu. Ilustracije su rađene u programu Inkscape te je napravljeno ukupno 7 ilustracija za glavnog lika, a to su 2 košarkaške lopte, teniska lopta, lopta za plažu, bilijarska kugla, odbojkaška lopta i nogometna lopta.

Slika 1 Prikaz ilustracija lopti



Slika 2 Finalni prikaz objekta glavnog lika

5.1.2. Izrada zamki i blokova za kreiranje mapa

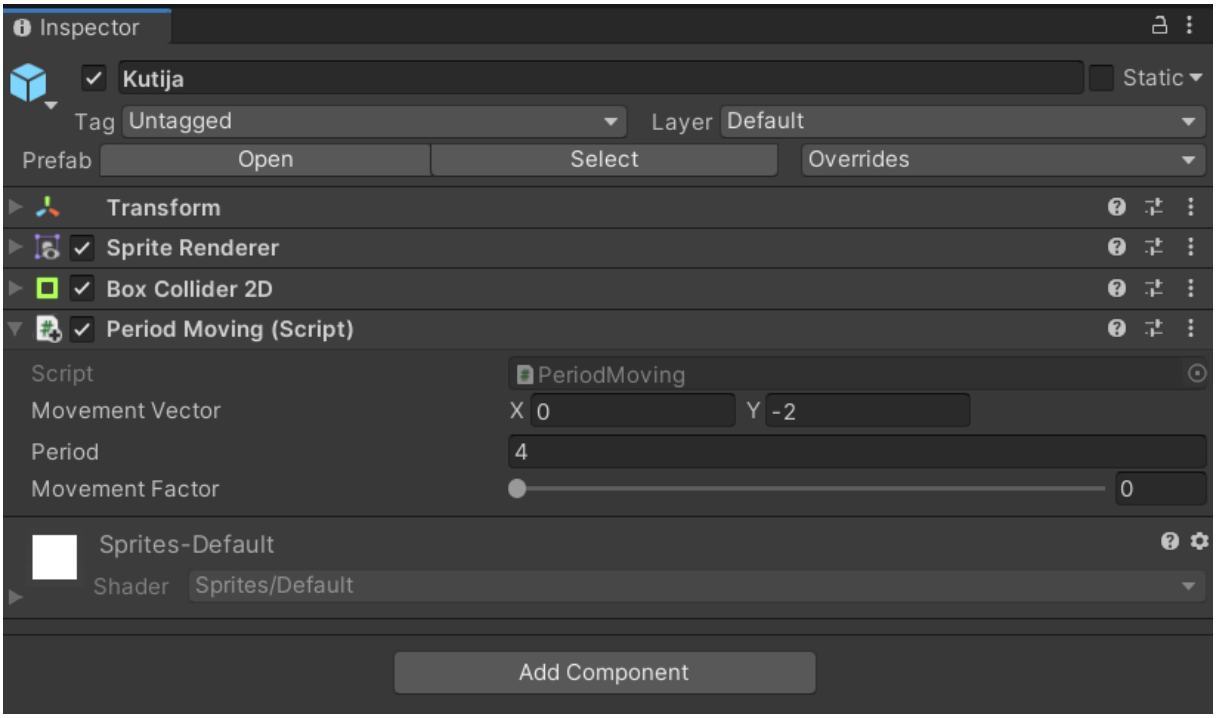
Nakon što je glavni lik bio napravljen bilo je potrebno urediti zamke sa kojima će se susretati na putu do cilja te napraviti blokove po kojima bi se glavni lik kretao. Kako je igrica podijeljena u 2 svijeta, prvo su napravljeni blokovi za oba svijeta, a zatim zamke.

5.1.2.1. Blokovi

Prvo su rađeni blokovi zato što su jednostavniji objekti za napravit. Stvoreni su prvo kao obični 2D kvadrati te im je dodana komponenta „Box Collider 2D“ ili „Polygon Collider 2D“ ovisno o ilustraciji bloka te je nakon toga testirano da li se lopta može sudariti sa blokom te hoće li se skripta Ball Particles na lopti izvršiti [8]. Kako je do sudara došlo kao završni dio dodana je komponenta „Sprite Renderer“ te je dodano više različitih ilustracija kako bi bilo više različitih blokova [32]. Posebnost kod blokova je što se neki blokovi također pomiču u određenom periodu, a za to je zadužena skripta Period Moving. U nastavku je prikazana navedena skripta.

```
[SerializeField]Vector2 movementVector;
[SerializeField] float period = 2f;
[Range(0, 1)] [SerializeField] float movementFactor;

Vector2 startingPos;

void Start()
{
    startingPos = transform.position;
}
void Update()
{
    float cycle = Time.time / period;

    const float tau = Mathf.PI * 2f;
    float rawSinWave = Mathf.Sin(cycle * tau);

    movementFactor = rawSinWave / 2f + 0.5f;

    Vector2 offset = movementVector * movementFactor;
    transform.position = startingPos + offset;
}
```

Skripta koristi dvije metode, a to su Start() i Update() te 4 varijable, od kojih se 3 mogu uređivati u editoru, a jedna ne može. Varijable koje se mogu uređivati u editoru su movementVector, period, i movementFactor jer sadrže atribut [SerializeField], a startingPos se dohvaća početkom scene [16]. Varijabla movementVector određuje u kojem smjeru će se objekt kretati, a varijabla period određuje brzinu kojom će se kretati do te točke i nazad do početne pozicije. Metoda Start() se pokreće prilikom starta scene te dohvaća početnu poziciju objekta na koji je dodana skripta, a metoda Update() je zadužena za pomicanje navedenog objekta tokom svake sekunde dok je igrač na toj razini [17].



Slika 3 Izgled bloka u editoru koji koristi skriptu Period Moving

Slika 4 Izgled blokova od oba svijeta

5.1.2.2. Zamke

Nakon što su napravljeni blokovi za oba svijeta i različite vrste glavnog lika, bile su na redu zamke koje će sputavati igrača u dolasku do cilja. Zamke su rađene drugačije za oba svijeta da budu u skladu sa svijetom koji predstavljaju.

Prva zamka sa kojom se igrač susreće se zove RotatingThing. To je objekt koji se okreće u smjeru kazaljke na satu ili u smjeru suprotnom od kazaljke na satu ovisno o tome

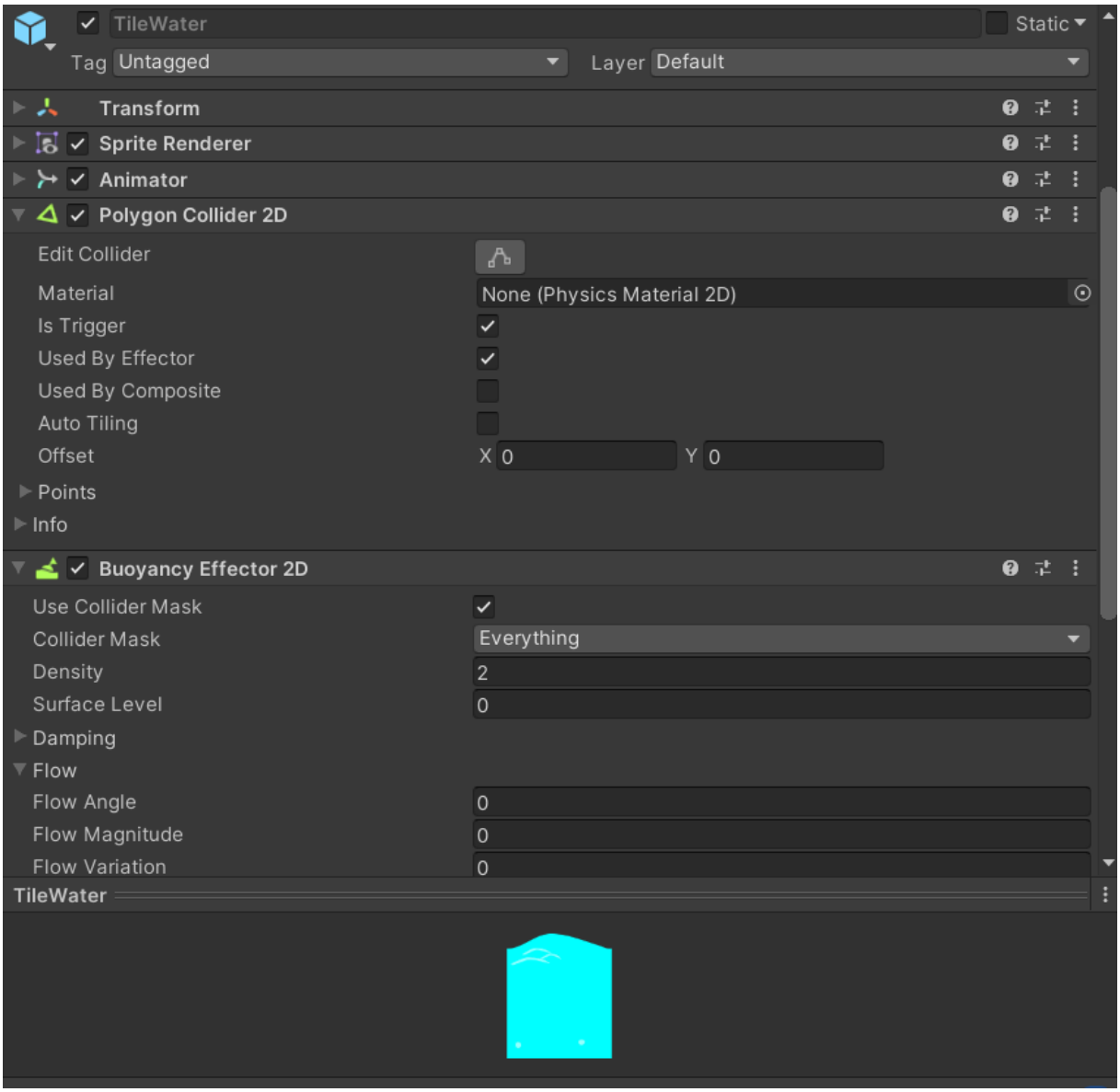
kako je postavljen te sprječava glavnog lika da prolazi kroz određeni prostor. Ukoliko dođe do kolizija sa glavnim likom, on će biti odbačen u smjeru ovisnom o tome do kakvog oblika kolizije je došlo. Ova zamka se sastoji od većinom do sad navedenih komponenti poput komponente „Sprite Renderer“, „Box Collider 2D“ te nove skripte koja se zove Rotate Clockwise. To je kratka skripta koja okreće objekata u smjeru kazaljke na satu ili obrnutom smjeru kao što se može vidjeti u nastavku.

```
private float rotZ;
    public float RotationSpeed;
    public bool ClockwiseRotation;

    void Update()
    {
        if(ClockwiseRotation == false){
            rotZ += Time.deltaTime * RotationSpeed;
        }
        else
        {
            rotZ -= Time.deltaTime * RotationSpeed;
        }

        transform.rotation = Quaternion.Euler(0, 0, rotZ);
```

U navedenoj skripti se koristi metoda Quaternion.Euler(x, y, z) koja je zadužena za to da se objekt rotira oko svojih xyz osi ovisno o tome koliko stupnjeva je dodano za koju os [18].

Iduća zamka je kamen od kojega se glavni lik odbija snažno te u većini slučajeva izlazi sa mape te korisnik gubi život. Kao i prijašnja zamka koristi komponente „Polygon Collider 2D“ i komponentu „Sprite Renderer“, a kako bi glavni lik odbijao od ovog objekta dodan je materijal sa velikom poskočnosti tako kada dođe do kolizije da se glavni lik može snažno odbiti [8].

Nakon te dvije zamke na desetoj razini igrač se susreće sa novom zamkom koja predstavlja vodu. Ta zamka, ukoliko glavni lik upadne u nju, gura korisnika ovisno o postavljenom smjeru. U nekim slučajevima ta zamka može pomoći igraču, a u nekim može i naštetiti njegovom pokušaju da riješi razinu. Kao i ostale zamke ima komponente „Sprite Renderer“ i „Polygon Collider“, no o ovom slučaju se kod „Collidera“ modificiraju vrijednosti atributa „Is Trigger“ i „Used By Effector“ koji su sada označeni potvrdno [8]. U tom slučaju se omogućava da objekt koristi komponentu „Buoyancy Effector 2D“ koji oponaša vodu u 2D svijetu [33]. Pomoću atributa „Surface Level“ određuje se površina objekta koja će predstavljati vrh vode, a modificirajući atribute „Flow Angle“ i „Flow Magnitude“ od varijable „Flow“ se odlučuje u kojem smjeru i koliko jako će ta voda odbacivati igrača kada upadne u nju. Posljednja komponenta koju sadrži je „Animator“ koji ima svoj „Animator Controller“ te pomoću njega animira vodu pomoću 3 slike koje se izmjenjuju svakih par sekundi. Izgled zamke se može vidjeti u narednoj slici [9].

