Izrada mobilne aplikacije koristeći programski okvir Godot game engine na studijskom slučaju trkaće igre

Komarec, Antonio

VELEUČILIŠTE U ŠIBENIKU

ODJEL RAČUNARSTVA
PREDDIPLOMSKI STRUČNI STUDIJ STRUČNI STUDIJ

Antonio Komarec

Izrada mobilne aplikacije koristeći programski okvir Godot *game engine* na studijskom slučaju trkaće igre

Završni rad

Šibenik, 2024.

VELEUČILIŠTE U ŠIBENIKU

ODJEL RAČUNARSTVA

PREDDIPLOMSKI STRUČNI STUDIJ ili SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ

Izrada mobilne aplikacije koristeći programski okvir Godot *game engine* na studijskom slučaju trkaće igre

Završni rad

Kolegij: Razvoj mobilnih aplikacija

Mentor: Marko Pavelić

Student: Antonio Komarec

Matični broj studenta: 0269155264

Šibenik, rujan 2024.

TEMELJNA DOKUMENTACIJSKA KARTICA

Veleučilište u Šibeniku Završni rad

Odjel Poslovna informatika

Preddiplomski stručni studij

Izrada mobilne aplikacije koristeći programski okvir Godot *game engine* na studijskom slučaju trkaće igre

Trkaća video igra za uređaje s operacijskim sustavom Android. Igrač može postaviti utrku s izborom nekoliko trkaćih staza i nekoliko automobila koje može kontrolirati, a na svakoj se stazi može natjecati s protivnikom kojeg kontrolira računalo. Igra također sadrži i nekoliko izbornika i postavki. Igra je napravljena korištenjem Godot game engina, dok su modeli napravljeni u softveru za modeliranje Blender.

(32 stranice / 19 slika /8 referenci/ jezik izvornika: hrvatski)
Rad je pohranjen u digitalnom repozitoriju Knjižnice Veleučilišta u Šibeniku
Ključne riječi :Android racing game
Mentor: Marko Pavelić mag. ing. inf. et comm. tech., predavač
Rad je prihvaćen za obranu

BASIC DOCUMENTATION CARD

Polytechnic of Šibenik
Batchelor/Graduation Thesis
Department of Bussines IT
Professional Undergraduate Studies

Mobile Aplication development using Godot *game engine* on case study Racing game

ANTONIO KOMAREC
Ćalete Cari 70C, antonio.komarec@vus.hr

Racing video game made for devices with Android operating system. Player has the ability to set up the race with a choice of several racetracks and several car models player can control, player can compete against an opponent controlled by the computer. The game features several menus and settings. The game was made using the Godot game engine, all the models used for the game were made in Blender.

(32 pages / 19 figures / 0 tables / 8 references / original in Croatian language)
Thesis deposited in Polytechnic of Šibenik Library digital repository
Keywords: Racing video game
Supervisor: Marko Pavelić
Paper accepted:

UVOD

Tema ovog rada je trkaća video igra za Uređaje s operacijskim sustavom android. Ova igra sadrži nekoliko tipova vozila te nekoliko trkaćih staza. Korisnik ima mogućnost odabrati tip staze i tip vozila koje će voziti. Također, korisnik bi imao mogućnost postaviti broj krugova za utrku, a za vrijeme same utrke, korisnik bi se mogao natjecati s protivnikom kojeg kontrolira računalo.

Za izradu video igre potreban je *game engine*. *Game engine* je programski okvir koji se koristi za izradu video igara. *Game engine,* među ostalim, daje alate za kreiranje grafike, animacija, fizike i logike video igara. *Game engine* odabran za ovaj rad je Godot. Specifično, Godot verzija 4.4 dev 1 standard. Ova je verzija odabrana zato što je najnovija te pruža najviše korisnih alata. Uz to, ova verzija, u odnosu na Godot 4.3, daje bolje performanse na Android uređajima. Odabran je Godot umjesto poznatijih konkurenata poput Unreal Engine ili Unity zato što već imam iskustva s Godot softverem te znam da Godot *game engine* ima dobru dokumentaciju u kojoj su zabilježeni podaci o svim alatima i mogućnostima koje Godot nudi. Također, Godot je, za razliku od konkurenata, u potpunosti besplatan i *open source*. Još jedna stvar koja je bila velika prednost ovog game *engina* jest mogućnost da se video igre napravljene u Godot-u mogu igrati i na Windows platformama i na Android uređajima, te na drugim popularnim sustavima poput macOS, web, i Linux platformama. Godot službeno podržava pisanje programskog koda u nekoliko različitih programskih jezika. Najpopularniji programski jezici za Godot su C, C++ te GDScript. Programski jezik GDScript je objektno orijentiran jezik napravljen za Godot *game engine*. Kako je programski jezik GDScript jednostavno koristiti uz Godot, odabran je baš taj jezik za potrebe ovog projekta.

Uz *game engine*, potrebne su i teksture. Sve su teksture napravljene u softveru za digitalno crtanje Krita. Nadalje, kako je ovo 3D igra, potrebni su i modeli. Modeli su kreirani u besplatnom softveru za 3D modeliranje Blender.

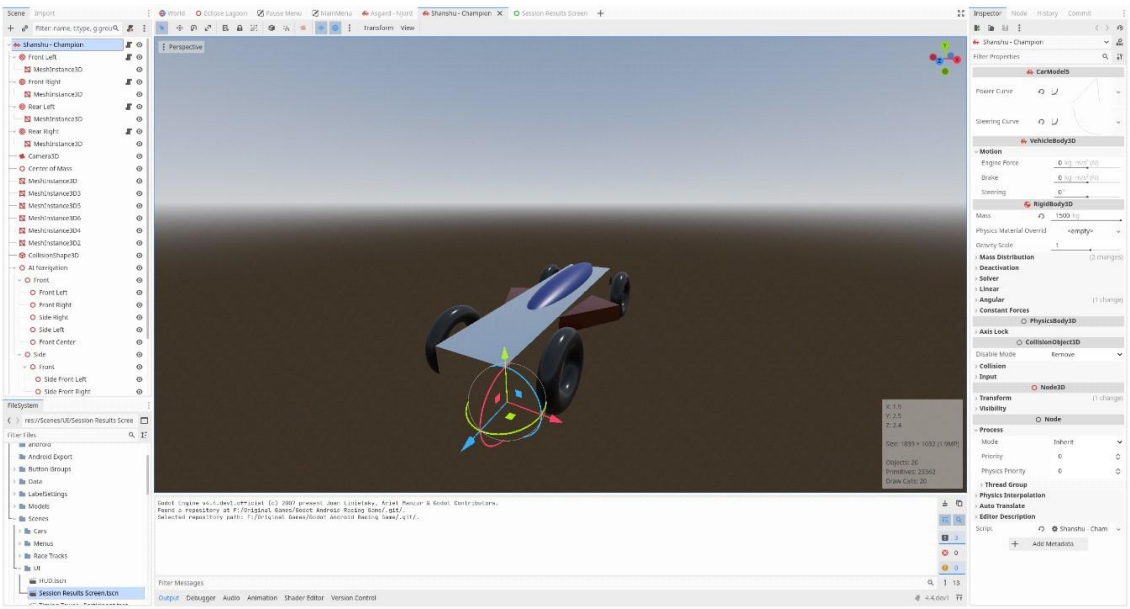
Cilj rada je karirati jednu jednostavnu 3D video igru u kojoj korisnik vozi trkaći automobil, te se natječe s računalnim protivnikom. Sam automobil sastoji se od nekoliko osnovnih komponenata: motor, mjenjač, kvačilo, osovine i kotači, kočnice i model za grafički prikaz. Osim toga potrebno je napraviti i kontrole za mobilne ekrane te korisničko sučelje.

