Izrada glazbene videoigre u programskom alatu Unity

Slavik, David

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

David Slavik

IZRADA GLAZBENE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

David Slavik

JMBAG: 0016148382

Studij: Informacijski i poslovni sustavi

IZRADA GLAZBENE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor :

Doc. dr. sc. Mladen Konecki

Varaždin, rujan 2023.

David Slavik

Izjava o izvornosti

Izjavljujem da je moj završni 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu prikazana je izrada glazbene videoigre u programskom alatu Unity. Najprije je kratko opisan ovaj žanr videoigara te alat Unity. Nakon toga objašnjena je problematika te dijelovi odnosno sustavi koji čine igru. Slijedi praktični dio odnosno izrada same videoigre pri čemu je korišten pristup objektno orijentiranog programiranja i programski jezik C#.

Ključne riječi: videoigra; Unity; glazba; ritam; objektno orijentirano programiranje; C#

Sadržaj

1. Uvod

Tema je ovog završnog rada izrada glazbene videoigre, točnije videoigre koja se temelji na ritmu (engl. *rhythm game*) u programskom alatu Unity. Prvom igrom tog žanra smatra se *Dance Aerobics* iz 1987. godine, razvijenom za Nintendo Entertainment System konzolu [1]. U igri se od igrača traži da u sklopu fitness treninga pritisne odgovarajuće tipke u odgovarajuće vrijeme kako bi ostvarivao pokrete.

Unatoč tome što je *Dance Aerobics* prva videoigra takvog žanra, Playstation igra *PaRappa the Rapper* iz 1996. godine smatra se prvom pravom igrom temeljenom na ritmu zbog velikog utjecaja na daljnji razvoj žanra. Igrač se u toj videoigri nalazi u ulozi psa PaRappe koji sudjeluje u rap bitkama protiv raznih protivnika i pobjedom prelazi na iduće razine koje uglavnom postaju progresivno teže. U igri je ukupno šest razina, a sama mehanika igre zahtijeva od igrača da u ritmu glazbe i u pravilno vrijeme pritisne odgovarajuće tipke koje mu se prikazuju na vremenskoj liniji na vrhu ekrana kako bi osvajao bodove (slika 1). Ako igrač sakupi dovoljno bodova i igra konzistentno, tada prelazi na iduću razinu. Osim navedenih videoigara, tu su još i drugi predstavnici ovog žanra: popularni serijali videoigara *Guitar Hero* i *Rocksmith*, *osu!*, *Beat Saber*, *Friday Night Funkin'* i brojne druge.

Slika 1: Snimka zaslona igre PaRappa the Rapper [2]

Videoigra izrađena u sklopu završnog rada inspirirana je igrom Guitar Hero. Osnovna mehanika je vrlo slična uz neke sitne razlike. Primjerice, u izrađenoj igri postoje četiri note odnosno tipke koje igrač koristi, dok je u Guitar Hero dostupno pet mogućih nota. Glavna ideja je ta da igrač odabire jednu od ponuđenih razina koje se temelje na različitim glazbenim žanrovima, glazba počinje svirati u pozadini te se istovremeno na ekranu pojavljuju note odnosno tipke koje igrač mora pritiskati u ritmu kako bi osvajao bodove.

Igrača je moguće kazniti za loše sviranje na način da se gube bodovi za prerano ili prekasno sviranje nota. Na ovaj način se potiče igrača da savlada kontrole i mehanike igre te da postane bolji. Osim toga, igrača se može i nagraditi za dobro sviranje: ako igrač dobro odsvira više nota zaredom, povećava se faktor kojime se množi broj osvojenih bodova. Na taj način igrač može postići puno više bodova ukoliko igra videoigru konzistentno dobro.

2. Metode i tehnike rada

Glavni alat pri izradi ove videoigre je programski alat Unity. Postoji više dostupnih programskih alata za razvoj videoigara poput: Unreal Engine, Godot i drugi. Za realizaciju završnog rada odabran je Unity zbog autorovog poznavanja istog te jer se alat primarno koristio na nastavi kolegija vezanih uz razvoj videoigara na Fakultetu. Neizbježan izvor informacija pri izradi ove videoigre bila je sama Unity dokumentacija. Koncepti i mehanike izrađene videoigre su neovisne o platformi koja se koristi za razvoj videoigara, principi su jednako primjenjivi. Korišten je uređivač programskog kôda Visual Studio Code uz razna proširenja za C# jezik koje poboljšavaju produktivnost i tijek rada. Osim samog programiranja, prijašnja znanja i vještine koja su pomogla pri razvoju videoigre odnose se na grafički dizajn i dizajn korisničkih sučelja. U igri je bilo potrebno dizajnirati razna sučelja i izbornike, a pritom je potrebno imati na umu korisničko iskustvo: navigacija kroz izbornike mora biti efikasna i jednostavna.

Programski jezik koji je korišten je C#, a time i pristup objektno orijentiranog programiranja jer je on pogodan za izradu videoigara.

Videoigra je izrađena za stolna računala odnosno PC, a operacijski sustavi na kojima je igra testirana jesu: Microsoft Windows, Mac OS X i Linux. Specifikacije računala autora završnog rada dane su u tablici 1 kako bi se dao kontekst okoline u kojoj je igra razvijena.

Tablica 1: Specifikacije računala autora završnog rada (vlastita izrada)

Procesor (CPU)	AMD Ryzen 5 1600 3.2GHz
Radna memorija (RAM)	Corsair Vengeance Dual-Channel 16GB 3200 MHz
Grafička kartica (GPU)	Nvidia GeForce GTX 1060 6GB
Operacijski sustav	Fedora Workstation 38 i Microsoft Windows 10

3. Programski alat Unity

Ovo poglavlje služi kako bi se čitatelja ukratko upoznalo s programskim alatom Unity za lakše čitanje i razumijevanje praktičnog dijela ovog rada odnosno izrade videoigre i programskog kôda.

3.1. Osnovne informacije

Unity je stroj igre (engl. *game engine*) koji omogućuje izradu videoigara za više platformi razvijen 2005. godine od strane tvrtke Unity Technologies. Strojevi igara su softverski okviri koji pomažu programerima strojeva igara stvarati i dizajnirati videoigre [3]. Koristeći strojeve igara, razvojnim programerima olakšani su brojni aspekti razvoja videoigre jer su im na raspolaganju gotovi alati i biblioteke. Razvojni programeri tako mogu lakše upravljati smještajem objekata u igri, svijetom odnosno terenom, fizikom, osvjetljenjem, zvukom, umjetnom inteligencijom i dr.

Unity je isprva bio najavljen kao stroj igre za razvoj videoigara za Mac OS X platformu, no kroz vrijeme dodana je podrška za razne platforme: Android, iOS, Microsoft Windows, Linux, WebGL... Videoigre koje se mogu razviti uz pomoć Unityja mogu biti 2D, 3D, igre prividne stvarnosti (engl. *virtual reality*) i dr.

3.2. Planovi korištenja

Cilj stroja igre Unity bio je omogućiti širem broju ljudi da sudjeluju u razvoju videoigara. Unity koristi širok spektar ljudi: od djece, studenata i hobista pa do velikih razvojnih studija. Tablica 2 prikazuje planove korištenja koje Unity nudi korisnicima.

Tablica 2: Planovi korištenja alata Unity [4]

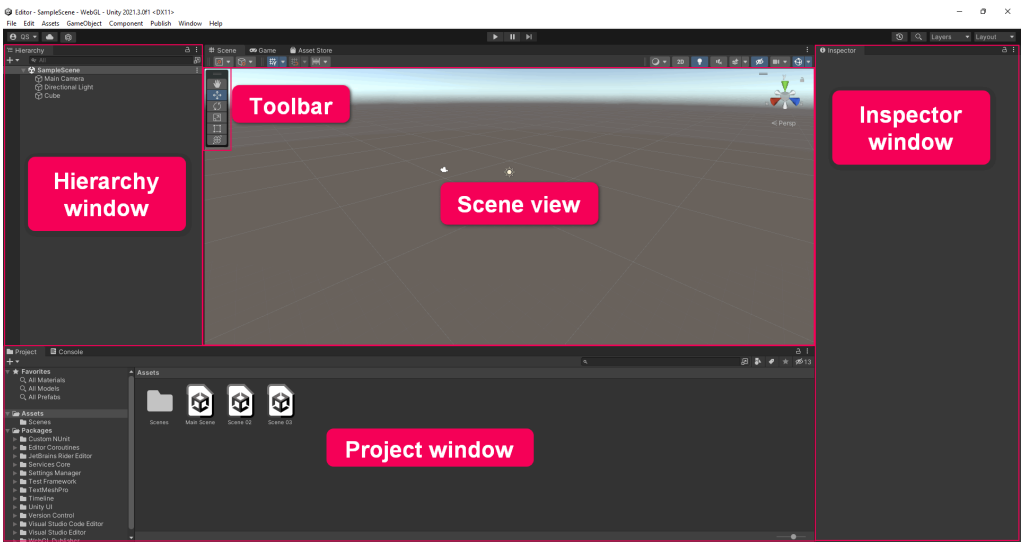
Plan	Cijena
Personal (osobni plan)	besplatno
Plus	od €369 godišnje
Enterprise (poduzeća)	potrebno kontaktirati Unity Technologies
Industry (industrija)	od €4.554 godišnje

Osobni plan je besplatan i dovoljan za korisnike da samostalno krenu s razvojem videoigara. Svi ostali planovi namijenjeni su za osobe i timove koji zahtijevaju više značajki i bave se profesionalnim razvojem videoigara. Autor završnog rada služio se osobnim planom Unityja za izradu videoigre.

3.3. Sučelje

Sučelje programskog alata Unity sastoji se od pet glavnih dijelova (slika 2):

- Hijerarhijski prozor (engl. *hierarchy view*)
  - u hijerarhijskom prozoru vidljiv je popis scena, objekata igre u sceni te njihova hijerarhija
  - korisnici mogu ugniježđivati objekte na koji god način žele na način da se odredi objekt roditelj i njegova djeca
- Alatna traka (engl. *toolbar*)
  - pod alatnom trakom grupirane su funkcije zaslužne za pregledavanje scene, transformaciju objekata i sl.
- Pogled scene (engl. *scene view*)
  - pogled scene je prozor u kojemu je vizualno vidljiva scena u kojoj se korisnik trenutno nalazi
- Prozor projekta (engl. *project window*)
  - u prozoru projekta vidljivi su svi resursi i datoteke projekta
  - to uključuje scene, objekte igre, predloške objekata igre, skripte i dr.
- Prozor inspektora (engl. *inspector window*)
  - u ovome prozoru prikazana su svojstva objekata igre i komponenata



Slika 2: Sučelje programskog alata Unity [5]

3.4. Koncepti

U ovoj sekciji bit će objašnjeni osnovni koncepti Unityja za lakše razumijevanje praktičnog rada. U sklopu završnog rada korištena je većina koncepata Unityja koji su bili potrebni za realizaciju mehanika videoigre.

3.4.1. Objekti igre i komponente

Objekti igre (engl. *game objects*) su osnovni elementi u Unityju. Oni se sastoje od komponenata koje dodaju funkcionalnost objektu. Komponenta (engl. *component*) je osnovna klasa za sve ono što je pridruženo objektu igre [6]. Neki od primjera komponenata su:

- transformacija (engl. *transformation*)
  - predstavlja poziciju, rotaciju i veličinu objekta prema X, Y i Z vrijednostima
  - svaki objekt igre u sceni sadrži komponentu transformacije
- oznaka (engl. *tag*)
  - objekti igre mogu se označavati oznakama
  - na taj način oni se mogu unikatno identificirati i razlikovati te se kasnije mogu ciljati skriptama
  - primjer: metoda `FindGameObjectsWithTag()` za pretraživanje objekata igre prema oznaci
- materijal (engl. *material*)
- skripta (engl. *script*)
  - skripte sadrže programski kôd i pomoću njih se mogu manipulirati objekti igre

Objekti igre mogu, ali i ne moraju biti vizualno vidljivi igraču. Primjerice, upravitelj razine može biti realiziran kao objekt igre koji sadrži samo skriptu za upravljanje razine odnosno programsku logiku koju igrač neće vidjeti doslovno na ekranu (npr. parsiranje JSON datoteke).

3.4.2. Scene

Scene su skupovi objekata igre koji se prikazuju zajedno. Svaka igra ima barem jednu scenu, a moguće je imati i više scena. Na primjer, najčešće postoje barem dvije scene: jedna za glavni izbornik, a druga za razinu igre.

3.4.3. Kamere

Kamere su odgovorne za prikazivanje scene. U videoigrama moguće je imati više postavljenih kamera od kojih svaka može prikazivati različitu perspektivu. Primjerice, moguće je imati dvije zasebne kamere: jednu za pogled iz prvog lica, a jednu za pogled iz trećeg lica.

Osim zadanih kamera koje Unity nudi, postoje i paketi poput Cinemachine koji proširuju mogućnosti kamera u Unityju. Cinemachine korišten je za realizaciju videoigre koju obuhvaća završni rad.

3.4.4. Predlošci objekata igre

Predlošci objekata igre (engl. *prefabs*) su posebna vrsta objekata igre koji dijele ista svojstva. Nakon što su predlošci objekata igre definirani, instance objekata lako se kreiraju putem skriptiranja metodom `Instantiate()`.

3.4.5. Skripte

Skripte su dijelovi programskog kôda koji se dodijeljuju objektima igre i na taj način im dodaju funkcionalnost. One se mogu koristiti za kontrolu kretanja igrača, animacije, instanciranje i uništavanje objekata igre i brojne druge funkcionalnosti.

Programski jezik koji je izvorno podržan i koji se koristi za skriptiranje unutar Unityja je C#.

3.4.6. Fizika

Unity sadrži ugrađeni sustav odnosno pogon za fiziku koji brine o koliziji odnosno sudaru objekata, gravitaciji, akceleraciji, simulaciji odnosno dinamici fluida i slično.

U sklopu izrađene videoigre korišteni su 3D objekti, no postavke fizike nisu mijenjane.