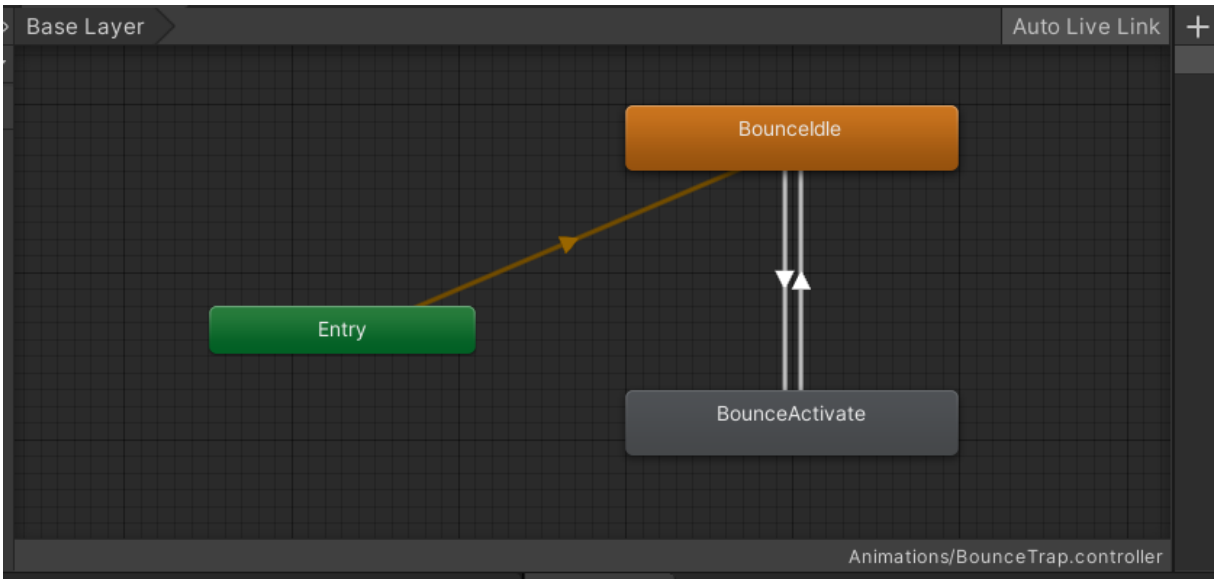


Slika 5 Prikaz bloka vode u editoru

Iduća zamka u prvom svijetu je BounceTrap. Kao i prethodne zamke ima komponente „Box Collider 2D“, „Sprite Renderer“ te također ima komponentu „Animator“ i skriptu Bouncing. Ta skripta detektira koliziju između te zamke i glavnog lika te kada dođe do kolizije, aktivira animaciju i odbacuje lika u zrak. Također ova zamka u nekim slučajevima može pomoći igraču, a u nekim slučajevima može i spriječiti ga na putu ka cilju. U narednoj skripti je opisano kako djeluje ta skripta.

```
[SerializeField] Animator anim;
private Vector2 Force;
private void OnCollisionEnter2D(Collision2D collision)
{
    if(collision.gameObject.tag == "Player")
    {
        anim.SetTrigger("onBounceAct");
        Force = new Vector2(0, 800);
        collision.rigidbody.AddForce(Force);
    }}
```

Skripta se sastoji od varijable anim tipa Animator koja se putem editora dodaje u skriptu, varijable Force tipa Vektor2 te metode OnCollisionEnter2D [9]. U slučaju kada dođe do kolizije sa glavnim likom, animacija se triggera te se dodaje sila objektu koji se sudario sa ovom zamkom, odnosno glavnom liku. Iz naredne slike se može vidjeti kako djeluje animator, zamka je u stanju mirovanja sve dok ne dođe do kolizije, nakon toga prelazi u stanje BounceActivate te odrađuje animaciju i vraća se u svoje početno stanje do iduće kolizije.



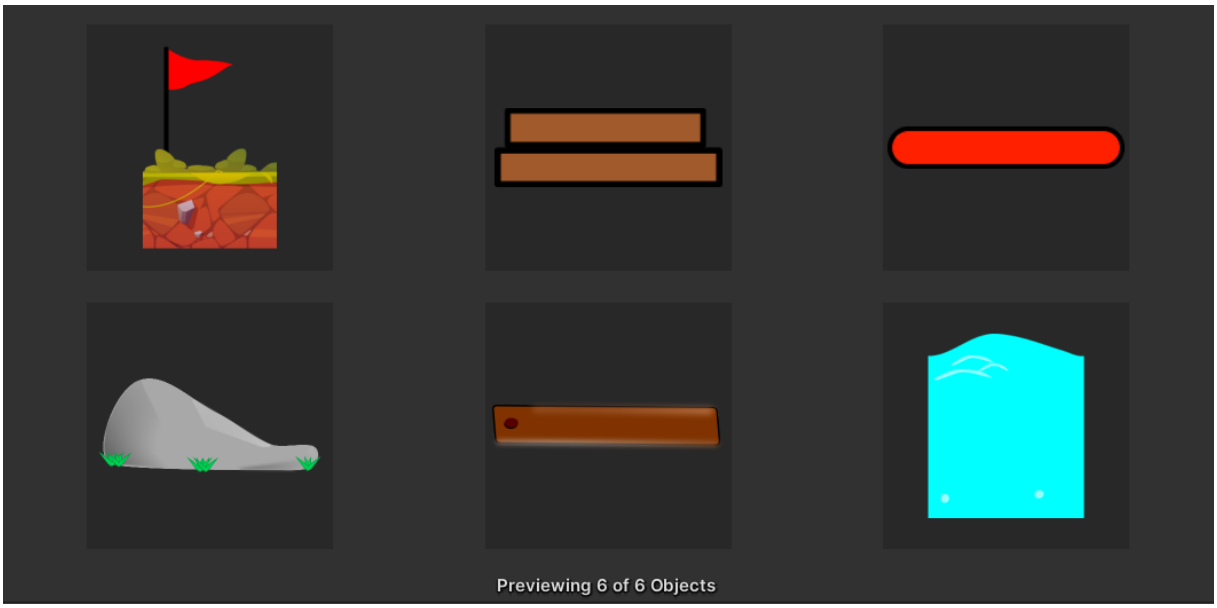
Slika 6 Animator zamke BounceTrap

Posljednja i najkompleksnija zamka prvog svijeta je zamka koja predstavlja oblik vrata koja se moraju otvoriti kako bi glavni lik mogao napredovati dalje kroz mapu. Vrata se otvaraju na način da prođe kroz zastavicu iste boje kao i što su vrata, te ukoliko ne uspije vrata se otvaraju i ne može proći dalje. Ova zamka se sastoji od 2 objekta te su različiti po strukturi komponenata. Prvi objekt je zastavica koja se sastoji od komponenata „Sprite Renderer“, „Polygon Collider“ koji ima postavljen atribut „Is Trigger“ te skripte „Blue Tile Script“. Skripta je jednostavnog oblika kao što se može vidjeti u nastavku.

```
[SerializeField] Animator animRed;
[SerializeField] GameObject go;

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Player")
    {
        animRed.SetTrigger("blueDoors");
        go.GetComponent<Collider2D>().enabled = false;
    }
}
```

Skripta se sastoji od varijabli animRed tipa Animator i go tipa GameObject. Varijabla animator predstavlja animator koji se nalazi na drugom objektu ove zamke, odnosno na vratima, a varijabla go predstavlja također ta vrata. U ovoj skripti se koristi nova metoda

OnTriggerEnter2D(Collider2D collision) koja provjerava da li je korisnik prošao kroz Collider te ukoliko je aktivira se animacija koja uklanja vrata te se uklanja komponenta „Box Collider 2D“ na tim vratima [19]. Drugi objekt zamke odnosno vrata se sastoje od komponenti „Box Collider 2D“, „Sprite Renderer“ i „Animator“.



Slika 7 Prikaz zamki prvog svijeta

Zamke drugog svijeta su bile zimske tematike kako je i sam svijet predstavljao zimu. Prva zamka su ledenice koje glavni lik ukoliko dotakne odmah gubi život te mora krenut ispočetka sa razinom. Komponente od kojih se sastoji su „Polygon Collider 2D“, „Sprite Renderer“ te skripta Spikes Script. Prikaz skripte se može vidjeti u nastavku.

```
 [SerializeField] GameObject lostPan;
[SerializeField] Animator lostAnim;
[SerializeField] Canvas canvas;
GameObject[] blokovi;
GameObject[] magneti;
GameObject[] trampolini;
GameObject[] ventilatori;
GameObject[] led;

void SakrijObjekte()
{  //Pronalazi sve objekte koje treba sakriti
    PronadjiSveBlokove();
    PronadjiSveMagnete();
    PronadjiSveTrampolie();
    PronadjiSveVentilatore();
    PronadjiLed();
    foreach (var item in blokovi)
    {
        item.GetComponent<SpriteRenderer>().sortingOrder = 0;
    }
    foreach (var item in magneti)
    {
```

```
            item.GetComponent<SpriteRenderer>().sortingOrder = 0;
        }
        foreach (var item in trampolini)
        {
            item.GetComponent<SpriteRenderer>().sortingOrder = 0;
        }
        foreach (var item in ventilatori)
        {
            item.GetComponent<SpriteRenderer>().sortingOrder = 0;
        }
        foreach (var item in led)
        {
            item.GetComponent<SpriteRenderer>().sortingOrder = 0;
        }
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if(collision.gameObject.tag == "Player")
        {
            DataBridge.Instance.dataInit.times_lost++;
            int a = DataBridge.Instance.dataInit.times_lost;
            DataBridge.Instance.UpdateTimesLost(a);
            canvas.planeDistance = 1;
            lostPan.SetActive(true);
            lostAnim.SetTrigger("lost");
            SakrijObjekte();
        }
     }
```

Kako glavni lik gubi život u slučaju da dođe u koliziju sa ovim objektom, u metodi OnCollisionEnter2D(Collision2D collision) potrebno je povećati u bazi broj puta koliko je korisnik izgubio te ažurirati taj broj, zatim prozor kada korisnik izgubi omogućiti, pokrenuti njegovu animaciju te na kraju sakriti objekte kako ne bi bili ispred toga prozora [15]. Ovo je skraćeni prikaz skripte jer nedostaje opis metoda koje pronalaze sve objekte koji se trebaju sakriti sa ekrana.

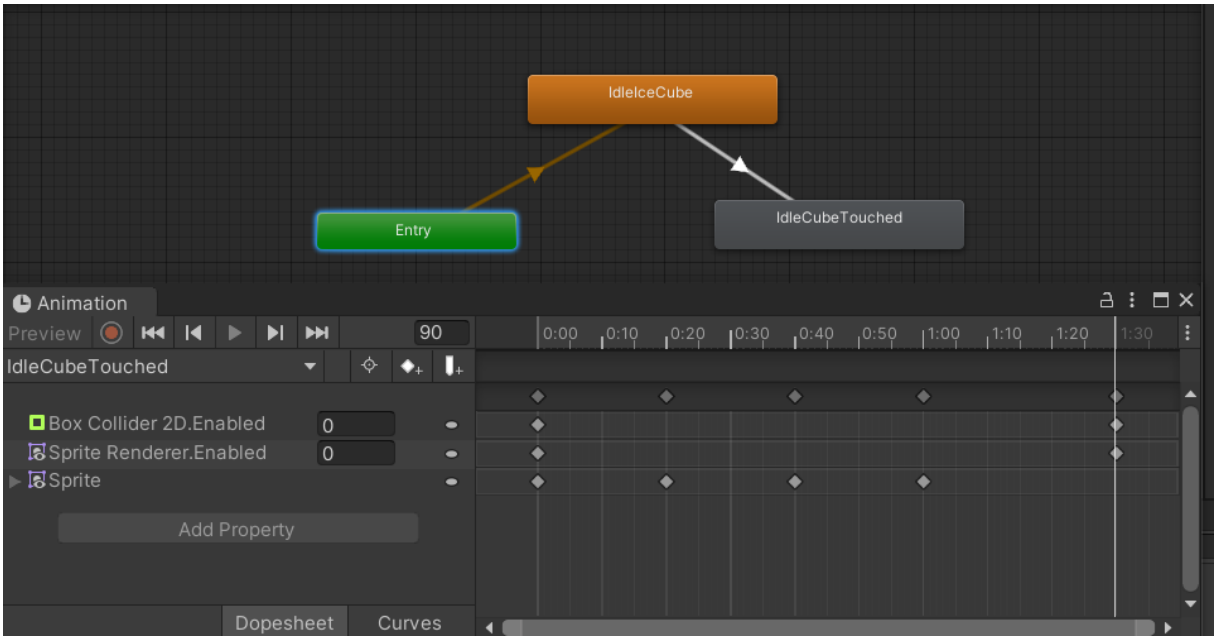
Iduća zamka je čekić koji se ljulja od jedne točke do druge točke te ukoliko dođe u doticaj sa glavnim likom odbacuje ga u smjer u kojem ga je udario. Komponente od kojih se sastoji su „Sprite Renderer“, „Box Collider 2D“ te skripte koja se zove Hummer Script. Na Collider-u je dodan materijal HummerBounce koji daje poskočnost glavnom liku u slučaju kolizije. Hummer Script služi za njihanje čekića od jedne točke do druge, a prikaz skripte je u nastavku.

```
    [SerializeField] protected Vector3 m_from = new Vector3(0.0f, 0.0f,
45.0f);
    [SerializeField] protected Vector3 m_to = new Vector3(0.0f, 0.0f, -
45.0f);
    [SerializeField] protected float m_frequency = 1.0F;
    void Update()
    {
        Quaternion from = Quaternion.Euler(this.m_from);
        Quaternion to = Quaternion.Euler(this.m_to);
```

```
        float lerp = 0.5F * (1.0F + Mathf.Sin(Mathf.PI *
Time.realtimeSinceStartup * this.m_frequency));
        this.transform.localRotation = Quaternion.Lerp(from, to, lerp);
    }
```

Skripta se sastoji od metode Update() i varijabli koje određuju frekvenciju koliko brzo će se čekić njihati te od koje do koje točke. Kako bi njihanje radilo koristi se metoda Quaternion.Lerp(x, y, z) koja određuje od koje točke će se objekt njihati te do koje točke i kojom brzinom [20].

