Izrada videoigre proklizavanja (driftanja) u programskom alatu Unity

Mikulan, Viktor

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:381750

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Viktor Mikulan

Izrada 3D videoigre proklizavanja (driftanja) u programskom alatu Unity

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Viktor Mikulan

Matični broj: 0016154974

Studij: Informacijski i poslovni sustavi – Umreženi sustavi i računalne igre

Izrada 3D videoigre proklizavanja (driftanja) u programskom alatu Unity

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Mladen Konecki

Varaždin, srpanj 2024.

Viktor Mikulan

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

Ovaj završni rad bavi se razvojem 3D igre proklizavanja koristeći Unity, popularni alat za razvoj igara. U radu ćemo najprije analizirati postojeće igre u ovom žanru kako bismo razumjeli njihove ključne komponente i dizajnerske odluke. Usporediti ću njihove mehanike, dizajn, te identificiranje ključnih značajki koje su utjecale na razvoj igre.

Nakon toga, rad se fokusira na osnovne aspekte rada u Unity okruženju, uključujući korištenje Unity Hub-a, importiranje asseta te osnovne postavke projekta. Detaljno ćemo opisati implementaciju ključnih komponenti igre, kao što su kontroler automobila, sustav kamere, zvučni efekti, finish line, upravljanje razinama i glavni izbornik. Uz to ću objasniti sve što se treba posložiti u Unity editoru.

Ključne riječi: 3D igra, Unity, kontroler automobila, proklizavanje/drift, zvučni efekti, finish line, upravljanje razinama, dizajn igre

Sadržaj

1. Uvod

Ovaj završni rad usmjeren je na razvoj 3D igre s elementima proklizavanja, koja uključuje implementaciju različitih ključnih komponenti igre u Unityu. Razvijena igra kombinira različite aspekte igre, uključujući kontrolu automobila, zvučne efekte, sustav vremena, završni ekran i glavne izbornike. Kroz ovaj rad, detaljno ćemo objasniti svaki korak razvoja, uključujući importiranje asseta, implementaciju skripti i podešavanje scena, kako bismo prikazali kako su svi ovi elementi povezani za stvaranje koherentne igre.

Prvo ćemo se fokusirati na osnove Unity-a, uključujući osnovne komponente i alate koje smo koristili. Razmotrit ćemo kako smo importirali assete kao što su modeli automobila, particle efekti za dim, low-poly ceste i okruženje. Zatim ćemo detaljno obraditi ključne aspekte razvoja igre, uključujući implementaciju kontrolera automobila, kamere, zvučnih efekata, tajmera, završetka nivoa, upravljanja nivoima i glavnog izbornika. Objasnit ću kako su ove komponente doprinijele stvaranju igre i kako su se integrirale u cjelokupni projekt.

2. Metode i tehnike rada

U razradi ovog završnog rada koristio sam različite metode i tehnike koje su mi omogućile razvoj Unity 3D igru proklizavanja(drift). Proces je obuhvatio nekoliko ključnih faza: istraživanje, dizajn, razvoj i kontinuirano testiranje.

2.1. Istraživanje

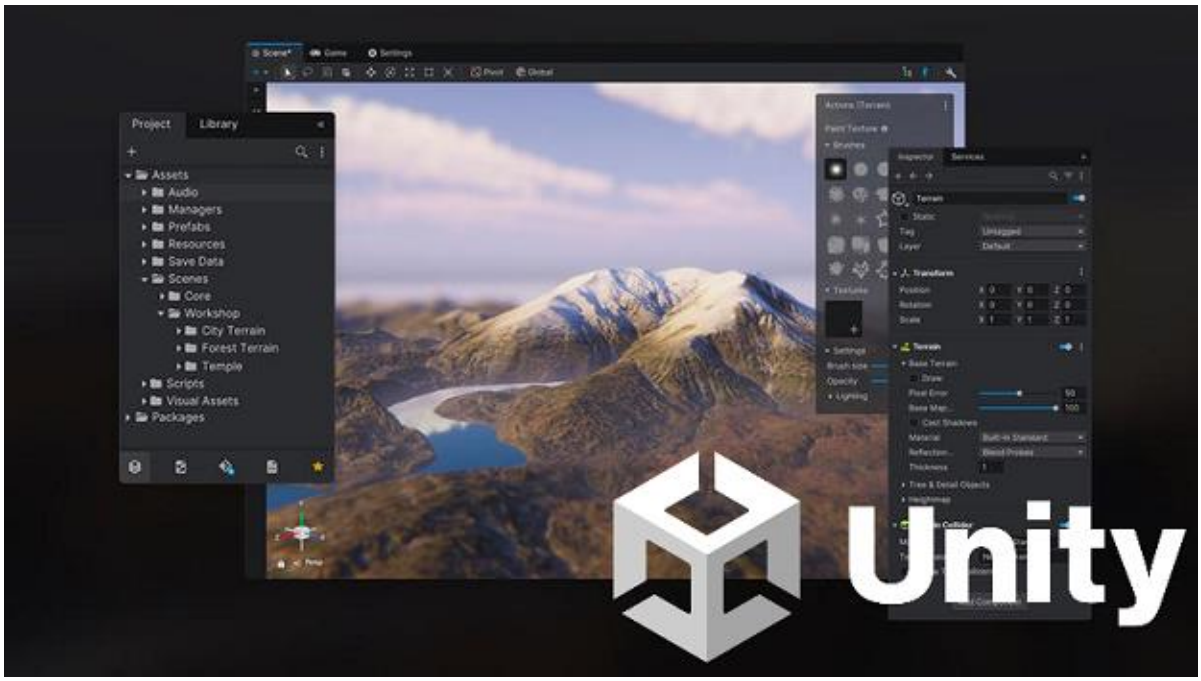
Na početku sam proveo temeljito istraživanje sličnih projekata, analizirajući mehanike igra sa autima, fizikalne modele vozila i tehnike optimizacije u drift igrama. Budući da već imam iskustva s drift igrama, to mi je omogućilo dublje razumijevanje ključnih elemenata i specifičnih izazova u razvoju ovog tipa igre. Proučavao sam relevantnu literaturu i online materijale kako bih dodatno proširio svoje znanje o razvoju igara u Unity 3D. Također sam istražio najbolje prakse za integraciju 3D modela i tekstura unutar Unity-a, što mi je bilo ključno za kreiranje vizualno atraktivnog okruženja i automobila unutar igre.

2.2. Programski alati i tehnologije

Za razvoj igre koristio sam Unity 3D, integrirano razvojno okruženje koje omogućuje izradu 3D igara. Sav kod sam pisao u programskom jeziku C#. Za potrebe igre, preuzeo sam 3D modele vozila i drugih objekata s Unity Asset Store-a. Također, koristio sam Photoshop za izradu i prilagodbu tekstura, te Audacity za obradu zvuka.

2.2.1.Unity 3d

Unity 3D je integrirano razvojno okruženje (IDE) koje omogućuje izradu 2D i 3D igara. Radi se o snažnom alatu koji pruža sve potrebno za razvoj igara, uključujući mogućnost kreiranja scena, upravljanja resursima, implementacije fizikalnih pravila, animacija i drugih aspekata igre.



Slika 1: Unity (Microsoft,2024)

U Unity-ju sam razvio kompletnu igru, od postavljanja osnovne scene do implementacije svih gameplay mehanika, uključujući fizikalni model, kontrole vozila i interakciju igrača s okolinom.

2.2.2.C# i Visual Studio

C# je programski jezik razvijen od strane Microsofta, poznat po svojoj svestranosti i korištenju u razvoju aplikacija i igara. Visual Studio je integrirano razvojno okruženje (IDE) koje nudi napredne alate za razvoj i debugiranje koda.

C# sam koristio za pisanje svih skripti koje pokreću logiku igre, dok sam Visual Studio koristio kao razvojno okruženje za pisanje, uređivanje i debugiranje koda.

2.2.3.Unity Asset Store

Unity Asset Store je online trgovina gdje programeri mogu kupovati i preuzimati razne resurse za svoje projekte, uključujući 3D modele, teksture, zvukove i druge alate.

2.2.4.Photoshop

Photoshop je vodeći softver za uređivanje grafike i fotografija, široko korišten u različitim industrijama, uključujući razvoj igara.

2.2.5. Audacity

Audacity je softver za obradu zvuka koji omogućuje snimanje i uređivanje audio zapisa.

2.3. Razvoj

- Fizikalni model: Razvio sam fizikalni model vozila koristeći Unity-jev fizikalni engine, s naglaskom na korištenje Rigidbody i WheelCollider komponenti. U igri, svaki kotač ima pridruženi WheelCollider i 3D model, a pomoću C# skripte upravljam kretanjem vozila, proklizavanjem, kočenjem i efektima.
- Programske tehnike: Fokusirao sam se na optimizaciju performansi kroz korišitenje „low poly“ 3D modela te pravilnu uporabu fizikalnih i grafičkih resursa. To je uključivalo preuzimanje optimiziranih modela s Unity Asset Store-a i smanjenje nepotrebnih kalkulacija tijekom igre.
- Kontrola igre i logika: Sva logika igre, uključujući upravljanje vozilom, implementirana je kroz skriptiranje u C# jeziku. Ova skripta kontrolira sve aspekte igre, od upravljanja vozilom do postavljanja ciljeva i mjerenja vremena te ostalih značajki poput level managera koji se koristi za početni izbornik i selektor razina.

2.4. Kontinuirano testiranje

Tijekom razvoja igre, istovremeno sam provodio kontinuirano testiranje kako bih osigurao ispravnost svake funkcionalnosti. Nakon implementacije novih značajki, odmah bih ih testirao, što mi je omogućilo brzo identificiranje i ispravljanje grešaka. Ovaj pristup omogućio je postepeno poboljšanje igre tijekom cijelog procesa razvoja.

3. Razrada teme

Ovo poglavlje predstavlja ključni dio rada u kojem ću detaljno razraditi proces razvoja Unity 3D drift igre. Razrada je podijeljena u nekoliko podpoglavlja koja pokrivaju sve aspekte razvoja, od istraživanja postojećih drift igara, definiranja mehanika igranja, tehničke implementacije, pa sve do testiranja i evaluacije rezultata.

3.1. Istraživanje i usporedba postojećih drift igara

Kako bih dao bolji uvid u mehanike koje su ključne za razvoj drift igre, istražio sam nekoliko popularnih igara u ovom žanru. Fokusirao sam se na mehanike poput driftanja, različitih razina i mapa, vizualnih efekata kao što su tragovi guma i dim, te zvuka automobila.

3.1.1. Assetto Corsa

Assetto Corsa je poznata po svojoj realističnoj simulaciji vožnje, a drift mehanika u ovoj igri je jedna od najpreciznijih na tržištu. Igra koristi napredan fizikalni model koji simulira ponašanje vozila na stazi s izuzetno visokom razinom detalja. Driftanje u Assetto Corsa zahtijeva precizno upravljanje, kontrolu gasa i kočenja.

Slika 2: Assetto Corsa (Steam,2024)

3.1.1.1. Razine

Assetto Corsa nudi razne staze, uključujući prave staze koje se nalaze u našem svijetu i izmišljene staze koje su prilagođene za različite vrste utrka, uključujući driftanje. Svaka staza je dizajnirana s pažnjom prema detaljima, što omogućuje igračima da vježbaju i usavršavaju svoje vještine driftanja na različitim vrstama terena.

3.1.1.2. Vizualni efekti

Vizualni efekti u Assetto Corsa su impresivni, posebno kada je riječ o tragovima guma i dimu tijekom driftanja. Igra precizno simulira kako se gume troše i ostavljaju tragove na asfaltu, dok se dim stvara izuzetno realistično, prateći pokrete vozila i reakcije vozača.