3.4.7. Materijali, shaderi i teksture

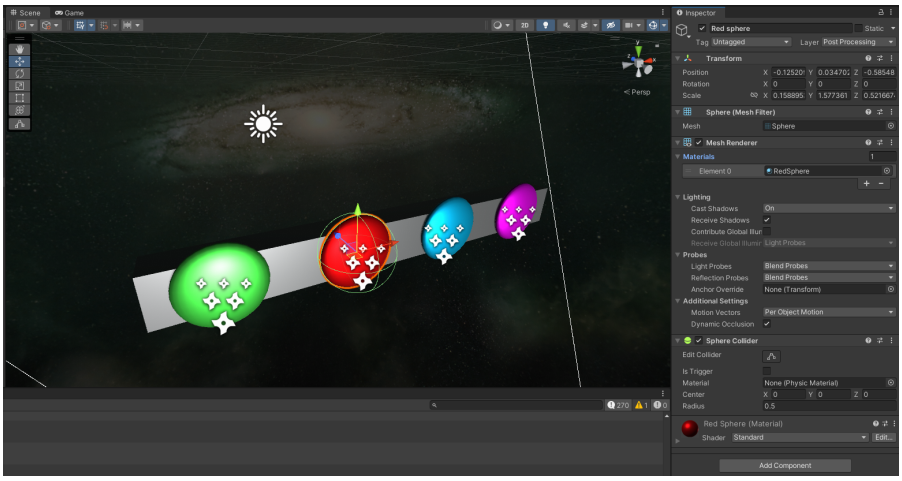
Postupak renderiranja odnosno iscrtavanja sličica videoigre postignut je uporabom materijala, shadera i tekstura. Izvor [7] pojašnjava navedene pojmove u nastavku.

Materijali definiraju kako bi površina trebala biti prikazana, uključujući reference na korištene teksture, informacije o popločavanju (engl. *tiling*), nijanse boja i dr. Dostupne opcije za materijal ovise o tome koji shader materijal koristi.

Shaderi su male skripte koje sadrže matematičke izračune i algoritme za izračun boje svakog prikazanog piksela, na temelju unosa osvjetljenja i konfiguracije materijala.

Teksture su bitmap slike. Materijal može sadržavati reference na teksture, tako da shader materijala može koristiti teksture dok izračunava boju površine objekta. Uz osnovnu boju (albedo) površine objekta, teksture mogu predstavljati mnoge druge aspekte površine materijala poput njegove refleksije ili hrapavosti. Neki od primjera tekstura su slike površina (poželjno visoke rezolucije) drva, trave, zemlje, kamena i dr.

U sklopu završnog rada kreirani su jednostavni materijali za potrebe davanja boje 3D objektima granice. Jedan od materijala prikazan je na slici 3.



Slika 3: Primjer materijala na 3D objektu granice (snimka zaslona)

3.4.8. Korisnička sučelja

Korisnička sučelja u Unityju moguće je kreirati na tri različita načina [8]:

- UI Toolkit
  - najnoviji sustav za izradu korisničkih sučelja
  - naglasak na fleksibilnosti i odvajanju sučelja od objekata igre u sceni
  - strukturiranje sučelja i njegovih elemenata koristeći HTML, XML i CSS
  - još uvijek je u eksperimentalnoj fazi pa neke mogućnosti nedostaju, poput podrške uređaja za unos koji nisu standardna tipkovnica i miš
- Unity UI paket (uGUI)
  - koristeći ovaj UI sustav temeljen na objektima igre, moguće je kreirati UI za igre i aplikacije tijekom izvršavanja
  - na taj način oni se mogu unikatno identificirati i razlikovati te kasnije ih na taj način ciljati skriptama
  - za kreiranje sučelja kombiniraju se objekti igre poput `Image`, `Panel`, `Slider`, `Scrollbar`, `TextMeshPRO` i dr.
  - brojne videoigre izrađene su koristeći ovaj sustav pa iz tog razloga postoji mnoštvo tutorijala, diskusija na forumima i sl. u vezi njegovog korištenja
- IMGUI
  - lagan (engl. *lightweight*)
  - služi za izradu prilagođenih inspektora za komponente i novih prozora te alata za proširenja unutar Unityja

Za videoigru završnog rada odlučeno je korištenje Unity UI paketa (uGUI). Iako UI Toolkit ima brojne prednosti i nudi razne mogućnosti, zasad još nije dobro integriran s novim Unity

sustavom za kontrolu unosa. Iz tog razloga otežano je omogućiti korisniku da se kroz izbornike navigira korištenjem različitih uređaja za unos istovremeno, npr. DS4 kontrolerom te mišem i tipkovnicom.

3.4.9. Zvuk

Unity omogućuje dodavanje raznih zvučnih efekata, pozadinske glazbe i sl. Izvorima zvuka prilažu se zvučne datoteke, a onda se njima mogu mijenjati svojstava poput brzine izvođenja, tona, i dr.

Izrađena videoigra sadrži datoteke pjesama u .ogg formatu. Svaka razina unutar videoigre završnog rada predstavljena je JSON datotekom, a u JSON datoteci nalazi se informacija o nazivu pjesme i ekstenziji.

3.4.10. Sustavi čestica

Sustavi čestica mogu se koristiti za stvaranje specijalnih efekata, kao što su vatra, dim, eksplozije i sl. Razvojnim programerima dostupne su brojni parametri koje mogu mijenjati kako bi dobili željeni efekt i ponašanje čestica.

U sklopu izrađene videoigre korišteni su za davanje vizualne povratne informacije (engl. *feedback*) korisniku u slučaju da je pritisnuo tipku u ispravno vrijeme i time pogodio notu.

4. Izrada videoigre

U ovom poglavlju biti će opisan praktični dio završnog rada, odnosno sama realizacija videoigre u platformi Unity. Prvo će biti opisana izrada projekta u Unityju, zatim će biti objašnjeni koraci, komponente i programski kôd odnosno skripte koje su potrebne za realizaciju jedne od razina. Nakon što je jedna razina spremna, ostale razine se vrlo jednostavno kreiraju na način da se parsira JSON datoteka koja sadrži podatke o razini. Na kraju završnog rada priloženi su isječci programskog kôda u programskom jeziku C# uz dodane komentare za lakše razumijevanje i snalaženje.

4.1. Pregled videoigre

Videoigra u sklopu završnog rada mehanikama je vrlo slična igri Guitar Hero. Neke od razlika u odnosu na Guitar Hero su te da u ovoj igri ne postoje note koje je potrebno držati pritisnutima te zbog jednostavnosti igra sadrži četiri note umjesto pet. Ono po čemu se razine razlikuju jesu informacije u JSON datoteci. JSON datoteka svake razine sadrži informacije poput naziva pjesme, izvođača, žanra, težine igranja i vremenskih oznaka nota.

Igra je 3D i zamišljena je da se odvija u svemiru, pa se iz tog razloga koristi svemir kao pozadina (engl. *skybox*) u glavnom izborniku, ali i u razinama. Po uzoru na svemirsku tematiku, igra nosi ime *AstroBeat*.

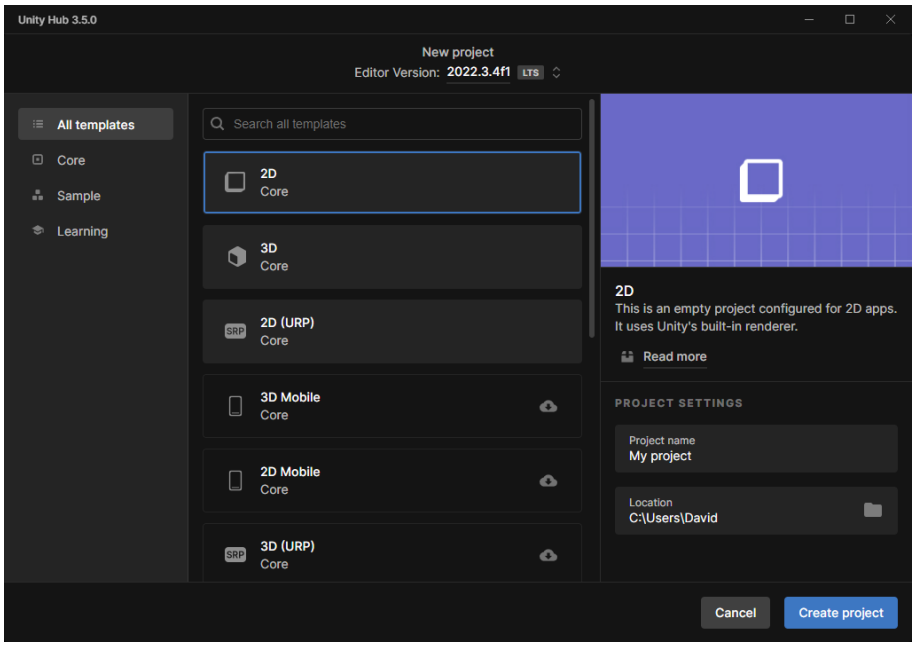
Što se tiče kontrola, videoigra podržava tipkovnicu i miš te kontroler. Igra je testirana koristeći DualShock 4 (dalje u tekstu skraćeno DS4) kontroler.

Igra se sastoji od više različitih klasa, pri čemu svaka klasa predstavlja određenu funkcionalnost ili podsustav videoigre, neke od njih su:

- Upravitelj igrom (engl. *game manager*)
- Upravitelj pjesmama (engl. *song manager*)
- Upravitelj glasnoćom (engl. *volume manager*)
- Nota (engl. *note*)
- Staza (engl. *lane*)

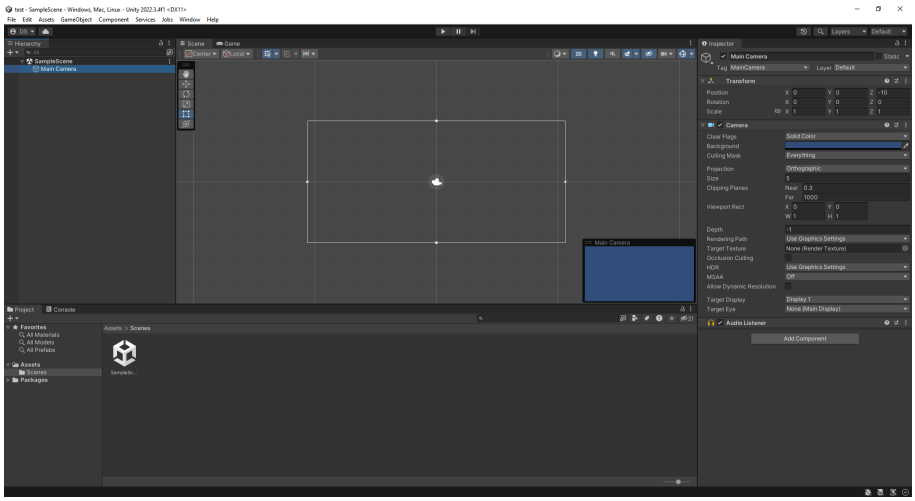
4.2. Izrada Unity projekta

Za početak, potrebno je izraditi projekt u Unityju. Osim same Unity platforme, korišten je i Unity Hub, dodatan, izborni program koji služi za prikaz postojećih Unity projekata, izradu novih projekata, preuzimanje predložaka projekata za videoigre raznih žanrova i sl. Predložak koji je izabran je "2D", a uz sam predložak potrebno je definirati još i ime projekta te lokaciju na disku na kojoj će se pohraniti projektne datoteke (slika 4).



Slika 4: Sučelje programa Unity Hub (snimka zaslona)

Odabirom ovog predloška, Unity kreira osnovni projekt za izradu 2D videoigre koji sadrži minimalne komponente odnosno scenu i kameru. Već sada uz ove minimalne komponente projekt se može pokrenuti. Na sučelju će biti dan prikaz kamere scene: prikaz kamere je samo plava boju odnosno boja pozadine neba koja je podešena u postavkama kamere (slika 5).



Slika 5: Projekt 2D predloška u Unityju (snimka zaslona)

4.3. Izrada razine

Za početak rada na videoigri, autor je najprije krenuo s izradom jedne razine. Kako je glavna logika igre jednaka za svaku razinu, nije potrebno kreirati više različitih scena za različite razine jer se one razlikuju samo podacima razine poput pjesme, žanra, izvođača, vremenskih oznaka i sl.

Prije izrade razine, odlučena je struktura datoteke koja će sadržavati informacije o pojedinoj razini. Odabran je format JSON, a podaci koji se odnose na razinu i čitaju iz datoteke su:

- naziv pjesme `name`
- izvođač `artist`
- žanr pjesme `genre`
- naslovnica albuma `cover_art`
- datoteka pjesme `stream`
- težina razine `difficulty`
- polje staza `lanes`
  - naziv staze `name`
  - vremenske oznake nota `timestamps`

4.3.1. Organizacija podataka razine

Pri izradi glazbene videoigre, potrebno je odlučiti o formatu i organizaciji podataka nota i staza za pojedinu razinu. Takvi podaci zapisani u datoteku zatim se parsiraju te se note stvaraju u odgovarajućim vremenskim intervalima. Za tu svrhu moguće je koristiti razne pristupe i različite vrste datoteka poput:

- **MIDI (Musical Instrument Digital Interface)**
  - komunikacijski standard koji omogućuje povezivanje i međusobnu komunikaciju uređaja poput sintisajzera, samplera i računala pomoću MIDI poruka [9]
  - MIDI standard definira i MIDI tip datoteke koja ne sadrži valove zvuka ni njihove uzorke već sadrži informacije koje su potrebne za reprodukciju zvuka
  - iz tog razloga, MIDI datoteka nije ograničena na zvuk instrumenta: iste note moguće je primijeniti na digitalni instrument gitare, sintisajzera, klavira, bubnjeva i dr.
  - u MIDI datoteku moguće je pohraniti note pjesme koja će svirati u razini videoigre i zatim MIDI datoteku čitati i obraditi informacije koja ona sadrži
  - primjer Unity paketa [10] i .NET knjižice (engl. *library*) [11] za tu svrhu je DryWetMIDI autora Melanchall koja znatno olakšava čitanje i obradu MIDI datoteka
  - MIDI datoteke se sa lakoćom kreiraju koristeći digitalne audio radne postaje (engl. *digital audio workstation*) poput Ableton Live, FL Studio, REAPER, LMMS i dr.
  - autor [12] služio se ovim pristupom pri izradi slične videoigre

- **JSON (Javascript Object Notation)**
  - lagani (engl. *lightweight*) format za razmjenu podataka [13]
  - prema službenoj stranici [13], JSON datoteku čine dvije strukture:
    * skup parova naziv/vrijednost koji se u raznim programskim jezicima realiziraju kao objekt, zapis, struktura, rječnik, hash tablica, popis s ključevima ili asocijativni niz
    * uređena lista vrijednosti koja se u većini programskih jezika realizira kao niz, vektor, lista ili niz
  - jedan od najzastupljenijih načina organizacije podataka u datotekama danas
- **CSV (Comma-seperated values)**
  - format tekstualne datoteke u kojima se vrijednosti pohranjuju nalik na tablični zapis
  - tekstualne ili numeričke vrijednosti su odvojene zarezima i zapisane su kao obični tekst (engl. *plain text*)

Na putanji `/Projekt/Assets/` organizirane su datoteke koje sadrže podatke o razinama.

U direktoriju `StreamingAssets` nalaze se poddirektoriji `Audio`, `Charts` i `CoverArt`.

Direktorij `Resources` sadrži poddirektorij `Audio` u kojemu su smještene zvučne datoteke (.ogg) pojedinih razina.

Direktorij `Charts` sadrži JSON datoteke pojedinih razina.