Iduća zamka se zove Icecube, a predstavlja ledenu kocku koju ukoliko glavni lik dotakne, ona nakon par sekundi potpuno nestaje. Zamka se sastoji od komponenti „Sprite Renderer“, „Animator“, „Box Collider 2D“ te skripte Cube Script.



Slika 8 Uklanjanje komponenti putem animacija

S obzirom da je kratka skripta te samo pokreće animaciju IdleCubeTouched, prikazano je na slici kako ta animacija djeluje. Kroz određeno vrijeme animacija mijenja ilustraciju koja se prikazuje u komponenti „Sprite Renderer“ te na kraju isključuje komponente „Box Collider 2D“ i komponentu „Sprite Renderer“ kako ne bi moglo više doći u koliziju sa istim objektom te kako se taj objekt više ne bi vidio.

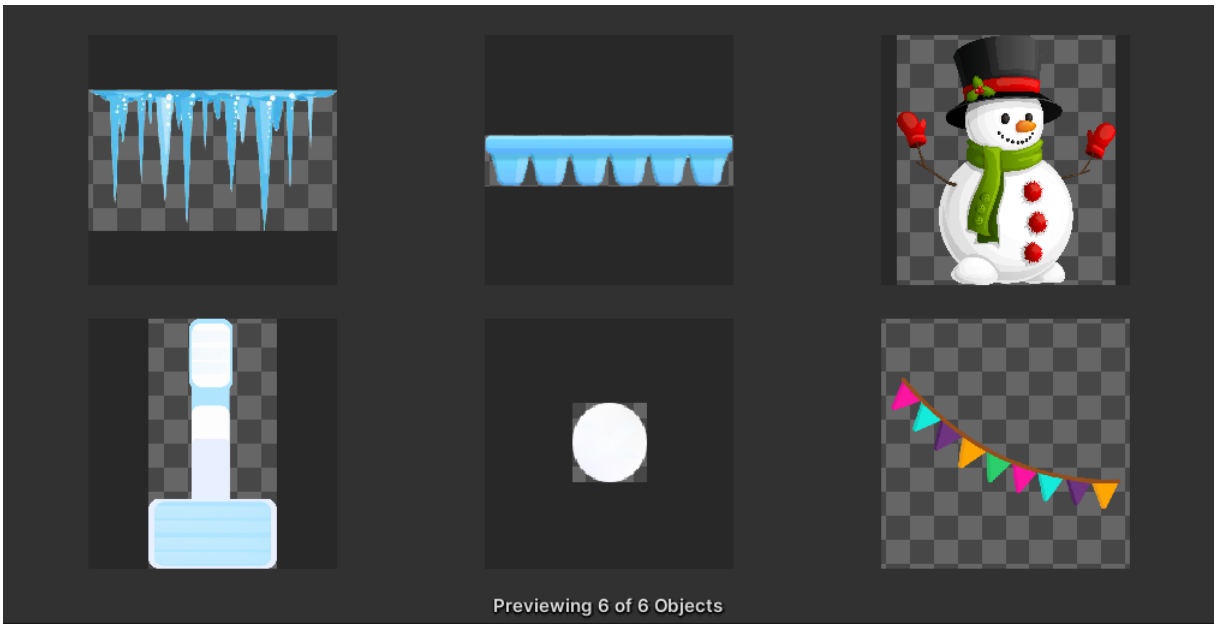
Iduća zamka sa kojom igrač dolazi u susret je ledena ploča koja se sastoji od komponenti „Sprite Renderer“, „Box Collider 2D“ te skripte Ice Script. Svrha zamke je da ukoliko glavni lik klizi po ploči dobije ubrzanje te da ga izbaci pod silom u stranu u kojoj se kreće. Izgled skripte se može vidjeti u nastavku te kao što je prikazano sastoji se od metode OnCollisionStay2D() koja u slučaju kada je kolizija neprekidna odrađuje neki zadatak, a u ovom slučaju je to dodavanje sile glavnom liku u stranu kojom se on kreće [21].

```
private void OnCollisionStay2D(Collision2D collision)
    {
        if(collision.gameObject.tag == "Player")
        {
            collision.rigidbody.AddForce(transform.right * 100f);
        }
    }
```

Sljedeća zamka je Snowman, odnosno snjegović, a cilj zamke je da privlači glavnog lika sebi te da mu ne dopušta da ide dalje kroz mapu. Sastoji se od komponenti „Sprite Renderer“, te dva Collider-a, „Circle Collider 2D“ i „Polygon Collider 2D“. Inače nije preporuka koristiti više Collider-a na jednom objektu, ali ovdje funkcionira. Razlog zašto je dodano više Collider-a je zato što se koristi komponenta „Point Effector 2D“, a ta je komponenta zadužena da u određenom radijusu privlači objekte sebi. Kako bi mogla privlačiti objekte komponentom „Circle Collider 2D“ je određen radijus u kojem će privlačiti objekte te se označeni atributi „Is Trigger“ i „Used By Effector“ [22]. Drugi Collider je zadužen za sami oblik objekta u koliko privuče loptu do snjegovića da dođe do kolizije sa njime.

Iduća zamka sa kojom se igrač susreće je uže koje ukoliko glavni lik dođe na njega, ono ga jako usporava da se ne može dalje kretati. Dodane komponente su „Sprite Renderer“ i „Polygon Collider 2D“. A za usporavanje lopti zadužen je materijal koji je dodan na Collider koji se zove DragMaterial te mu je povećana frikcija kako bi zaustavljao glavnog lika.

Posljednja zamka u zimskom svijetu su ledene grude koje se stvaraju i padaju na blokove te nakon par sekundi nestaju. Zamka se sastoji samo od komponente „Particle System. Ilustracija je dodana putem opcija „Renderer“ u tom sustavu, a kolizije su sređene putem opcija „Collision“. Nakon toga je bilo potrebno samo odabrati oblik i emisiju kojom brzinom će se stvarati ledena gruda te kojom brzinom će padati na zemlju. To je uređeno putem opcija „Emission“ i „Shape [23].



Slika 9 Prikaz zamki drugog svijeta

5.1.3. Izrada pomoćnih objekata za rješavanje razina

Nakon izraze zamki i blokova za razine slijedio je finalni dio početne izrade, a to je izrada objekata pomoću kojih bi igrač rješavao razine. Postoje četiri objekta uz pomoć kojih igrač rješava razinu, a to su: magnet, trampolin, kutija i ventilator.

Prvi i svojom vrijednosti najskuplji objekt je trampolin. Njegova vrijednost je jednaka pet jer je najkorisniji objekt stoga je i najskuplji. Komponente od kojih se sastoji su „Sprite Renderer“, „Capsule Collider 2D“, „Circle Collider 2D“, „Animator“ te tri skripte Trampolin Animation, Move Script i Rotation Script. Animator je zadužen za animaciju trampolina u slučaju kada dođe do kolizije glavnog lika sa njime, a skripta Trampolin Animation je zadužena za pokretanje te animacije kao što se može vidjeti u nastavku.

```
private Animator animator;

    private void Awake()
    {
        animator = GetComponent<Animator>();
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            animator.SetTrigger("isTouched");

        }
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        animator.ResetTrigger("isTouched");
    }
```

Skripta je jednostavnog oblika te se sastoji od već poznatih metoda OnCollisionEnter2D() i OnCollisionExit2D(). U slučaju kada dođe do kolizije sa glavnim likom pokreće se animacija, te kada prestane biti kolizije, trampolin se vraća na početnu animaciju. Iduća skripta od koje sa sastoji svaki od pomoćnih objekata za rješavanje razina je Move Script. Kao što naziv kaže skripta je zadužena za pomicanje objekta kada ga igrač želi pomjerat, izgled skripte je u nastavku.

```
    float deltaX, deltaY;
    bool moveAllowed = false;

    private Vector2 mousePos;
    void Update()
    {
        if (Input.touchCount > 0)
        {
           Touch touch = Input.GetTouch(0);
           Vector2 touchPos=Camera.main.ScreenToWorldPoint(touch.position);
```

```
            switch (touch.phase)
            {
                case TouchPhase.Began:
                    if (GetComponent<CircleCollider2D>() ==
Physics2D.OverlapPoint(touchPos))
                    {
                        deltaX = touchPos.x - transform.position.x;
                        deltaY = touchPos.y - transform.position.y;
                        moveAllowed = true;
                    }
                    break;
                case TouchPhase.Moved:
                    if (moveAllowed)
                    {
                        mousePos =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
                        transform.position = new Vector2(mousePos.x -
deltaX, mousePos.y - deltaY);
                    }

                    break;
                case TouchPhase.Ended:
                    moveAllowed = false;
                    break;
            }
        }
    }
```

Skripta radi na način da se koristi Vector2 mousePos varijabla kao pozicija koju igrač trenutačno dodiruje na ekranu [24]. U slučaju kada korisnik dodirne ekrane dohvaća se ta pozicija i kreće slučaj „TouchPhase.Began“. Tu se koristi „Circle Collider 2D“ koji se nalazi na objektu te u slučaju ako korisnik dodirne taj Collider i pomakne prstom prelazi se u slučaj „TouchPhase.Moved“ te se računa pozicija na kojoj bi se taj objekt trebao nalaziti i tako se taj objekt pomjera. Na kraju kada korisnik podigne prst sa ekrana onemogućava se kretanje do daljnjeg dodira [25]. Iduća skripta je Rotation Script koja se koristi za rotiranje objekta, a skripta je prikazana u nastavku.

```
    public float rotatespeed = 30f;
    private float _startingPositionX;
    bool allowed = false;

    void Update()
    {
        if (Input.touchCount > 0)
        {
            Touch touch = Input.GetTouch(0);
            Vector2 touchPos =
Camera.main.ScreenToWorldPoint(touch.position);
            RaycastHit2D hitInfo = Physics2D.Raycast(touchPos,
Camera.main.transform.forward);
            switch (touch.phase)
            {
                case TouchPhase.Began:
                    if (hitInfo.collider ==
GetComponent<CircleCollider2D>())
                    {
```

```
                        allowed = true;
                        _startingPositionX = touch.position.x;

                    }
                    else
                    {
                        allowed = false;
                    }
                    break;
                case TouchPhase.Moved:
                    if(allowed)
                    {
                        if (_startingPositionX > touch.position.x)
                        {
                            transform.Rotate(Vector3.forward * rotatespeed
* Time.deltaTime);
                        }
                        else if (_startingPositionX < touch.position.x)
                        {
                            transform.Rotate(Vector3.forward * -rotatespeed
* Time.deltaTime);
                        }


                    }
                    break;
                case TouchPhase.Ended:
                    allowed = true;
                    break;
            }
        }
    }
```

Skripta funkcionira na sličan način kao i Move Script. Očitava se dodir korisnika te se provjerava da li je korisnik dotaknuo prstom komponentu „Circle Collider 2D“. Razlika je što se provjerava da li je korisnik dodirnuo Collider putem varijable RaycastHit2D hitInfo. On djeluje na način da se provjerava korisnikov dodir na način kao da je laser uperen u ekran i provjerava koji objekt je dodirnut [26]. Ukoliko je došlo do dodira i pomaka prstom računa se za koliko bi se objekt trebao rotirati te se rotira ovisno o igračevom kretanju prstom.

Idući objekt je kutija koja ima vrijednost dva, a komponente od koje se sastoji su „Sprite Renderer“, „Circle Collider 2D“, „Box Collider 2D“ te već spomenutih skripti Rotation Script i Move Script. Kao što je spomenuto, komponenta „Circle Collider 2D“ služi za pomicanje i rotiranje samog objekta, a komponenta „Box Collider 2D“ služi da se glavni lik može sudarati sa njime [8].

Sljedeći objekt je ventilator sa vrijednošću tri. Komponenta koje ga čine su „Sprite Renderer“, „Animator“, „Circle Collider 2D“, „Box Collider 2D“, „Particle System“ te spomenutih skripti Rotation Script i Move Script te skripte vezane samo za ovaj objekt, a to je Blowing Fan skripta. Komponenta „Animator“ animira objekt cijelo vrijeme te prikazuje ventilator kako puše, „Particle System“ služi tomu da prikazuje zrak u kojem smjeru trenutačno ventilator puše.