3.1.1.3. Zvuk vozila

Zvuk u Assetto Corsa igra ključnu ulogu u stvaranju osjećaja realizma. Zvuk motora, guma i okoline precizno je prilagođen kako bi reflektirao stanje vozila, promjene u brzini i intenzitet driftanja. To omogućuje igračima da intuitivno prate ponašanje vozila i reagiraju na promjene u vožnji.

3.1.2. BeamNG.drive

BeamNG.drive je poznata po svojoj naprednoj fizici i mogućnostima simulacije sudara. Iako nije striktno drift igra, BeamNG.drive omogućuje vrlo realistične simulacije driftanja zahvaljujući svojoj fizici. Ova igra omogućuje igračima da osjete težinu i dinamiku vozila dok driftaju, pružajući vrlo autentično iskustvo.

Slika 3: BeamNG.drive (Steam,2024)

3.1.2.1. Razine

BeamNG.drive nudi veliki broj mapa koje uključuju raznolike terene, od gradskih ulica do ruralnih područja i planinskih cesta. Ove mape su vrlo detaljne i omogućuju igračima da istražuju različite uvjete za driftanje.

3.1.2.2. Vizualni efekti

Efekti poput tragova guma i dima su vrlo detaljni u BeamNG.drive. Tragovi guma ostaju na cesti i mijenjaju se ovisno o intenzitetu driftanja, dok se dim stvara realistično i prilagođava brzini i kutu driftanja, što doprinosi impresivnoj vizualnoj prezentaciji igre.

3.1.2.3. Zvuk vozila

U BeamNG.drive, zvuk automobila je pažljivo usklađen s fizikom vozila. Zvukovi motora, guma koje klize po cesti, te sudara, doprinose realističnosti i uranjanju igrača u igru. Svaka promjena u dinamici vožnje jasno se čuje kroz promjene u zvuku, što pomaže igračima u kontroliranju vozila.

3.1.3. Forza Horizon 3

Forza Horizon 3 nudi manje realistični pristup driftanju u usporedbi s Assetto Corsom, ali i dalje pruža zadovoljavajuće iskustvo za igrače koji vole drift igre. Igra nudi različite postavke za upravljanje koje omogućuju igračima prilagodbu vozila kako bi se postigli željeni drift rezultati, kombinirajući realizam i pristupačnost.

Slika 4: Forza Horizon 3 (Microsoft,2024)

3.1.3.1. Razine

Forza Horizon 3 ističe se po svom otvorenom svijetu koji igračima omogućava slobodno istraživanje različitih okruženja, od urbanih područja do pustinjskih cesta i tropskih plaža.

Svaka mapa pruža jedinstvene izazove za driftanje, s različitim vrstama cesta i terena koji igračima omogućuju vježbanje i natjecanje u raznim uvjetima.

3.1.3.2. Vizualni efekti

Vizualni efekti u Forza Horizon 3 su vrlo impresivni, posebno kada je riječ o tragovima guma i dimu. Igra koristi napredne grafičke tehnologije za prikazivanje realističnih tragova guma koje ostaju na cesti i postupno nestaju, dok dim iz guma stvara dinamične oblake koji prate pokrete vozila. Ovi efekti doprinose ukupnoj atmosferi igre i pomažu u stvaranju dojma brzine i intenziteta driftanja.

3.1.3.3. Zvuk vozila

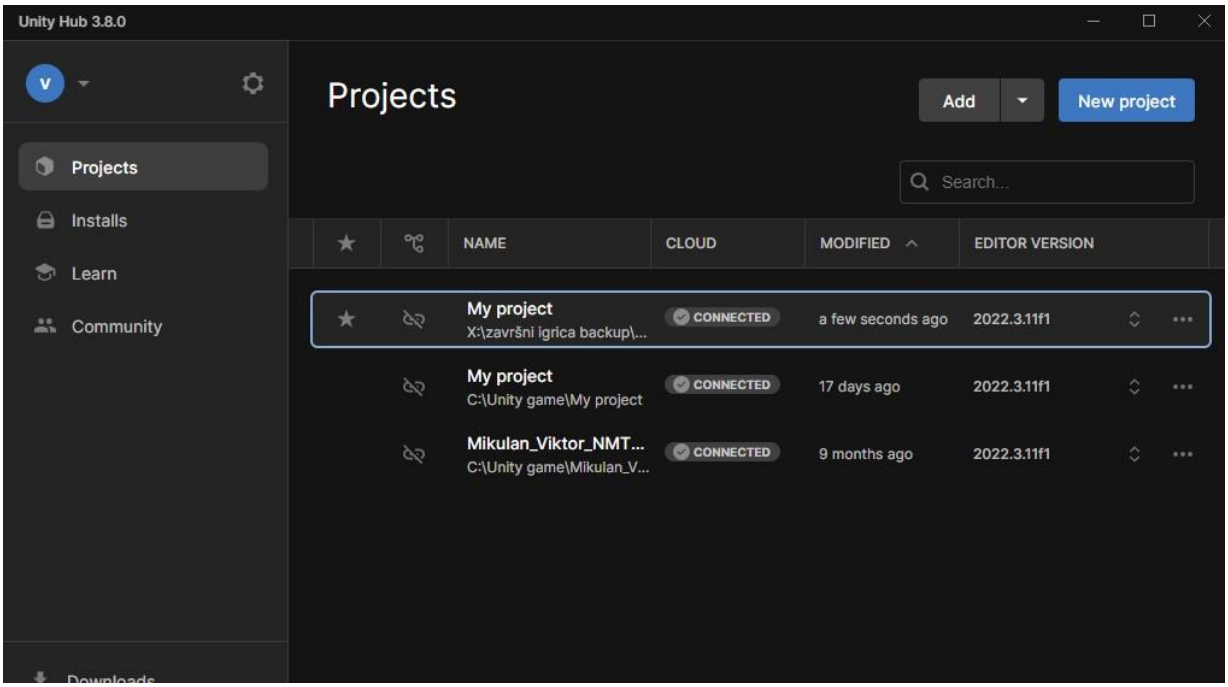
Zvuk u Forza Horizon 3 je izuzetno važan za stvaranje uranjajućeg iskustva. Zvukovi motora su autentični i prilagođeni svakom tipu automobila, dok zvuk guma i okoline pomaže igračima da osjete kako se vozilo ponaša na različitim podlogama. Ova pažnja prema detaljima u zvuku doprinosi općem osjećaju realističnosti i dinamike u igri.

3.2. Osnove Unity-a

U ovom dijelu ćemo prvo ukratko objasniti osnovne funkcionalnosti Unity-a, to uključuje korisničko sučelje unity-a i ključne elemente koji su sudjelovali u razvoju igre.

3.2.1.Unity Hub

Unity Hub je alat koji služi kao središnje mjesto za upravljanje Unity projektima. Kroz Unity Hub korisnici mogu lako kreirati nove projekte, upravljati postojećima, instalirati i ažurirati verzije Unity-a te integrirati dodatne alate poput paketa iz Unity Asset Store-a. Također nudi mogućnost podešavanja postavki za izvoz igara na različite platforme.



Slika 5: Unity Hub

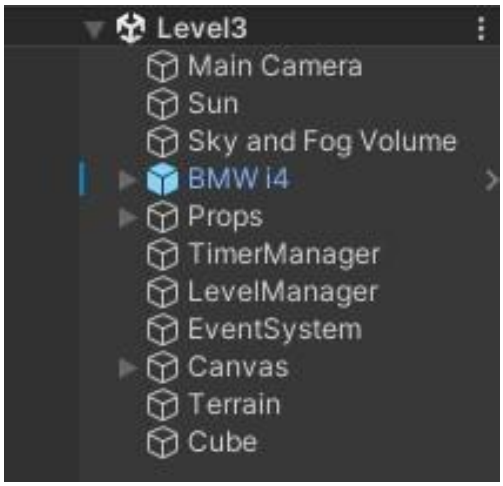
3.2.2. Sučelje Unity-a

Unity sučelje je vrlo intuitivno i sastoji se od više ključnih dijelova koji omogućuju razvoj igara.

Slika 6: Prikaz cjelokupnog sučelja

3.2.2.1. Hierarhija

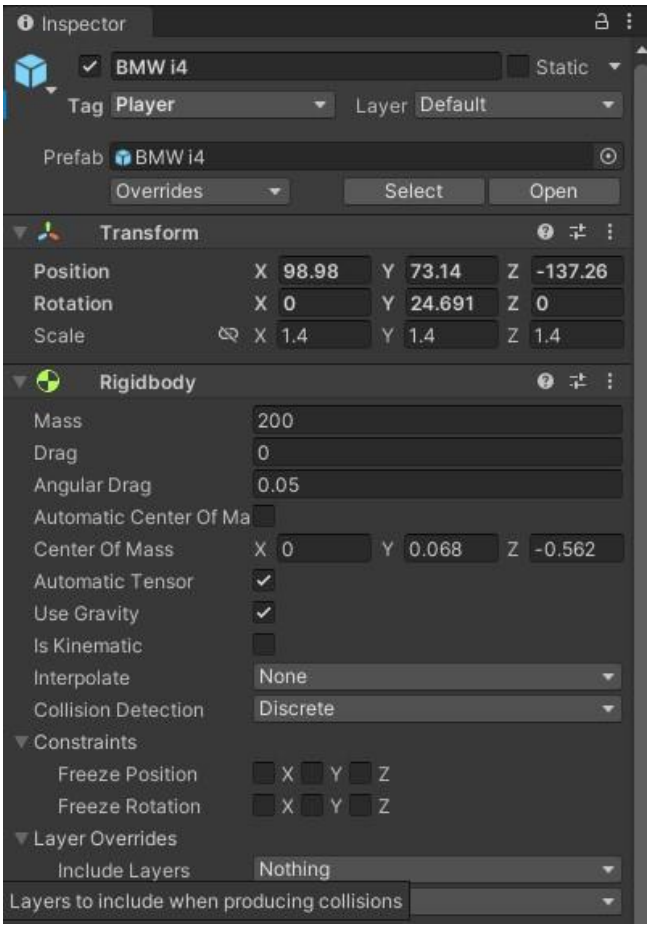
Hierarchy prikazuje sve objekte unutar trenutne scene. Svaki objekt u sceni, poput kamera, svjetla, modela, zvukova ili UI elemenata, je hijerarhijski prikazan. Ovaj panel omogućava organizaciju objekata, grupiranje i jednostavan pristup svakoj komponenti u sceni.



Slika 7: Prikaz hierarhije

3.2.2.2. Inspektor

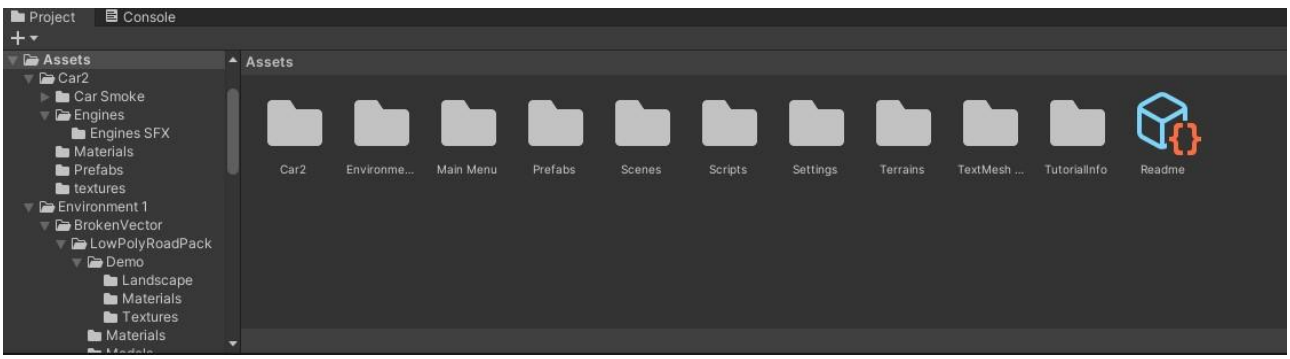
Inspektor je panel koji prikazuje sve komponente odabranog objekta u hijerarhiji. Ovdje mijenjamo svojstva objekata, prilagođavamo skripte, podešavamo fiziku, materijale i animacije. Svaki objekt u Unity-u može sadržavati više komponenti koje su vidljive i prilagodljive kroz Inspector.