Direktorij `CoverArt` sadrži slike (.jpg) naslovnice albuma na kojemu se pjesma pojedine razine nalazi.

Kod glazbenih videoigara obično se igračima nudi sloboda da samostalno kreiraju vlastite razine. Takvu mogućnost nude videoigre poput Guitar Hero, Clone Hero, osu! i sl.

U sklopu videoigre ovog završnog rada takva mogućnost se također nudi igraču no on zapravo mora ručno pisati sve informacije o pjesmi u JSON datoteku, a time i podatke poput vremenskih oznaka nota. Potencijalno proširenje videoigre bilo bi kreirati softver s grafičkim sučeljem s kojim bi se lakše vizualizirale note i njihove vremenske oznake.

4.3.2. Dodavanje izvora zvuka

Dodan je objekt igre (engl. *game object*) naziva `Audio Source` koji će biti zaslužan za reproduciranje zvuka na izlazni uređaj odnosno zvučnu karticu računala. Neki od parametara odnosno svojstava (engl. *properties*) ove komponente su:

- `AudioClip` - ovo svojstvo sadrži referencu na audio datoteku koja će se reproducirati tijekom razine

- `Mute` - pomoću ovog svojstva moguće je u potpunosti utišati reprodukciju datoteke zvuka
- `Volume` - raspon vrijednosti od 0 do 1 koji određuje glasnoću zvuka
- `Pitch` - raspon vrijednosti od -3 do 3 koji određuje visinu zvuka

Svojstva komponenata mogu se mijenjati ručno prije ili tijekom izvođenja igre. Ona se također mogu mijenjati i programski, odnosno kreiranjem skripti koje određuju svojstva komponenata, a time i ponašanje komponenata odnosno objekata igre. Svojstva se pomoću skripti mogu mijenjati prije početka izvođenja prve sličice igre (engl. *frame*), to znači da se blok naredbi nalazi u funkciji `void Start()` ili nakon svake sličice u funkciji `void Update()`.

Referenca na audio datoteku koja će se reproducirati tijekom razine dobivena je parsiranjem JSON datoteke. Vrijednost pod `stream` u JSON datoteci je naziv i ekstenzija pjesme.

4.3.3. Kreiranje objekta granice

Za kreiranje objekta granice, korišteni su 3D objekti: kocka i četiri kugle. Ovaj objekt vizualno daje do znanja igraču gdje se od njega traži da pritisne tipku za odgovarajuću notu. Ukoliko igrač prekasno ili prerano pritisne note, one će se bilježiti kao promašaj. Nakon što su kreirani objekti kocke i četiri kugle, na njima su obavljene transformacije. Kocka je skalirana da bude kvadar, a kugle su skalirane da budu plosnate. Nakon toga, kugle su pomaknute po X osi da odgovaraju X osi nota koje se stvaraju.

Kako bi kreirani 3D objekti imali boju ili teksturu, njima su pridruženi odgovarajući materijali. Neka od svojstava materijala koja su mijenjana u ovoj videoigri su: `Albedo` (osnovna boja materijala), `Metallic` (određuje u kojoj mjeri će se svjetlost odbijati od materijala), `Emission` (mogućnost materijala da emitira svjetlost) i dr.

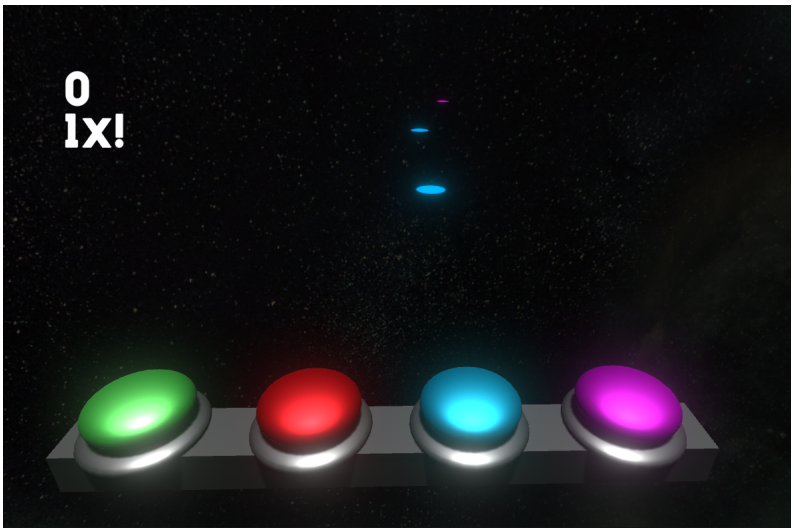
Slika 6 u nastavku prikazuje snimku zaslona igre na kojoj su vidljive note (prikazane kao kružići) od kojih se svaka pomiče po svojoj stazi prema objektu granice. Objekt granice igraču daje do znanja u kojem trenutku se od njega zahtijeva da pritisne tipku za pogađanje note.

4.3.4. Kreiranje objekata nota

Objekti nota vidljivi su igraču i daju mu vizualni znak u kojem trenutku je potrebno pritisnuti određenu tipku kako bi ostvarili bodove. Za stvaranje objekata nota izrađen je predložak objekta igre. U objektu staze se zatim prema vremenskim oznakama instanciraju objekti nota.

Predložak objekta note kreiran je na sljedeći način: `Assets` > `Create` > `Prefab`. Za svaku notu pridružena je slika (engl. *sprite*) te skripta `Note.cs`.

Svaka nota sadrži sljedeće varijable: vremena instanciranja, pridruženo vrijeme note, Y vrijednost pozicije na kojoj je nota stvorena i na kojoj će se uništiti, brzina kretanja note i slika note.



Slika 6: Objekt granice i objekti nota u videoigri (snimka zaslona)

```
double timeInstantiated;
public float assignedTime;
float startY;
float endY;
float startTime;
float noteMoveSpeed;
private SpriteRenderer spriteRenderer;
```

U metodi `Start()` potrebno je podesiti početne vrijednosti nota:

```
void Start() {
    timeInstantiated = SongManager.GetAudioSourceTime();
    // Pozicija na kojoj je nota stvorena
    startY = SongManager.Instance.noteSpawnY;
    // Pozicija na kojoj ce se nota unistiti
    endY = SongManager.Instance.noteDespawnY;
    startTime = Time.time;
    noteMoveSpeed = (endY - startY) / (SongManager.Instance.noteTime * 2);
    // Postavljanje pocetne pozicije note
    transform.localPosition = new Vector3(transform.localPosition.x, startY,
    transform.localPosition.z);
    // Aktiviranje objekta note i postavljanje da nota bude vidljiva
    gameObject.SetActive(true);
    spriteRenderer = GetComponent<SpriteRenderer>();
    spriteRenderer.enabled = true;
}
```

U metodi `Update()` note se svake sličice igre pomiču i uništavaju:

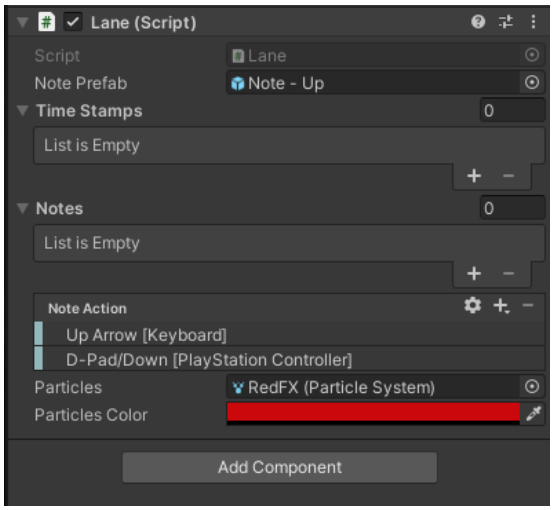
```
void Update() {
    // Provjera je li instanca upravitelja pjesme null
```

```
    if (SongManager.Instance != null) {
        float timeSinceStart = (float)SongManager.GetAudioSourceTime() - startTime;
        float t = timeSinceStart / (SongManager.Instance.noteTime * 2);
        // Unistavanje nota
        if (t > 1) {
            Destroy(gameObject);
        }
        else {
            // Pomicanje nota zeljenom brzinom
            transform.localPosition += Vector3.up * noteMoveSpeed * Time.deltaTime;
        }
    }
}
```

4.3.5. Kreiranje objekata staza

Objekt staze odgovoran je za popunjavanje polja nota koje sadrži vremenske oznake nota iz JSON datoteke. Na temelju tih vremenskih oznaka moguće je usporediti vrijeme pritisnute tipke igrača i provjeriti je li ona pritisnuta za odgovarajuću stazu u ispravno vrijeme. Kako bi iskustvo igrača bilo bolje, dodana je varijabla tolerancije koja predstavlja toleranciju na vremensku pogrešku pritiska tipke.

Svaka staza ima pridruženu `Lane.cs` skriptu, predložak objekta note, polje vremenskih oznaka, polje objekata nota, definirane tipke kojom se sviraju note na toj stazi i sustav čestica (slika 7).

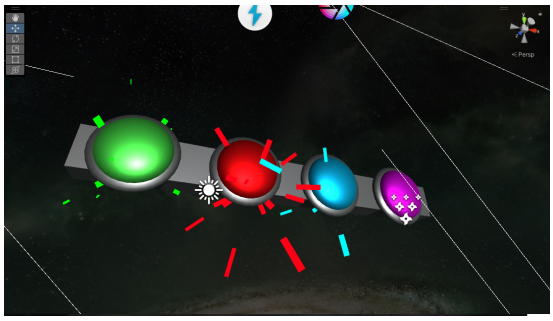


Slika 7: Svojstva objekta igre staze (snimka zaslona)

4.3.6. Kreiranje sustava čestica

Svakoj stazi u videoigri pridružen je njezin sustav čestica. Sustav čestica će vizualno igraču dati do znanja ukoliko je pritisnuo notu u ispravno vrijeme (slika 8). Postavke odnosno

svojstva koje je moguće mijenjati u sklopu sustava čestica su brojne te ovise o efektu koji se želi postići (slika 9).

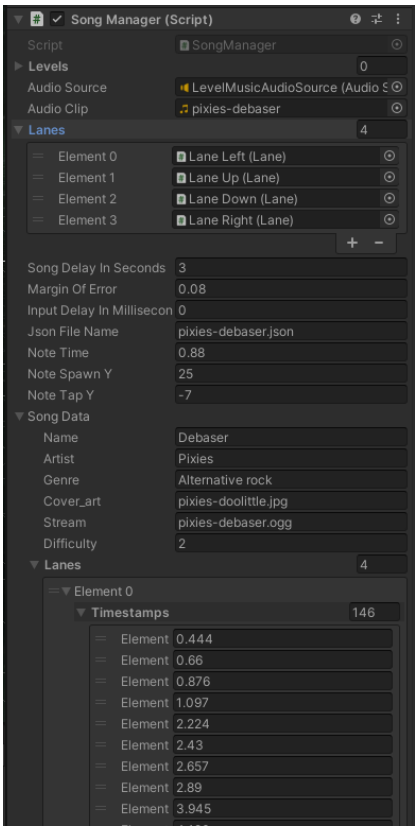


Slika 8: Sustav čestica u videoigri (snimka zaslona)

Slika 9: Svojstva sustava čestica u videoigri (snimka zaslona)

4.3.7. Kreiranje upravitelja pjesme

Glavna mehanika igre nalazi se u klasama upravitelja pjesme i klasi staza. Primjer svojstava koje sadrži skripta upravitelja pjesme nakon parsiranja JSON datoteke nalazi se na slici 10.



Slika 10: Svojstva upravitelja pjesme (snimka zaslona)

Klase odnosno sustavi zaslužni za mehaniku igre realizirani su po uzoru na [12]:

- Klasa nota `Note`
- Klasa staza `Lane`
- Klasa upravitelja pjesme `SongManager`
- Klasa upravitelja rezultatom `ScoreManager`

Uz navedene klase, autor završnog rada realizirao je još neke poput:

- Klase upravitelja postavkama `SettingsManager`
- Klase za postprocesiranje `PostProcessing`

Inicijalizacija upravitelja pjesme:

```
1 void Start() {
2     Instance = this;
3     string levelDataJson = PlayerPrefs.GetString("SelectedLevelData");
4     if (!string.IsNullOrEmpty(levelDataJson)) {
5         SelectedLevelData levelDataHolder = JsonUtility.FromJson<SelectedLevelData>(
    levelDataJson);
6         SongData selectedLevel = levelDataHolder.selectedLevel;
```

```
        string songName = Path.GetFileNameWithoutExtension(selectedLevel.stream);
        jsonFileName = songName + ".json";
        ReadFromJSON();
        LoadAudio(songName);
        Invoke(nameof(StartSong), songDelayInSeconds);
    }
    else {
        Debug.Log("Ne postoje podaci o razini.");
    }
}
```

Parsiranje JSON datoteke razine:

```

// Funkcija za citanje podataka iz JSON datoteke
private void ReadFromJSON() {
    string jsonText = File.ReadAllText(Path.Combine(Application.streamingAssetsPath, "Charts", jsonFileName));
    songData = JsonUtility.FromJson<SongData>(jsonText);
    string audioFileName = songData.stream;
    string audioPath = Path.Combine(Application.streamingAssetsPath, "Audio", audioFileName).Replace("\\", "/");
    LoadAudio(audioFileName);
    // Provjera podudaranja broja objekata staza i broja staza u JSON datoteci
    if (songData.lanes.Count == lanes.Length) {
        for (int i = 0; i < lanes.Length; i++) {
            // Postavljanje vremenskih oznaka prema podacima iz datoteke
            if (songData.lanes[i].timestamps != null) {
                lanes[i].SetTimeStamps(songData.lanes[i].timestamps);
            }
            else {
                Debug.Log("Staza " + i + " nema definirane vremenske oznake.");
            }
        }
    }
    else {
        Debug.Log("Broj objekata staza i staza u JSON datoteci se ne podudara.");
    }
}
```

Funkcije vezane za upravljanje zvukom:

```
// Ucitavanje datoteke iz /Resources/Audio direktorija
private void LoadAudio(string audioName) {
    audioClip = Resources.Load<AudioClip>("Audio/" + audioName);
    audioSource.clip = audioClip;
}

// Funkcija za sviranje pjesme
public void StartSong() {
```

```
9       // Pocetak sviranja pjesme - odnosi se na mp3/ogg/wav datoteku
10      if (audioSource != null) {
11          audioSource.Play();
12      }
13  }
14
15  // Funkcija za dohvacanje trenutnog vremena sviranja u sekundama
16  public static double GetAudioSourceTime() {
17      // Potrebno je pretvoriti vrijeme iz vremenskih uzoraka (engl. samples) u
        sekunde
18      return (double)Instance.audioSource.timeSamples / Instance.audioSource.clip.
        frequency;
19  }
20
21  // Provjera je li pjesma zavrsila
22  private void CheckSongEnd() {
23      if (GetAudioSourceTime() >= Instance.audioSource.clip.length) {
24          Debug.Log("Pjesma/razina je gotova");
25      }
26  }
```

4.4. Postprocesiranje

Postprocesiranje (engl. *post-processing*) postupak je kojim se slikama, videozapisima, filmovima ili igrama naknadno dodaju razni efekti. U kontekstu videoigara, postprocesiranjem se efekti dodaju sličicama igre nakon što su one renderirane. Cilj postprocesiranja je taj da se raznim efektima igra uljepša i stvori bolji vizualni ugođaj.