Korisničko sučelje sastoji se od dva osnovna elementa: *heads-up-display* (skr. HUD, grafički prikaz koji se prikazuje za vrijeme same utrke) i korisnički izbornici. HUD će dati korisniku informacije o svojoj poziciji (u usporedbi s pozicijom protivnika), broju krugova utrke, te podatke o stanju vozila (trenutna brzina, broj okretaja motora, i stupanj prijenosa automobila). Igra će imati nekoliko izbornika: glavni izbornik, izbornik za postavke te izbornik za postavke o utrci te izbornik koji se može pokazati za vrijeme utrke, kada se utrka stavi pod pauzom. Za samu video igru potreban je još i svijet, odnosno, trkaće staze na kojima će se voziti.

- **PROGRAMSKI OKVIR GODOT**

Kada se otvori računalni program Godot, prvo se prikaže sučelje na kojem stoji niz Godot projekata. Ovdje se nalazi i mogućnost kreiranja novog projekta. Nakon što je novi projekt napravljen, otvara se Godot Editor. Ovdje se igra stvarno može kodirati, animirati i kreirati.

Kao i svaki *game engine*, Godot nudi velik broj već gotovih klasa koje korisnici mogu koristiti u svojim video igrama. Video igre napravljene u Godotu sastoje se od niza scena. Scena je organiziran skup *node* klasa. Svaka scena spremljena je kao svoja vlastita datoteka s ekstenzijom tscn. *Node* i osnovni element Godota. Ova se klasa koristi kao baza za gotovu svaku klasu u Godotu. U konkretnom primjeru, model automobila nalazi se u sceni Shanshu - Champion.tscn.



Slika 1. Prikaz Godot editora

Glavni *node* u ovoj sceni je `VehicleBody3D`. Ova klasa daje niz korisnih postavki, poput distribucije mase, iznosu snage motora, sili kočnica i slično. Osim `VehicleBody3D`, ova scena sastoji se od još nekoliko *node* klasa, a sve su te klase djeca `VehicleBody3D` klase u hijerarhiji scene. Te ostale klase definiraju izgled i fizički model automobila, podatke o kotačima i kamera. Više o automobilima u cjelini 4. Automobili.

Što se tiče programiranja, odabran je programski jezik GDScript. Programski jezik GDScript je po izgledu sličniji programskom jeziku Python nego programskom jeziku C ili programskom jeziku Java, no programski jezik GDScript je ipak objektno orijentirani jezik. Programski kod pisao sam u samom Godot Editoru. Programski jezik GDScript datoteke imaju ekstenziju gd. Ove će se skripte obično izvoditi samo ukoliko su skripte vezane za neki *node*, te ukoliko je taj *node* aktivan u sceni koja se trenutno koristi. U programskom jeziku GDScript varijable deklarirane su s ključnom riječi var, dok su konstante vrijednosti deklarirane s ključnom riječi const. Kako bi se neke varijable mogle jednostavnije mijenjati, mogu se postaviti u sam editor, koristeći dekorator `@export`. Uz ovaj dekorator, često sam koristio i `@onready`, koji će prikupiti neke podatke jednom kada se ovaj kod počne koristiti za vrijeme izvođenja igre. U programskom jeziku GDScript nije potrebno odraditi tip varijable, no moguće je te poboljšava performanse igre. Tip varijable zadaje se nakon deklaracije, koristeći dvotočke i ključni riječ tipa varijable. Na prvoj liniji svake skripte obično stoji *node* klasa elementa koji koristi skripti. Ovdje se može definirati nova klasa upisom `class_name` i imena nove klase. Na sljedećoj slici prikazana je cijela skripta HUD.gd. Funkcije u programskom jeziku GDScriptu definirane su koristeći `func`, te nakon imena funkcije moraju sadržavati zagrade. U zagrade moguće je staviti parametre.

```gdscript
extends CanvasLayer

@onready var label_speedometer : Label = %Speedometer
@onready var speedometer_unit_label : Label = %SpeedometerUnitLabel
@onready var label_tahometer : Label = %Tahometer
@onready var label_current_gear : Label = %CurrentGear
@onready var timing_tower : ColorRect = %"Timing Tower"

func _ready() -> void:
	Globals.hud = self
	set_hud_speed_unit()

func _exit_tree() -> void:
	Globals.hud = null

func set_hud_speed_unit() -> void:
	if Globals.speed_unit_multiplier == Globals.mps_to_kph_multiplier:
		speedometer_unit_label.text = "KPH"
	else:
		speedometer_unit_label.text = "MPH"

func set_hud(speed : float, rpm : int, gear : String) -> void:
	label_speedometer.text = str(int(Globals.calculate_speed_in_units(speed)))
	label_tahometer.text = str(abs(rpm))
	label_current_gear.text = gear
```

Slika 2. Skripta za *heads-up-display*

Funkcije također mogu biti statične ukoliko ispred `func` piše `static`. Ovakve se funkcije mogu koristiti čak i ako se instance te skripte momentalno ne koristi. Ovakve funkcije koristim u skripti sa postavkama kako bi učitao postavke.

Jedna od važnijih tipova skripti u Godotu su global skripte. Ove skripte će se uvijek izvoditi ne bitno o aktivnoj sceni. Također, postoje skripte koje se mogu aktivirati čim se igra otvori. Ove se skripte nazivaju autoload skripte. Jedna skripta može biti globalna i autoload u isto vrijeme. Primjer takve skripte je Globals.gd skripta ove igre. Ova će se skripta otvoriti jednom kada se otvori igra, te će koristiti dok se god igra ne zatvori. U ovoj skripti imam funkciju koja preračunava metre po sekundi u željene jedinice. Također, u ovoj se skripti, pri otvaranju igre pronađu spremljene postavke te se one učitaju, uz pomoć već navedenih `static func` funkcija iz skripte s postavkama.

```gdscript
static func save_window_mode() -> void:
	var config_file : ConfigFile = ConfigFile.new()
	config_file.load(Globals.save_path_config)
	config_file.set_value("Graphics", "DisplayMode", DisplayServer.window_get_mode())
	config_file.save(Globals.save_path_config)

static func load_window_mode() -> void:
	var config_file : ConfigFile = ConfigFile.new()
	config_file.load(Globals.save_path_config)
	DisplayServer.window_set_mode(config_file.get_value("Graphics", "DisplayMode", 3))

static func load_speed_unit_multiplier() -> void:
	var config_file : ConfigFile = ConfigFile.new()
	config_file.load(Globals.save_path_config)
	if config_file.get_value("Gameplay", "SpeedUnit", true):
		Globals.speed_unit_multiplier = Globals.mps_to_kph_multiplier
	else:
		Globals.speed_unit_multiplier = Globals.mps_to_mph_multiplier

static func load_vsync_mode() -> void:
	var config_file : ConfigFile = ConfigFile.new()
	config_file.load(Globals.save_path_config)
	DisplayServer.window_set_vsync_mode(config_file.get_value("Graphics", "VSyncMode", 1))

static func load_resolution() -> void:
	var config_file : ConfigFile = ConfigFile.new()
	config_file.load(Globals.save_path_config)
	var resolution : Vector2 = config_file.get_value("Graphics", "Resolution", DisplayServer.screen_get_size())
	ProjectSettings.set_setting("display/window/size/viewport_width", resolution.x)
	ProjectSettings.set_setting("display/window/size/viewport_height", resolution.y)

static func load_locale() -> void:
	var config_file : ConfigFile = ConfigFile.new()
	config_file.load(Globals.save_path_config)
	var locale : String = config_file.get_value("Gameplay", "Locale", "en_US")
	TranslationServer.set_locale(locale)
```

Slika 3. Primjer skripte koja koristi statične funkcije, Settings.gd

- **SVIJET**

Jedna od osnovnih komponenti svake video igre je svijet, odnosno, prostor u kojem se vrši glavni dio video igre. U ovom slučaju, najvažniji dio svijeta su trkaće staze. Osim trkaćih staza, svijet sadrži svijetlost, te instance klase `WorldEnvironment`. Bez svijetlosti, svijet bi bio pre mračan te se u njemu ne bi moglo vidjeti. Bez `WorldEnvironment` klase, svijet ne bi imao nebo.