Slika 8: Primjer inspektora za model auta

3.2.2.3. Prozor projekta

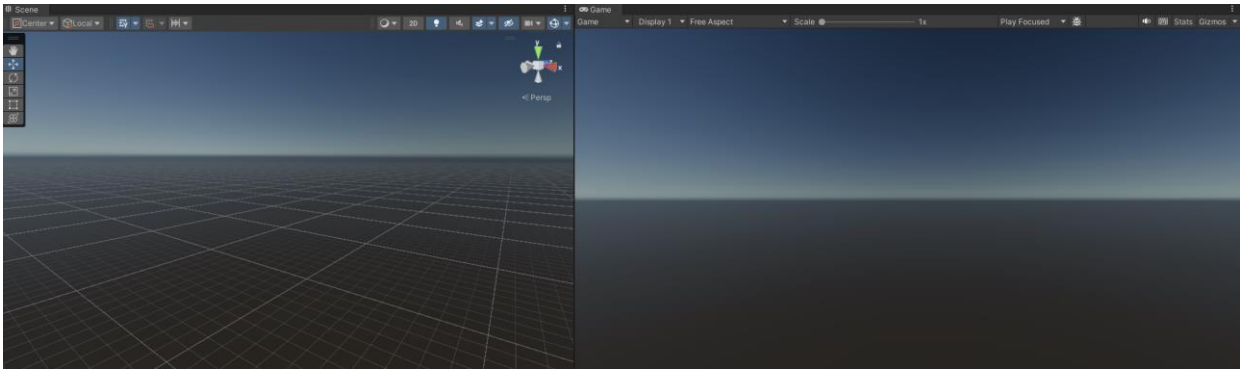
Prozor projekta (Project Window) je mjesto gdje su prikazani svi resursi projekta, poput 3D modela, tekstura, zvukova, skripti i drugih datoteka. Resursi su organizirani u mape, a iz ovog prozora se lako mogu dodavati u scenu putem drag-and-drop funkcionalnosti.



Slika 9: Prikaz prozora projekta

3.2.2.4. Prikaz scene i prikaz igre

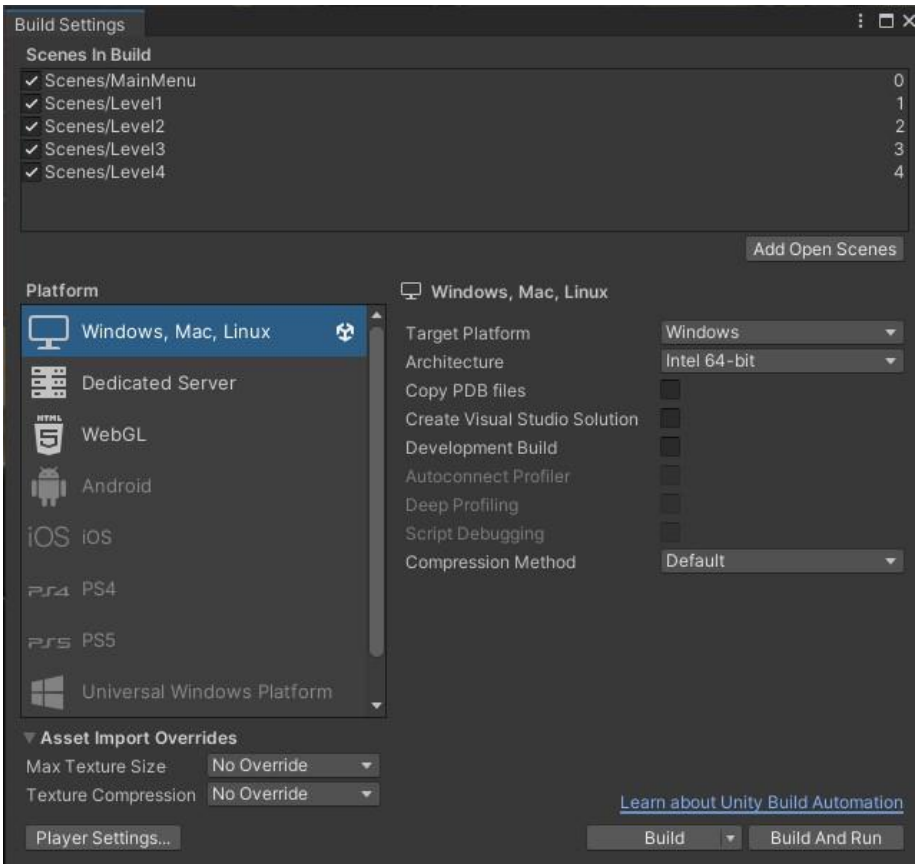
Prikaz scene(Scene View) je prostor u kojem se kreira i uređuje scena igre. Ovdje postavljamo objekte, manipuliramo elementima igre. Prikaz igre(Game View) je pregled kako će igra izgledati prilikom izvođenja što nam omogućuje da testiramo scenu.



Slika 10: Prikaz scene i prikaz igre

3.2.2.5. Postavke izgradnje

Postavke izgradnje (Build Settings) panel omogućuje konfiguraciju izvoza igre za različite platforme, poput PC-a, mobilnih uređaja, konzola i weba. Ovdje odabiremo platformu na koju želimo izvoziti igru, možemo podesiti kvalitetu i druge postavke. Pritiskom na gumb build pokrećemo proces izvoza igre.



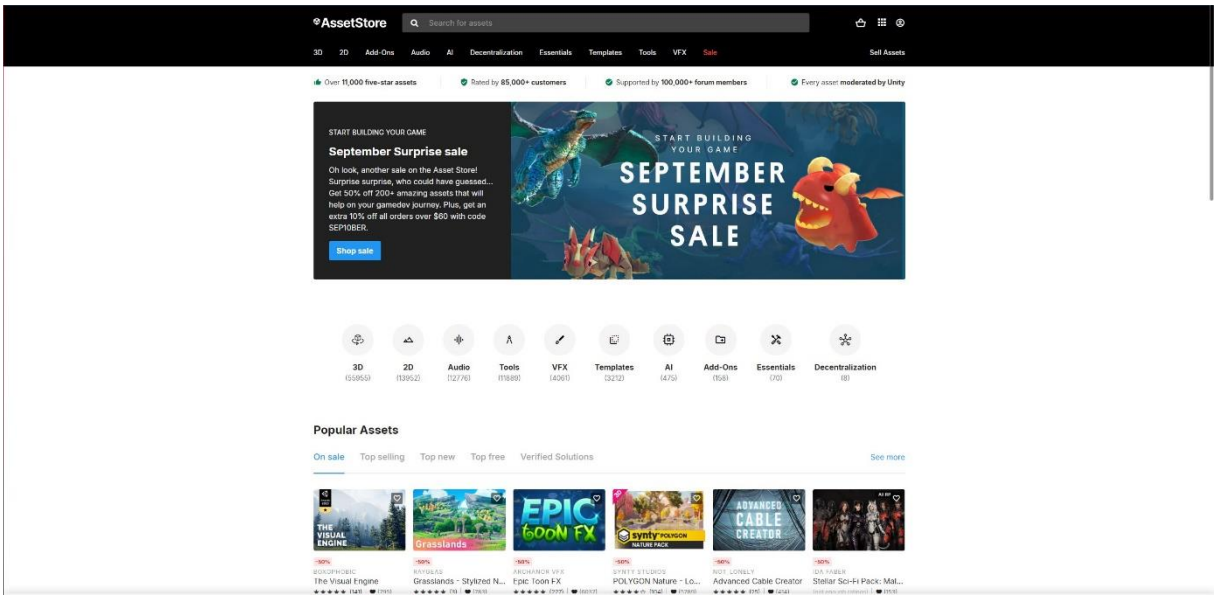
Slika 11: Prikaz prozora za postavke izgradnje

3.3. Razvoj 3D igre proklizavanja

U ovom poglavlju detaljno ću objasniti cijeli proces razvoja 3D igre proklizavanja (drift). Proći ćemo kroz sve ključne faze, od odabira i importiranja potrebnih asseta kao što su 3D modeli automobila, efekti dima i low-poly asseti, do tehničke implementacije mehanika unutar Unity okruženja. Također, opisat ću kako sam koristio Unity Asset Store za preuzimanje resursa, te metodu importiranja asseta preko Package Managera. Na kraju, prikazat ću implementaciju igre, što uključuje sve ključne elemente potrebne za stvaranje funkcionalne drift igre.

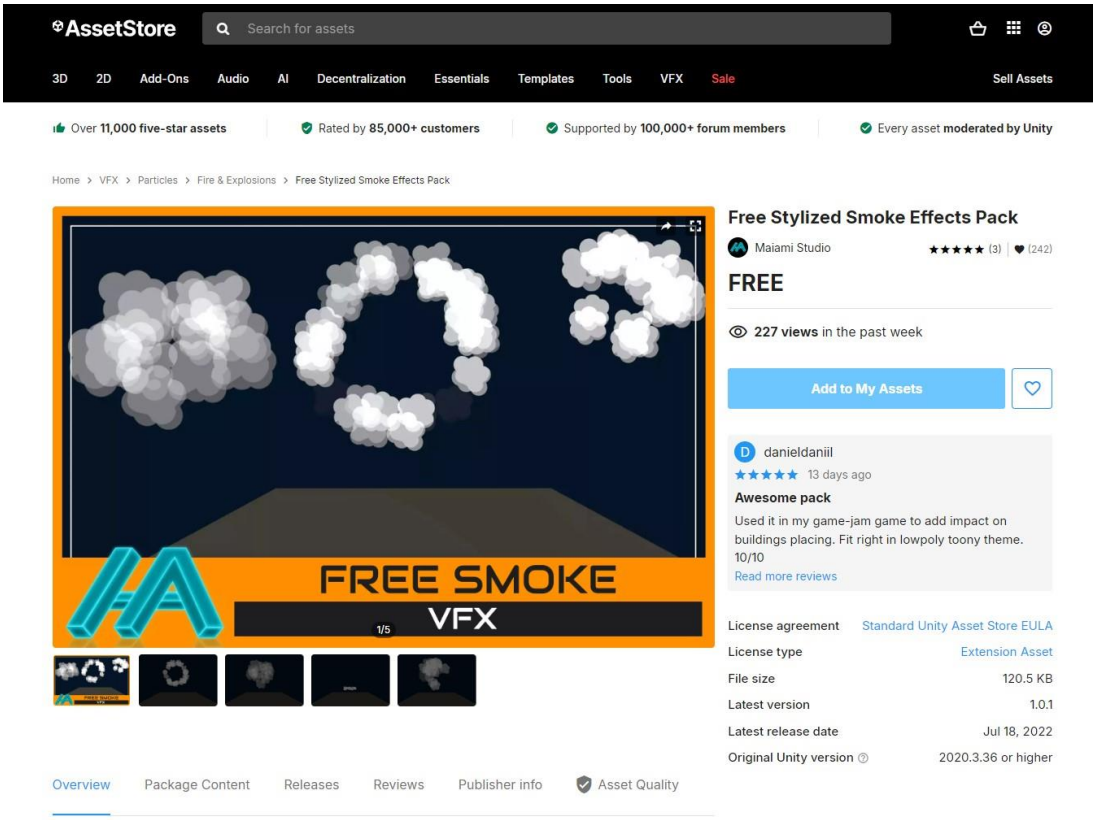
3.3.1. Unity Asset Store

Prilikom razvoja, Unity Asset Store mi je omogućio brz i jednostavan pristup velikom broju gotovih materijala. Ovdje sam preuzeo ključne elemente poput low-poly modela automobila, cesta,te efekte poput dima koji sam koristio za vizualne efekte automobila.