Također komponenta „Box Collider 2D“ ima označen atribut „Is Trigger“ kako bi u trenutku kada glavni lik uđe u taj Collider sila vjetra djelovala na njega. Skripta Blowing Fan je zadužena za puhanje objekta, a izgled se može vidjeti u nastavku.

```
private Vector2 Force;
    private List<Collider2D> objects = new List<Collider2D>();
    void FixedUpdate()
    {
        for (int i = 0; i < objects.Count; i++)
        {
            Rigidbody2D body = objects[i].attachedRigidbody;
            if((transform.rotation.eulerAngles.z % 360 >= 0 &&
transform.rotation.eulerAngles.z % 360 <= 22.5) ||
(transform.rotation.eulerAngles.z % 360 <= -337.5 &&
transform.rotation.eulerAngles.z % 360 >= -360))
            {
                Force = new Vector2(40, 10);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 22.6 &&
transform.rotation.eulerAngles.z % 360 <= 45) ||
(transform.rotation.eulerAngles.z % 360 <= -315 &&
transform.rotation.eulerAngles.z % 360 >= -337.4))
            {
                Force = new Vector2(40, 20);
                body.AddForce(Force);
            }
            else if((transform.rotation.eulerAngles.z % 360 >= 45.1 &&
transform.rotation.eulerAngles.z % 360 <= 90) ||
(transform.rotation.eulerAngles.z % 360 <= -270 &&
transform.rotation.eulerAngles.z % 360 >= -314.9))
            {
                Force = new Vector2(30, 60);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 90.1 &&
transform.rotation.eulerAngles.z % 360 <= 135) ||
(transform.rotation.eulerAngles.z % 360 <= -225 &&
transform.rotation.eulerAngles.z % 360 >= -269.9))
            {
                Force = new Vector2(-30, 60);
                body.AddForce(Force);
            }
            else if((transform.rotation.eulerAngles.z % 360 >= 135.1 &&
transform.rotation.eulerAngles.z % 360 <= 157.5) ||
(transform.rotation.eulerAngles.z % 360 <= -202.5 &&
transform.rotation.eulerAngles.z % 360 >= -224.9))
            {
                Force = new Vector2(-40, 20);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 157.6 &&
transform.rotation.eulerAngles.z % 360 <= 180) ||
(transform.rotation.eulerAngles.z % 360 <= -180 &&
transform.rotation.eulerAngles.z % 360 >= -202.4))
            {
                Force = new Vector2(-40, 10);
                body.AddForce(Force);
            }
```

```
            else if ((transform.rotation.eulerAngles.z % 360 >= 180.1 &&
transform.rotation.eulerAngles.z % 360 <= 202.5) ||
(transform.rotation.eulerAngles.z % 360 <= -157.5 &&
transform.rotation.eulerAngles.z % 360 >= -179.9))
            {
                Force = new Vector2(-40, -10);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 202.5 &&
transform.rotation.eulerAngles.z % 360 <= 225) ||
(transform.rotation.eulerAngles.z % 360 <= -135 &&
transform.rotation.eulerAngles.z % 360 >= -157.4))
            {
                Force = new Vector2(-40, -20);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 225.1 &&
transform.rotation.eulerAngles.z % 360 <= 270) ||
(transform.rotation.eulerAngles.z % 360 <= -90 &&
transform.rotation.eulerAngles.z % 360 >= -134.9))
            {
                Force = new Vector2(-30, -60);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 270.1 &&
transform.rotation.eulerAngles.z % 360 <= 315) ||
(transform.rotation.eulerAngles.z % 360 <= -45 &&
transform.rotation.eulerAngles.z % 360 >= -89.9))
            {
                Force = new Vector2(30, -60);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 315.1 &&
transform.rotation.eulerAngles.z % 360 <= 337.5) ||
(transform.rotation.eulerAngles.z % 360 <= -22.5 &&
transform.rotation.eulerAngles.z % 360 >= -44.9))
            {
                Force = new Vector2( 40, -20);
                body.AddForce(Force);
            }
            else if ((transform.rotation.eulerAngles.z % 360 >= 337.6 &&
transform.rotation.eulerAngles.z % 360 <= 360) ||
(transform.rotation.eulerAngles.z % 360 <= 0 &&
transform.rotation.eulerAngles.z % 360 >= -22.4))
            {
                Force = new Vector2(40, -10);
                body.AddForce(Force);
            }

        }
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        objects.Add(other);
    }

    void OnTriggerExit2D(Collider2D other)
    {
        objects.Remove(other);
    }
```

Iako je skripta ogromna u suštini je jednostavna. Koristi se metoda FixedUpdate() koja je prilagođenija radu sa fizikom u 2D i 3D svjetovima od metode Update() [16]. U toj metodi se provjerava trenutačni kut ventilatora te ovisno o kutu ventilatora se dodaje sila glavnom liku koji ulazi u „Box Collider 2D“. Prilikom ulaska u tu komponentu poziva se metoda OnTriggerEnter2D() te se objekt dodaje u listu koja kasnije se poziva u metodi FixedUpdate(). Nakon izlaska iz Collider-a, objekt se miče iz liste u metodi OnTriggerExit2D() [16].

Posljednji objekt uz pomoću kojeg se rješavaju razine je magnet. Magnet je sličan već spomenutom snjegoviću te je njegova vrijednost jedan. Sastoji se od komponenti „Sprite Renderer“, „Particle System“, „Circle Collider 2D“, „Box Collider 2D“, „Point Effector 2D“ te skripti Move Script i Rotation Script. „Particle System“ komponenta prikazuje u kojem smjeru magnet trenutačno djeluje te pridodaje estetici samog objekta. „Circle Collider 2D“ služi za rotiranje i za pomicanje kao i kod prijašnjih objekata. Komponenta „Box Collider 2D“ ima označene atribute „Is Trigger“ i „Used By Effector“ kako bi komponenta „Point Effector 2D“ mogla koristiti taj Collider. Kao što je spomenuto „Point Effector 2D“ djeluje na način da stvara privlačnu silu te tako objekt pomjera u smjeru prema sebi.

Slika 10 Prikaz pomoćnih objekata za rješavanje razina

5.2. Izrada razina i korisničkog sučelja za razine

Nakon što su pripremljeni svi blokovi, zamke, mogući likovi, te pomoćni objekti za rješavanje razina slijedila je izrada razina i korisničkog sučelja za razine. Kako bi razine funkcionirale također je bilo potrebno napisati skripte za određene funkcionalnosti u razinama.

5.2.1. Korisničko sučelje za razine

Izrada korisničkog sučelja je počela sa kreiranjem kanvasa na prvoj razini. Nakon što je kreiran kanvas prva dodana stvar je gumb za vraćanje nazad. Kako bi taj gumb i ostali funkcionirali bilo je potrebno stvoriti prazni objekt koji bi služio kao GameManager odnosno koji bi upravljao sa većinom funkcionalnosti na svakoj razini. Kreiravši taj upravitelj za razine

dodana je skripta naziva Skripta za Lopte u kojoj je dodana metoda koja se poziva pritiskom na gumb nazad, a izgled metode se može vidjeti u nastavku. Metoda se zove OnBackPress() te koristi metodu StartCoroutine() [27]. Tom metodom se poziva IEnumerator Povratak() koji se također može vidjeti u nastavku.

```
public void OnBackPress()
    {
        StartCoroutine(Povratak());
        int b = PlayerPrefs.GetInt("boja");
        if (b == 0)
        {
            music.gameObject.GetComponent<AudioSource>().clip = mainM;
            music.gameObject.GetComponent<AudioSource>().volume = 0.3f;
            music.gameObject.GetComponent<AudioSource>().Play();
        }
        else
        {
            music.gameObject.GetComponent<AudioSource>().clip = mainM;
            music.gameObject.GetComponent<AudioSource>().volume = 0f;
            music.gameObject.GetComponent<AudioSource>().Play();
      }
    }
private IEnumerator Povratak()
    {
        canvas.planeDistance = 1;
        canvas.sortingOrder = 10;
        gameeObj.GetComponent<Animator>().SetTrigger("end");
        yield return new WaitForSeconds(3f);
        SceneManager.LoadScene("LevelMenuScene");
    }
```

O metodi Povratak() se poziva animator od objekta koji je zadužen za animiranje otvaranja i zatvaranja razina i nakon 3 sekunde se otvara scena za razine koja se zove „LevelMenuScene“ pomoću metode SceneManager.LoadScene() [28]. A prije nego što se to izvrti u metodi OnBackPress() putem PlayerPrefs.GetInt() se provjerava da li je korisnik na MainMenu scene odabrao da svira zvuk ili je ugasio zvuk i ovisno o tom slučaju se dalje izvršava metoda. PlayerPrefs je klasa koja se inače koristi za spremanje preferencija igrača između sesija [29].

Nakon izrade tog gumba slijedilo je stvaranje prikaza sa zvjezdicama i sa rezultatom koji je korisnik trenutačno skupio. Nakon što je to sve kreirano kasnije je dodano u skripti naziva Win Script metoda ProvjeraZvijezda() koja je provjeravala korisnikov rezultat, spremala ga u bazu, provjeravala je da li je korisnik riješio neki od postignuća itd, a ova metoda se pozivala samo u slučaju ako glavni lik dođe do cilja. S obzirom da je metoda ogromna, a i cijela skripta prikazan je samo mali dio koda u nastavku.

```
void ProvjeraZvijezda()
    {
        int score, threeScore, twoScore, oneScore;
        Metoda(out score, out threeScore, out twoScore, out oneScore);
        foreach (var item in data.GetComponent<SkriptaZaPod>().lvls)
```

```
        {
            if (item.levelName == name)
            {
                if (item.score > score)
                {
                    if (score >= 0 && score <= threeScore)
                    {
                        canvas.planeDistance = 1;
                        canvas.sortingOrder = 10;
                        winPanel.SetActive(true);
                        anim.SetTrigger("threeStars");
                        panelScore.text = currScore.text;
                        coins.text = currCoins.text;
                        diamonds.text = currDiamonds.text;
                        coins.SetAllDirty();
                        diamonds.SetAllDirty();
                        n = 3;
                        UpLeveli(score, item);
                        if (item.solved == 0)
                        {
                            DataBridge.Instance.dataInit.levels_solved++;
                            n = DataBridge.Instance.dataInit.levels_solved;
                            DataBridge.Instance.UpdateLevelsSolved(n);
                            if (name == "Level1")
                            {
                                var obj1 =
data.GetComponent<SkriptaZaPod>().achi.FirstOrDefault(x => x.id_user ==
DataBridge.Instance.dataInit.id &&
x.achievementName.Equals("Achievement1"));
                                obj1.unlocked = 1;

DataBridge.Instance.UpdateAchievementLocked("1");
                                achAnim.SetTrigger("noEnergy");
                            }
                            else if (name == "Level15")
                            {
                                var obj1 =
data.GetComponent<SkriptaZaPod>().achi.FirstOrDefault(x => x.id_user ==
DataBridge.Instance.dataInit.id &&
x.achievementName.Equals("Achievement2"));
                                obj1.unlocked = 1;

DataBridge.Instance.UpdateAchievementLocked("2");
                                achAnim.SetTrigger("noEnergy");
                            }
                            else if (name == "Level24")
                            {
                                var obj1 =
data.GetComponent<SkriptaZaPod>().achi.FirstOrDefault(x => x.id_user ==
DataBridge.Instance.dataInit.id &&
x.achievementName.Equals("Achievement3"));
                                obj1.unlocked = 1;

DataBridge.Instance.UpdateAchievementLocked("3");
                                achAnim.SetTrigger("noEnergy");
                            }
                            else if (name == "Level25")
                            {
                                var obj1 =
data.GetComponent<SkriptaZaPod>().achi.FirstOrDefault(x => x.id_user ==
```

```
DataBridge.Instance.dataInit.id &&
x.achievementName.Equals("Achievement12"));
                                obj1.unlocked = 1;

DataBridge.Instance.UpdateAchievementLocked("12");
                                achAnim.SetTrigger("noEnergy");
                            }
                            else if (name == "Level33")
                            {
                                var obj1 =
data.GetComponent<SkriptaZaPod>().achi.FirstOrDefault(x => x.id_user ==
DataBridge.Instance.dataInit.id &&
x.achievementName.Equals("Achievement13"));
                                obj1.unlocked = 1;

DataBridge.Instance.UpdateAchievementLocked("13");
                                achAnim.SetTrigger("noEnergy");
                            }
                            else if (name == "Level48")
                            {
                                var obj1 =
data.GetComponent<SkriptaZaPod>().achi.FirstOrDefault(x => x.id_user ==
DataBridge.Instance.dataInit.id &&
x.achievementName.Equals("Achievement14"));
                                obj1.unlocked = 1;

DataBridge.Instance.UpdateAchievementLocked("14");
                                achAnim.SetTrigger("noEnergy");
                            }
                        }
                        ProvjeraAchievementsaUlisti();
                        item.score = score;
                        item.oneStar = 1;
                        item.twoStar = 1;
                        item.threeStar = 1;
                        item.solved = 1;
                        DataBridge.Instance.UpdateOneStar(name);
                        DataBridge.Instance.UpdateTwoStar(name);
                        DataBridge.Instance.UpdateThreeStar(name);
                        DataBridge.Instance.UpdateSolved(name);
                        DataBridge.Instance.UpdateScore(name,
int.Parse(currScore.text.ToString()));
                        SakrijObjekte();
                        brojac++;
                          }
```

Ukratko ovaj dio koda provjerava da li je korisnik riješio razinu za 3 zvjezdice, te ako je riješio onda poziva animaciju za pobjedu, ažurira podatke na toj animaciji, te zatim provjerava da li je ova razina jedna od razina koja treba biti riješena kako bi otključao postignuće te ako je ažurira podatke u bazi za to postignuće i na kraju ažurira podatke u bazi za rezultat, koliko zvijezda je skupio te da je razina riješena.