3.1 Trkaće staze

Kako su trkaće staze jedna od glavnih dijelova ove video igre, staze moraju biti prilično dobro napravljene. Pod tim, smatra se da je dobra trkaća staza, staza koja može biti jednostavna za početnike, a opet iskusniji igrači mogli bi pronaći različite trkaće linije kroz zavoje kako bi dovršili krug u najkraćem vremenu. Trkaće staze sastoje se od samog 3D modela koji definira asfalt staze. Svaka staza ima i *grid*. Grid se odnosi na linije na glavnoj ravnini staze, ispred startne linije, na mjesta gdje se postavljaju automobili na startu utrke. Svaka staza ima barem 2 ova mjesta na *gridu*, kako bi se na njih postavili automobil kojeg kontrolira računalo, te automobil kojeg kontrolira igrač. Nadalje, trkaće staze imaju i modele koji služe kao zid na stazi, tako da igrač ne pobjegne sa staze. Zidovi postoje na unutarnjoj i na vanjskoj strani staze, te prate sam asfalt trkaće staze. Svaka staza ima i startnu liniju, te dekorativni element iznad linije, kako bi igrač lakše uočio startnu liniju. Startna linija na stazama služi i kao ciljna linija, kao što je često slučaj u stvarnom moto sportu. Svaka staza ima i niz *checkpoint* tijela, koji služe kako bi se pronašla pozicija automobila za vrijeme utrke te kako bi se izračunala vremenska razlika između automobila. Broj *checkpoint* tijela je relativno visok, oko 300 na svakoj stazi. To su sve ključne komponente jedne staze, uz njih, svaka staza sadrži i neke dekorativne elemente, poput mostova ili rasvjete.

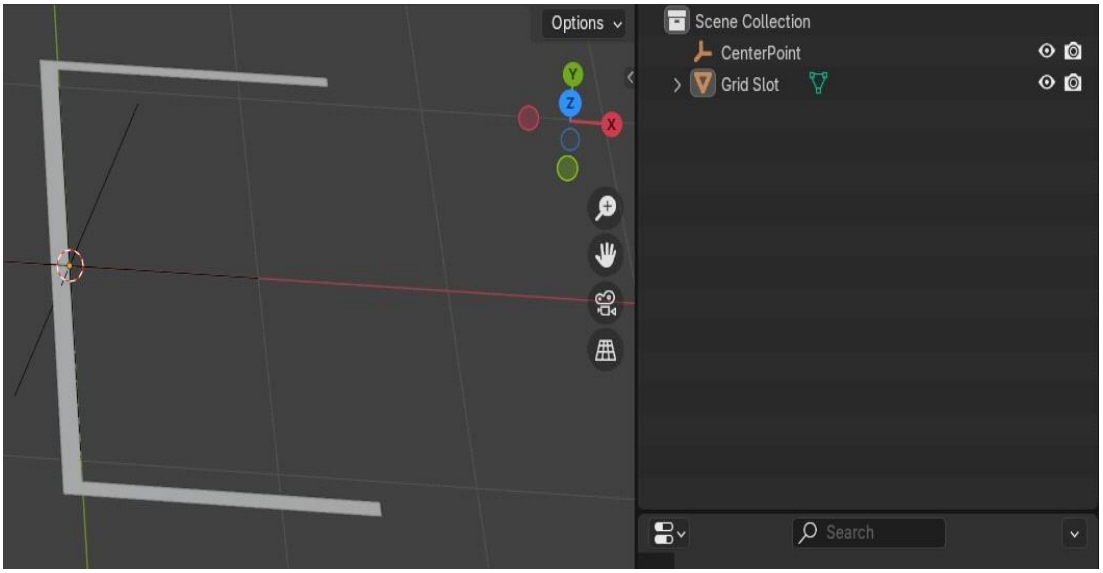
Sve trkaće staze napravljene su koristeći softver Blender. Model same trkaće staze, odnosno, asfalta, je jedan običan plane *mesh,* dok su zidovi staze napravljeni koristeći *cube mesh* kao bazu. Za stazu, odabrano je postaviti širinu na oko 12 metara, što je standard za staze koje se koriste za Formulu 1. Sama dužina objekata za zidove i za samu stazu, zadana je na jedan metar, no zbog načina na koji radim staze, dužina ovih objekata mora se smanjiti na 10

centimetara kako bi rubovi staze kasnije izgledali glatko. Kako bi mi bilo jednostavnije modificirati ove objekte, te tako kreirati zavoje, ovi su objekti spojeni na Bézier *curve*. Bézier *curve* omogućuje da se na jednostavni način kreiraju točke koji čine oblik, odnosno, zavoje trkaće staze. Na kraju je samo potrebno spojiti početak i kraj Bézier *curve* objekta na način da se završna točka stavi na istu poziciju na kojoj se nalazi početna točka. Potom se na elemente koji čine asfalt staze i zidove staze stave dva modifikatora. Potrebno je imati *curve* modifikator, te namjestiti da prati završen Bézier *curve,* kako bi modeli pratili zaobljenost *curve* objekta. Uz *curve modifier,* potrebno je imati i array modifikator. Array modifikator će napraviti niz objekata istog tipa, te se može namjestiti da se kreira onoliko kopija objekata koliko je potrebno da se popuni Bézier *curve*. Bitno je postaviti array modifikator iznad *curve* modifikatora, jer se u suprotnom kreira niz modela koji prate oblik bézier *curve* objekta, no nisu međusobno povezani. Sada je moguće detaljnije oblikovati stazu kroz manipuliranje Bézier *curve* objekta. Tako je moguće promijeniti visinu staze na pojedinim dijelovima, kao i nagnuće staze. Promjenom ovih elemenata, trkaća staze može postati vrlo složena za vožnju pri visokim brzinama u pojedinim automobilima. Glavni razlog za to je čvrstoća ovjesa pojedinih vrlo brzih automobila. Ovaj element daje priliku automobilima koji su sporiji na ravnim dijelovima, da prođu brže kroz zavoje zbog svojeg relativno mekšeg ovjesa.

Kako bi se trkaća staza dovršila, modifikatori se moraju primijeniti, pritiskom na *apply modifier* gumb. Sada je potrebno na stazu postaviti *checkpoint* elemente. *checkpoint* je kreiran u zasebnoj Blender datoteci. U njoj se nalazi samo jedan *cube mesh* čija je visina postavljena na 12 metara. Potrebno je u datoteci s trkaćom stazom, unijeti *checkpoint* datoteku, te je postaviti na željenu poziciju, pod željenom rotacijom. Kako svaka staza ima velik broj *checkpoint* objekata, odlučio sam napraviti Blender skriptu koje će postaviti *checkpoint* objekte na željene pozicije. Blender omogućuje korištenje skripti napisane u programskom jeziku Python kako bi se promijenile informacije o Blender datoteci. U ovoj skripti, potrebno je pronaći datoteku s *checkpoint* objektom, te zabilježiti dužinu Bézier *curve* objekta staze. Python skripta sadrži i varijablu i kojoj stoji željeni razmak izađu *checkpoint* objekata. Broj *checkpoint* objekata koje je potrebno kreirati dobije se dijeljenjem dužine staze s željenim razmakom između *checkpoint* objekata. U ovom slučaju, željeni razmak iznosi 10 metara. Skripta sada prolazi kroz for petlju, u kojoj se za svaki potrebni *checkpoint,* dodaje *checkpoint* element iz *checkpoint* Blender datoteke. Na novi se *checkpoint* tada postavlja *curve* modifikator. Širina *checkpointa* postavlja se na širinu tog dijela trkaće staze, tako da *checkpoint* prekriva čitavu širinu staze. Nadalje, na modifikatoru se definira lokacija novog

checkpoint objekta, i za kraj se na svakom *checkpoint* objekta primijeni *curve modifikator*. Još je bitno napomenuti kako se ime objekta svakog objekta postavlja kao „*Checkpoint* X", gdje X predstavlja redni broj *checkpointa*. Redni broj *checkpoint mesha* kasnije će se koristiti za identifikaciju *checkpointa* kojeg je prošao automobil. Sada staza postoji, sadrži zidove te *checkpoint* objekte. Sada je potrebno još postaviti i *grid* te startnu liniju. Ovaj sam problem također riješio Blender skriptom.