Sa efektima postprocesiranja ne treba pretjerivati jer postprocesiranje može značajno negativno utjecati na performanse zbog svoje prirode (dodavanje efekta sličicama nakon renderiranja).

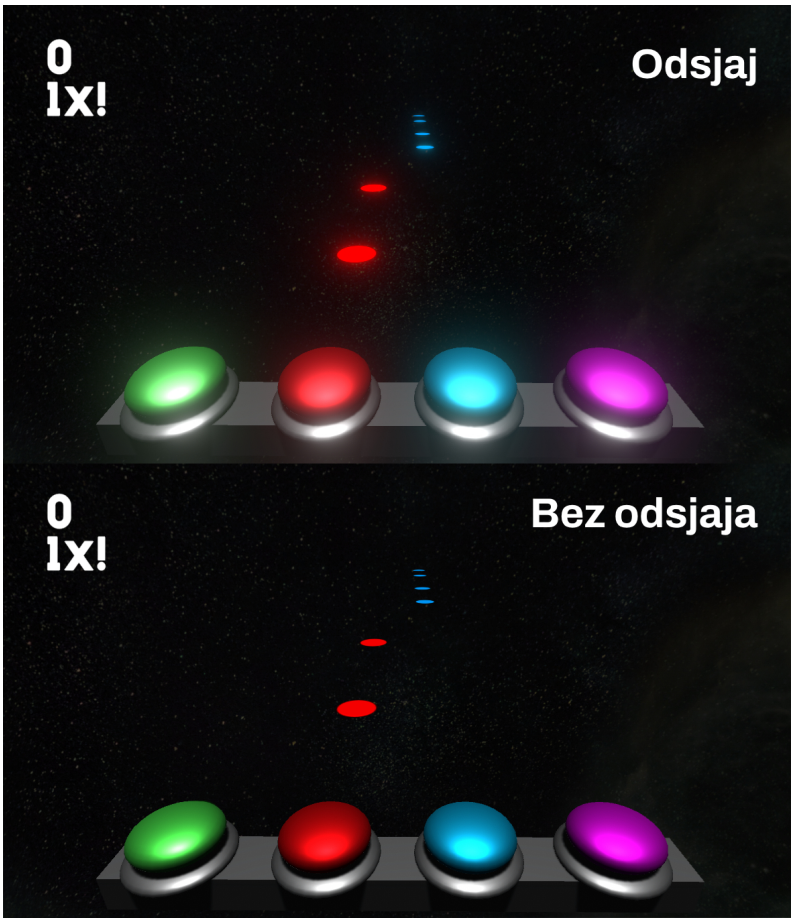
4.4.1. Postprocesiranje u alatu Unity

Neki od često korištenih efekata postprocesiranja u videoigrama su:

- Dubina polja (engl. *depth of field*)
- Odsjaj (engl. *bloom*)
- Zamućenje pokreta (engl. *motion blur*)
- Vinjeta (engl. *vignette*)

Videoigra završnog rada koristi efekte dubine polja i odsjaja kojima upravlja klasa odnosno sustav za postprocesiranje. Efekt dubine polja korišten je kod pauziranja igre (slika **??**),

a efekt odsjaja korišten je da bi se sinkronizirala glazba s količinom odsjaja. Ukoliko korisnik pritisne tipku za pauzu, slika na ekranu će se 'zamutiti', odnosno promijeniti će se apertura kamere razine. Efekt odsjaja sinkroniziran je s valnim podacima zvučne datoteke (engl. *waveform data*) pa se time dobiva efekt odsjaja u ritmu glazbe (slika 11).

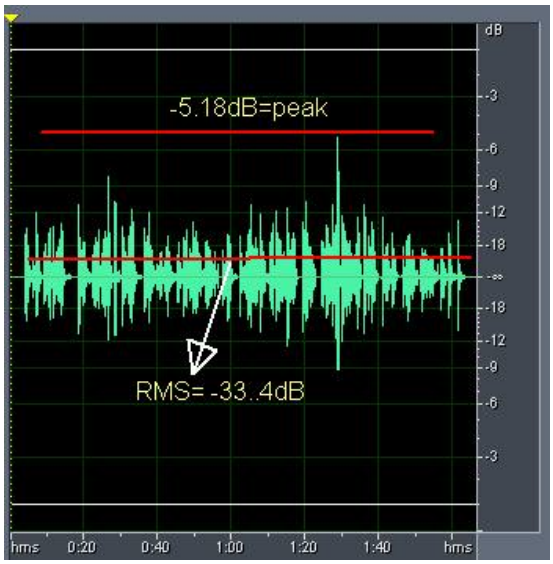


Slika 11: Efekt odsjaja u videoigri (vlastita izrada)

Slika 12: Korisničko sučelje pauzirane igre (snimka zaslona)

Za izračun vrijednosti koje se koriste za postprocesiranje korištena je metoda kvadratne

sredine (engl. *root mean square*). RMS je sredstvo za mjerenje koje mjeri prosječnu glasnoću audio zapisa unutar okvira od otprilike 300 milisekundi [14]. Koristeći tu metodu, svake sličice sekunde videoigre izračunava se percipirana glasnoća pjesme. Vrijednosti koje se dobe tim mjerenjem zatim se koriste kao faktor umnoška s jačinom vibracije kontrolera i efektom odsjaja, a to rezultira vizualnim i taktilnim dojmom jer jačina efekata vibracije i odsjaja direktno ovisi o glasnoći glazbe. Slika 13 prikazuje primjer valnih oblika zvučne datoteke i odnos između vršne (engl. *peak*) vrijednosti signala i RMS vrijednosti.



Slika 13: Odnos vršne i RMS vrijednosti zvučnog signala na primjeru (snimka zaslona)

U nastavku slijedi isječak skripte `PostProcessing` koji prikazuje izračunavanje RMS vrijednosti i korištenje iste za efekte postprocesiranja:

```
// Varijable za efekt sjaja u postprocesiranju i vibraciju kontrolera
float intensity = 0.0f;
float rms = 0.0f;
float duration = 0.04f;

for (int i = 0; i < waveformData.Length; i++) {
    intensity += Mathf.Abs(waveformData[i]);
    rms += waveformData[i] * waveformData[i];
}

intensity /= waveformData.Length;

// Primjena logaritamske skale za jacinu efekta sjaja odnosno bloom efekta
    postprocesiranja
const float INTENSITY_SCALE = 4.0f; // Proizvoljna vrijednost za skalu jacine efekta
intensity = Mathf.Log10(intensity * INTENSITY_SCALE + 1.0f);

// Postavljanje jacine efekta odsjaja
Bloom bloom = volume.profile.GetSetting<Bloom>();
bloom.intensity.value = intensity * 60;

rms = Mathf.Sqrt(rms / waveformData.Length);

```

```
23 const float RMS_THRESHOLD = 0.06f;  // Varijabla osjetljivosti
24 if (rms > RMS_THRESHOLD) {
25     // Jacina vibriranja
26     float rumbleStrength = intensity;
27 
28     // Slanje vibracije na kontroler i zaustavljanje vibracije nakon odredenog
       vremena
29     if (Gamepad.current != null) {
30         Gamepad.current.SetMotorSpeeds(rumbleStrength, rumbleStrength);
31 
32     }
33     if (Gamepad.current != null) {
34         Invoke("StopRumble", duration);
35     }
36 }
```

4.5. Podešavanje sustava za kontrole

Sustav za kontrole služi kako bi se definirale tipke koje će igrač koristiti postizanje određenih radnji u igri. Unity nudi dva sustava za kontrole:

- Upravitelj unosa (Input Manager) - zastarjeli
- Sustav unosa (Input System) paket - noviji

U sklopu izrađene videoigre korišten je noviji sustav za kontrole alata Unity no biti će prokomentirane razlike između starog i novog sustava za kontrole te njihove prednosti i nedostaci.

4.5.1. Unity Input Manager

Upravitelj unosa dio je jezgre Unity platforme i nije potrebno poduzimati dodatne korake kako bi se on instalirao. Način rada upravitelja je jednostavan i preporučljivo je da početnici krenu s njim, a kasnije da pređu na paket sustava unosa.

Sučelje upravitelja unosa nalazi se u postavkama projekta. Sučelje sadrži padajuće izbornike osi (engl. *axes*) od kojih svaka sadrži svojstva koja su prikazana u tablici 3. Slika 14 prikazuje primjer svojstava za os `Horizontal`.

Nakon što su definirana svojstva osi, u nastavku će biti dani isječci programskog kôda kako bi se objasnilo kako programski upravljati pritiskom tipki igrača.

4.5.1.1. Korištenje

U metodi `Update()`, svake sličice igre provjerava se uvjet koristeći `GetButtonDown` metodu klase unosa `Input`, a argument koji se prosljeđuje metodi je naziv osi. Ukoliko je

Tablica 3: Svojstva upravitelja unosa [15]

Svojstvo	Opis
Osi (Axes)	Sadrži sve definirane osi unosa za trenutni projekt: Veličina je broj različitih ulaznih osi u projektu, Element 0, 1, ... su posebne osi za izmjenu.
Naziv (Name)	Niz znakova koji se odnosi na os u pokretaču igre i kod skriptiranja.
Opisni naziv (Descriptive Name)	Detaljni opis funkcije tipke koja se prikazuje u pokretaču igre.
Opisni naziv suprotne tipke (Descriptive Negative Name)	Detaljni opis funkcije suprotne tipke koja se prikazuje u pokretaču igre.
Suprotna tipka (Negative Button)	Tipka koja će poslati negativnu vrijednost na os.
Pozitivna tipka (Positive Button)	Tipka koja će poslati pozitivnu vrijednost na os.
Zamjenska suprotna tipka (Alt Negative Button)	Zamjenska tipka koja će poslati negativnu vrijednost na os.
Zamjenska tipka (Alt Button)	Zamjenska tipka koja će poslati pozitivnu vrijednost na os.
Gravitacija (Gravity)	Koliko brzo će se unos ponovno centrirati. Koristi se samo kada je vrsta osi tipka na tipkovnici ili tipka miša.
Tolerancija (Dead)	Sve pozitivne ili negativne vrijednosti koje su manje od ovog broja bit će registrirane kao nula. Korisno za joysticke.
Suprotnost osi (Invert)	Ako je omogućeno, tipke će slati negativne vrijednosti na os, a suprotne tipke pozitivne vrijednosti na os.
Vrsta (Type)	Koristite tipku tipkovnice/gumb miša za bilo koju vrstu gumba, pokret miša za deltu miša i kotačiće za pomicanje, os joysticka za analogne osi joysticka i pokret prozora kada korisnik protrese prozor.
Os (Axis)	Os s ulaznog uređaja (joystick, miš, gamepad, itd.)
Broj joysticka (Joy Num)	Koji joystick treba koristiti. Prema zadanim postavkama ovo je postavljeno za dohvaćanje unosa sa svih joysticka. Koristi se samo za ulazne osi, a ne za tipke.

uvjet ispunjen, odnosno pritisnuta je tipka `Fire1`, poziva se metoda `NoteHit`.

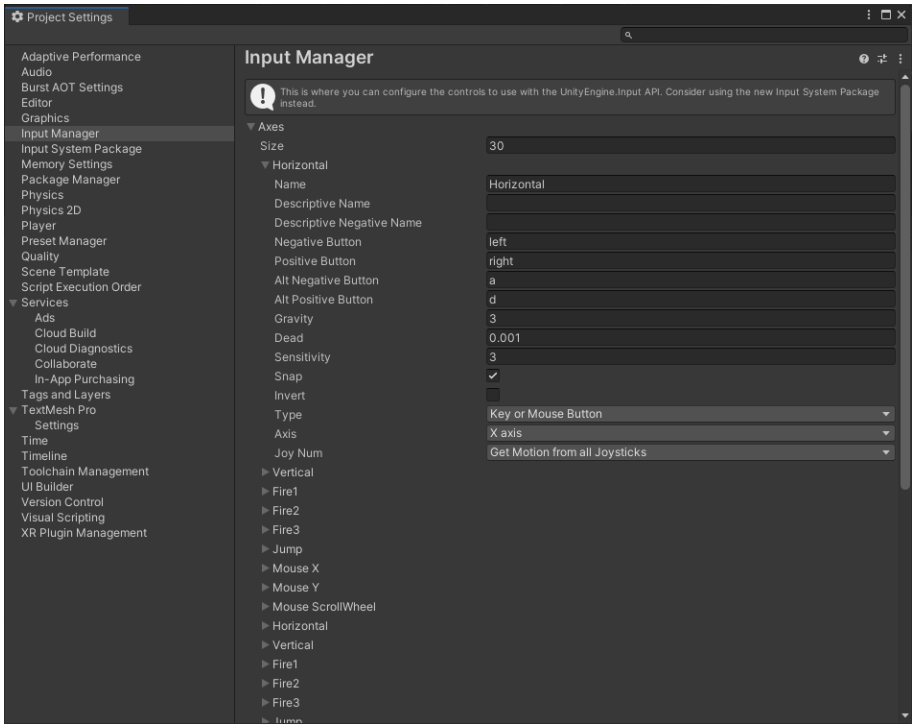
```
void Update() {
    if (Input.GetButtonDown("Fire1")) {
        NoteHit();
    }
}
```

Za igre koje imaju mehaniku kretanja igrača, prikladnije je koristiti metodu `GetAxis` umjesto `GetButtonDown` jer će na taj način kretanje biti uglađenije.

```
void Update() {
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");

    Vector3 movement = new Vector3(horizontalInput, 0.0f, verticalInput) * moveSpeed
     * Time.deltaTime;
    transform.Translate(movement);
}
```

Osim navedenih metoda, tu su još brojne druge poput `GetTouch`, `GetMouseButton`, `GetAccelerationEvent` i dr. koje čitatelji mogu proučiti.



Slika 14: Primjer svojstava osi upravitelja unosa (snimka zaslona)

4.5.1.2. Prednosti

Jednostavnost

Najveća prednost upravitelja unosa je njegova jednostavnost. Osi i njihova svojstva se vrlo lako i brzo postave.

Upoznatost korisnika

Pošto je upravitelj unosa dio Unityja još od samih početaka, većina korisnika je upoznata s njim te su na temelju njega izrađeni brojni projekti i tutorijali.