Poslije sučelja za rezultat napravljeni su gumbi za pomicanje i za rotiranje koji su opisani već prije. U slučaju da korisnik pritisne gumb za rotiranje, svi objekti za rješavanje razina se mogu samo rotirati, ne mogu se micati, te tako i suprotno radi. Osim tih gumba dodan je i gumb za ponavljanje razine. Prilikom pritiska na taj gumb poziva se metoda OnPressRetry() koja provjerava da li korisnik ima dovoljno energije, te ako ima dovoljno

energije onda restarta razinu u suprotnom se pojavljuje notifikacija na ekranu da igrač nema više energije. Prikaz kako skripta točno radi se može vidjet u nastavku. MetodaProvjere() ažurira bazu podataka za vrijednosti koliko korisnik ima energije, odnosno oduzima mu se jedna energija, te se povećava za jedan koliko je sveukupno energije iskoristio kako bi otključao postignuća vezana uz potrošnju energije.

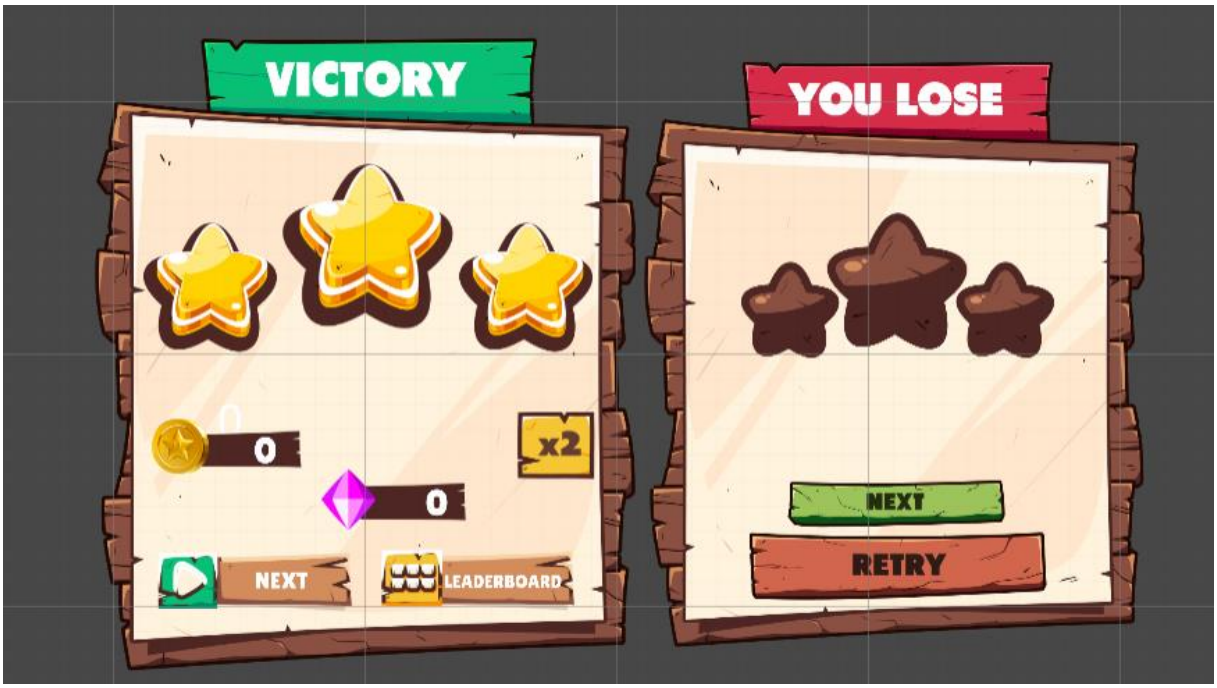
```
public void OnPressRetry()
    {
        if(DataBridge.Instance.dataInit.current_energy > 1)
        {
            MetodaProvjere();
            StartCoroutine(MetodaScene());
        }
        else
        {
            anim.SetTrigger("noEnergy");
        }
    }

    private IEnumerator MetodaScene()
    {
        yield return new WaitForSeconds(0.1f);
        SceneManager.LoadScene(name);
    }
```

Iduća komponenta u kanvasu su bile ilustracije za objekte te kako pritiskom na njih povući i stvoriti kopiju njihovog objekta. Stvorena su četiri gumba za svaki objekata te pritiskom na njih i povlačeći prstom se stvara taj objekt i istovremeno se ažurira rezultat u sučelju. Na kraju je dodano dugme Play. Prilikom pritiska na to dugme poziva se više metoda. Jedna od metoda je EnableMoveRotate.DisableAll() koja je već spomenuta i ona onemogućava da se pomoćni objekti više mogu pomjerati ili rotirati. Zatim metoda AnimationButtons.BtnPlayPressed() koja je zadužena za animiranje cijelog sučelja te se svaka komponenta sučelja animira u tome trenutku osim gumba za restart razine, gumba za povratak nazad i panele sa zvjezdicama i rezultatom. U nastavku je prikazana kako izgleda metoda BtnPlayPressed().

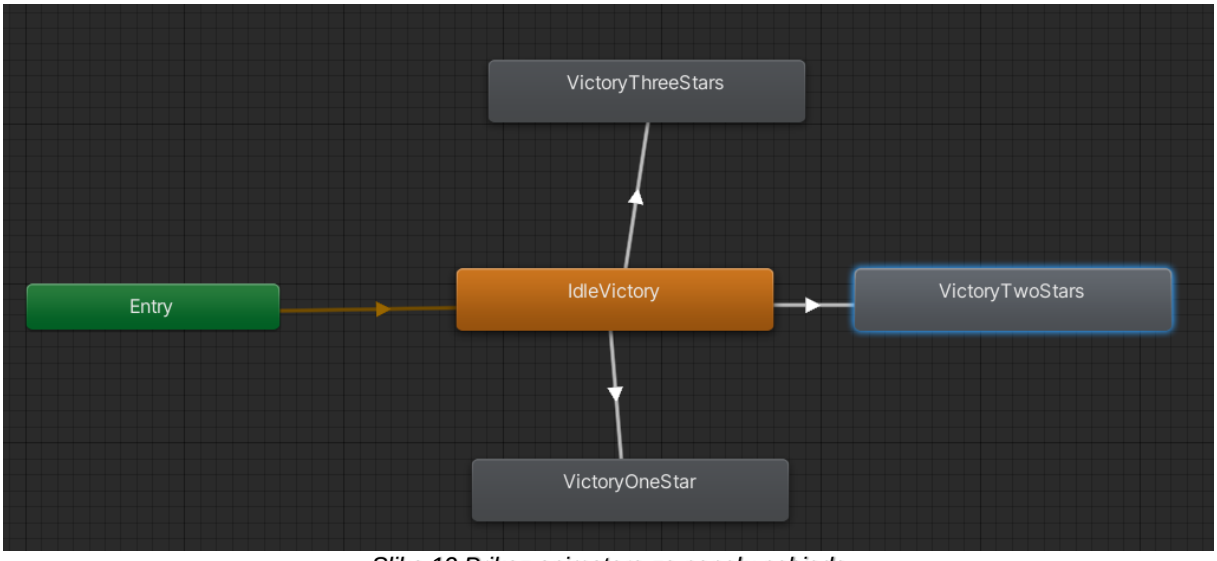
```
public void BtnPlayPressed()
    {
        animBtnPlay.SetTrigger("onPress");
        btnPlay.enabled = false;
        animBtnMove.SetTrigger("onPress");
        btnMove.enabled = false;
        animBtnRotate.SetTrigger("onPress");
        btnRotate.enabled = false;
        animDaske.SetTrigger("onPress");
        animGOimg.SetTrigger("onPress");
        animBtnPrav.SetTrigger("onPress");
        animBtnTramp.SetTrigger("onPress");
        animBtnFan.SetTrigger("onPress");
        animBtnMagn.SetTrigger("onPress");
        animArrow.SetTrigger("onPress");
        btnArrow.enabled = false;
```

```
smece.SetActive(false);
  }
```

Nakon što je dodano sve u sučelje na kraju su dodani paneli za pobjedu i paneli kada igrač izgubi. Uz to su paneli i animirani, a pogotovo panel za pobjedu koji je animiran ovisno o tome da li je korisnik riješio razinu za jednu, dvi ili tri zvijezde.



Slika 11 Paneli za pobjedu i gubitak

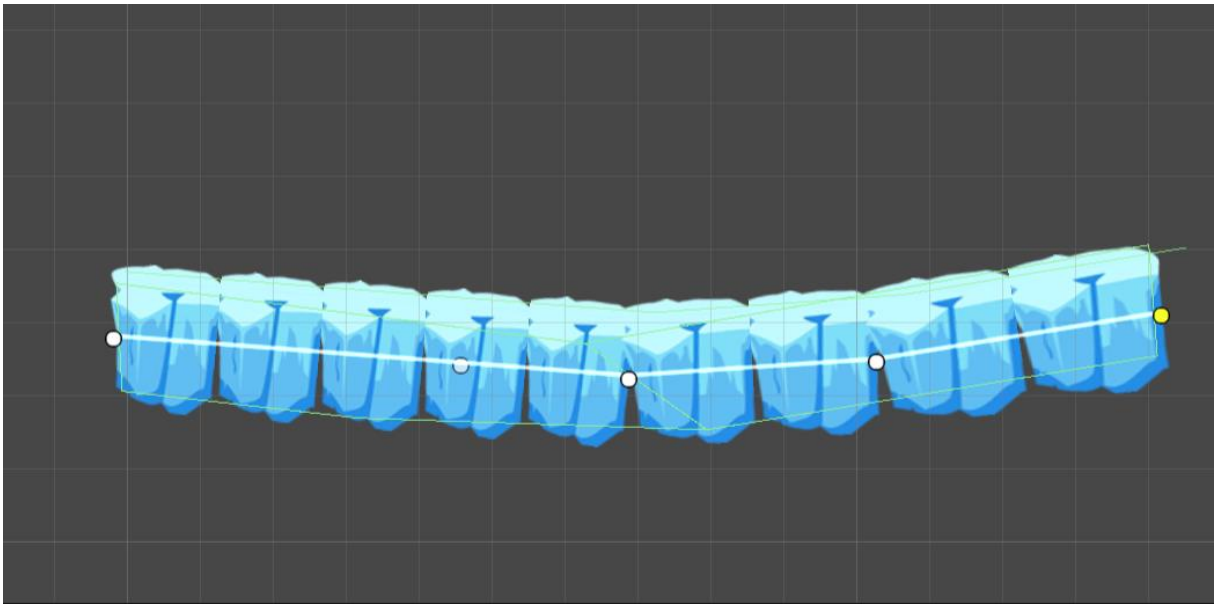


Slika 12 Prikaz animatora za panelu pobjede

Nakon što je riješeno cijelo korisničko sučelje dodana je još jedna metoda koja bi provjeravala koju loptu korisnik trenutačno koristi i na osnovu toga bi stvaralo loptu te vrste kao glavnog lika, a metoda se zove ProvjeraLopte(). Sa time je riješeno korisničko sučelje za razine i krenula je izrada razina za oba svijeta krenuvši od prvog.

5.2.2.Izrada razina

Prije početka izrade svake razine bilo je prvo potrebno smisliti kako će svaka razina funkcionirati, koje bi zamke trebale biti na kojem mjestu te otprilike smisliti neko od rješenja kako bi se mogao zadati rezultat koji igrač treba dostići za tri zvijezde, a zatim i za dvi zvijezde te za jednu zvijezdu. Kada bi se odredili cilj i početak dalje bi išla izrada razine dodavanjem blokova koji će se nalaziti na toj razini. Kako bi bilo olakšano dodavanje blokova korišten je alat Unity Sprite Shape uz kojeg je olakšano uređivanje oblika platformi. Na slici ispod je prikazano kako izgleda korištenje tog alata.