Kako bi se napravio *grid* potrebno je prvo napraviti sam *mesh* objekt koji se koristi za svako pojedino mjesto na *gridu*. Taj *mesh* je napravljen u svojoj posebnoj Blender datoteci.



Slika 4. Blender datoteka s jednim grid mjestom

U skripti je najprije potrebno kreirati startnu liniju. Startna linija je jednostavan plane *mesh* objekt bijele boje, dužine 10 centimetara. U skripti, ova će se linija postaviti na mjesto gdje je prvi *checkpoint* objekt, dok će se *grid* nalaziti iza ove pozicije. To znači da je potrebno ostaviti dovoljno prostora iza zadnjeg zavoja, kako bi se sva *grid* mjesta mogla postaviti na ravan dio staze. U suprotnom, neki bi automobili mogli imati prednost na startu, ovisno o smjeru i kutu zaobljenosti zavoja. Sada je potrebno na stazu postaviti *grid* mjesta u željenom broju na željenu lokaciju. U Formuli 1, dužinski razmak između svakog *grid* mjesta je 8 metara, odlučio postaviti taj razmak u ovoj igri. Razmak u širini između *grid* mjesta ovisi o širini staze. Potrebno je definirati željeni broj *grid* mjesta. Ovaj broj ovisi o pojedinoj trkaćoj stazi, a obično je postavljen oko 30. Te zadnja potreban varijabla u skripti je razmak dužine između prvog *grid* mjesta i startne linije. Iznos ovog razmaka također je inspiriran Formulom

1, te ovisi o pojedinoj trkaćoj stazi. Obično iznosi oko 15 metara. Uz sve ove podatke jednostavno je napraviti skriptu koja će pronaći startnu liniju, od nje će otići 15 metara, te će stvoriti prvo *grid* mjesto. Zatim će se na tu poziciju dodati još 8 metara kako bi se stvorilo drugo *grid* mjesto. Te se kroz ovu for petlju kreće stalno, dok broj *grid* mjesta nije jednak iznosu varijable s brojem željenih *grid* mjesta. Na kraju izvođenja ove skripte, trkaća staza ima sve potrebne dijelove kako bi se koristila u igri. Međutim, ova trkaća staza se i dalje može poboljšati.

Postoje dvije glavne stvari koje se mogu poboljšati na ovoj stazi: dodavanje dekoracija, i prilagodba staze. Što se tiče dekoracija, na stazama se mogu dodati elementi kako bi se sama staza uljepšala. Neki od tih elemenata su most koji prolazi iznad trkaće staze, jezero koje se nalazi usred staze, rasvjeta uz stazu i slično. Uz to, jedna stvar koju je jednostavno izvesti, a uveliko uljepšava stazu su materijali na *mesh* objektima. Bez ovih materijala, *mesh* objekti su uvijek bijele boje. Prvi očiti materijal koji postoji je materijal koji koristim za asfalt. Svaka staza koristi malo drugačiju nijansu crne boje. Zidovi na rubu staze će također dobiti svoj materijal, čija boja ovisi o pojedinoj trkaćoj stazi. I ostali dekorativni elementi imaju svoj materijal, između kojih izdvajam materijal koji koristim za metalne površine poput stupova rasvjete. Osim same boje materijala, ovi materijali sadrže i postavku `metallic`, koja materijalu daje metalni sjaj. Uz `metallic` postavku, ovi materijali sadrže i `roughness` postavku koja namještena na float vrijednost između nule i jedan. Ukoliko je `roughness` namješten na nulu, materijal će odbijati svjetlost poput zrcala. Na drugoj krajnjoj vrijednosti od jedan, materijal uopće ne odbija svijetlost. U ovom slučaju, `roughness` vrijednost iznosi 0.5.

Kako bi napravio jezero, osim samog modela površine jezera, potrebna je i shader skripta. Shader je softver koji se koristi za kreaciju efekata. Korištenjem shader skripte, moguće je promijeniti boju svakog pojedinačnog piksela na ekranu. Godot *game engine* koristi vlastiti jezik za shader skripte. Cilj ove skripte je napraviti realističnu površinu jezera, što znači da je potrebno kreirati malene valove te obojiti piksele u nijanse plave boje.

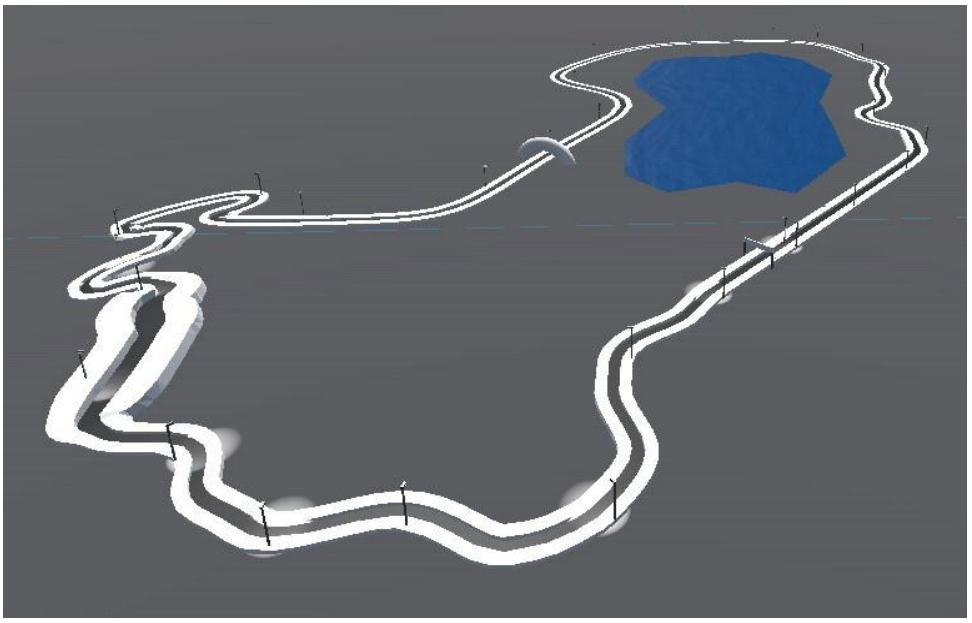
```
shader_type spatial;

uniform vec3 albedo : source_color;
uniform float metallic : hint_range(0.0, 1.0) = 0;
uniform float roughness : hint_range(0.0, 1.0) = 0.02;
uniform sampler2D texture_normal_map;
uniform sampler2D texture_normal_map_2;
uniform vec2 wave_direction = vec2(1.9, 0.0);
uniform vec2 wave_direction_2 = vec2(0.0, 0.9);
uniform float time_scale : hint_range(0.0, 0.2, 0.005) = 0.025;

void fragment() {
    ALBEDO = albedo;
    METALLIC = metallic;
    ROUGHNESS = roughness;
    vec3 normal = texture(texture_normal_map, UV).rgb;
    vec2 time = (TIME * wave_direction) * time_scale;
    vec2 time_2 = (TIME * wave_direction_2) * time_scale;
    vec3 normal_blend = mix(texture(texture_normal_map, UV + time).rgb,
    texture(texture_normal_map_2, UV + time_2).rgb, 0.5);
    NORMAL = normal_blend;
}
```