4.5.1.3. Mane

Manjak fleksibilnosti

Velik broj videoigara danas nudi korisniku slobodu da mijenja tipke koje želi koristiti za unos (engl. *key rebinding*). Problem kod upravitelja unosa je da je izazovno postići taj učinak zbog otežane mogućnosti mijenjanja tipki tijekom izvršavanja igre (engl. *runtime*)

Kašnjenje

Kod upravitelja unosa može doći do manje responzivnosti unosa, odnosno kašnjenja jedne sličice slike (engl. *single frame delay*). To može izrazito negativno utjecati na iskustvo igrača, posebice ako se radi o videoigrama gdje je bitna brza reakcija igrača i responzivan unos.

4.5.2. Unity Input System

Unity je uveo novi sustav unosa u preglednom (engl. *preview*) izdanju verzije 2019.1. Godinu dana kasnije, u verziji 2020.1 sustav je službeno uveden kao paket.

Novi sustav dizajniran je da bude fleksibilan te da se razriješe mane starijeg upravitelja unosa. Osim tipkovnice, miša i kontrolera, sustav podržava brojne uređaje poput dodirnih zaslona i VR naočala.

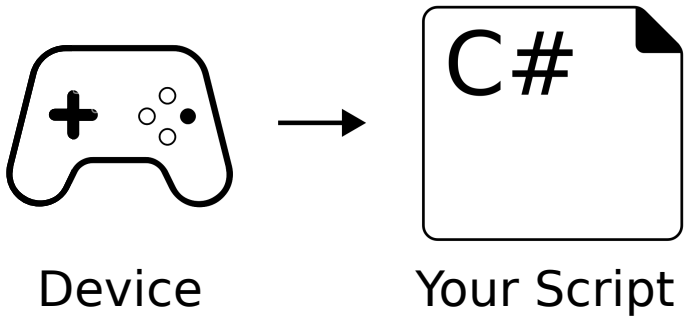
4.5.2.1. Korištenje

Sustav unosa sa sobom nosi četiri moguća tijeka rada [16]:

- Direktan način (Workflow - Direct)
- Ugrađene radnje (Workflow - Embedded Actions)
- Skup radnji (Workflow - Actions Asset)
- Komponenta `PlayerInput` (Workflow - `PlayerInput` Component)

Direktan način

Direktan način rada je najjednostavniji i zapravo podsjeća na stariji upravitelj unosa. Ovaj način rada je i najmanje fleksibilan jer se ne ostvaruju prednosti akcija i interakcija koje novi sustav unosa nudi. Osim toga, koristeći ovaj način teže je ostvariti podršku za više ulaznih uređaja.

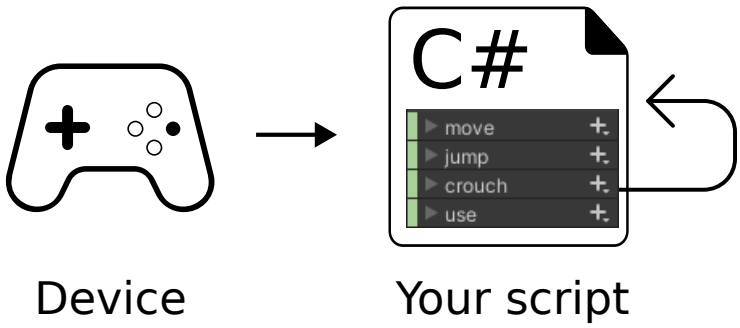


Slika 15: Sustav unosa - direktan način rada [16]

Ugrađene radnje

Korištenje ugrađenih radnji fleksibilniji je način od direktnog načina rada, no manje fleksibilan od korištenja skupa radnji.

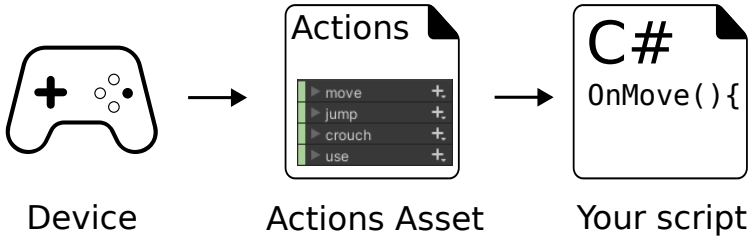
To znači da umjesto izravnog čitanja stanja uređaja, kontrole i njihove radnje se ne navode eksplicitno (npr. gumb na kontroleru, tipka na tipkovnici...). Umjesto toga definiraju se radnje, one se vežu za kontrole, a zatim se u programskom kôdu čitaju vrijednosti radnji.



Slika 16: Sustav unosa - ugrađene radnje [16]

Skup radnji

Korištenje skupa radnji omogućuje da se radnje definiraju i grupiraju pod jednim skupom umjesto da se definiraju u programskom kôdu. Na ovaj način postiže se dodatna razina apstrakcije i bolja organizacija u usporedbi s korištenjem ugrađenih radnji jer su radnje odvojene od objekata igre.

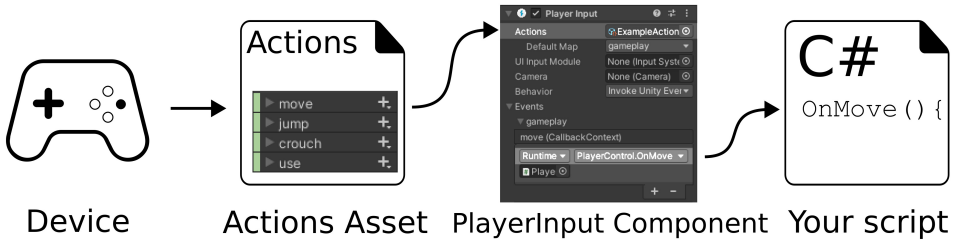


Slika 17: Sustav unosa - skup radnji [16]

Komponenta `PlayerInput`

Korištenje `PlayerInput` komponente predstavlja najvišu razinu apstrakcije što se tiče upravljanja unosom u Unityju. Ta komponenta prima referencu na skup radnji i pruža način za uspostavljanje veza između radnji definiranih u tom resursu i C# metoda u vlastitim MonoBehaviour skriptama, pa će se željene C# metode pozivati kad korisnik obavlja radnje.

Ovaj način omogućuje da korisnici postave veze pomoću korisničkog sučelja u inspektoru, umjesto da zahtijeva pisanje programskog kôda za uspostavu tih veza (kao što je to u slučaj u prethodnom primjeru sa skupom radnji). Ovaj način dopušta i da korisnici sami odaberu kako će se te metode pozivati.



Slika 18: Sustav unosa - komponenta `PlayerInput` [16]

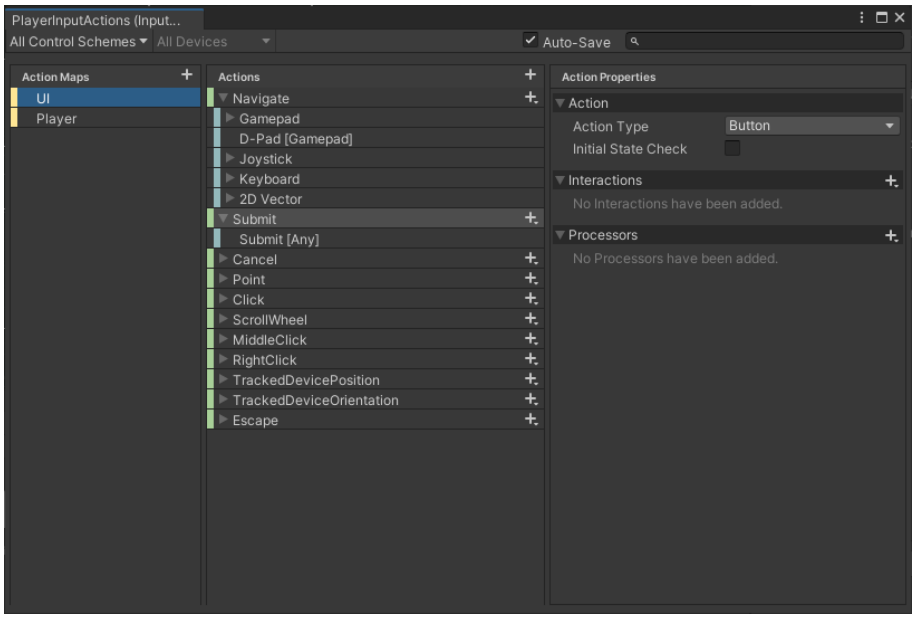
Osim tipki za note, definirane su i tipke za pauziranje igre: tipka `Escape` u slučaju da se koristi tipkovnica te tipka `Options` za PlayStation kontroler.

Ono što valja naglasiti je to da korisnik može u bilokojem trenutku promijeniti koji će ulazni uređaj za kontrole koristiti. Primjerice, ukoliko korisnik pritisne neku stavku u izborniku klikom miša, a zatim odluči kroz opcije izbornika listati tipkama PlayStation kontrolera, ta promjena kontrola biti će glatka i bez zastoja igre ili slično.

Autor završnog rada služio se komponentom `PlayerInput` te skupom radnji. Korištenjem komponente `PlayerInput` lakše je u budućnosti izmijeniti odnosno proširiti kontrole u igri, omogućiti korisniku da mijenja kontrole tijekom izvođenja igre, i sl.

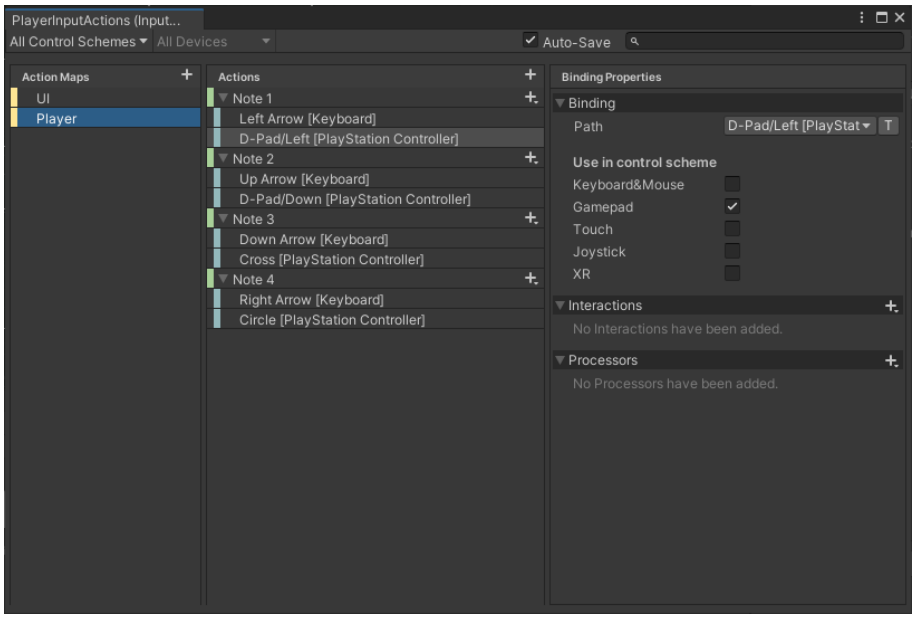
Snimke zaslona u nastavku prikazuju skup radnji. Pri prvom korištenju Unityjevog novog paketa za kontrole ponuđena je opcija za automatsko stvaranje skupa radnji. Ovo je korisno jer će se po zadanome automatski stvoriti radnje odnosno kontrole vezane za korisnička sučelja. Zbog toga nije uopće potrebno definirati vlastite tipke za korisnička sučelja već će se koristiti neke standardne tipke, a u isto vrijeme će biti podržani različiti uređaji poput miša i tipkovnice,

gamepada i sl.



Slika 19: Skup radnji - UI (snimka zaslona)

Osim korištenja već postojećih, moguće je dodati i vlastita mapiranja radnji (engl. *Action Maps*). Na slici 20 vidljiv je skup radnji i odgovarajuće tipke koje je autor završnog rada definirao. Za svaku stazu kreirana je radnja s nazivom note i odgovarajućim tipkama (odabrani uređaji su tipkovnica i PlayStation kontroler).



Slika 20: Skup radnji - Player (snimka zaslona)

4.5.2.2. Prednosti

Fleksibilnost i prilagodljivost

Novi sustav unosa nudi veću fleksibilnost i prilagodbu u usporedbi s starijim sustavom. Omogućuje definiranje vlastitih radnji unosa, stvaranje složenih kontrolnih shema i mijenjanje kontrola tijekom izvođenja igre.

Podrška za više platformi

Unityjev novi sustav unosa dizajniran je za besprijekoran rad na različitim platformama: PC, konzole, mobilni uređaje i VR. Razvoj igara za više platformi time je znatno olakšan.

Sustav temeljen na događajima

Novi sustav unosa koristi arhitekturu vođenu događajima, što olakšava rukovanje ulaznim događajima i odgovaranje na njih na učinkovitiji način.

4.5.2.3. Mane

Krivulja učenja

Novi sustav unosa ima strmiju krivulju učenja u usporedbi sa starijim sustavom unosa, posebno za programere koji su već upoznati sa starim sustavom i koji su izradili brojne projekte služeći se njime.

Kompatibilnost postojećih projekata

U slučaju postojećih projekata koji se uvelike oslanjaju na stari sustav unosa, migracija na novi sustav može zahtijevati značajan napor i prilagodbe.

4.6. Izrada sustava za bilježenje rezultata

Kraj reproduciranja pjesme označava i završetak razine. Uspoređujući vremenske oznake i vrijeme pritiska određuje se je li igrač pritisnuo notu u ispravno vrijeme. Ako je nota pritisnuta u ispravno vrijeme, povećava se rezultat i faktor kojime se množi rezultat. Ukoliko nije, rezultat se smanjuje, a faktor kojime se množi rezultat se resetira na vrijednost 1. Potencijalno proširenje videoigre je kreiranje sustava čitanja i pisanja rezultata u datoteke te prikaz tih informacija putem korisničkog sučelja korisniku nakon što razina završi.