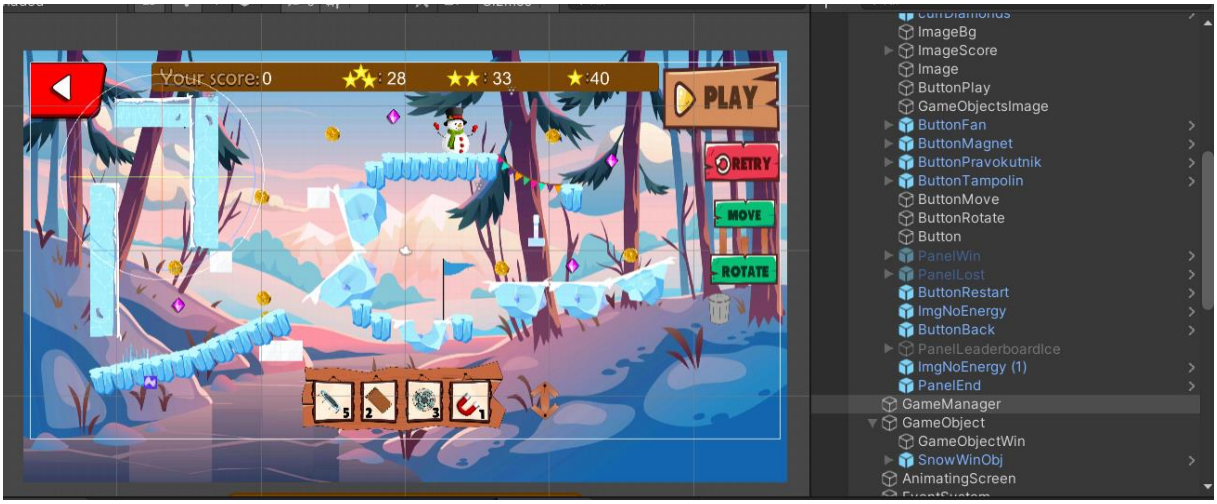
Slika 13 Prikaz alaza Unity Sprite Shape

Na slici je prikazano kako imamo jedan blok koji ukoliko razdužujemo sa klikom miša, automatski stvara nove blokove iste prethodnima te ih povezuje međusobno da izgledaju kao jedan objekt. Iduća stvar koju je bilo potrebno dodati nakon blokova i cilja bili su novčići i dijamanti koje će igrač skupljati. Svaka razina ovisno o svojoj težini je imala zadan broj novčića, a dijamanti su se javljali svaku četvrtu razinu u oba svijeta s obzirom da im je vrijednost veća od novčića. Na kraju uređivanja svake razine kada su svi blokovi bili postavljeni bilo je potrebno dobro odrediti početak gdje će se lopta nalaziti zato što je se lopta učitavala iz baze putem skripte te da ne bi došlo do greške u kojoj lopta završi u bloku te se ne može pomjerati. Metoda koja je zadužena za taj dio nalazi se na GameManager-u te se zove ProvjeraLopte(). Pozicija na koju je lopta bila kreirana bila je pozicija di je GameManager bio postavljen stoga je na svakoj razini se GameManager postavljao kao početna pozicija gdje će se lopta nalaziti. Mali isječak iz koda je u nastavku kako metoda ProvjeraLopte() radi.

```
private void ProvjeraLopte()
    {
        foreach (var item in data.GetComponent<SkriptaZaPod>().shops)
        {
```

```
            if(item.used == 1)
            {
                if (item.ballname == "Kosarkaska")
                {
                    if(GameObject.FindGameObjectWithTag("Player") != null)
                    {
                        //Do nothing
                    }
                    else
                    {
                        Vector3 newVec = transform.position;
                        Instantiate(kosarksaska, newVec,
Quaternion.identity);
                    }

                }
```

Kao što je prikazano u skripti prvo se provjeravaju sve lopte koje korisnik ima u bazi, te ona lopta koju je korisnik postavio da se koristi ima vrijednost Used jednak jedan, a ostale imaju tu vrijednost na nuli. Kada dođe do lopte sa tom vrijednosti provjerava se njen naziv te kada dođe do istog naziva ulazi se u metodu koja stvara novu loptu te vrste koja je odabrana. Prijašnji if uvjet je zadan iz razloga što je znalo doći do situacije kada je više lopti bilo kreirano iste vrste te je s tim uvjetom taj problem riješen. Sa time je jedna razina bila riješena te su se onda istim redoslijedom kreirale druge razine, a primjer razine u editoru se može vidjeti na idućoj slici.



Slika 14 Primjer razine u editoru

5.3. Izrada ostalih scena

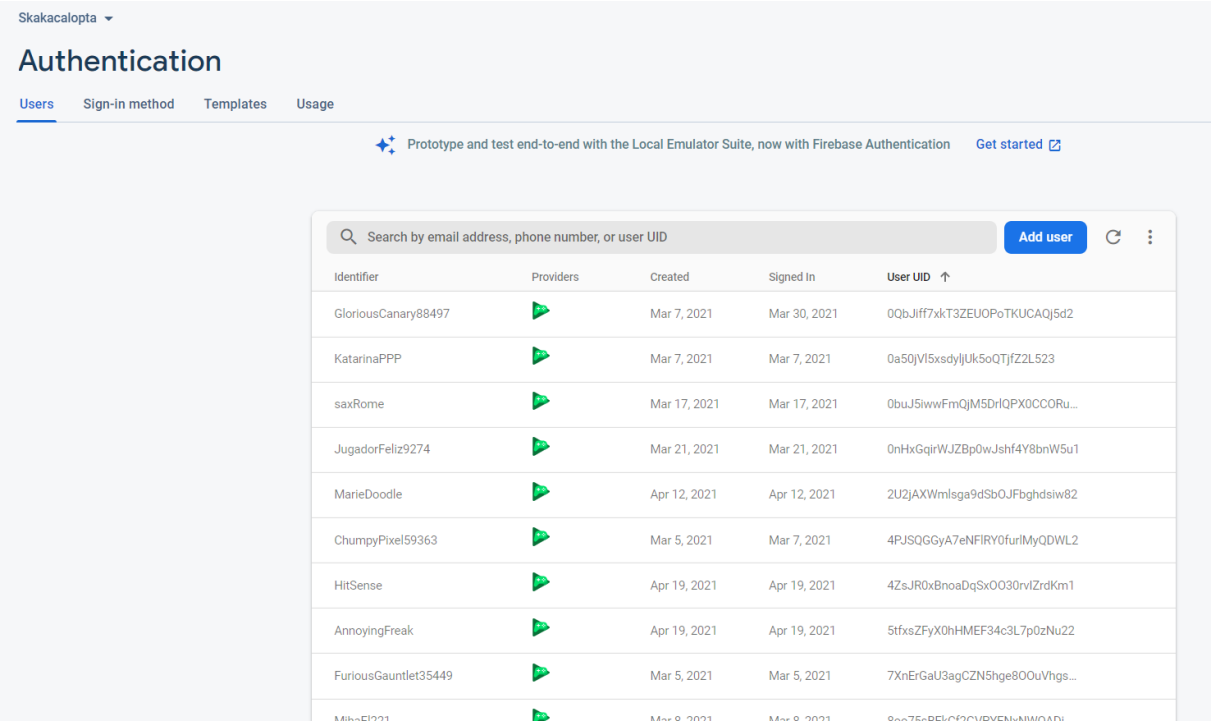
Nakon što su sređene sve razine bilo je potrebno napraviti Login scenu, Main Menu scenu i Level scenu. Također su se morale urediti i omogućiti funkcionalnosti koje trebaju nuditi.

5.3.1. Login scena

Prva na redu je bila Login scena za koju je bilo potrebno prvo odrediti na koji način će korisnici se prijavljivati u igricu. S obzirom da je korišten Firebase u projektu dodana je Firebase autentikacija putem Google Play-a. Autentikacija putem Firebase-a je jednostavna te kada se prijavi projekt na Firebase i odabere se ta autentikacija postoje upute kako izvršiti autentikaciju te se doda skripta koja će to odrađivati. Skripta zadužena za prijavljivanje je GooglePlayAuthentication, a glavna metoda je FirebasePlayGames() koja je prikazana u nastavku.

```
void FirebasePlayGames()
    {
        Firebase.Auth.FirebaseAuth auth =
Firebase.Auth.FirebaseAuth.DefaultInstance;


        Social.localUser.Authenticate((bool success) => {
            if (!success)
            {

                return;
            }
            authCode = PlayGamesPlatform.Instance.GetServerAuthCode();
            if (string.IsNullOrEmpty(authCode))
            {

                return;
            }
            Firebase.Auth.Credential credential =

Firebase.Auth.PlayGamesAuthProvider.GetCredential(authCode);
            auth.SignInWithCredentialAsync(credential).ContinueWith(task =>
{
                if (task.IsCanceled)
                {

                    Debug.LogError("SignInWithCredentialAsync was
canceled.");
                    return;
                }
                if (task.IsFaulted)
                {

                    Debug.LogError("SignInWithCredentialAsync encountered
an error: " + task.Exception);
                    return;
                }
                Firebase.Auth.FirebaseUser newUser = task.Result;
                DataBridge.Instance.UserID = newUser.UserId;
                DataBridge.Instance.UserEmail = newUser.Email;
                DataBridge.Instance.UserName = newUser.DisplayName;
                DataBridge.Instance.ProvjeraK();
                StartCoroutine(LoadScenes());

            });
        });
    }
```

Novi kod koji je dodan u ovu skriptu je povezan sa spremanjem novih korisnika u bazu i staranje nove scene koji je na kraju ove metode. Prikaz kako izgleda spremanje korisnika putem Google Play računa u bazu se vidi na idućoj slici.



Slika 15 Prikaz spremljenih podataka od prijave korisnika u bazi

Slika 16 Login scena

5.3.2. Main Menu scena

Nakon što se prijava automatski izvrši u Login sceni otvara se nova scena, a to je Main Menu scena. U toj sceni korisnik može vidjeti trenutačnu energiju koju ima, koliko novčića i

dijamanata posjeduje te u slučaju ako je riješio neko postignuće može skupiti novčiće i dijamante predodređene za rješavanje tog postignuća. Ukoliko ima dovoljnu količinu novaca može otići u trgovinu i kupiti novu loptu sa kojom će igrati. Posljednje opcije koje se nude korisniku su da može ugasiti zvuk ili upaliti ovisno o njegovoj želji te također može pritisnuti gumb Play i otići u Level Menu scenu te igrati razine koje slijede. Kako bi sve opcije funkcionirale dodan je objekt GameManager sa skriptom koja upravlja sa svim aktivnostima na toj sceni. Zadužena je za skupljanje postignuća, za kupovanje novih lopti i za ažuriranje podataka u bazi. Izgled Main Menu scene je prikazan u nastavku.

Slika 17 Main Menu scena

Ukoliko korisnik riješi jedno od postignuća, dobit će notifikaciju dok rješava razine. Kada dobije tu notifikaciju, u bazi se ažuriralo navedeno postignuće te mu je omogućeno da skupi zaslužene novčiće i kovanice. Pritiskom na gumb Achievements otvorit će se postignuća i postignuće koje je otključano će biti izrazito zelene boje te će se razlikovati od ostalih postignuća kako bi korisnik znao koje postignuće može pokupiti. Jednom kada pokupi neko postignuće, ne može ga ponovno skupiti. Izgled panele postignuća se može vidjeti nastavku na slici.

Slika 18 Prikaz postignuća

Iduća stvar poslije postignuća je panela trgovine na kojoj korisnik može kupovati nove likove sa kojima će igrati. Korisnik na Main Menu sceni u svakom trenutku može vidjeti točan broj novčića i dijamanata koji posjeduje. Kada skupi dovoljno pritiskom na gumb Store se otvara panela trgovine sa svim mogućim loptama koje može kupiti. Ukoliko nema dovoljno novčića ili dijamanata, a pokuša kupiti neku loptu dobiti će notifikaciju kako trenutačno nema dovoljno novčića ili dijamanata ovisno o lopti koju je pokušao kupiti. Izgled panele trgovine se može vidjeti na idućoj slici.

Slika 19 Prikaz trgovine

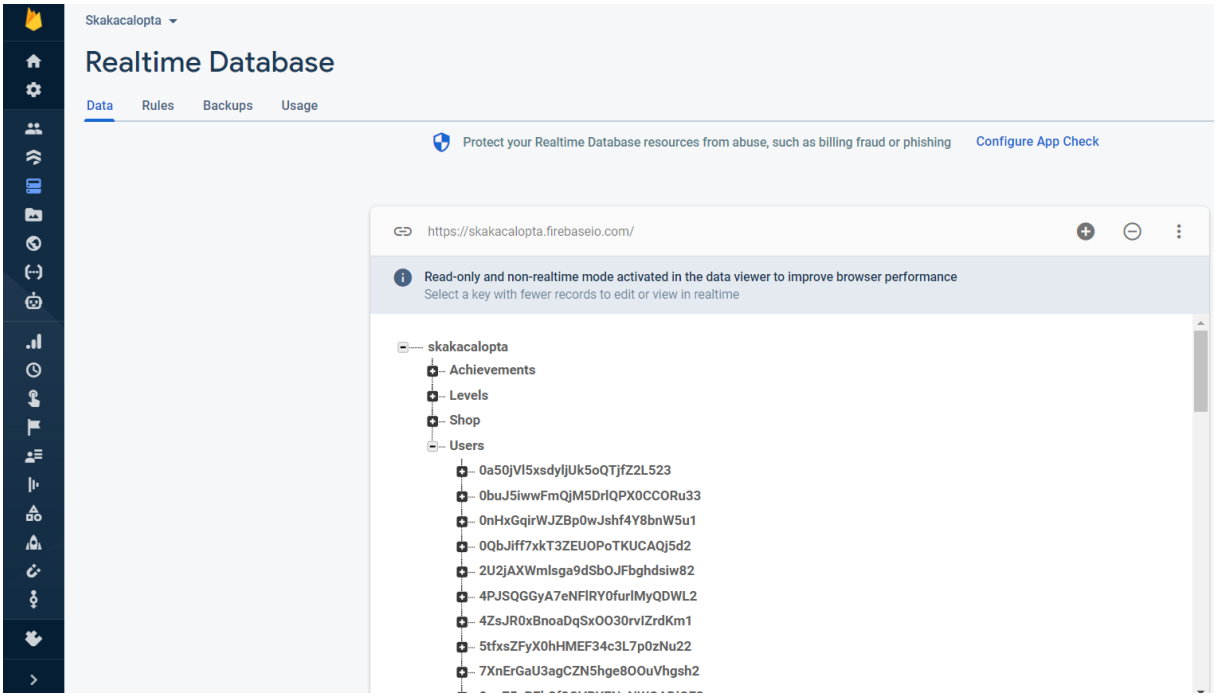
5.3.3. Level Menu scena

Pritiskom na gumb Play otvara se Level Menu scena koju čine dva svijeta sa svojim razinama. Ta scena također sadrži objekt GameManager koji je zadužen za upravljanje sa svim opcijama na toj sceni. Igrač kada riješi jednu razinu tek tada je u mogućnosti da ide na iduću razinu, a da bi došao do drugog svijeta potrebno je riješiti sve razine od prethodnog svijeta. Prikaz Level Menu scene je vidljiv na idućoj slici.