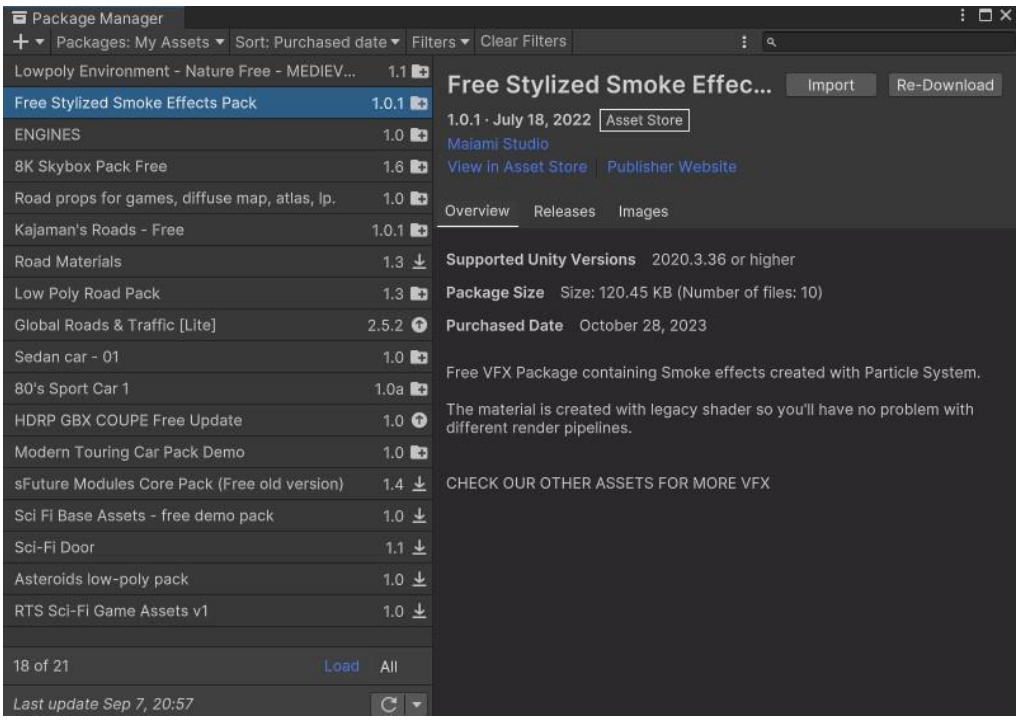
Slika 12: Unity Asset Store

Kada na stranici pronađemo asset koji nam se sviđa, možemo ga lako dodati pod „My Assets“ pritiskom na gumb „Add to my Assets“.



Slika 13: Dodavanje materijala preko Unity Asset Store-a

Nakon što smo pronašli materijale, možemo ih uvesti u projekt koristeći Package Manager. Klikom na gumb Import dodajemo resurse u svoj projekt.



Slika 14: Package Manager

3.3.2. Upravljanje automobilom

Sljedeća skripta se koristi za kontrolu automobila unutar igre proklizavanja. U njoj su implementirane sve ključne funkcionalnosti koje upravljaju kretanjem automobila, simulacijom driftanja, prikupljanjem drift bodova i prikazom efekata poput dima i tragova guma.

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

public enum WheelPosition
{
    Front,
    Rear
}
```

Na početku su prikazane biblioteke koje se koriste. To su biblioteke UnityEngine, System, System.Collections.Generic i TMPro.

Ova enumeracija (enum) koristi se za definiranje pozicije kotača na automobilu. Prednji i stražnji kotači imaju različite funkcionalnosti - prednji kotači služe za skretanje, dok stražnji za pogon i efekat proklizavanja.

```
[Serializable]
public struct Wheel
{
    public GameObject model;
    public WheelCollider collider;
    public GameObject effectObject;
    public ParticleSystem smoke;
    public WheelPosition position;
}
```

Wheel je struktura koja grupira sve komponente jednog kotača:

- model: 3D model kotača koji je vidljiv u igri.
- collider: Komponenta WheelCollider, odgovorna za fiziku i interakciju kotača s podlogom.
- effectObject: Objekt koji sadrži vizualne efekte, u ovom slučaju za tragove guma.
- smoke: ParticleSystem koji generira dim prilikom proklizavanja kotača.
- position: Definira da li je kotač prednji ili stražnji, što utječe na ponašanje kotača u skripti.

[Serializable] omogućava da direktno postavljamo vrijednosti poput modela kotača, collidera i efekta u sučelju bez potrebe za kodiranjem.

```
[Header("Car Settings")]
public float Speed = 55000.0f;
public float brakeForce = 12000000.0f;
public float Steering = 0.8f;
public float MaxSteering = 30.0f;
```

U ovom djelu postavljamo varijable za postavke auta.

- Speed: Maksimalna snaga koja se koristi za ubrzanje automobila. Ova snaga se primjenjuje na motorTorque svakog kotača.
- brakeForce: Maksimalna snaga kočenja. Kad je aktivirano kočenje, ova snaga se primjenjuje na kotače kako bi usporila ili zaustavila automobil.
- Steering: Omjer upravljanja koji definira osjetljivost upravljanja, omogućavajući manje ili veće zakretanje kotača ovisno o unosu.
- MaxSteering: Maksimalni kut zakretanja prednjih kotača, koji ograničava koliko automobil može skrenuti.

```
[Header("Wheel Settings")]
public List<Wheel> wheels = new List<Wheel>();
```

Lista svih kotača na automobilu. U Unity editoru, dodajemo svaki kotač (njegov model, collider itd.) u ovu listu. (definirano ranije u kodu kao struktura).

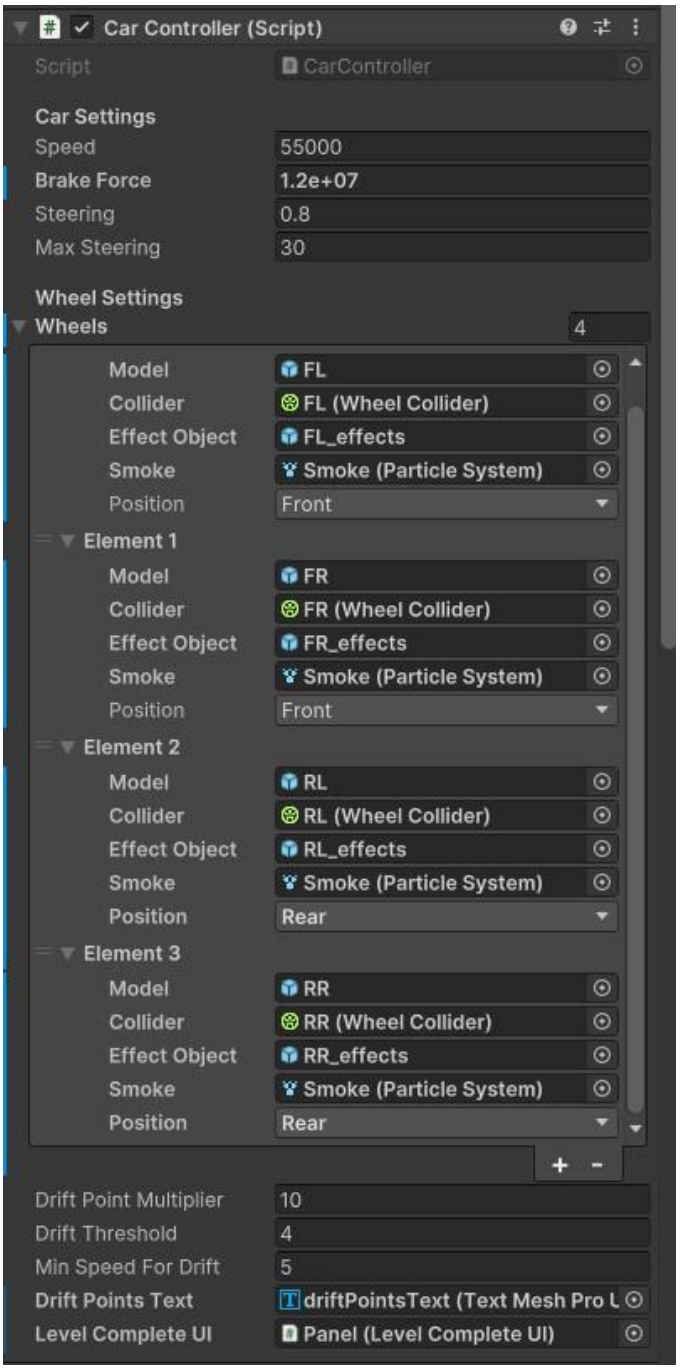
```
private float accelerationInput;
private float steeringInput;
```

Ove privatne varijable pohranjuju trenutne vrijednosti unosa od igrača za ubrzanje i upravljanje, koje dolaze od tipkovnice.

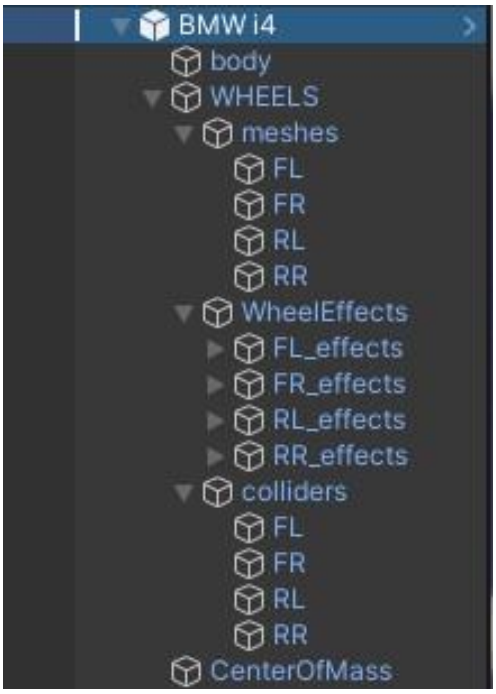
```
[Header("Drift Points")]
private float driftPoints = 0.0f;
private float driftTimer = 0.0f;
public float driftPointMultiplier = 10f;
public float driftThreshold = 4.0f;
public float minSpeedForDrift = 5.0f;
public TMP_Text driftPointsText;
public LevelCompleteUI levelCompleteUI;
```

U ovom djelu postavljamo varijable za drift bodove.

- driftPoints: Broj bodova koje igrač osvaja tijekom driftanja. Ovi bodovi se povećavaju ovisno o vremenu provedenom u driftanju i brzini automobila.
- driftTimer: Mjeri vrijeme trajanja driftanja koje se koristi za izračun drift bodova.
- driftPointMultiplier: Faktor koji pojačava broj bodova za svaku sekundu driftanja.
- driftThreshold: Prag za bočno klizanje. Ako bočna brzina prijeđe ovu vrijednost, automobil se smatra u driftu.
- minSpeedForDrift: Minimalna brzina koju automobil mora postići da bi drift bio moguć.
- driftPointsText: Referenca na UI tekst koji prikazuje trenutne drift bodove na zaslonu.



Slika 15: Inspektor skripe za upravljanje automobilom



Slika 16: Hierarhija auta

```
void Start()
{
    rb = GetComponent<Rigidbody>();
}
void Update()
{
    HandleInputs();
    UpdateWheelAnimations();
    HandleWheelEffects();
}
void LateUpdate()
{
    ApplyMovement();
    ApplyBraking();
    ApplySteering();
}
```

Start metoda inicijalizira Rigidbody komponentu koja se koristi za upravljanje fizikom automobila.

Update metoda se poziva jednom po frame-u (okviru) i koristi se za obradu svih operacija vezanih uz korisnički unos i ažuriranje vizualnih elemenata. Ovdje se vrše zadaci koji moraju biti u skladu s brzinom osvježavanja ekrana. U ovom slučaju metode:

- HandleInputs();
- UpdateWheelAnimations();
- HandleWheelEffects();

LateUpdate se koristi kako bi se osiguralo da se fizičke promjene (kretanje, kočenje, skretanje) primijene tek nakon što su svi korisnički unosi i vizualni efekti ažurirani u Update metodi. Time se postiže veća preciznost u odnosu između unosa korisnika i fizičkog ponašanja vozila, kao i glatkiji tijek igre.