Varijable i početne vrijednosti klase `ScoreManager`:

```
public static ScoreManager Instance;
public Canvas scoreUI;
public TextMeshProUGUI scoreText;
public TextMeshProUGUI comboText;
public static int comboScore;
public static int comboMultiplier;
void Start() {
    Instance = this;
    comboScore = 0;
```

```
    comboMultiplier = 1;

    scoreText = scoreUI.GetComponentInChildren<TextMeshProUGUI>();
    comboText = scoreUI.transform.Find("ComboText").GetComponent<TextMeshProUGUI>();
}
```

Metode za bilježenje pogotka i promašaja:

```
public static void Hit() {
    comboScore += 100 * comboMultiplier;
    comboMultiplier++;
}

public static void Miss() {
    comboMultiplier = 1;
    if (comboScore > 0)
        comboScore -= 100 * comboMultiplier;
}
```

Ažuriranje TextMeshPro objekta koji vizualno igraču prikazuje rezultat:

```
private void Update() {
    scoreText.text = comboScore.ToString("n0");
    comboText.text = comboMultiplier.ToString() + "x!";
}
```

4.7. Izrada glavnog izbornika

Pošto je scena s glavnom mehanikom igre `Level` gotova, dalje je potrebno još izraditi novu scenu `Main Menu` koja će sadržavati glavni izbornik i podizbornike. Tipični glavni izbornik u videoigrama sastoji se od sljedećih opcija:

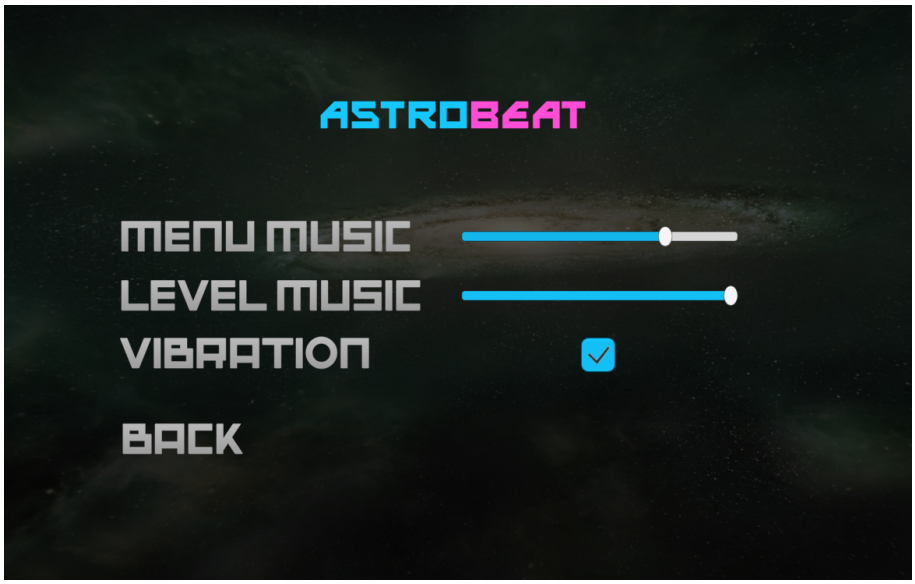
- Play (početak igre)
- Options (postavke)
- Quit (izlaz odnosno povratak na radnu površinu)

Ukoliko igrač odabere opciju započinjanja igre, biti će mu prikazan novi izbornik. U novootvorenom izborniku korisnik može birati između ponuđenih razina te vidjeti informacije o razinama poput naziva izvođača i pjesme, slike naslovnice albuma, težine razine na skali od 1 do 5 te žanr pjesme. Slika 21 u nastavku prikazuje izbornik na kojemu igrač bira jednu od više ponuđenih razina.

Slika 21: Izbornik s razinama (snimka zaslona)

Za upravljanje izbornicima i podizbornicima korištena je metoda `SetActive()` uz argumente `true` ili `false` ovisno o tome je li potrebno sakriti ili prikazati element na klik gumba u izborniku.

Igrač u postavkama igre može mijenjati glasnoću razine, glasnoću glazbe u glavnom izborniku te uključiti/isključiti vibraciju kontrolera. Slika 22 prikazuje zaslon s postavkama videoigre završnog rada.



Slika 22: Zaslon postavki videoigre završnog rada (snimka zaslona)

Osim izlaska iz igre (povratak na radnu površinu), korisnik može u bilokojem trenutku prekinuti igranje razine i vratiti se u glavni izbornik.

Objekti igre koji su korišteni za izradu glavnog izbornika su:

- Izvor zvuka

  - odnosi se na glazbu koja svira u glavnom izborniku.
  - izvor zvuka podešen je na način da se pjesma ponavlja kad završi (engl. *loop*).
- Kamera
- Modul za sustav unosa korisničkog sučelja (*Input System UI Input Module*)
  - korišten je skup radnji koji je bio definiran u sekciji 4.5.2.1.
  - upravljanje unosom u korisničkom sučelju (i za DS4 kontroler i za miš i tipkovnicu)
- Upravitelj igrom
  - sadrži jednostavne funkcije za upravljanje igrom poput započinjanja razine i učitavanja scene, izlaska iz razine i učitavanja glavnog izbornika te funkcija za potpuni izlaz iz videoigre (povratak na radnu površinu)
- Upravitelj postavkama
  - postavke koje korisnik mijenja spremaju se koristeći `PlayerPrefs` klasu
- Upravitelj razinama
  - sadrži jednostavne funkcije za upravljanje igrom poput započinjanja razine i učitavanja scene, izlaska iz razine i učitavanja glavnog izbornika te funkcija za potpuni izlaz iz videoigre (povratak na radnu površinu)
- Platno (*Canvas*)
  - odnosi se na korisničko sučelje i čine ga elementi poput teksta, klizača i potvrdnog okvira

4.8. Izrada podizbornika

Opcije izbornika s razinama i upravitelja postavkama u glavnom izborniku primjeri su podizbornika: glavni izbornik se dalje grana i korisniku se nudi više stavki. Postavke koje korisnik može mijenjati u sklopu izrađene videoigre su:

- Glasnoća glavnog izbornika (klizač)
- Glasnoća razine (klizač)
- Vibracija kontrolera (potvrdni okvir)

Vrijednosti navedenih opcija spremati će se koristeći `PlayerPrefs()` klasu. Na taj način će ih biti moguće čitati u drugoj sceni odnosno sceni razine. Kreirana je klasa `SettingsManager` za upravljanje postavkama.

Inicijalizacija vrijednosti:

```
[SerializeField] Slider menuMusicVolumeSlider;
[SerializeField] Slider levelMusicVolumeSlider;
[SerializeField] Toggle gamepadVibrationToggle;

private void Start() {
    if (!PlayerPrefs.HasKey("MenuMusicVolume")) {
        PlayerPrefs.SetFloat("MenuMusicVolume", 1f);
    }

    if (!PlayerPrefs.HasKey("LevelMusicVolume")) {
        PlayerPrefs.SetFloat("LevelMusicVolume", 1f);
    }

    Load();

    gamepadVibrationToggle.isOn = PlayerPrefs.GetInt("GamepadVibrationEnabled", 1)
    == 1;
    gamepadVibrationToggle.onValueChanged.AddListener(ChangeGamepadVibration);
}

private void Load() {
    menuMusicVolumeSlider.value = PlayerPrefs.GetFloat("MenuMusicVolume");
    levelMusicVolumeSlider.value = PlayerPrefs.GetFloat("LevelMusicVolume");
}
```

Metode za promjenu vrijednosti klizača i potvrdnog okvira:

```
public void ChangeMenuMusicVolume() {
    MusicVolumeManager.MenuMusicVolume = menuMusicVolumeSlider.value;
}

public void ChangeLevelMusicVolume() {
    MusicVolumeManager.LevelMusicVolume = levelMusicVolumeSlider.value;
}

public void ChangeGamepadVibration(bool isOn) {
    int vibrationEnabled = isOn ? 1 : 0;
    PlayerPrefs.SetInt("GamepadVibrationEnabled", vibrationEnabled);
}
```

4.9. Izrada sustava za postavke

U videoigrama najčešće se korisniku nudi da u glavnom izborniku ili tijekom pauziranja igre mijenja određene postavke. Neke od postavki koje se često nude korisniku da ih mijenja su postavke prikaza (rezolucija, kvaliteta grafike, itd.), postavke zvuka (glasnoća glazbe, glasnoća zvučnih efekata), težina igre i sl.

Primjer glavnog izbornika igre Guitar Hero III: Legends of Rock prikazan je na slici 23.

U videoigri koju obuhvaća završni rad, igrač može u postavkama podešavati glasnoću glazbe u glavnom izborniku, glasnoću glazbe razine, uključiti ili isključiti vibraciju kontrolera.

Postavke koje korisnik mijenja spremaju se koristeći `PlayerPrefs` klasu. Ta klasa sprema preference igrača kroz sesije igre. Mogu se spremati cjelobrojne i decimalne vrijednosti te znakovi. Ti podaci se spremaju bez enkripcije pa je preporučljivo da oni ne budu osjetljivi [17].

Slika 23: Glavni izbornik igre Guitar Hero III: Legends of Rock [18]

Programski kôd upravitelja postavkama svodi se na čitanje vrijednosti klizača odnosno potvrdnog okvira i ažuriranje tih vrijednosti u `PlayerPrefs` klasi:

```
1  private void Start() {
2          if (!PlayerPrefs.HasKey("MenuMusicVolume")) {
3              PlayerPrefs.SetFloat("MenuMusicVolume", 1f);
4          }
5          if (!PlayerPrefs.HasKey("LevelMusicVolume")) {
6              PlayerPrefs.SetFloat("LevelMusicVolume", 1f);
7          }
8          Load();
9          gamepadVibrationToggle.isOn = PlayerPrefs.GetInt("GamepadVibrationEnabled",
       1) == 1;
10         gamepadVibrationToggle.onValueChanged.AddListener(ChangeGamepadVibration);
11     }
12
13     private void Load() {
14         menuMusicVolumeSlider.value = PlayerPrefs.GetFloat("MenuMusicVolume");
15         levelMusicVolumeSlider.value = PlayerPrefs.GetFloat("LevelMusicVolume");
16     }
17
18     public void ChangeMenuMusicVolume() {
19         MusicVolumeManager.MenuMusicVolume = menuMusicVolumeSlider.value;
20     }
21
22     public void ChangeLevelMusicVolume() {
```

```
        MusicVolumeManager.LevelMusicVolume = levelMusicVolumeSlider.value;
    }

    public void ChangeGamepadVibration(bool isOn) {
        int vibrationEnabled = isOn ? 1 : 0;
        PlayerPrefs.SetInt("GamepadVibrationEnabled", vibrationEnabled);
    }
```

4.10. Izrada klase za upravljanje razinama

Kako se razine u ovoj igri razlikuju samo po podacima iz JSON datoteke, pogodno je kreirati klasu koja će imati metode poput iteriranja kroz razine, prikaz informacija odabrane razine i sl. Neke od funkcionalnosti takve klase `LevelSelection` dani su u nastavku, a potpuni kôd nalazi se u prilozima završnog rada. Klasa koja je izrazito korisna je `JsonUtility` jer sadrži metode za olakšano baratanje JSON podacima. Metodom `ToJson` pročitani tekst se pretvara u JSON reprezentaciju, a metodom `FromJson` se na temelju učitane JSON reprezentacije mogu kreirati objekti.

Funkcija za pokretanje razine koju je korisnik odabrao u glavnom izborniku:

```
public void PlaySelectedLevel() {
    SongData selectedLevel = levels[currentLevelIndex];

    SelectedLevelData levelDataHolder = new SelectedLevelData {
        selectedLevel = selectedLevel
    };

    string levelDataJson = JsonUtility.ToJson(levelDataHolder);

    PlayerPrefs.SetString("SelectedLevelData", levelDataJson);
    PlayerPrefs.Save();

    SceneManager.LoadScene("Game");
}
```

Funkcija za učitavanje i parsiranje svih razina:

```
private void LoadAllLevels() {
    string[] jsonFiles = Directory.GetFiles(Path.Combine(Application.
    streamingAssetsPath, "Charts"), "*.json");

    foreach (string jsonFilePath in jsonFiles) {
        string jsonText = File.ReadAllText(jsonFilePath);
        SongData levelData = JsonUtility.FromJson<SongData>(jsonText);
        levels.Add(levelData);

        Sprite coverArtSprite = LoadCoverArt(levelData.cover_art);
```

```
        if (coverArtSprite != null) {
            levelData.coverArtSprite = coverArtSprite;
        }
    }
}
```

Funkcija za prikaz detalja o pojedinoj razini:

```
private void LoadLevelData(int levelIndex) {
    levelNameText.text = levels[levelIndex].name.ToUpper();
    artistText.text = levels[levelIndex].artist.ToUpper();
    difficulty.text = "DIFFICULTY " + levels[levelIndex].difficulty.ToString() + " / 5";
    coverArtImage.sprite = levels[levelIndex].coverArtSprite;

    backButton.interactable = (levelIndex > 0);
    nextButton.interactable = (levelIndex < levels.Count - 1);
}
```

Funkcije za iteriranje kroz razine:

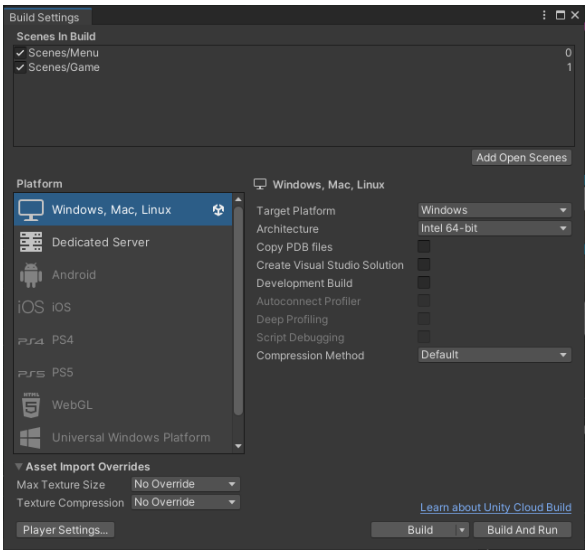
```
private void PreviousLevel() {
    currentLevelIndex--;
    currentLevelIndex = Mathf.Clamp(currentLevelIndex, 0, levels.Count - 1);
    LoadLevelData(currentLevelIndex);
}

private void NextLevel() {
    currentLevelIndex++;
    currentLevelIndex = Mathf.Clamp(currentLevelIndex, 0, levels.Count - 1);
    LoadLevelData(currentLevelIndex);
}
```

4.11. Izvoz videoigre

Trenutno je razvijena videoigra dostupna za pokretanje isključivo unutar alata Unity. Nakon što je videoigra razvijena, potrebno je obaviti *build* proces kojim se kreira izvršna datoteka videoigre te će igrači moći takvu igru pokrenuti na svojim računalima. Izvršne datoteke za videoigru moguće je kreirati za razne platforme, a u sklopu završnog rada to su: Linux, Mac OS X i Microsoft Windows.

Postavke izvoza videoigre dostupne su pod `File` > `Build Settings`. Tu je i prečac `File` > `Build And Run` kojime je moguće odmah pokrenuti postupak izvoza i pokretanja igre.