Slika 5. Shader skripta za površinu jezera

Kako bi se jezero obojalo koristim osnovnu plavu boju, te postavljam `metallic` i `rougness` parametre kako bi jezero izgledalo kao da odbija, ali i upija neku količinu svjetlosti. Kako bi jezero imalo valove, napravio sam dvije dvodimenzionalne teksture šuma. Teksture šuma napravljene su od niza pseudo nasumičnih vrijednosti. Koristim dvije teksture šuma kako bi voda izgledala kao da se spajaju valovi koji dolaze iz dva različita izvora. Kako bi se valovi kretali, koristim jednostavnu animaciju koja će postepeno prolaziti kroz teksturu šuma. Zatim se dvije teksture šuma spajaju u jednu, te se kroz tu jednu teksturu šuma kreira `normal`. `Normal` je tekstura koja predstavlja vektor okomit na površinu objekta. `Normal` vektor se koristi za kalkulaciju svjetlosti, te se u slučajevima kada jedan objekt ima `normal` s nekoliko različitih vrijednost, može vidjeti da je površina objekta neravna. Korištenjem `normal` vektora na površini jezera, dobije se iluzija trodimenzionalnih valova na površini.



Slika 6. Dovršeno jezero

Sada je trkaća staza ima sve potrebne dijelove kako bi bila funkcionalna u igri, te sadrži niz dekoracija kako bi igra ljepše izgledala. Međutim, kako je ovo igra koja mora funkcionirati na mobilnim uređajima, uređajima s lošim performansama, pametno je pojednostavniti trkaće staze. Sama trkaća staza, kao i zidovi vanjskog ruba staze, napravljeni su od niza *vertex* točaka. *Vertex* točke su među povezane točke u trodimenzionalnom prostoru. Kako je osnovni model staze dug 10 centimetara, a svaka trkaća staza duga je najmanje 4 kilometara, a svakih 10 centimetara potrebne su najmanje dvije točke, to znači da svaka staza sadrži najmanje 80,000 točaka. Zidovi staze također imaju velik broj *vertex* točaka. Kako je ovo velik broj *vertex* točaka, moguće je da će se na slabijim uređajima dogoditi pad broja sličica po sekundi. Stoga je bitno na neki način smanjiti broj *vertex* točaka, bez pada kvalitete trkaće staze. Vratimo se u Blender. Broj *vertex* točaka može se smanjiti na jednostavan način, korištenjem *merge by* distance funkcije u Blenderu. Potrebno je unijeti željenu udaljenost, a Blender će automatski izbrisati *vertex* točke koje bliže od unesene udaljenosti. Udaljenost koju sam unio, ovisila je o trkaćoj stazi, a obično je bila oko 3 metra. Što znači da sada, umjesto svakih 10 centimetara, staza sadrži dvije *vertex* točke svaka 3 metra. Ova promjena uvelike smanjuje broj *vertex* točaka na stazi se smanjuje napor uređaja pri učitavanju i kalkulaciji fizike na stazi. *Merge by distance* koristio sam na samom asfaltu staze kao i na zidovima staze. Uz ove promjene trkaće su staze dovršene. Staze sada imaju svoju asfaltiranu površinu, *grid*, startnu liniju, *checkpoint* objekte, zidove, dekorativne elemente, te su staze pojednostavljene. Sljedeći je korak importirati Blender objekte u Godot.

3.2. Importiranje Blender objekata u Godot

Kako bi se staze mogle koristiti u igri, potrebno ih je prenijeti u Godot. Trkaće se staze trenutno nalaze u datotekama s Blenderovom ekstenzijom blend. Staze je moguće eksportirati iz Blendera u neki drugi oblik, među kojima su poznatiji Collada, Wavefront, FBX i glTF 2.0. Ukoliko se model u Godot unosi iz jednog od tih oblika, bilo bi ponovno vršiti eksportiranje u Blenderu, svaki put kada bi se Blender datoteka promijenila. Ovan način zahtjeva nešto više vremena, no posti jednostavniji način. Godot ima sposobnost automatski pretvoriti samu Blender datoteku u glTF 2.0, te se kroz ovaj način importiranja modela, model mijenja u Godot scenama automatski, bez potrebe ručnog eksportiranja glTF
2.0 datoteke iz Blendera. Automatski importer Godota može importirati objekte te ih postaviti kao *mesh* neke klase, ovisno o sufiksu naziva samog *mesh* objekta. Ovo je vrlo korisna funkcionalnost. Jedan čest primjer gdje se ona koristi u ovoj igri je kod importiranja *checkpoint mesh* objekta. *Checkpoint mesh* mora biti importiran kao objekt `Area3D` klase u Godotu. Kako bi se *mesh* automatski importirao kao Area3D, potrebno je samo u Blenderu postaviti sufiks „-Area3D" na svaki *mesh* objekt koji se koristi kao *checkpoint*. Međutim automatski importer Godota nije dovoljan. Kako bi se hijerarhija komponenti scene organizirala na način koji bi mi omogućio jednostavnije snalaženje u hijerarhiji, koristio sam vlastitu import skriptu klase `EditorScenePostImport`. Jednom kada Godot završi automatski import objekta, pokrenut će se moja import skripta. U njoj se najprije stvaraju dva objekta klase `Node3D`, jedan od njih će sadržavati sve objekte kosi se tiču *grida*, dok će drugi imati sve *checkpoint* objekte. Kako bi se doista promijenila hijerarhija objekata, potrebno je proći kroz svaki objekt u sceni, prepoznati po nazivu objekta ima li objekt veze s *gridom* ili je li objekt *checkpoint*. Ukoliko je objekt *checkpoint*, postavlja se kao dijete `Node3D` klase koja sadržava *checkpoint* objekte, a ukoliko objekt ima veze s *gridom*, objekt će biti dijete `Node3D` klase koja sadrži sve Grid objekte. Nakon izvođenja ove skripte, staze su uspješno importirane u Godot. Svaka se staza zatim postavlja u glavnu scenu svijeta igre, čime je dovršen sam svijet. Međutim, kako bi se staze mogle koristiti u igri, potrebno je dodati skriptu koju će koristiti svaka trkaća staza. Ova skripta morat će pokrenuti trkaću stazu na početku utrke, povezati potrebne funkcije kako bi *checkpoint* elementi bili funkcionalni, te mora omogućiti korištenje startne linije kako bi se brojali prijeđeni krugovi svakog automobila.