```
private void HandleInputs()
{
accelerationInput = Input.GetAxis("Vertical");
steeringInput = Input.GetAxis("Horizontal");
}
```

Ova funkcija koristi Unity-ov Input sustav za dohvaćanje vrijednosti osovina Vertical (W/S ili tipke strelica gore/dolje) i Horizontal (A/D ili tipke strelica lijevo/desno)

```
private void ApplyMovement()
{
 foreach (var wheel in wheels)
  {
wheel.collider.motorTorque = accelerationInput * Speed * Time.deltaTime;
   }
}
private void ApplyBraking()
{
    bool isBraking = Input.GetKey(KeyCode.Space) ||
Mathf.Approximately(accelerationInput, 0);

    foreach (var wheel in wheels)
    {
        wheel.collider.brakeTorque = isBraking ? brakeForce *
Time.deltaTime : 0;
    }
}
private void ApplySteering()
{
    foreach (var wheel in wheels)
    {
        if (wheel.position == WheelPosition.Front)
        {
            float targetSteerAngle = steeringInput * Steering *
MaxSteering;
            wheel.collider.steerAngle =
Mathf.Lerp(wheel.collider.steerAngle, targetSteerAngle, 0.6f);
        }
    }
}
```

ApplyMovement() : Prolazi kroz sve kotače i primjenjuje moment (motorTorque) na svaki kotač na temelju unosa ubrzanja.

ApplyBraking() : Ako je kočenje aktivirano (tipka Space ili nula ubrzanja), primjenjuje silu kočenja na sve kotače.

ApplySteering() : Ažurira kut prednjih kotača koristeći unose za skretanje i interpolira između trenutnog i ciljanog kuta.

```
private void HandleWheelEffects()
{
    foreach (var wheel in wheels)
    {
        if (wheel.position != WheelPosition.Rear ||
!wheel.collider.isGrounded) continue;

        float forwardSpeed = Vector3.Dot(rb.velocity, transform.forward);
        float sidewaysSpeed = Vector3.Dot(rb.velocity, transform.right);

        bool isDrifting = forwardSpeed >= minSpeedForDrift &&
Mathf.Abs(sidewaysSpeed) >= driftThreshold;

        if (isDrifting)
        {
            driftTimer += Time.deltaTime;
            float pointsThisFrame = Mathf.Abs(sidewaysSpeed) *
driftPointMultiplier * Time.deltaTime;
            driftPoints += pointsThisFrame;

            if (driftPointsText != null)
            {
                driftPointsText.text = "Drift Points: " +
Mathf.RoundToInt(driftPoints);
            }
        }
        else
        {
            driftTimer = 0.0f;
        }

        bool shouldEmitTrail = forwardSpeed >= 5.0f &&
Mathf.Abs(sidewaysSpeed) >= 4.0f;
        bool shouldEmitSmoke = forwardSpeed >= 5.0f &&
Mathf.Abs(sidewaysSpeed) >= 10.0f;

        var trailRenderer =
wheel.effectObject.GetComponentInChildren<TrailRenderer>();
        if (trailRenderer != null)
        {
            trailRenderer.emitting = shouldEmitTrail;
        }


        if (shouldEmitSmoke)
        {
            wheel.smoke.Emit(1);
        }

    }
}
```

Metoda HandleWheelEffects odgovorna je za prikazivanje vizualnih efekata poput tragova kotača i dima tijekom driftanja ili proklizavanja automobila. Također izračunava drift bodove. U nastavku ću objasniti djelove ove metode.

```
if (wheel.position != WheelPosition.Rear || !wheel.collider.isGrounded)
continue;
```

Metoda prvo provjerava je li trenutni kotač stražnji i nalazi li se na tlu. Efekti driftanja primjenjuju se samo na stražnje kotače koji dodiruju tlo.

```
float forwardSpeed = Vector3.Dot(rb.velocity, transform.forward);
float sidewaysSpeed = Vector3.Dot(rb.velocity, transform.right);
```

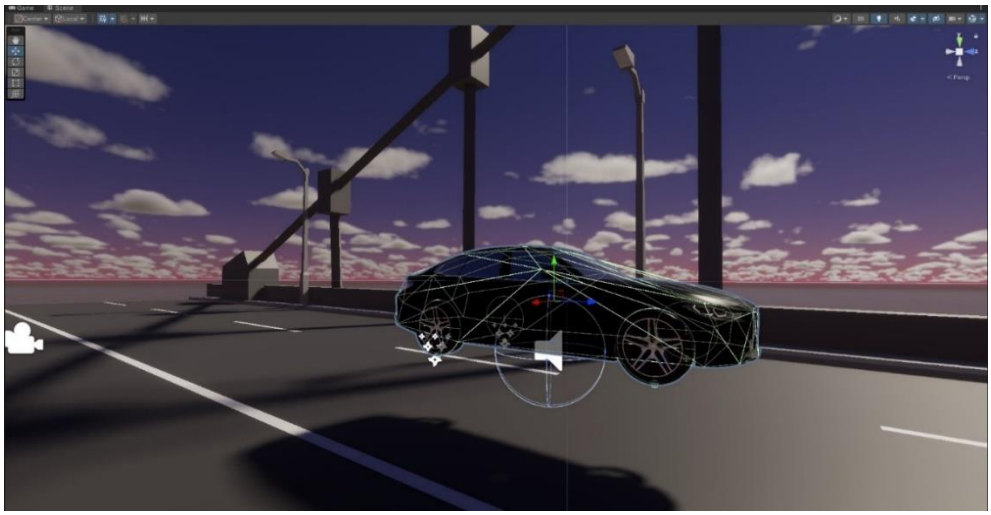
Izračunavaju se dvije ključne brzine, forwardSpeed(brzina prema naprijed) i sidewaysSpeed, koja je ključna za detekciju proklizavanja i driftanja.

```
bool isDrifting = forwardSpeed >= minSpeedForDrift &&
Mathf.Abs(sidewaysSpeed) >= driftThreshold;
if (isDrifting)
{
    driftTimer += Time.deltaTime;
    float pointsThisFrame = Mathf.Abs(sidewaysSpeed) * driftPointMultiplier
* Time.deltaTime;
    driftPoints += pointsThisFrame;
}
```

Ovdje se provjerava se je li automobil u stanju proklizavanja, na temelju minimalne brzine prema naprijed i bočne brzine. Ako automobil proklizuje, računa se broj drift bodova na temelju bočne brzine i multiplicira s driftPointMultiplier. Ovi bodovi prikazuju se na ekranu pomoću driftPointsText koji smo priložili u inspectoru.

```
bool shouldEmitTrail = forwardSpeed >= 5.0f && Mathf.Abs(sidewaysSpeed) >=
4.0f;
if (trailRenderer != null)
{
    trailRenderer.emitting = shouldEmitTrail;
}

bool shouldEmitSmoke = forwardSpeed >= 5.0f && Mathf.Abs(sidewaysSpeed) >=
10.0f;
if (shouldEmitSmoke)
{
    wheel.smoke.Emit(1);
}
```

Ako su zadovoljeni uvjeti, aktiviraju se tragovi kotača, a efekt dima se aktivira ako je bočna brzina veća od 10.



Slika 17: Model auta

Slika 18: Efekt guma

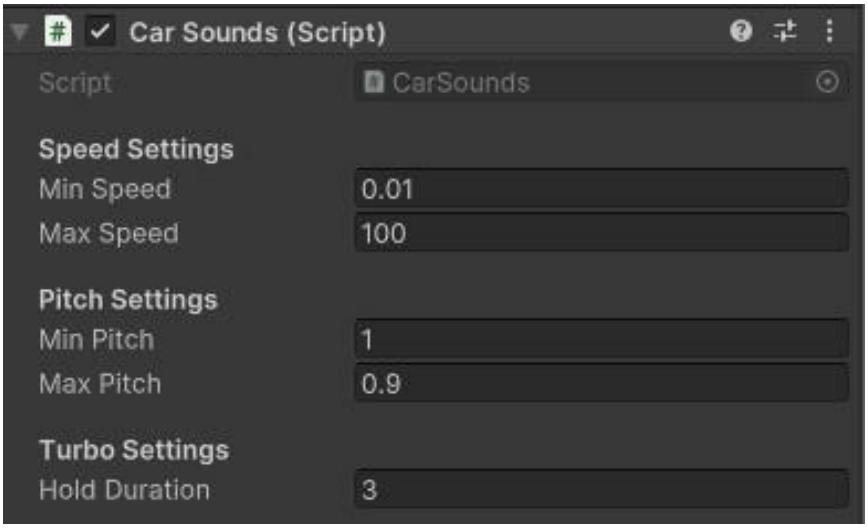
Slika 19: Efekt dima

3.3.3.Zvuk automobila

Zvukovi automobila se sastoje od 3 djela. Zvuk automobila kada je na mjestu(idle), zvuk automobila kada se kreće, i turbo zvuk.

```
public class CarSounds : MonoBehaviour
{
    [Header("Speed Settings")]
    public float minSpeed = 0f;
    public float maxSpeed = 100f;

    [Header("Pitch Settings")]
    public float minPitch = 0.5f;
    public float maxPitch = 2.0f;

    [Header("Turbo Settings")]
    public float holdDuration = 1f;

    private Rigidbody carRigidbody;
    private AudioSource[] carAudioSources;
    private bool isWKeyPressed = false;
    private float keyPressedTime = 0f;
```

Speed Settings definiraju minimalnu i maksimalnu brzinu automobila (minSpeed i maxSpeed). Ove vrijednosti će odrediti kako će se zvuk motora mijenjati ovisno o brzini vozila. Pitch Settings koriste minPitch i maxPitch kako bi prilagodili ton (pitch) zvuka motora, oponašajući promjenu zvuka pri ubrzavanju ili usporavanju. Turbo Settings uključuju holdDuration, što označava koliko dugo igrač mora držati tipku W za aktivaciju turbo zvuka. Varijabla carRigidbody dohvaća Rigidbody komponentu koja omogućava skripti pristup brzini automobile, carAudioSources sadrži sve AudioSource komponente koje su pridružene automobile, a isWKeyPressed i keyPressedTime koriste se za praćenje je li tipka za ubrzanje (W) pritisnuta i koliko dugo je pritisnuta.



Slika 20: Car sounds inspector

```
void Start()
{
    carRigidbody = GetComponent<Rigidbody>();
    carAudioSources = GetComponents<AudioSource>();
}
void Update()
{
    UpdateEngineSound();
    CheckTurboSound();
}
```

U Start metodi dohvaćaju se potrebne komponente iz automobile, carRigidbody dohvaća Rigidbody komponentu kako bi mogao pratiti brzinu automobile,a carAudioSources dohvaća sve AudioSource komponente na automobilu, od kojih svaka reproducira specifičan zvuk (npr. idle zvuk, aktivni motor, turbo zvuk).

U Update metodi su dvije ključne metode koje se stalno izvršavaju. UpdateEngineSound prilagođava zvuk motora ovisno o trenutnoj brzini automobile,a CheckTurboSound provjerava pritisak tipke W i uvjete za aktivaciju turbo zvuka.