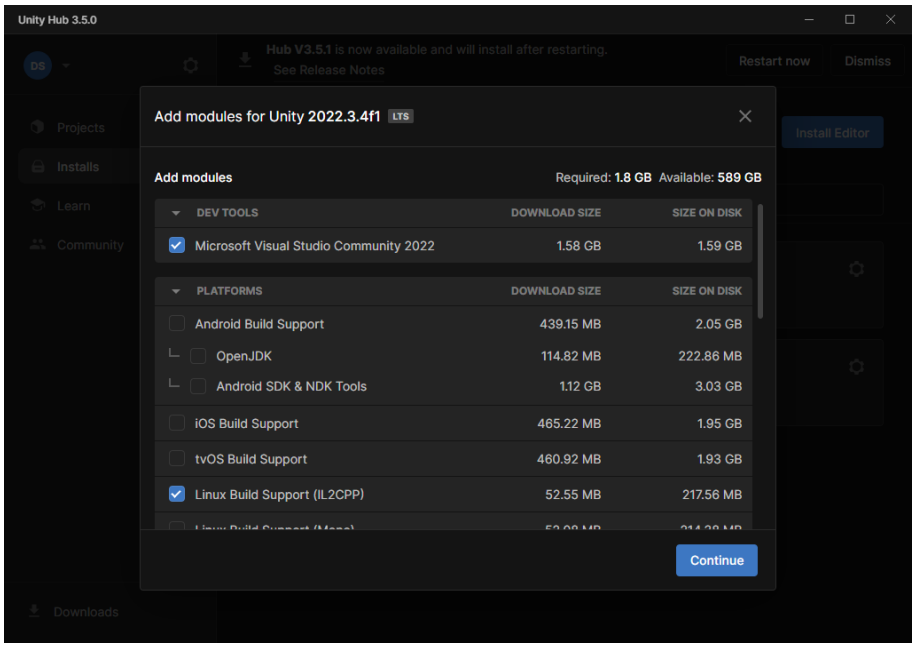
Slika 24: Postavke izvoza videoigre (snimka zaslona)

Slika 24 prikazuje prozor postavki izvoza videoigre sa odabranom opcijom izvoza za platformu Microsoft Windows. Ono što je prvo uočljivo je popis scena videoigre pod **Scenes In Build**. Svaka scena uz sebe ima potvrdni okvir jer je moguće birati koje će se scene nalaziti u sklopu izvezene videoigre. Osim toga, vidljiv je i indeks svake scene: za scenu `Menu` to je vrijednost 0, a za scenu `Game` vrijednost 1.

U nastavku se nude opcije platformi za koju će se obaviti izvoz videoigre. Potrebno je naglasiti da pri samoj instalaciji alata Unity korisnici mogu odabrati koje module za izvoz žele preuzeti i instalirati, a to je moguće obaviti i naknadno nakon instalacije. Da bi se igre mogle izvoziti na određene platforme, potrebno je preuzeti odgovarajuće module za izvoz. Autor završnog rada ima module za izvoz videoigara za platforme Linux, Mac OS X i Windows te za namjenske poslužitelje (engl. *dedicated server*) pa su mu iz tog razloga opcije za izvoz na te dvije platforme omogućene. Na slici 24 vidljivo je da su ostale opcije odnosno platforme osjenčane, odnosno nisu omogućene jer je potrebno preuzeti odgovarajuće module kako bi se igre izvozile na te platforme. Module je moguće preuzimati putem Unity Hub softvera (slika 25).

Neke od opcija koje se mogu mijenjati su: arhitektura procesora za koju se izvoz radi (32 ili 64-bit), kreiranje razvojnog izvoza (engl. *development build*), metoda kompresije i dr.

Nakon odabranih postavki, potrebno je kliknuti gumb `Build` ili `Build And Run` te će izvoz igre započeti. Kad izvoz završi, korisnik će biti obaviješten putem poruke o tome je li izvoz uspio te koliko je bilo potrebno vremena za izvoz (slika 26).



Slika 25: Moduli dostupni za preuzimanje u Unity Hub-u (snimka zaslona)

Slika 26: Obavijest o statusu izvoza videoigre (snimka zaslona)

5. Zaključak

U sklopu završnog rada kreirana je glazbena videoigra u programskom alatu u Unity. Najprije su obrađeni koncepti u programskom alatu Unity koje je potrebno razumijeti kako bi se krenulo u izradu videoigre. Zatim je korišten holistički pristup te su određeni objekti, komponente i sustavi koji će činiti videoigru. Videoigru u ovom završnom radu čine razni sustavi, od sustava za upravljanje rezultatom, sustava za upravljanje razinom, sustava za upravljanje postavkama i dr. Objašnjena je uloga pojedinog sustava te su pružene odgovarajuće skripte odnosno isječci programskog kôda.

Nakon što je videoigra izrađena, potrebno je obaviti izvoz videoigre odnosno kreirati izvršne datoteke. U sklopu završnog rada kreirane su izvršne datoteke za platforme Linux, Mac OS X i Microsoft Windows.

Najkorisniji resurs odnosno literatura pri izradi videoigre bila je Unity dokumentacija jer je vrlo detaljno napisana i čitljiva kako za iskusne korisnike tako i za početnike. Tu su još i tutorijali drugih korisnika koji su se susreli sa sličnim problemima ili izazovima pri izradi videoigara slične tematike.

Potencijalna poboljšanja razvijene videoigre mogu biti daljnje peglanje mehanika igre da ona bude još ugodnija za igranje, bolje dizajnirani grafički elementi, dodavanje mogućnosti igranja s više igrača (engl. *multiplayer*), itd. Moguće je uz samu videoigru izraditi i softver s grafičkim sučeljem za lakše kreiranje nota za pjesme pošto su u trenutnoj inačici njihove vremenske oznake pisane ručno u JSON datoteku.

Popis literature

[1] Acer Corner, *What are Rhythm Games?* Srpanj 2022. adresa: https://blog.acer.com/en/discussion/188/what-are-rhythm-games (pogledano 5. 8. 2023.).

[2] A. Frank, *PaRappa the Rapper isn't perfect, but his 20th anniversary still marks something special*, prosinac 2016. adresa: https://www.polygon.com/2016/12/6/13856718/parappa-the-rapper-ps4-20th-anniversary (pogledano 8. 9. 2023.).

[3] M. Schatten, *Bilješke s predavanja kolegija Platforme za razvoj računalnih igara - 01 Elementi platformi za razvoj igara*, ožujak 2023.

[4] Unity Technologies, *Compare plans*. adresa: https://unity.com/compare-plans (pogledano 5. 8. 2023.).

[5] Unity Learn, *Explore the Unity Editor*. adresa: https://learn.unity.com/tutorial/explore-the-unity-editor-1 (pogledano 8. 9. 2023.).

[6] Unity Technologies, *Unity - Scripting API: Component*. adresa: https://docs.unity3d.com/ScriptReference/Component.html (pogledano 15. 8. 2023.).

[7] Unity Technologies, *Unity - Manual: Materials, Shaders & Textures*. adresa: https://docs.unity3d.com/550/Documentation/Manual/Shaders.html (pogledano 27. 8. 2023.).

[8] Unity Technologies, *Unity - Manual: Creating User Interfaces (UI)*. adresa: https://docs.unity3d.com/2020.2/Documentation/Manual/UIToolkits.html (pogledano 27. 8. 2023.).

[9] LANDR Blog, *What is MIDI? How to use the most powerful tool in music*, en-US, siječanj 2020. adresa: https://blog.landr.com/what-is-midi/ (pogledano 2. 9. 2023.).

[10] Melanchall, *DryWetMIDI | Audio | Unity Asset Store*. adresa: https://assetstore.unity.com/packages/tools/audio/drywetmidi-222171 (pogledano 2. 9. 2023.).

[11] Melanchall, *DryWetMIDI*. adresa: https://github.com/melanchall/drywetmidi (pogledano 2. 9. 2023.).

[12] *How to make a Rhythm Game in Unity (Using MIDI)*. adresa: https://www.youtube.com/watch?v=ev0HsmgLScg (pogledano 5. 9. 2023.).

[13] *JSON - Introducing JSON*. adresa: https://www.json.org/json-en.html (pogledano 2. 9. 2023.).

[14] eMastered, *RMS Level for Mastering: Achieving the Perfect Loudness*. adresa: https://emastered.com/blog/rms-level-for-mastering (pogledano 8. 9. 2023.).

[15] Unity Technologies, *Unity - Manual: Input Manager*. adresa: https://docs.unity.cn/2017.3/Documentation/Manual/class-InputManager.html (pogledano 10. 8. 2023.).

[16] Unity Technologies, *Input System Workflows | Input System | 1. 6. 3*. adresa: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.6/manual/Workflows.html (pogledano 12. 8. 2023.).

[17] Unity Technologies, *Unity - Scripting API: PlayerPrefs*. adresa: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html (pogledano 3. 9. 2023.).

[18] The Cutting Room Floor, *Guitar Hero III: Legends of Rock (PlayStation 2) - The Cutting Room Floor*. adresa: https://tcrf.net/Guitar_Hero_III:_Legends_of_Rock_%28PlayStation_2%29 (pogledano 3. 9. 2023.).

Popis slika

Popis tablica

Popis priloga

Prilog 1: Klasa za upravljanje pjesmama

Prilagodba i proširenje izvornog kôda autora [12].

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Rendering.PostProcessing;
using UnityEngine.InputSystem;
using System.IO;
using System;
using Cinemachine; // Koristenje Cinemachine kamera

// Klasa koja predstavlja vremenske oznake nota za pojedinu stazu
[Serializable]
public class LaneData {
    public List<double> timestamps;
}


// Klasa koja predstavlja strukturu JSON datoteke pjesme odnosno razine
[Serializable]
public class SongData {
    // Naziv pjesme
    public string name;
    // Izvodac pjesme
    public string artist;
    // Zanr pjesme
    public string genre;
    // Naslovnica albuma (naziv slike i ekstenzija)
    public string cover_art;
    // Naziv i ekstenzija zvucne datoteke
    public string stream;
    // Tezina razine
    public int difficulty;
    // Popis vremenskih oznaka za pojedinu stazu
    public List<LaneData> lanes;
}

// Klasa za upravljanje pjesmama
```

```
38 public class SongManager : MonoBehaviour {
39     public PlayerInput playerInput;
40     // Singleton instanca
41     public static SongManager Instance;
42     // Polje podataka o razinama
43     public List<SongData> levels = new List<SongData>();
44     // Objekt zasluzan za reprodukciju pjesme u razini
45     public AudioSource audioSource;
46     // Referenca na datoteku pjesme koja svira u razini
47     public AudioClip audioClip;
48     // Polje podataka o valnom obliku pjesme
49     float[] waveformData = new float[4096];
50     // Polje proizvoljnog broja objekata staza
51     public Lane[] lanes;
52     // Parametar kasnjenja prije nego sto pjesma pocne
53     public float songDelayInSeconds;
54     // Decimalni broj tolerancije na dozvoljenu pogresku pritiska tipke i
       pojavljivanja note na ekranu
55     public double marginOfError;
56     // Kasnjenje od trenutka kad je pritisnuta tipka u milisekundama i kada je ona
       registrirana kao pritisnuta
57     public int inputDelayInMilliseconds;
58     // Putanja do JSON datoteke
59     public string jsonFileLocation;
60     // Vrijeme izmedu nota, "brzina" izvodenja igre
61     public float noteTime;
62     // Mjesto stvaranja note na Y osi
63     public float noteSpawnY;
64     // Mjesto gdje se tipka registrira kao pritisnuta na Y osi
65     public float noteTapY;
66
67     public SongData songData;
68     // Sakrivanje odnosno unistavanje nota na Y osi
69     public float noteDespawnY {
70        get {
71             // Pozicija sakrivanja note kalkulira se kao razlika mjesta stvaranja i
       mjesta registriranja
72             return 2 * noteTapY - noteSpawnY;
73         }
74     }
75
76     void Start() {
77         // Postavljanje instance upravitelja pjesmama
78         Instance = this;
79         ReadFromJSON();
80         Invoke(nameof(StartSong), songDelayInSeconds);
81     }
82
83     // Funkcija za citanje podataka iz JSON datoteke
84     private void ReadFromJSON() {
85         string jsonText = File.ReadAllText(Path.Combine(Application.
       streamingAssetsPath, jsonFileLocation));
86         songData = JsonUtility.FromJson<SongData>(jsonText);
```

```
87
88          string audioFileName = songData.stream;
89          string audioPath = Path.Combine(Application.streamingAssetsPath,
    audioFileName).Replace("\\", "/");
90          Debug.Log(audioPath);
91          StartCoroutine(LoadAudio(audioPath));
92
93          // Provjera podudaranja broja objekata staza i broja staza u JSON datoteci
94          if (songData.lanes.Count == lanes.Length) {
95              for (int i = 0; i < lanes.Length; i++) {
96                  // Provjera sadrzi li staza vremenske oznake
97                  if (songData.lanes[i].timestamps != null) {
98                      lanes[i].SetTimeStamps(songData.lanes[i].timestamps);
99                  }
100                 else {
101                     Debug.Log("Staza " + i + " nema definiranih vremenskih oznaka.")
    ;
102                 }
103             }
104         }
105         else {
106             Debug.Log("Broj objekata staza i staza u JSON datoteci se ne podudara.")
    ;
107         }
108     }
109
110     private IEnumerator LoadAudio(string audioPath) {
111         // Ucitavanje zvucne datoteke pomocu UnityWebRequest zahtjeva
112         using (UnityWebRequest www = UnityWebRequestMultimedia.GetAudioClip(
    audioPath, AudioType.OGGVORBIS)) {
113             yield return www.SendWebRequest();
114
115             // Uspjesno ucitavanje zvucne datoteke
116             if (www.result == UnityWebRequest.Result.Success) {
117                 // Dodjela zvucnog zapisa izvoru zvuka
118                 AudioClip audioClip = DownloadHandlerAudioClip.GetContent(www);
119                 audioSource.clip = audioClip;
120             }
121             // Neuspjesno ucitavanje zvucne datoteke
122             else {
123                 Debug.Log("Neuspjesno ucitavanje zvucne datoteke: " + www.error);
124             }
125         }
126     }
127
128     // Funkcija za sviranje pjesme
129     public void StartSong() {
130         // Pocetak sviranja pjesme - odnosi se na mp3/ogg/wav datoteku
131         if (audioSource != null) {
132             audioSource.Play();
133         }
134     }
135
```

```
136     // Funkcija za dohvacanje trenutnog vremena sviranja u sekundama
137     public static double GetAudioSourceTime() {
138         // Potrebno je pretvoriti vrijeme iz vremenskih uzoraka (engl. samples) u
        sekunde
139         return (double)Instance.audioSource.timeSamples / Instance.audioSource.clip.
        frequency;
140     }
141
142     private void CheckSongEnd() {
143         // Provjera je li pjesma gotova
144         if (GetAudioSourceTime() >= Instance.audioSource.clip.length) {
145             Debug.Log("Pjesma je gotova");
146         }
147     }
148
149     void Update() {
150         CheckSongEnd();
151         // Dohvacanje waveform podataka odnosno signala zvuka iz datoteke
152         audioSource.GetOutputData(waveformData, 0);
153     }
154 }
```

Programski kôd 1: Klasa za upravljanje pjesmama

Prilog 2: Klasa za upravljanje rezultatom

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System.IO;

public class ScoreManager : MonoBehaviour {
    public static ScoreManager Instance;
    public Canvas scoreUI;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI comboText;
    public static int comboScore;
    public static int comboMultiplier;
    void Start() {
        Instance = this;
        comboScore = 0;
        comboMultiplier = 1;

        scoreText = scoreUI.GetComponentInChildren<TextMeshProUGUI>();
        comboText = scoreUI.transform.Find("ComboText").GetComponent<TextMeshProUGUI
    >();
    }
    public static void Hit() {
        comboScore += 100 * comboMultiplier;
        comboMultiplier++;
    }

    public static void Miss() {
        comboMultiplier = 1;

        if (comboScore > 0)
            comboScore -= 100 * comboMultiplier;
    }

    private void Update() {
        scoreText.text = comboScore.ToString("n0");
        comboText.text = comboMultiplier.ToString() + "x!";
    }
}
```