3.3. Skripta trkaće staze

Iako su staze dovršene, kada bi korisnik sada pokrenuo igru ne bi bilo moguće računati razmak između protivnika niti broj prijeđenih krugova. To znači da se jedna utrka nikada ne bi mogla dovršiti. Stoga je potrebno stvoriti novu skriptu, Race Track, te nju postaviti kao glavnu skriptu svake trkaće staze. Ova će skripta, jednom kada je trkaća staza odabrana, ali prije nego što utrka započne, omogućiti detekciju prijelaza automobila kroz *checkpoint* objekte. Ovo je moguće napraviti korištenjem signala `body_entered`, kojeg ima `Area3D` klasa. Ovaj signal pozvat će se jednom kada automobil uđe u *checkpoint*, s parametrom tog automobila. Kako će svaki *checkpoint* aktivirati taj signal, potrebno je prepoznati koji je *checkpoint* pravi pozivatelj. Stoga se koristi `bind` metoda, čiji je parametar naziv *checkpointa*. Ovaj će signal aktivirati funkciju `on_checkpoint_body_entered`. Kako je će ova funkcija biti vrlo slična funkciji koja bi se trebala pozvati jednom kada automobil prođe startnu liniju, i startna ću linija pozvati funkciju `on_checkpoint_body_entered`, no u tom će slučaju drugi parametar biti string "Start Line", dok će prvi parametar i dalje biti automobil koje je prošao kroz ovaj objekt. Detalji `on_checkpoint_body_entered` funkcije objašnjeni su poglavlju Podaci o utrci, cjeline 6. Heads-up-display. Ukratko, ova će funkcija pogledati je li drugi parametar ime *checkpointa* ili "Start Line". Ukoliko se radi o startnoj liniji, te ukoliko je gotov krug automobila legitiman, automobil je dovršio krug, te započinje novi. Ukoliko se radi o zadnjem krugu utrke, na ekranu će se pojaviti scena s rezultatom utrke. Ukoliko je prvi parametar ipak *checkpoint*, potrebno je obaviti pregled informacija. U pregledu, bitno je pronaći zadnji prijeđeni *checkpoint* tog automobila, ukoliko je identifikacijski broj tog *checkpointa* manji od ovog, računat će se da je ovaj *checkpoint* prijeđen. Nebitno o drugom parametru, nakraju `on_checkpoint_body_entered` funkcije pozvat će se funkcija koja će izračunati razmak između automobila.

U Race Track skripti, na početku utrke također započinje mjerenje vremena koje je se koristi za računanje razmaka između automobila, te se pregleda koliko se *checkpoint* objekata automobil mora proći u jednom krugu kako bi se krug smatrao dovršenim. Više o ovom u poglavlju Podaci o utrci, cjeline 6. Heads-up-display. Skripta trkaće staze također sadrži samu funkciju koja kalkulira udaljenost između automobila, no i ovaj dio skripte bit će opisan kasnije.

S ovim je funkcionalnostima skripta trkaće staze dovršene, a s njom su dovršene i sve radnje koje što se tiše postavljanja i definiranja trkaćih staza i samog svijeta. S gotovim stazama i gotovim svijetom, igra se bliži stanju kada se stvarno može koristiti, no za to su potrebni automobili.

- **AUTOMOBILI**

Automobili su najvažniji dio ove video igre. Kako je ovo trkaća igra, automobili moraju biti kvalitetno napravljeni, a kako mi je cilj napraviti igru koje je relativno realistična, automobili moraju biti adekvatno programirani, bez ne realistično jednostavnih sustava i dijelova automobila. Svaki model trkaćeg automobila sastoji se od 2 različita modela: modela za vizualni prikaz auta i modela koji se koristi za fizička svojstva auta. Bez modela za fizička svojstva auta, ne bi bilo moguće kalkulirati doticaje i sudare s ovim autom. Odnosno, kada bi se auto bez fizičkog modela zaletio u zid, auto bi samo prošao kroz zid zato što na fizičkom modelu provode kalkulacije o sudarima. Kao i kod trkaćih staza, modeli automobila su napravljeni u Blenderu, te sadrže materijale različitih boja i drugih postavki. Fizički modeli su jednostavniji od vizualnih iz razloga što je za kalkuliranje fizičkih svojstva kompleksnih modela potrebno imati bolje uređaje. A kako kalkulacije svojstava vizualnih modela nisu toliko zahtjevne, ovi modeli sadrže veći broj *vertex* točaka.

Osim modela, automobili imaju i dvije kamere. Kamere su objekti koji daju mogućnost prikazati modele za vizualni prikaz. Automobili imaju jednu kameru koja gleda prema naprijed i kameru koja gleda prema nazad, tako da igrač može vidjeti što se iza događa iza svog auta. Glavna kamera je kamera koje gleda prema naprijed, do će se kamera koristiti samo kada igrač stišće gumb za pogled prema nazad.

Svaki automobil ima i četiri kotača, od kojih su samo prednji mogu okretati lijevo i desno, dok su stražnji kotači pogonski. Prednji kotači sadržavati podatke o poziciji na koju se automobil postavlja na *grid* za start utrke. Područje gdje guma dotiče tlo ne svije dotaknuti bijelu liniju *grida,* stoga se automobili postavljaju na *grid* na lokaciji prednjih kotača uz ukalkuliran razmak koji se dobije kroz radijus kotača. Svaki objekt kotača na sebi sadrži i podatke o ovjesu. Ovjes svakog kotača svakog auta ima svoju tvrdoću, te najveću udaljenost koju smije proći od mjesta gdje ovjes započinje. Mekši ovjes može podnijeti lošiji teren pod većom brzinom od tvrđeg, ali mekši ovjes će biti teže kontrolirati kroz zavoje s većim brzom. Stvarni trkaći automobili koji se utrkuju na glatkim površinama općenito imaju tvrd ovjes, dok je mekši ovjes čest kod automobila koji voze po neravnom terenu.

Sam automobil ima podatke o svojoj masi te o lokaciji centra mase. Nadalje, svaki model automobila ima svoju vlastitu dvodimenzionalnu krivulju snage koja se koristi za kalkuliranje faktora snage pri nekom broju okretaja motora.

Ova video igra ima pet različitih trkaćih automobila. Svaki od tih pet automobila nasljeđuje klasu `Car`. U klasi `Car` opisani su osnovni podaci koje ima svaki model automobila, podaci o kontrolnoj skripti automobila, te metode koji modeli automobila koriste. U kodu samog modela automobila deklarirane su vrijednosti koje se koriste u metodama klase `Car`. Te su vrijednost snaga motora, podaci o mjenjaču, i dužina međuosovinskog razmaka. U skripti modela automobila još stoji i metoda koja se stalno izvodi. U toj se metodi obavljaju operacije s upravljanjem auta te se dobavljaju informacije o trenutnoj brzini auta i trenutnom broju rotacija motora.

4.1 Kontrolne skripte

Kao što je već napisano, klasa `Car` sadrži varijablu koja mora dobiti instancu kontrolne skripte koju koristi automobil. Kontrolna skripta daje informacije o inputu, koji se koristi u ostalim skriptama. Postoje dva tipa kontrolnih skripti, skripta koja se koristi ukoliko automobil kontrolira igrač, te skripta koja se koristi ukoliko automobil kontrola računalo. Kod kontrolne skripte za igrača, sav input dolazi od samog igrača te je ovaj dio bilo jednostavno programirati.

S druge strane, kontrola skripta za računalo je, u ovoj izvedbi jednostavan. Svaka trkaća staza sadrži put kojeg čini niz 3D točaka, a automobil prati ove točke pod konstantom brzinom. Postoji nekoliko očitih problema kod ovog načina rada. Ukoliko se automobil kojeg kontrola računalo sudari s automobil kojeg kontrolira vozač, računalo automobilom će nastaviti voziti kao da se ništa nije dogodilo. Nadalje, kako ovaj automobil uvijek prati jedu liniju, ovaj automobil nije reaktivan, odnosno, ne može niti braniti svoju poziciju, niti napadati više pozicije na realističan i dinamičan način. Rezultat ovoga je jedan dosadan i predvidljiv protivnik. Možda najveći problem ovog načina rada, što se tiče zabave igre i realizma, je činjenica da ovaj automobil uvijek vozi istom brzinom. Ovaj problem je očigledan na startu utrke. Dok igrač stoji na startu, te treba lagano proći kroz različite stupnjeve prijenosa, protivnik započinje utrku s istom brzinom koju uvijek koristi. To znači da ovaj automobil