```
private void UpdateEngineSound()
{
    float currentSpeed = carRigidbody.velocity.magnitude;
    float pitchFromSpeed = currentSpeed / 60f;

    if (currentSpeed < minSpeed)
    {
        SetEngineIdle();
    }
    else
    {
        SetEngineActive(currentSpeed, pitchFromSpeed);
    }
}
```

Metoda UpdateEngineSound koristi trenutnu brzinu automobila (carRigidbody.velocity.magnitude) kako bi odredila odgovarajući pitch zvuka motora. Ako je brzina manja od minSpeed, zvuk motora prelazi u idle stanje, što znači da se ne čuje aktivno ubrzanje. Ako je brzina iznad minimalne, zvuk se mijenja ovisno o brzini.

```
private void SetEngineIdle()
{
    carAudioSources[0].volume = 1f;
    carAudioSources[1].volume = 0f;
    carAudioSources[1].pitch = minPitch;
}
```

Ako je brzina automobila ispod minimalne vrijednosti (minSpeed), motor prelazi u idle stanje kako je spomenuto prije. To radi ova metoda gdje prvi AudioSource (index 0) reproducira idle zvuk motora s punim volumenom, dok drugi (index 1) prestaje s reprodukcijom.

```
private void SetEngineActive(float currentSpeed, float pitchFromSpeed)
{
    carAudioSources[0].volume = 0f;
    carAudioSources[1].volume = 1f;

    if (currentSpeed > minSpeed && currentSpeed < maxSpeed)
    {
        carAudioSources[1].pitch = minPitch + pitchFromSpeed;
    }
    else if (currentSpeed >= maxSpeed)
    {
        carAudioSources[1].pitch = maxPitch;
    }
}
```

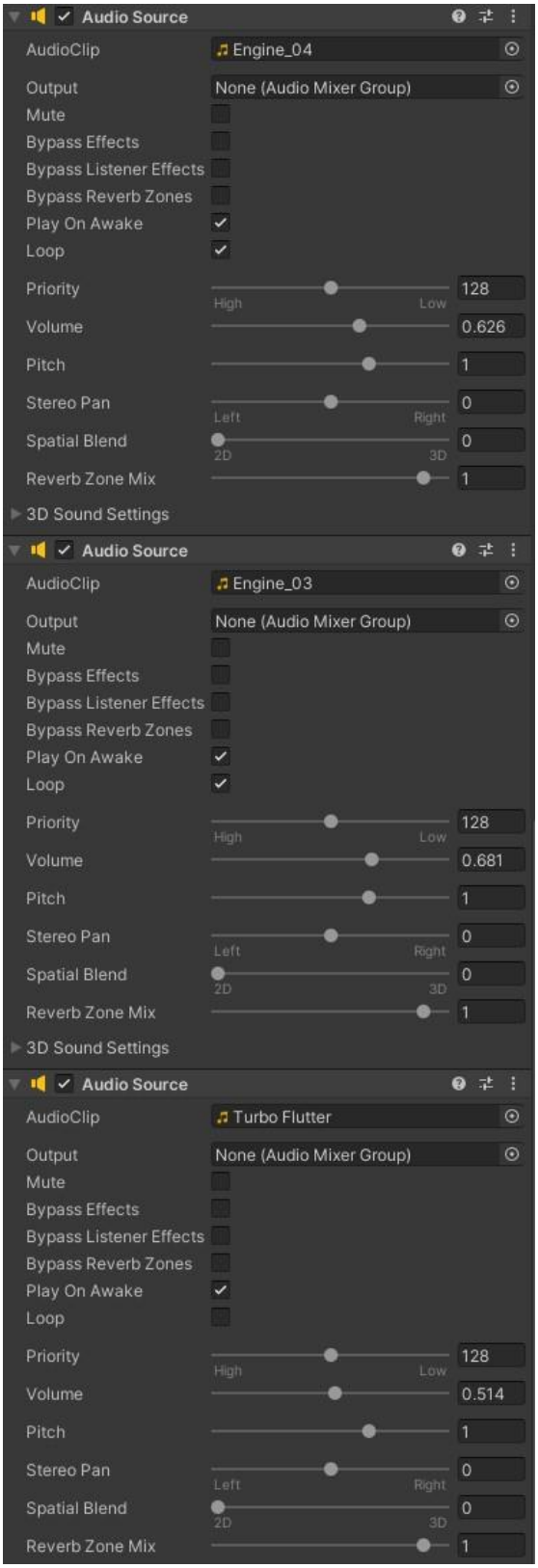
Ako je brzina automobila unutar raspona između minSpeed i maxSpeed, zvuk motora postaje aktivan. Zvuk se prilagođava (pitch se povećava) ovisno o brzini vozila, a ako automobil dostigne ili premaši maksimalnu brzinu, pitch dostiže svoju maksimalnu vrijednost (maxPitch).

```
private void CheckTurboSound()
{
    if (Input.GetKeyDown(KeyCode.W))
    {
        isWKeyPressed = true;
        keyPressedTime = Time.time;
    }

    if (Input.GetKeyUp(KeyCode.W))
    {
        TryPlayTurboSound();
    }
}
```

Metoda CheckTurboSound prati pritisak i puštanje tipke W. Kada igrač pritisne W, bilježi se trenutak kada je tipka pritisnuta, a nakon što je puštena, poziva se TryPlayTurboSound.

```
private void TryPlayTurboSound()
{
    if (isWKeyPressed && (Time.time - keyPressedTime) >= holdDuration)
    {
        carAudioSources[2].Play();
    }

    isWKeyPressed = false;
}
```

U ovoj metodi, ako je tipka W bila pritisnuta dovoljno dugo (više od holdDuration), reproducira se turbo zvuk (treći AudioSource).



Slika 21: Zvukovi automobila

3.3.4.Kamera

Skripta CameraFollow omogućava glatko praćenje kamere koja se prilagođava poziciji i rotaciji ciljanog objekta, obično automobila u igri. Korištenjem glatkih prijelaza za pomicanje i rotaciju kamere, postiže se fluidno praćenje objekta.

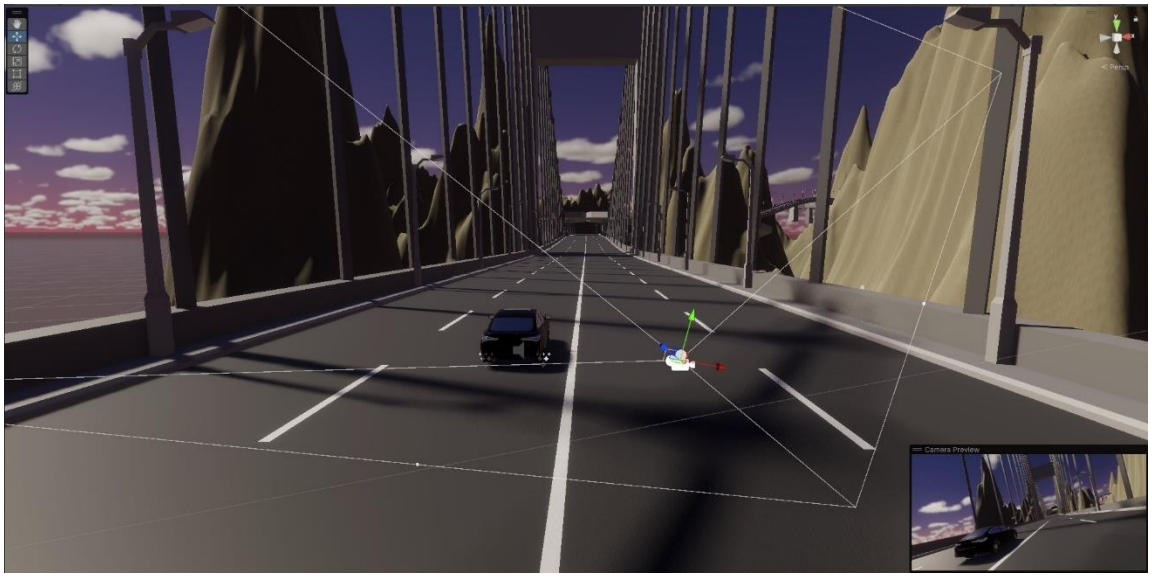
```
[Header("Follow Settings")]
public float movementSmoothness = 0.125f;
public float rotationSmoothness = 0.1f;

[Header("Target Settings")]
public Transform target;
public Vector3 positionOffset;
public Vector3 rotationOffset;
```

U follow settings movementSmoothness i rotationSmoothness predstavljaju brzinu prilagodbe kamere prilikom praćenja objekta, čineći prijelaze glatkim. U target settings target predstavlja objekt koji kamera prati, dok su positionOffset i rotationOffset odgovorni za definiranje udaljenosti kamere od objekta i njenu orijentaciju u odnosu na njega.

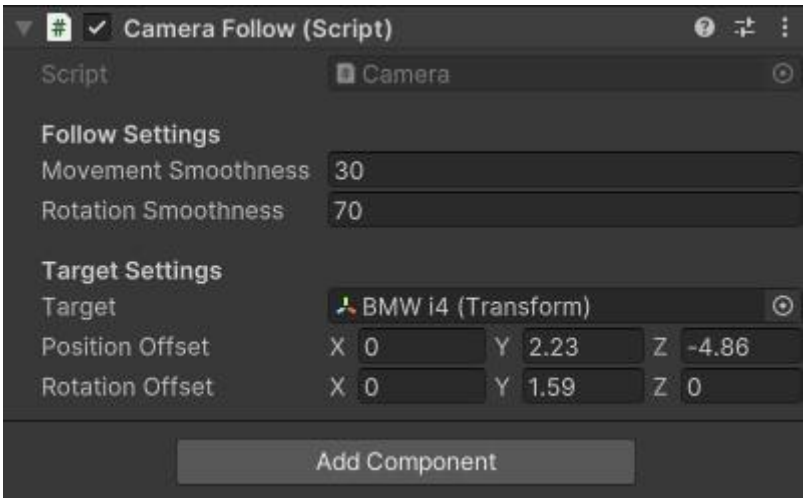
```
void FixedUpdate()
    {
        FollowTarget();
    }
    private void FollowTarget()
    {
        SmoothMovement();
        SmoothRotation();
    }
    private void SmoothMovement()
    {
        Vector3 targetPosition = target.TransformPoint(positionOffset);
        transform.position = Vector3.Lerp(transform.position,
targetPosition, movementSmoothness * Time.deltaTime);
    }

    private void SmoothRotation()
    {
        Vector3 direction = target.position - transform.position;
        Quaternion targetRotation = Quaternion.LookRotation(direction +
rotationOffset, Vector3.up);
        transform.rotation = Quaternion.Lerp(transform.rotation,
targetRotation, rotationSmoothness * Time.deltaTime);
    }
}
```

Skripta koristi metodu FixedUpdate za kontinuirano praćenje ciljanog objekta. FixedUpdate se poziva u određenim intervalima i osigurava konzistentnost u kretanju kamere. Unutar metode FollowTarget, skripta poziva dvije pomoćne metode, SmoothMovement i SmoothRotation. SmoothMovement koristi Lerp funkciju za interpolaciju između trenutne pozicije kamere i ciljne pozicije (koja se određuje na osnovi pozicije ciljanog objekta plus offset). Ova metoda omogućuje glatko i postupno kretanje kamere prema autu. SmoothRotation koristi Quaternion.LookRotation kako bi kamera bila usmjerena prema

ciljanom objektu, prilagođavajući rotaciju prema njemu. I ovdje se koristi Lerp za glatko prilagođavanje rotacije.



Slika 22: Kamera u editor



Slika 23: Camera Follow inspector

3.3.5.Mjerenje vremena i završetak razine

Ovaj kod implementira mjerač vremena za razinu igre u Unityju, koji prati proteklo vrijeme dok igrač upravlja vozilom. Ovaj sustav tajminga je povezan s prikazom vremena na ekranu, kao i sa sustavom završetka razine koji prikazuje igraču konačno vrijeme i broj drift bodova.