Programski kôd 2: Klasa za upravljanje rezultatom

Prilog 3: Klasa za upravljanje razinama

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using System;
using System.IO;
using TMPro;
[Serializable] public class SelectedLevelData {
    public SongData selectedLevel;
}

public class LevelSelection : MonoBehaviour {
    public TextMeshProUGUI levelNameText;
    public TextMeshProUGUI artistText;
    public TextMeshProUGUI difficulty;
    public Image coverArtImage;
    public Button backButton;
    public Button nextButton;

    public List<SongData> levels;
    private int currentLevelIndex = 0;

    private void Start() {
        LoadAllLevels();
        LoadLevelData(currentLevelIndex);

        backButton.onClick.AddListener(() => PreviousLevel());
        nextButton.onClick.AddListener(() => NextLevel());
    }

    public void PlaySelectedLevel() {
        SongData selectedLevel = levels[currentLevelIndex];

        SelectedLevelData levelDataHolder = new SelectedLevelData {
            selectedLevel = selectedLevel
        };

        string levelDataJson = JsonUtility.ToJson(levelDataHolder);

```

```
40          PlayerPrefs.SetString("SelectedLevelData", levelDataJson);
41          PlayerPrefs.Save();
42
43          SceneManager.LoadScene("Game");
44      }
45
46      private void LoadAllLevels() {
47          string[] jsonFiles = Directory.GetFiles(Path.Combine(Application.
    streamingAssetsPath, "Charts"), "*.json");
48
49          foreach (string jsonFilePath in jsonFiles) {
50              string jsonText = File.ReadAllText(jsonFilePath);
51              SongData levelData = JsonUtility.FromJson<SongData>(jsonText);
52              levels.Add(levelData);
53
54              Sprite coverArtSprite = LoadCoverArt(levelData.cover_art);
55              if (coverArtSprite != null) {
56                  levelData.coverArtSprite = coverArtSprite;
57              }
58          }
59      }
60
61      private void LoadLevelData(int levelIndex) {
62          levelNameText.text = levels[levelIndex].name.ToUpper();
63          artistText.text = levels[levelIndex].artist.ToUpper();
64          difficulty.text = "DIFFICULTY " + levels[levelIndex].difficulty.ToString() +
     " / 5";
65          coverArtImage.sprite = levels[levelIndex].coverArtSprite;
66
67          backButton.interactable = (levelIndex > 0);
68          nextButton.interactable = (levelIndex < levels.Count - 1);
69      }
70
71      private void PreviousLevel() {
72          currentLevelIndex--;
73          currentLevelIndex = Mathf.Clamp(currentLevelIndex, 0, levels.Count - 1);
74          LoadLevelData(currentLevelIndex);
75      }
76
77      private void NextLevel() {
78          currentLevelIndex++;
79          currentLevelIndex = Mathf.Clamp(currentLevelIndex, 0, levels.Count - 1);
80          LoadLevelData(currentLevelIndex);
81      }
82
83      private void ReturnToMainMenu() {
84          SceneManager.LoadScene("MainMenu");
85      }
86
87      private Sprite LoadCoverArt(string coverArtFilePath) {
88          string fullPath = Path.Combine(Application.streamingAssetsPath, "CoverArt",
    coverArtFilePath).Replace("\\", "/");
89          Debug.Log(fullPath);
```

```
90
91          if (File.Exists(fullPath)) {
92              byte[] data = File.ReadAllBytes(fullPath);
93              Texture2D texture = new Texture2D(2, 2);
94              if (texture.LoadImage(data)) {
95                  return Sprite.Create(texture, new Rect(0, 0, texture.width, texture.
    height), Vector2.zero);
96              }
97          }
98
99          Debug.Log("Naslovnica albuma nije pronadena: " + coverArtFilePath);
100         return null;
101     }
102 }
```

Programski kôd 3: Klasa za upravljanje razinama

Prilog 4: Klasa nota

Prilagodba i proširenje izvornog kôda autora [12].

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Note : MonoBehaviour {
    double timeInstantiated;
    public float assignedTime;
    float startY;
    float endY;
    float startTime;
    float noteMoveSpeed;
    private SpriteRenderer spriteRenderer;

    void Start() {
        timeInstantiated = assignedTime - SongManager.Instance.noteTime;
        // Pozicija na kojoj je nota stvorena
        startY = SongManager.Instance.noteSpawnY;
        // Pozicija na kojoj ce se nota unistiti
        endY = SongManager.Instance.noteDespawnY;
        startTime = Time.time;
        noteMoveSpeed = (endY - startY) / (SongManager.Instance.noteTime * 2);
        // Postavljanje pocetne pozicije note
        transform.localPosition = new Vector3(transform.localPosition.x, startY,
    transform.localPosition.z);
        // Aktiviranje objekta note i postavljanje da nota bude vidljiva
        gameObject.SetActive(true);
        spriteRenderer = GetComponent<SpriteRenderer>();
        spriteRenderer.enabled = true;
    }

    void Update() {
        // Provjera je li instanca upravitelja pjesme null
        if (SongManager.Instance != null) {
            float timeSinceStart = (float)SongManager.GetAudioSourceTime() -
    startTime;
            float t = timeSinceStart / (SongManager.Instance.noteTime * 2);
            // Unistavanje nota
            if (t > 1) {
                Destroy(gameObject);
            }
```

```
39              else {
40                  // Pomicanje nota zeljenom brzinom
41                  transform.localPosition += Vector3.up * noteMoveSpeed * Time.
    deltaTime;
42              }
43          }
44      }
45  }
```

Programski kôd 4: Klasa nota

Prilog 5: Klasa staza

Prilagodba i proširenje izvornog kôda autora [12].

```
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class Lane : MonoBehaviour {
    public GameObject notePrefab;
    public List<double> timeStamps = new List<double>();
    public List<Note> notes = new List<Note>();

    int spawnIndex = 0;
    int inputIndex = 0;

    private bool isButtonPressed = false;
    private float buttonMoveStartTime;
    public Transform buttonTransform;
    private Vector3 buttonOriginalPosition;
    private float buttonMoveDuration = 0.2f;

    public InputAction noteAction;

    public ParticleSystem particles;
    [SerializeField] private Color particlesColor;

    public void SetTimeStamps(List<double> laneTimestamps) {
        timeStamps = laneTimestamps;
    }

    void Start() {
        noteAction.Enable();
        buttonOriginalPosition = buttonTransform.position;
        buttonTransform.position -= new Vector3(0f, 0.0f, 0.2f);
    }

    void Update() {
        if (Time.timeScale == 0) {
            return;
        }

        if (spawnIndex < timeStamps.Count) {
```

```
41              if (SongManager.GetAudioSourceTime() >= timeStamps[spawnIndex] -
    SongManager.Instance.noteTime) {
42                  var note = Instantiate(notePrefab, transform);
43                  notes.Add(note.GetComponent<Note>());
44                  note.GetComponent<Note>().assignedTime = (float)timeStamps[
    spawnIndex];
45                  spawnIndex++;
46              }
47          }
48
49          if (inputIndex < timeStamps.Count) {
50              double timeStamp = timeStamps[inputIndex];
51              double marginOfError = SongManager.Instance.marginOfError;
52              double audioTime = SongManager.GetAudioSourceTime() - (SongManager.
    Instance.inputDelayInMilliseconds / 1000.0);
53
54              if (noteAction.triggered) {
55                  if (!isButtonPressed) {
56                      buttonMoveStartTime = Time.time;
57                      isButtonPressed = true;
58                  }
59                  if (Math.Abs(audioTime - timeStamp) < marginOfError) {
60                      Hit();
61                  }
62              }
63
64              if (timeStamp + marginOfError <= audioTime) {
65                   Miss();
66              }
67
68              if (isButtonPressed) {
69                  float t = (Time.time - buttonMoveStartTime) / buttonMoveDuration;
70                  buttonTransform.position = Vector3.Lerp(buttonOriginalPosition,
    buttonOriginalPosition - new Vector3(0f, 0.0f, 0.2f), t);
71                  if (t >= 1.0f) {
72                      isButtonPressed = false;
73                  }
74              }
75          }
76      }
77
78
79      void Hit() {
80          ScoreManager.Hit();
81          Debug.Log($"Pogodak note");
82          Destroy(notes[inputIndex].gameObject);
83          inputIndex++;
84          particles.Play();
85      }
86
87      void Miss() {
88          ScoreManager.Miss();
89          inputIndex++;
```

```
90          Debug.Log($"Promasaj note");
91          particles.Stop();
92      }
93  }
```

Programski kôd 5: Klasa staza

Prilog 6: Klasa za upravljanje postavkama

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public static class MusicVolumeManager {
    private static float menuMusicVolume = 1f;
    private static float levelMusicVolume = 1f;

    public static float MenuMusicVolume {
        get { return menuMusicVolume; }
        set
        {
            menuMusicVolume = Mathf.Clamp01(value);
            PlayerPrefs.SetFloat("MenuMusicVolume", menuMusicVolume);
            AudioSource menuMusic = GameObject.Find("MenuMusic").GetComponent<
    AudioSource>();
            if (menuMusic != null) {
                menuMusic.volume = menuMusicVolume;
            }
        }
    }

    public static float LevelMusicVolume {
        get { return levelMusicVolume; }
        set {
            levelMusicVolume = Mathf.Clamp01(value);
            PlayerPrefs.SetFloat("LevelMusicVolume", levelMusicVolume);
        }
    }
}
public class SettingsManager : MonoBehaviour {
    [SerializeField] Slider menuMusicVolumeSlider;
    [SerializeField] Slider levelMusicVolumeSlider;
    [SerializeField] Toggle gamepadVibrationToggle;

    private void Start() {
        if (!PlayerPrefs.HasKey("MenuMusicVolume")) {
            PlayerPrefs.SetFloat("MenuMusicVolume", 1f);
```

```
39          }
40
41          if (!PlayerPrefs.HasKey("LevelMusicVolume")) {
42              PlayerPrefs.SetFloat("LevelMusicVolume", 1f);
43          }
44
45          Load();
46
47          gamepadVibrationToggle.isOn = PlayerPrefs.GetInt("GamepadVibrationEnabled",
    1) == 1;
48          gamepadVibrationToggle.onValueChanged.AddListener(ChangeGamepadVibration);
49      }
50
51      private void Load() {
52          menuMusicVolumeSlider.value = PlayerPrefs.GetFloat("MenuMusicVolume");
53          levelMusicVolumeSlider.value = PlayerPrefs.GetFloat("LevelMusicVolume");
54      }
55
56      public void ChangeMenuMusicVolume() {
57          MusicVolumeManager.MenuMusicVolume = menuMusicVolumeSlider.value;
58      }
59
60      public void ChangeLevelMusicVolume() {
61          MusicVolumeManager.LevelMusicVolume = levelMusicVolumeSlider.value;
62      }
63
64      public void ChangeGamepadVibration(bool isOn) {
65          int vibrationEnabled = isOn ? 1 : 0;
66          PlayerPrefs.SetInt("GamepadVibrationEnabled", vibrationEnabled);
67      }
68  }
```

Programski kôd 6: Klasa za upravljanje postavkama

Prilog 7: Klasa za postprocesiranje

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Rendering.PostProcessing;
using UnityEngine.InputSystem;
using System.IO;
using System;
public class PostProcessing : MonoBehaviour {
    public PlayerInput playerInput;
    public AudioSource audioSource;
    public float frequency = 1.0f;
    public PostProcessVolume volume;
    float[] waveformData = new float[4096];

    void Update() {
        // Dohvacanje waveform podataka odnosno signala zvuka iz datoteke
        audioSource.GetOutputData(waveformData, 0);

        // Provjera je li igra u fokusu / pauzirana - ako igra nije u fokusu ili je pauzirana, ugasi vibraciju kontrolera
        if (playerInput.currentControlScheme != "Gamepad") {
            StopRumble();
        }

        // Provjera je li vibracija ukljucena u postavkama
        int vibrationEnabled = PlayerPrefs.GetInt("GamepadVibrationEnabled", 1);

        if (Time.timeScale == 0) {
            if (Gamepad.current != null) {
                StopRumble();
            }
            return;
        }

        // Varijable za efekt sjaja u postprocesiranju i vibraciju kontrolera
        float intensity = 0.0f;
        float rms = 0.0f;
        float duration = 0.04f;

        for (int i = 0; i < waveformData.Length; i++) {
            intensity += Mathf.Abs(waveformData[i]);
```

```
42              rms += waveformData[i] * waveformData[i];
43          }
44
45          intensity /= waveformData.Length;
46
47          // Primjena logaritamske skale za jacinu efekta sjaja odnosno bloom efekta postprocesiranja
48          const float INTENSITY_SCALE = 4.0f; // Proizvoljna vrijednost za skalu jacine efekta
49          intensity = Mathf.Log10(intensity * INTENSITY_SCALE + 1.0f);
50
51          // Postavljanje jacine bloom efekta
52          Bloom bloom = volume.profile.GetSetting<Bloom>();
53          bloom.intensity.value = intensity * 60;
54
55          rms = Mathf.Sqrt(rms / waveformData.Length);
56
57          const float RMS_THRESHOLD = 0.06f;  // Varijabla osjetljivosti na zvucne valove
58          if (rms > RMS_THRESHOLD) {
59              // Jacina vibriranja
60              float rumbleStrength = intensity;
61
62              // Slanje vibracije na kontroler i zaustavljanje vibracije nakon odredenog vremena
63              if (Gamepad.current != null && vibrationEnabled == 1) {
64                  Gamepad.current.SetMotorSpeeds(rumbleStrength, rumbleStrength);
65              }
66              if (Gamepad.current != null) {
67                  Invoke("StopRumble", duration);
68              }
69          }
70      }
71
72      // Metoda za prestanak vibracija kontrolera, postavljanje vibracije lijevog i desnog motora na vrijednost 0.0
73      void StopRumble() {
74          Gamepad.current.SetMotorSpeeds(0.0f, 0.0f);
75      }
76  }
```

Programski kôd 7: Klasa za postprocesiranje