koristi različiti model fizike od automobila kojeg kontrolira igrač. Automobil računala ne koristi isti kod kao automobil igrača što se tiče pogonskog sklopa i kočnica. Kako ovaj automobil uvijek vozi istom brzinom, automobil računala ne koristi kočnice. Rezultat ovoga je automobil koje je izraziti spor na ravnim dijelovima staze, a opet izrazito brz kroz spore zavoje. U teoriji, ovaj bi se problem mogao riješiti ukoliko bi se gledala udaljenost između ruba trkaće staze i automobila. Na prednjoj strani automobila može se dodati nešto poput lasera, koji će prepoznati rub staze. Kod tog bi se modela, ukoliko je rub staze dovoljno blizu, mogle aktivirati kočnice. No postoji problem. Recimo da se ispred automobila nalaze dva zavoja, prvi zavoj koji je relativno brz, te je drugi zavoj nakon njega, zavoj koji je izuzetno spor. Ukoliko model samo računa udaljenost do prvog zavoja, model bi zaključio da nije potrebno kočiti toliko jako, te će automobil doći do drugog zavoja s velikom brzinom. Kada automobil prođe ovaj zavoj, automobil će računati udaljenost za drugi zavoj te će zaključiti da ide prebrzo te da je potrebno kočiti. Kako se drugi zavoj nalazi neposredno iza prvog, automobil nema dovoljno vremena usporiti, te će u zavoj uči prebrzo, a nakraju će izletiti sa staze. Rješenje je model koji će naći razliku ne samo za prvi zavoj, već i za ostale zavoje. Ovakav bi sustav trebao biti relativno složen, čak i u svojem najjednostavnijem obliku.

Konkurenti ovog koncepta automobila kojeg vozi računalo je sustav koji koristi strojno učenje. Ovakav bi sustav koristio velik niz pokušaja, kako bi kao rezultat dao kvalitetan, dovršen krug na jednoj trkaćoj stazi na kojoj se ovaj sustav trenira. Prednost ovog sustava je, potencijalno, vrlo dobro brzo vrijeme kruga na stazi. Mana ovog sustave je potreba za velikim brojem pokušaja. No, čak i s tom manom, ovaj bi sustav mogao donijeti veliku prednost ovoj igri. Međutim, ovaj bi sustav zahtijevao dovršenu trkaću stazu, te bi funkcionirao samo ukoliko je model treniran na toj konfiguraciji staze. Ukoliko se staza promijeni, pa makar i za samo jedan zavoj, model bi se ponovno trebao trenirati. Kako sam kroz čitavo vrijeme rada na ovom projektu mijenjao trkaće staze, ovo nije adekvatno rješenje.

Osim problema koji nastaju kao posljedica osnovnog koncepta ove skripte, automobil kojeg kontrolira računalo često je prespor. Djelomičnu, ovaj je problem rezultat ne dinamičnog koncepta, no ovaj je problem čest bez obzira na koncept. Jedno od čestih rješenja ovog problema je dati automobilu ne realističnu prednost kroz povećanje snage motora (engl. *rubberbanding*). Ovakav sustav često se koristi kako bi riješio i suprotan problem. Ukoliko je igrač pre spor, računalo će usporiti, bez realističnog razloga, te tako dati igraču priliku da dostigne svog protivnika. Cilj ovog projekta je video igra koja je barem relativno realistična,

te je ovo rješenje često nepopularno kod poznatih trkaćih igrica. To znači da je spor protivnik realnost ovog projekta.

4.2. Skretanje automobila

Automobili u ovoj video igri koriste model skretanja pod nazivom Ackermann *steering geometry*. Kada automobil prolazi kroz zavoj, kotači na unutarnjoj strani zavoja prolaze kroz manji radijus zavoja od kotača na vanjskoj strani zavoja. Zbog toga jer potrebno rotirati kotače s unutarnje strane zavoja pod oštrijim kutom od kotača s vanjske strane zavoja, te je ovo problem koji se rješava s Ackerman *steering* modelom. Ovakav model upravljanja automobila omogućuje veću stabilnost pri skretanju, posebice pri većim brzinama te kod oštrijih zavoja. Kako se lijevi kotač okreće pod različitim kutom od desnoga, potrebno je prilagoditi kod.

```
179    if abs(control_script.steering_input) > 0.001:
180        turning_radius = wheel_base / tan(control_script.steering_input)
181        left_steering = atan(wheel_base / (turning_radius - rear_track / 2))
182        right_steering = atan(wheel_base / (turning_radius + rear_track / 2))
```

Slika 7. Osnovan kod za skretanje koji baziran na Akcermann modelu

Kod radi sljedeće. Ukoliko je apsolutna vrijednost inputa za skretanje veće od 0.001, izračunat će se željeni radijus skretanja. Željeni radijus skretanja je rezultat dijeljenja razmaka osovina s tangensom inputa za skretanja. Zatim se željeni radijus skretanja koristi kako bi se došlo do potrebnog radijusa skretanja, koji je poseban za lijevi, te poseban za desni kotač. Ovo je sama baza Ackermann *steering* modela. Međutim, automobili će skretati pod različitim kutom, ovisno o trenutnoj brzini automobila. Pod niskim brzinama, automobil može rotirati kotače pod većim kutovima bez problema. Problemi nastaju kada se kotač rotira pod istim kutom kao i prije, no pod mnogo većom brzinom. U ovom slučaju velika je vjerojatnost da će automobil biti nestabilan, te dolazi do pojave prevrtanja automobila. Stoga će sljedeći dio koda skretanja automobila morati prilagoditi skretanje samog automobila ovisno o njegovoj trenutnoj brzini, na način da se input igrača ne mijenja. Odnosno, recimo da igrač vozi sporo, te daje input za skretanje od 90 stupnjeva prema lijevo. Automobil će kotače okrenuti za oko 60 stupnjeva na željenu stranu. Nadalje, ako igrač vozi brzo, te opet daje input za skretanje od 90 stupnjeva prema lijevo, automobil će okrenuto kotače za oko 15 stupnjeva

prema lijevo. Ovakva će primjena koda također omogućiti jednostavnijim načinom upravljanja za sama igrače, jer bi se bez ovog koda moglo desiti da se i maleni input za skretanje protumači kao relativno velik kut skretanja pri brzoj vožnji. Tako da se odmah ispod gore navedene slike koda, nalazi slijedeći kod.

```
184        speed_factor = clamp(vehicle_body.linear_velocity.length() / 100, 0.5, 30.0)
185        left_steering *= speed_factor
186        right_steering *= speed_factor
187        steering_factor = clamp(vehicle_body.linear_velocity.length() / 100, 0.03, 1.0)
188        max_steering_angle = vehicle_body.steering_curve.sample_baked(steering_factor)
189        max_steering_angle_clamped = clamp(max_steering_angle, 0.03, 1.0)
```

Slika 8. Kod za skretanje čija vrijednost ovisi o trenutnoj brzini automobila

Definirana je varijabla `speed_factor`, koja će prvo poprimiti vrijednost trenutne brzine automobila u metrima po sekundi, a zatim se ta vrijednost dijeli sa sto. Vrijednost varijable će zatim poprimiti vrijednost rezultata, ili ukoliko je rezultat niži od donjeg limita (0.5), ili veći od gornjeg limita (30.0), rezultat će biti taj limit. Zatim će se `speed_factor` koristiti kao množitelj za kutove skretanja koji su definirani u Ackermann *steering* dijelu koda. Zatim, potrebno je doći do faktora za skretanje, kojeg definira trenutna brzina automobila. Ovaj dio koda je sličan definiciji `speed_factor` varijable, no u ovom se slučaju vrijednost varijable definira na donji limit od 0.03, te gornji limit koji iznosi 1.0. Faktor za skretanje koristit će se kao pozicija na X osi krivulje za skretanje, te se koristi kao parametar `sample_baked` metode kako bi se došlo do vrijednosti na Y osi te iste krivulje. Ta je float vrijednost uzeta kao baza za najveći kut skretanja moguć pri nekoj brzini. U slučaju da je krivulja netočno postavljena, odlučio sam limitirati ovu vrijednost između 0.03 i 1.0, te je to čitav kod za postavljanje vrijednost `max_steering_angle_clamped` varijable.