Slika 20 Level Menu scena

5.4. Baza podataka

Kako bi sve funkcioniralo i kako bi podaci bili ažurni koristila se „Realtime Database“ baza podataka od Firebase-a. Kao što je već spomenuto baza sprema podatke u obliku JSON objekata te je oblika NoSQL baze podataka. Baza se sastoji od četiri entiteta, a to su Users, Shop, Levels i Achievements. Entiteti su povezani na način da je u svaki entitet spreman korisnikov id i onda putem tog id-a su se dohvaćali podaci ovisno o potrebi. Primjer kako izgleda baza podataka može se vidjeti na idućoj slici.



Slika 21 Prikaz baze podataka

Metode za rad sa bazom podataka su spremane u posebnu klasu te su se dohvaćale iz te klase po potrebi. Klasa se zove DataBridge, a primjeri nekih od metoda za rad sa bazom podataka se mogu vidjeti u nastavku.

```
public void UpdateThreeStar(string ime1)
    {
        string ime = ime1;
        string json = "1";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Levels"
).Child(UserID).Child(ime).Child("threeStar").SetValueAsync(json);
    }
    public void UpdateSolved(string ime1)
    {
        string ime = ime1;
        string json = "1";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Levels"
).Child(UserID).Child(ime).Child("solved").SetValueAsync(json);
    }
```

```
    public void UpdateScore(string ime1, int score)
    {
        string ime = ime1;
        string json = score.ToString(); ;

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Levels"
).Child(UserID).Child(ime).Child("score").SetValueAsync(json);
    }

    public void SaveData()
    {
        DataInit data = new DataInit(UserID, UserEmail, UserName, 0, 0, 0,
21, 0, 0);
        var jsonData = JsonUtility.ToJson(data);

databaseReference.Child("Users").Child(UserID).SetRawJsonValueAsync(jsonDat
a);
    }


    public void UpdateDataBallsBought(string ime1)
    {
        string ime = ime1;
        string json = "1";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Shop").
Child(UserID).Child(ime).Child("bought").SetValueAsync(json);
    }

    public void UpdateDataBallsUsedOn(string ime1)
    {
        string ime = ime1;
        string json = "1";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Shop").
Child(UserID).Child(ime).Child("used").SetValueAsync(json);
    }

    public void UpdateDataBallsUsedOff(string ime1)
    {
        string ime = ime1;
        string json = "0";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Shop").
Child(UserID).Child(ime).Child("used").SetValueAsync(json);
    }



    public void UpdateAchievementLocked(string i)
    {
        string n = "Achievement" + i;
        string json = "1";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Achieve
ments").Child(UserID).Child(n).Child("unlocked").SetValueAsync(json);
    }

    public void UpdateAchievementCollect(string i)
    {
        string n = "Achievement" + i;
```

```
        string json = "1";

FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child("Achieve
ments").Child(UserID).Child(n).Child("collected").SetValueAsync(json);
    }
```

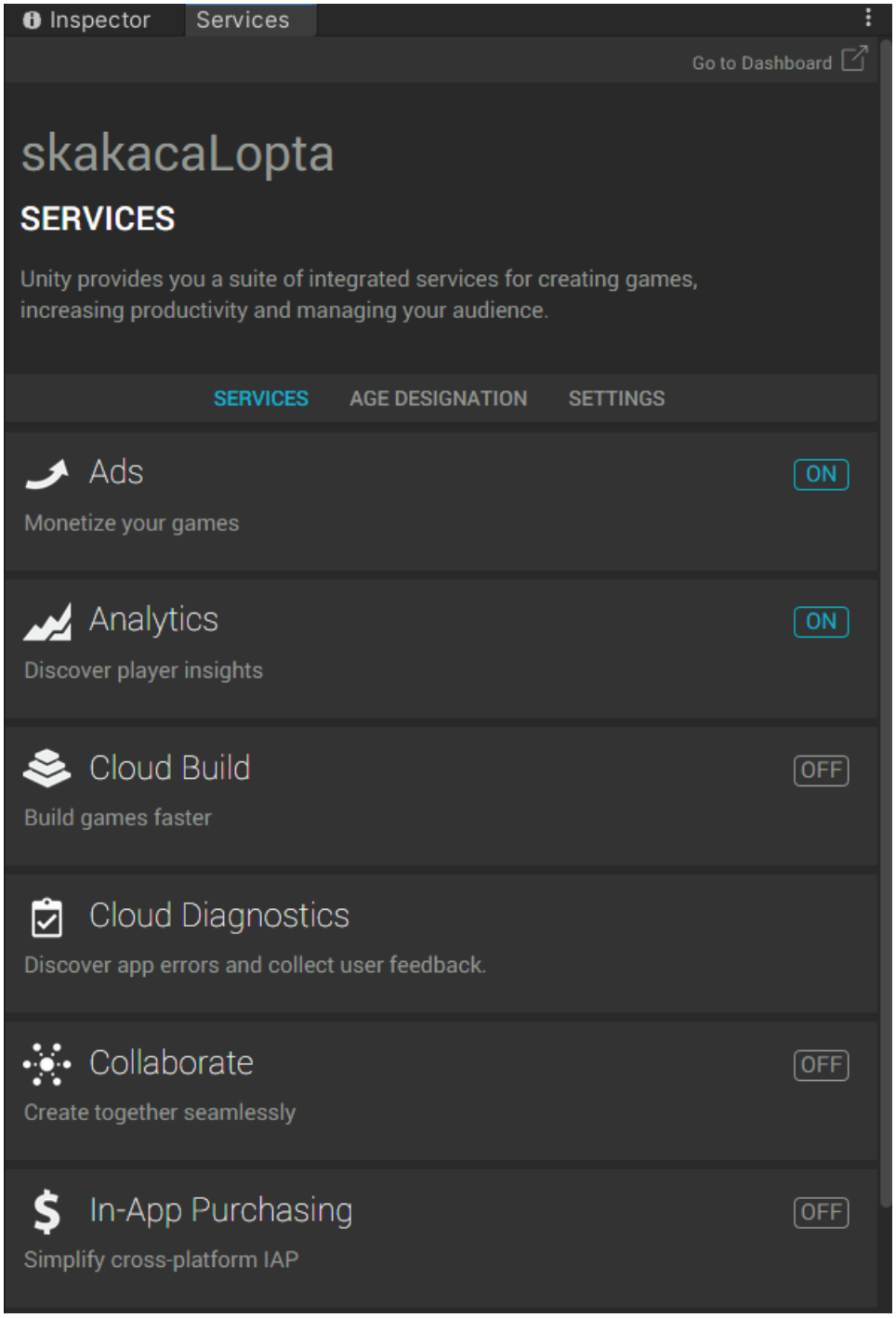
Za ažuriranje podataka koristi se metoda sa podacima kao u nastavku: FirebaseDatabase.DefaultInstance.GetReferenceFromUrl(DB_URL).Child(„Entitet").Child(„korisnikov id“).Child(„ime razine/postignuća“).Child(„ime podatka koji se sprema").SetValueAsync(json);.

6. Izdavanje i monetizacija igre

Nakon što je igra napravljena te testirana na vlastitom uređaju slijedilo je izdavanje same igre te dodavanje reklama u igru kako bi igrica zarađivala na reklamama koje igrači gledaju.

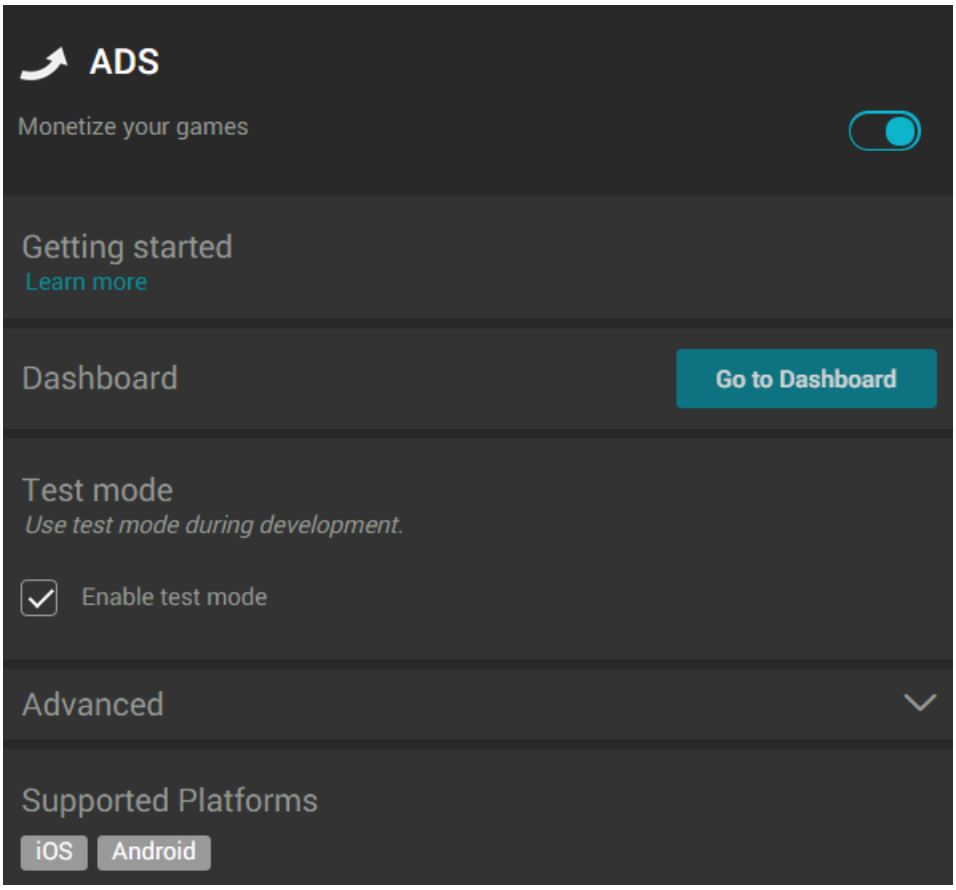
6.1. Reklame

Za dodavanje reklama u igricu najprije je potrebno omogućiti servise za igru. Servisi se omogućuju tako da se ode na opciju „Window“ te nakon toga „Unity Services“ u glavnom izborniku. Prilikom odabira te opcije otvara se novi prozor sa servisima koje nudi Unity kao što je prikazano na idućoj slici [34].



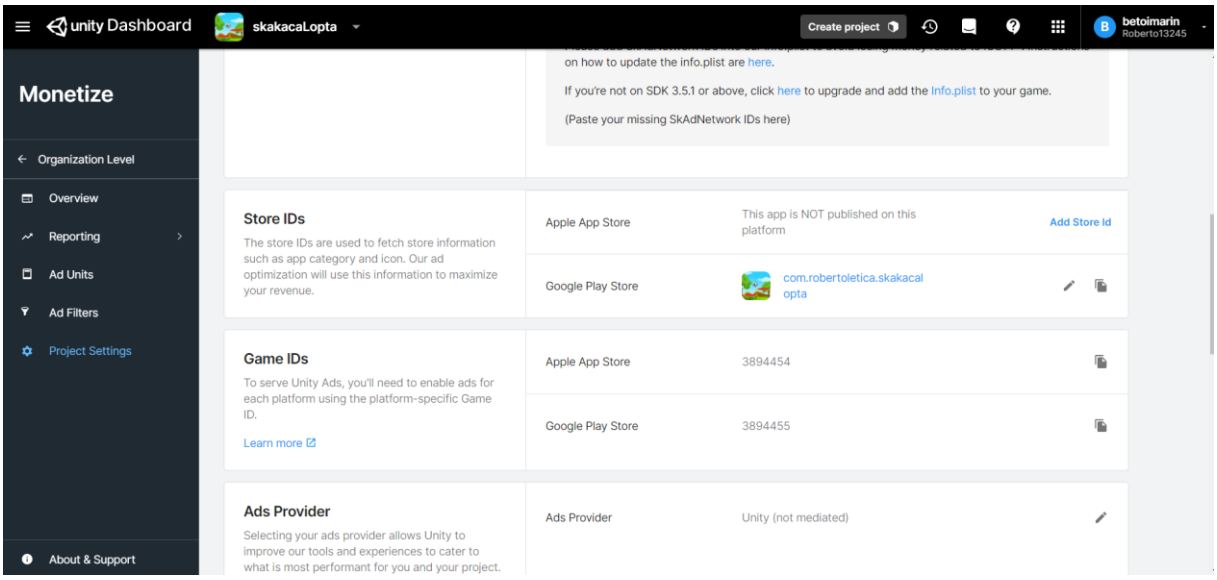
Slika 22 Unity servisi

U priloženom izborniku se odabere opcija Ads i otvori se novi izbornik te se uključi servis za reklame, a izbornik je prikazan na idućoj slici.