```
using UnityEngine;
using TMPro;
```

```
public class LevelTimer : MonoBehaviour
{
    private float startTime;
    private float finalTime;
    private bool isTiming;

    public TextMeshProUGUI timerText;
    public LevelCompleteUI levelCompleteUI;
    private CarController carController;
    void Start()
    {
        StartTimer();
        carController = FindObjectOfType<CarController>();
    }
    void Update()
    {
        if (isTiming)
        {
            float timeElapsed = Time.time - startTime;
            UpdateTimerDisplay(timeElapsed);
        }
    }
    public void StartTimer()
    {
        startTime = Time.time;
        isTiming = true;
    }
    public void StopTimer()
    {
        isTiming = false;

        finalTime = Time.time - startTime;

        UpdateTimerDisplay(finalTime);

        float driftPoints = carController != null ?
carController.GetDriftPoints() : 0;

        levelCompleteUI.Show(finalTime, driftPoints);
    }
    private void UpdateTimerDisplay(float timeElapsed)
    {
        if (timerText != null)
        {
            timerText.text = FormatTime(timeElapsed);
        }
    }

    private string FormatTime(float time)
    {
        int minutes = Mathf.FloorToInt(time / 60F);
        int seconds = Mathf.FloorToInt(time - minutes * 60);
        float milliseconds = (time - Mathf.Floor(time)) * 1000;
        return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds,
milliseconds);
    }
}
```

LevelTimer skripta koristi TextMeshProUGUI komponentu za prikaz vremena u obliku teksta na ekranu, dok LevelCompleteUI omogućava prikaz informacija nakon završetka razine. Skripta također komunicira s CarController komponentom za dohvaćanje drift bodova.

Kada se igra započne, Start() metoda automatski pokreće mjerenje vremena pomoću StartTimer() metode. Svaki frame, kroz Update(), provjerava se je li tajmer aktivan, a zatim ažurira prikaz vremena pomoću UpdateTimerDisplay(), koji formatira i prikazuje proteklo vrijeme. Kad igrač završi razinu, metoda StopTimer() zaustavlja mjerenje vremena, izračunava proteklo vrijeme te ažurira prikaz. Osim toga, skripta poziva levelCompleteUI.Show() kako bi prikazala konačno vrijeme i bodove koje je igrač sakupio tijekom igre. FormatTime() metoda formatira vrijeme u oblik "minute:sekunde", što omogućava igraču da vidi vrijeme u preciznom formatu.

```
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;
using UnityEngine.UI;
public class LevelCompleteUI : MonoBehaviour
{
    public TextMeshProUGUI timeText;
    public Button nextLevelButton;
    public TextMeshProUGUI driftPointsText;
    void Start()
    {
        gameObject.SetActive(false);

        nextLevelButton.onClick.AddListener(LoadNextLevel);
    }
    public void Show(float timeToComplete, float driftPoints)
    {
        gameObject.SetActive(true);
        timeText.text = "Time: " + FormatTime(timeToComplete);

        driftPointsText.text = "Drift Points: " +
Mathf.RoundToInt(driftPoints);
    }
    private string FormatTime(float time)
    {
        int minutes = Mathf.FloorToInt(time / 60F);
        int seconds = Mathf.FloorToInt(time - minutes * 60);
        float milliseconds = (time - Mathf.Floor(time)) * 1000;
        return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds,
milliseconds);
    }
    void LoadNextLevel()
    {
        Int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
        if (currentSceneIndex + 1 < SceneManager.sceneCountInBuildSettings)
        {
            SceneManager.LoadScene(currentSceneIndex + 1);
        }
        else
        {
            SceneManager.LoadScene(0);
        }
    }
}
```

Ovaj kod implementira korisničko sučelje (UI) za prikaz završetka razine u igri Unity, pod nazivom LevelCompleteUI. Kada igrač završi razinu, ovaj UI se aktivira i prikazuje vrijeme koje je bilo potrebno za dovršetak razine, kao i drift bodove koje je igrač osvojio.

Na početku igre, Start() metoda deaktivira UI element pomoću gameObject.SetActive(false), jer je UI vidljiv samo kada igrač završi razinu. Također, dodaje funkcionalnost gumba nextLevelButton, koji će učitati sljedeću razinu kada ga igrač pritisne. Kada se razina završi, metoda Show() se poziva, aktivira UI element, te ažurira tekstualna polja za prikaz vremena i drift bodova. Vrijeme se prikazuje u formatu "minute:sekunde" pomoću metode FormatTime(), dok drift bodovi prikazuju zaokruženi rezultat pomoću Mathf.RoundToInt(driftPoints). Gumb nextLevelButton koristi LoadNextLevel() metodu za učitavanje sljedeće scene u igri, koristeći Unityjev SceneManager. Ako je trenutna scena posljednja u igri, gumb vraća igrača na početnu scenu (index 0).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class FinishLine : MonoBehaviour
{
    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            FindObjectOfType<LevelTimer>().StopTimer();
        }
    }
}
```

Ova skripta predstavlja jednostavan FinishLine sustav u igri. Svrha joj je detektirati kada igrač, odnosno objekt označen tagom "Player", prođe kroz cilj, a tada zaustaviti tajmer igre. Metoda OnTriggerEnter(Collider other) se automatski poziva kada neki objekt s Collider komponentom uđe u Trigger zonu ovog objekta. U ovom slučaju, ako objekt koji uđe ima tag "Player", poziva se metoda StopTimer() iz klase LevelTimer, koja zaustavlja tajmer i prikazuje završne informacije o igri. U editoru finish line je ploha koja ima isključeni mesh renderer ali još uvijek prima kolizije putem box collidera.

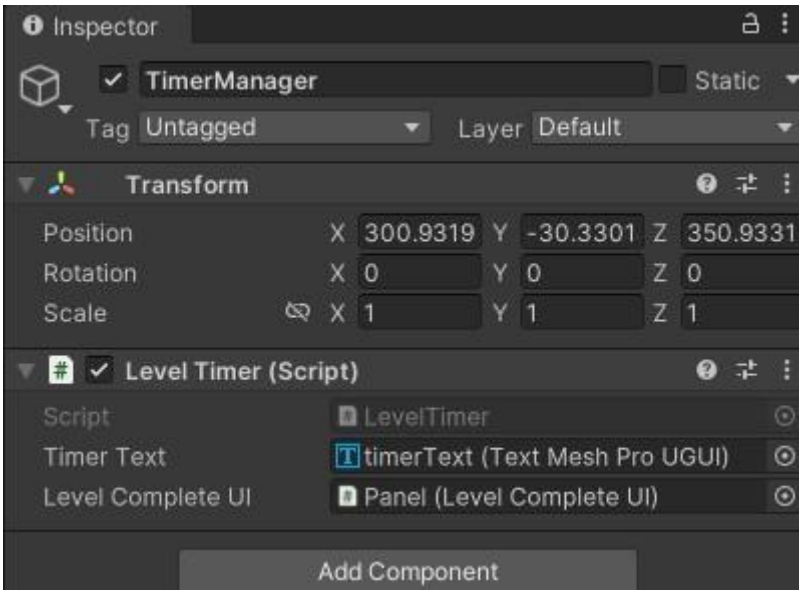
```
using UnityEngine;
using UnityEngine.SceneManagement;
public class LevelManager : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            RestartLevel();
        }
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            LoadMainMenu();
        }
    }
```

```
    void RestartLevel()
    {
        Scene currentScene = SceneManager.GetActiveScene();
        SceneManager.LoadScene(currentScene.name);
    }
    public void LoadNextLevel()
    {
        int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;

        if (currentSceneIndex + 1 < SceneManager.sceneCountInBuildSettings)
        {
            SceneManager.LoadScene(currentSceneIndex + 1);
        }
    }
    void LoadMainMenu()
    {
        SceneManager.LoadScene(0);
    }
}
```

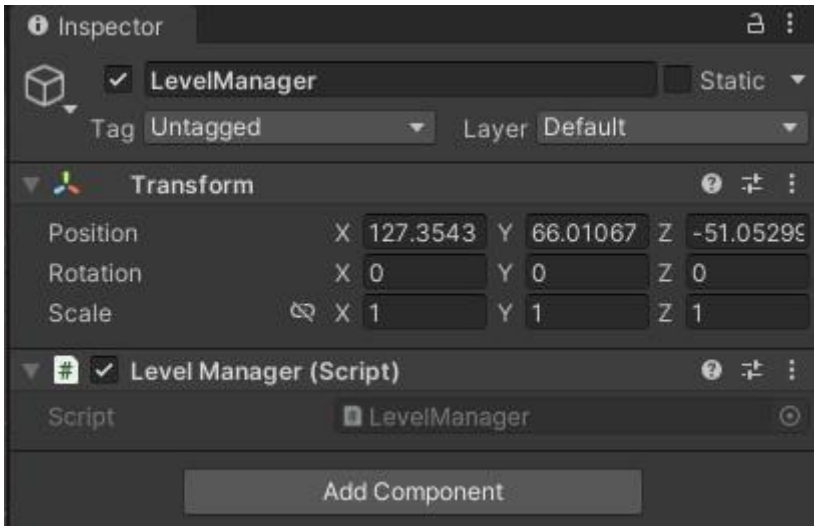
Ova skripta upravlja osnovnim funkcijama prelaska između nivoa u igri. U Update() metodi detektiraju se pritisci na tipke:

- Tipka R za ponovno pokretanje trenutne razine (pozivom metode RestartLevel(), koja učitava aktivnu scenu ponovno).
- Tipka Escape za povratak na glavni izbornik (metoda LoadMainMenu() učitava scenu na indeksu 0, koji obično predstavlja glavni izbornik).

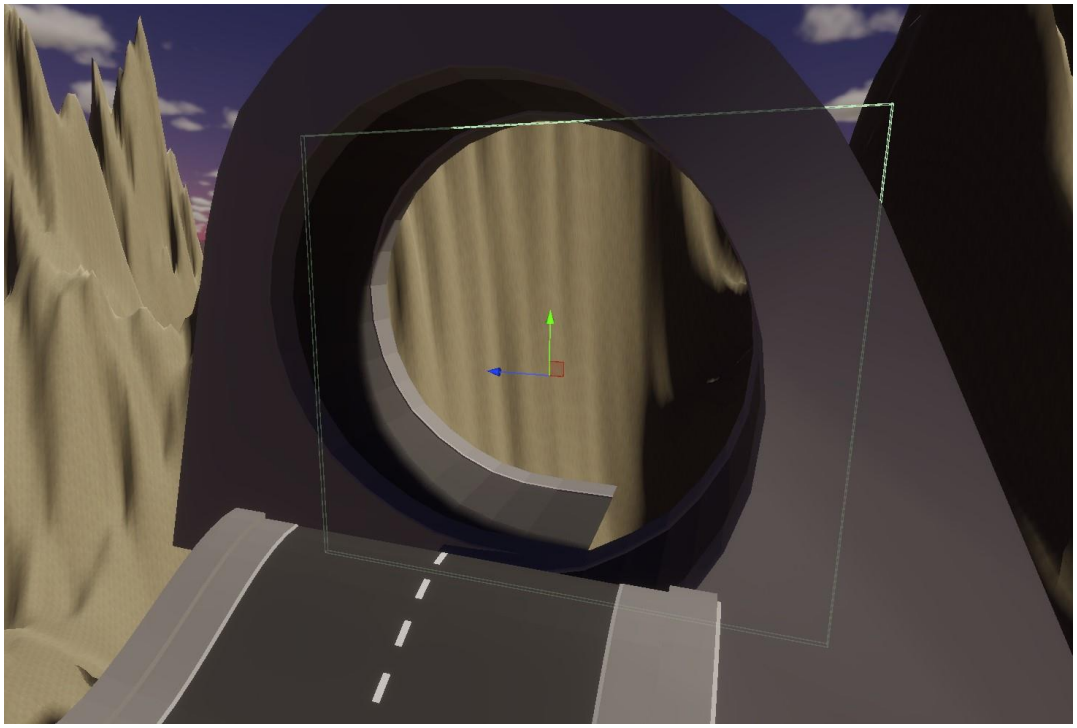
Također, skripta omogućuje prelazak na sljedeću razinu koristeći LoadNextLevel(), koja provjerava trenutni indeks scene i učitava sljedeću ako postoji.



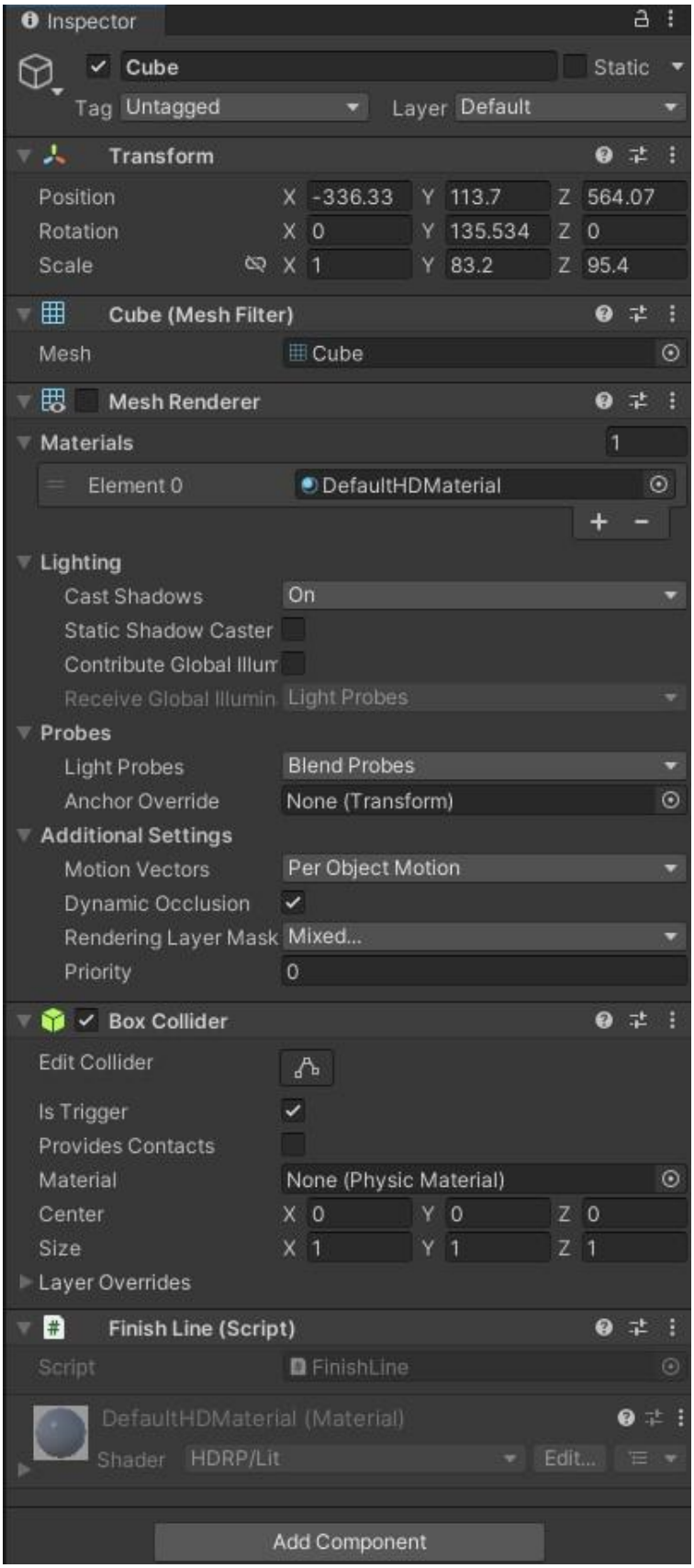
Slika 24: TimerManager Inspektor



Slika 25: LevelManager inspector



Slika 26: Finish line unutar editora



Slika 27: Finish line inspector

Slika 28: Kraj razine

3.3.6. Početni zaslon

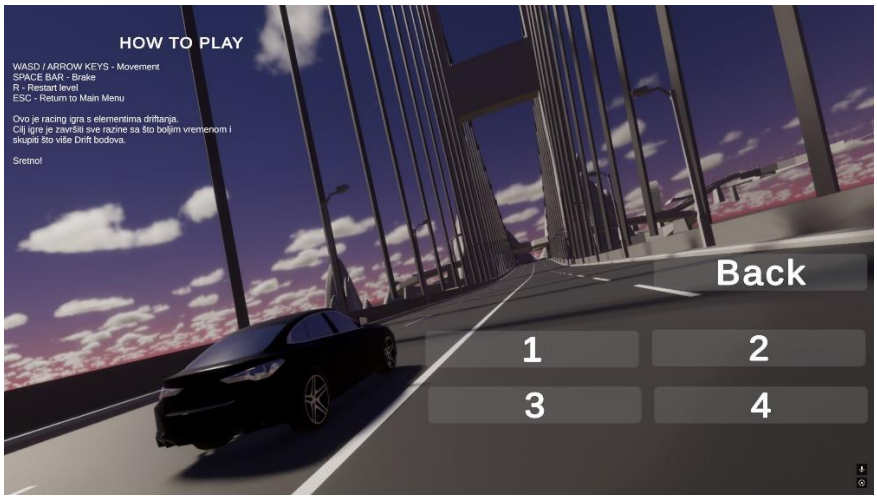
Sljedeća skripta upravlja opcijama u glavnom izborniku igre. Ova skripta omogućava korisnicima da odaberu različite razine za igranje ili izađu iz igre.

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


public class MainMenu : MonoBehaviour
{
    public void PlayLevel1()
    {
        SceneManager.LoadScene(1);
    }
    public void PlayLevel2()
    {
        SceneManager.LoadScene(2);
    }
    public void PlayLevel3()
    {
        SceneManager.LoadScene(3);
    }
    public void PlayLevel4()
    {
        SceneManager.LoadScene(4);
    }

    public void Quit()
    {
        Application.Quit();
```

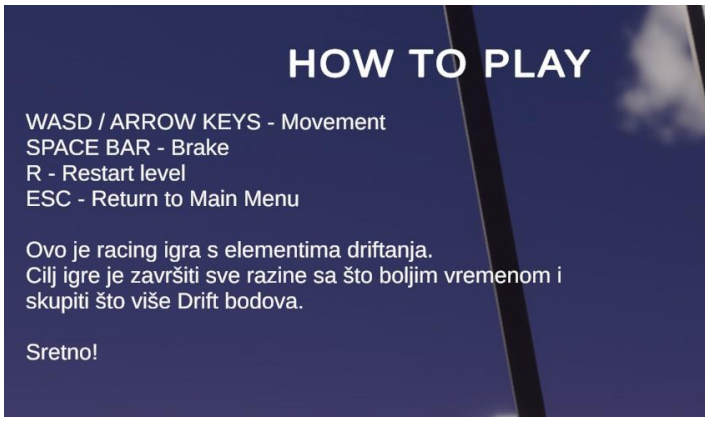
```
    }
}
```

SceneManager.LoadScene() se koristi za učitavanje različitih scena prema njihovim indeksima. U ovom slučaju, svakom metodom dodjeljuje se odgovarajući broj koji odgovara indeksu scene u Build Settingsu u Unityju. Application.Quit() se koristi za zatvaranje igre. Ova metoda je korisna za izlazak iz igre, ali se ne izvršava u editoru Unityja—samo u izgrađenoj igri.

Slika 29: Početni zaslon



Slika 30: Odabir razina



Slika 31: Objašnjenje igre

4. Zaključak

Ovaj završni rad istražuje proces razvoja 3D igre proklizavanja u Unity okruženju, s naglaskom na ključne komponente igre kao što su kontroler automobila, sustav kamere, zvučni efekti i upravljanje razinama. Na početku rada, provedena je analiza postojećih igara iz žanra kako bi se razumjeli njihovi osnovni mehanizmi i dizajnerske odluke, što je poslužilo kao polazište za razvoj našeg projekta.

Korišteni su različiti programski alati, poput Unity-a, Unity Hub-a, te dodatnih komponenti za zvuk, fiziku i upravljanje korisničkim sučeljem. U radu su detaljno opisane metode implementacije i prilagodbe tih alata, a osobita pažnja posvećena je praktičnom razvoju igre i izazovima s kojima smo se susreli. Pokazalo se da je Unity vrlo fleksibilan alat za razvoj 3D igara te omogućuje širok spektar prilagodbi.

Rezultati rada pokazuju da su sve ključne funkcionalnosti uspješno implementirane, a postavljene pretpostavke, poput mogućnosti integracije različitih komponenti (zvuka, fizike i sučelja) u cjelovitu igru, potvrđene su. Analiza usporedbe s postojećim igrama pokazala je da je projekt uspješno implementirao osnovne mehanike proklizavanja, s prostorom za daljnje poboljšanje vizualnih elemenata i kompleksnosti razina. Na kraju, projekt je pokazao da je korištenje Unity-a vrlo intuitivno i zanimljivo. Mislim da je Unity vrlo moćan alat za izradu video igara i odličan je za samostalno razvijanje video igra, dok su neki drugi alati možda bolji u razvoju video igra u timu.

Popis literature

[1] Unity Technologies. (n.d.). *Unity Asset Store*. Unity. Dostupno na: **https://assetstore.unity.com** (Pristupano: 31.8.2024)

[2] Technologies, U. (2022.) WheelCollider, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/WheelCollider.html** (Pristupano: 31.8.2024)

[3] Technologies, U. (2022.) Vector3, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/Vector3.html** (Pristupano: 31.8.2024)

[4] Technologies, U. (2022.) TrailRenderer, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/TrailRenderer.html** (Pristupano: 31.8.2024)

[5] Technologies, U. (2022.) Rigidbody, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/Rigidbody.html** (Pristupano: 31.8.2024)

[6] Technologies, U. (2022.) Quaternion, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/Quaternion.html** (Pristupano: 31.8.2024)

[7] Technologies, U. (2022.) ParticleSystem, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/ParticleSystem.html** (Pristupano: 31.8.2024)

[8] Technologies, U. (2022.) Canvas, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/Canvas.html** (Pristupano: 31.8.2024)

[9] Technologies, U. (2022.) Camera, Unity. Dostupno na: **https://docs.unity3d.com/ScriptReference/Camera.html** (Pristupano: 31.8.2024)

[10] Assetto Corsa. (n.d.). *Assetto Corsa Official Website*. Dostupno na: **https://assettocorsa.gg** (Pristupano: 31.8.2024)

[11] BeamNG. (n.d.). *BeamNG Game*. Dostupno na: **https://www.beamng.com/game/** (Pristupano: 31.8.2024)

[12] Playground Games. (n.d.). Forza. Xbox Game Studios. Dostupno na: **https://forza.net/horizon** (Pristupano: 31.8.2024)

Popis slika