Slika 23 Ads izbornik

Nakon podešavanja servisa slijedi dodavanje koda. Kod je već generiran od strane Unity-a te ga je potrebno samo implementirat u vlastiti projekt. Klasa se zove AdManager() koja se dodaje na objekt koji će se prenositi kroz sve scene te će se ta klasa pozivati u onim slučajevima kad je zadano da se reklame pozivaju. Kao finalni dio dodavanja reklama treba ispuniti par stavki u Unity dashboard-u na web-u. Primjer podataka koji se trebaju ispuniti su vidljivi na slici, ali oni se mogu dodati tek nakon što se igrica izbaci na trg play.

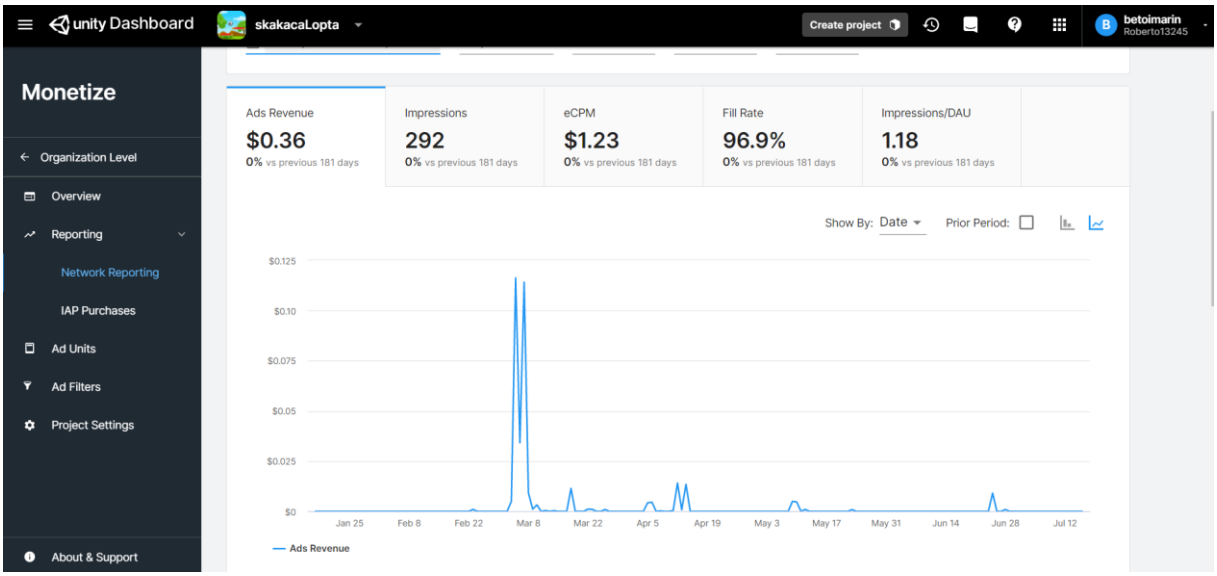


Slika 24 Unity dashboard

6.2. Izdavanje i zarada na igri

Za izdavanje igre potrebno je prvo napraviti programerski račun na Google Play konzoli, a poslije izrade računa izdavanje igre je jednostavno. Prate se osnovni koraci te kada se doda aplikacijska verzija od igrice, Google Play provjerava da li je sve u redu sa igrom te u slučaju ako ne krši nikakva pravila izdaje se kroz nekih tjedan dana na Google Play Store.

Zarada na reklamama se provjerava na Unity dashboard-u u opciji Network Reporting u izborniku. Kao što se vidi na idućoj slici, igra je zaradila ukupno 0.36$ na 292 pogledane reklame s time da je većina reklama bila nagradnog tipa te se te reklame iz toga vrednuju više s obzirom da korisnik mora pogledati cijelu reklamu kako bi dobio nagradu, ako ne pogleda cijelu reklamu neće biti nagrađen. Dodana su 2 tipa nagradnih reklama, gdje korisnik može odabrati nakon svake razine da mu se uduplaju novčići i dijamant ili da u Main Menu sceni pritisne gumb za dodavanje energije. Reklame koje nisu nagradnog tipa su dodane kod razina te svaki sedmi put kada korisnik izgubi pojavljuje se reklama no tu reklamu nakon 5 sekundi korisnik može preskočiti stoga one puno ne doprinose kod zarade.



Slika 25 Zarada na igrici

7. Zaključak

Za izradu igre u Unity programu bez ikakvog predznanja o samom software-u potrebno je predznanje barem u pisanju skripti u C#-u koje je meni uvelike olakšalo izradu ovog završnog rada. Osim predznanja za kodiranje dobro je znati koristit resurse koje nudi Internet danas, mnogo različite kurseve, online knjige i ostale primjere koji nakon 10 sati provođenja na njima mogu korisnika dobro upoznati sa programom i samim razvojem igara. Također prije početka izrade same igre najbitnije je dobro odrediti sve funkcionalnosti i kako će igrica raditi tako da ne bi došlo do nekih velikih izmjena nakon par mjeseci koje mogu dodatno zakomplicirati završavanje igre. Za izradu ove igre trebalo mi je 6 mjeseci no u tom razdoblju je vrijeme korišteno na učenje dosta stvari, pogotovo na učenje drugih programa uz Unity koji su pomagali pri izradi igre. Odabrao sam ovaj tip igrice zato što mi se sviđaju misaone igre te sam vjerovao da ovakav tip igrice može uspjeti na tržištu zato što nisam vidio takav primjer do sada. No, kao što je vrijeme pokazalo svoje, igrica nije uspjela no postoji mnogo razloga zašto nije uspjela. Za izradu uspješne igre potrebno je dosta više vremena ukoliko osoba sama programira i radi te dosta više znanja kako bi to sve brže išlo. Također postoje mnoge stvari u ovoj igri koje su mogle biti unaprijeđene, ali svejedno sumnjam da bez dobre marketinške kampanje igrica može uspjeti na svjetskoj razini. Kao glavni zaključak bi izvukao to da se ulaganje vremena u izradu igre isplati, jer se dobije puno više od gotovog proizvoda na kraju. Dobije se odlično znanje i iskustvo kako bi rad na igricama trebao teći te nakon što sve bude napravljeno, uvidi se kolike promijene pri razvoju su mogle ići drugačije i kako na kraju bi najbolje bilo, a smatram da je to najbolja nagrada, naučiti na vlastitim greškama.

8. Literatura

[1]*„What is Firebase?“* [Na internetu] Dostupno: https://howtofirebase.com/what-is-firebase-fcb8614ba442 Pristupano: 11.07.2021

[2] „*What is Visual Studio?*“ [Na internetu] Dostupno: https://www.incredibuild.com/integrations/visual-studio Pristupano: 12.07.2021

[3] „*What is Unity?*“ [Na internetu] Dostupno: https://www.androidauthority.com/what-is-unity-1131558/ Pristupano: 12.07.2021

[4] „*2D and 3D mode settings*“ [Na internetu] Dostupno: https://docs.unity3d.com/Manual/2DAnd3DModeSettings.html Pristupano: 12.07.2021

[5] „*Camera*“ [Na internetu] Dostupno: https://docs.unity3d.com/Manual/class-Camera.html Pristupano: 12.07.2021

[6] „*Sprites“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/Sprites.html Pristupano: 12.07.2021

[7] „*Rigidbody 2D*“ [Na internetu] Dostupno: https://docs.unity3d.com/Manual/class-Rigidbody2D.html Pristupano: 12.07.2021

[8] „*Collider 2D“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/Collider2D.html Pristupano: 12.07.2021

[9] „*Animator“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/class-Animator.html Pristupano: 12.07.2021

[10] „Animator Controller“ [Na internetu] Dostupno: https://docs.unity3d.com/Manual/class-AnimatorController.html Pristupano: 12.07.2021

[11] „*Audio Source“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/class-AudioSource.html Pristupano: 12.07.2021

[12] „*Action video games“* [Na internetu] Dostupno: https://gamicus.fandom.com/wiki/Action_video_games Pristupano: 13.07.2021

[13] „*Strategy video games“* [Na internetu] Dostupno: https://gamicus.fandom.com/wiki/Strategy_video_games Pristupano: 13.07.2021

[14] „*Puzzle video games“* [Na internetu] Dostupno: https://gamicus.fandom.com/wiki/Puzzle_video_games Pristupano: 13.07.2021

[15] *Collider2D.OnCollisionEnter2D*() Dostupno: https://docs.unity3d.com/ScriptReference/Collider2D.OnCollisionEnter2D.html Pristupano: 14.07.2021

[16] *ExecutionOrder* Dostupno: https://docs.unity3d.com/Manual/ExecutionOrder.html Pristupano: 14.07.2021
[17] [*SerializeField]* Dostupno: https://docs.unity3d.com/ScriptReference/SerializeField.html Pristupano: 14.07.2021
[18] *Quaternion.Euler()* Dostupno: https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html Pristupano: 14.07.2021
[19] *Collider2D.OnTriggerEnter2D() Dostupno: https://docs.unity3d.com/ScriptReference/Collider2D.OnTriggerEnter2D.html Pristupano: 14.07.2021*
[20] *Quaternion.Lerp() Dostupno: https://docs.unity3d.com/ScriptReference/Quaternion.Lerp.html Pristupano: 14.07.2021*
[21] Collider2D.*OnCollisionStay2D() Dostupno:* https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnCollisionStay2D.html *Pristupano: 15.07.2021*
[22] *Point Effector 2D Dostupno:* https://docs.unity3d.com/Manual/class-PointEffector2D.html *Pristupano: 15.07.2021*
[23] *Particle System Dostupno:* https://docs.unity3d.com/ScriptReference/ParticleSystem.html *Pristupano: 15.07.2021*
[24] *Vector2 Dostupno:* https://docs.unity3d.com/ScriptReference/Vector2.html *Pristupano: 15.07.2021*
[25] *TouchPhase Dostupno:* https://docs.unity3d.com/ScriptReference/TouchPhase.html *Pristupano: 15.07.2021*
[26] *RaycastHit2D Dostupno:* https://docs.unity3d.com/ScriptReference/RaycastHit2D.html *Pristupano: 15.07.2021*
[27] *StartCoroutine Dostupno:* https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html *Pristupano: 15.07.2021*
[28] *SceneManager.LoadScene() Dostupno:* https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html *Pristupano: 15.07.2021*
[29] *PlayerPrefs Dostupno:* https://docs.unity3d.com/ScriptReference/PlayerPrefs.html *Pristupano: 15.07.2021*
[30] *2D Sprite Shape Dostupno:* https://docs.unity3d.com/Packages/com.unity.2d.spriteshape@3.0/manual/index.html *Pristupano: 16.07.2021*
[31] *Trail Renderer Dostupno:* https://docs.unity3d.com/Manual/class-TrailRenderer.html *Pristupano: 17.07.2021*

[32] *Sprite Renderer Dostupno:* https://docs.unity3d.com/ScriptReference/SpriteRenderer.html *Pristupano: 17.07.2021*

[33] *Sprite Renderer Dostupno:* https://docs.unity3d.com/Manual/class-BuoyancyEffector2D.html *Pristupano: 17.07.2021*

[34] *Unity Ads Dostupno:* https://docs.unity3d.com/550/Documentation/Manual/UnityAdsHowTo.html *Pristupano: 17.07.2021*

9. Popis slika

10. Popis korištenih resursa

10.1. Programski Alati

- Program za izradu vlastite glazbe: FL Studio, https://www.image-line.com/ Pristupano: 16.02.2021
- Program za pisanje i uređivanje koda: Microsoft Visual Studio, https://visualstudio.microsoft.com/ Pristupano: 17.07.2021
- Program za uređivanje i izradu ilustracija: Inkscape, https://inkscape.org/hr/ Pristupano: 16.07.2021
- Web stranica za rad s bazom: Firebase, https://firebase.google.com/ Pristupano: 17.07.2021
- Web stranica za izdavanje igre i dodavanje servisa: Google Play Console https://play.google.com/apps/publish Pristupano: 17.07.2021
- Program za izradu igre: Unity, https://unity.com/ Pristupano: 17.07.2021

10.2. Unity Assets

- Pjesme u igri: vlastita izrada
- Ilustracije za lopte i pomoćne objekte: vlastita izrada
- Blokovi i pozadine: naručeno i kupljeno putem stranice Fiverr, https://www.fiverr.com/ Pristupano: 15.03.2021