Naposljetku, potrebno je postaviti vrijednosti skretanja svakog kotača. Kako bi se to dogodilo prvo je potrebno naći baznu vrijednost za taj kotač. Ta će se vrijednost pomnožiti s vrijednosti `max_steering_angle_clamped` varijable, te je rezultat željena rotacija kotača. Kako nije poželjno da se kotač teleportira na novu zadanu rotaciju, potrebno je postepeno pomaknuti rotaciju kotača iz trenutne, u novu željenu rotaciju. Kako bi se to dogodilo, koristim `move_towards` metodu. Ova metoda prima tri parametra, gdje prvi definira početno stanje varijable, drugi parametar je željena vrijednost na kraju korištenja metode, te treći parametar označava delta vrijednost, za iznos koje će se vrijednost prvog parametra

mijenjati dok god vrijednost nije jednaka vrijednosti drugog parametra. Ova se funkcija vodi dva puta, jednom za prednji lijevi kotač, te jednom za prednji desni kotač. U kodu, prvi parametar je trenutna rotacija kotača. Drugi parametar iznosi umnožak željene rotacije kotača s maksimalnom vrijednosti rotacije kotača. Treći parametar je umnožak varijable delta s brojem koji utječe na brzinu položaja kotača. Ova varijabla ovisi o samom modelu automobila. Delta ovdje označava razliku između pojedinih sličica na kojima se računa fizika ove video igre. Kako se sav kod opisan iznad, odvija samo u slučaju da je input kontrolne skripte veći od nule, bitno je definirati postupak za rotaciju kotača ukoliko input kontrolne skripte iznosi nula. U tom se slučaju, oba kotača rotiraju od svoje trenutne pozicije, prema rotacijske pozicije nula korištenjem `move_toward` metode.

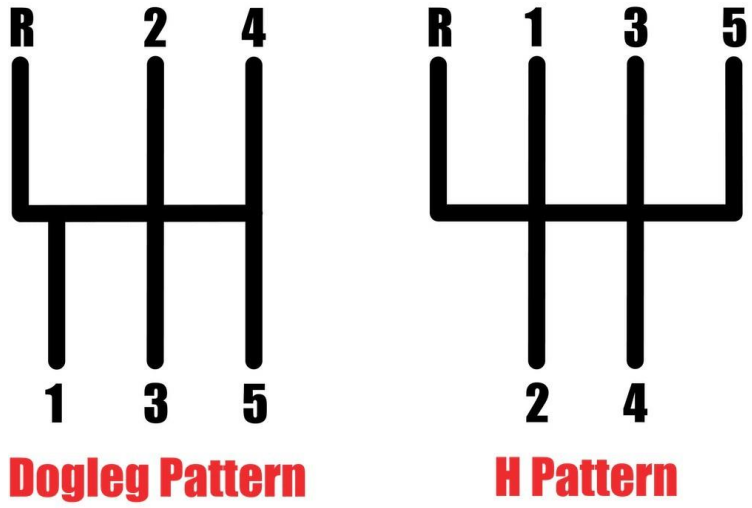
```
179  if abs(control_script.steering_input) > 0.001:
180      turning_radius = wheel_base / tan(control_script.steering_input)
181      left_steering = atan(wheel_base / (turning_radius - rear_track / 2))
182      right_steering = atan(wheel_base / (turning_radius + rear_track / 2))
183
184      speed_factor = clamp(vehicle_body.linear_velocity.length() / 100, 0.5, 30.0)
185      left_steering *= speed_factor
186      right_steering *= speed_factor
187      steering_factor = clamp(vehicle_body.linear_velocity.length() / 100, 0.03, 1.0)
188      max_steering_angle = vehicle_body.steering_curve.sample_baked(steering_factor)
189      max_steering_angle_clamped = clamp(max_steering_angle, 0.03, 1.0)
190
191      $"Front Left".steering = move_toward($"Front Left".steering, left_steering *
192      max_steering_angle_clamped, delta * steering_delta_multiplier)
193
194      $"Front Right".steering = move_toward($"Front Right".steering, right_steering *
195      max_steering_angle_clamped, delta * steering_delta_multiplier)
196  else:
197      $"Front Left".steering = move_toward($"Front Left".steering, 0, delta *
198      steering_delta_multiplier)
199      $"Front Right".steering = move_toward($"Front Right".steering, 0, delta *
200      steering_delta_multiplier)
```

Slika 9. Dovršena funkcija za skretanje

Ovakav model skretanja je poprilično realističan, te rezultira jednostavnoj vožnji automobila za igrače. Automobili pružaju već stabilnost pri skretanju pod velikoj brzini kretnje te veću preciznost pri skretanju u svakom slučaju. Sljedeći važni dio automobila je pogonski sklop, ovaj će element omogućiti automobilu pokret te mijenjanje stupnjeva prijenosa.

4.3. Pogonski sklop

Pogonski sklop automobila sastoji se od dva glavna elementa: mjenjač i, motor. Što se tiče mjenjača, definirano je nekoliko klasa. Glavna klasa mjenjača zove se `Gearbox`, te se ona nasljeđuje u klasama `GearboxManualHPattern` i `GearboxManualDogLeg`, te se ove klase koriste u klasi `Car` kako bi se definirao tip mjenjača u modelu automobila. Razlika ova dva mjenjača je u smještaju pojedinih stupnjeva prijenosa, te kao posljedica toga, u računanju vremena potrebnog za mijenjanje pojedinih brzina. U stvarnom svijetu, kod ručnih mjenjača, stupnjevi prijenosa često su smještana na jednom od ova dva sustava.



Slika 10. Usporedba *dogleg* i *H pattern* sustava, Izvor: www.bimmertips.com

Takozvani *dogleg* sustav čest je kod trkaćih automobila, dok je H *pattern* standard za osobne automobile. Prava razlika između ova dva sustava nastaje zbog različitog položaja stupnjeva prijenosa. Kod standardnog *H pattern* mjenjača, prijenos iz trećeg stupnja prijenosa u četvrti je vertikalan, potrebno je samo spustiti ruku, te je zato i vremenski kraći. Kod *dogleg* sustava, ova promjenu stupnja prijenosa zahtjeva promjenu smjera ruke prema gore, prema desno, pa ponovo prema gore. Rezultat je da sama promjena brzine traje malo duže.

Mjenjač se u ovoj igrici na uređajima s operacijskim sustavom Android, koristi automatski. Ukoliko je broj okretaja motora dovoljno visok, za taj model automobila, te ukoliko je igračev input za snagu veći od 0, automobil će promijeniti stupanj prijenosa za 1, prema gore. No, ukoliko igračev input za snagu iznosi 0, te je broj okretaja motora manji od definiranog minimuma za taj model automobila, stupanj prijenosa promijenit će se za 1, prema dolje.

Cilj sustava mjenjača je da promijeni prijenosni omjer. Prijenosni omjer je iznos koji se koristi za danje radnje s motorom, a ovisi o trenutnom stupnju prijenosa automobila. Svaki stupanj prijenosa automobila ima svoj prijenosni omjer. Prijenosni omjer obično je pozitivna vrijednost, a ukoliko je negativna, označava da je automobil u stupnju brzine za vožnju unazad.

U klasi `Car`, snaga automobila dodaje se ovisno o nekoliko parametara. Parametri su: input igrača, maksimalna snaga automobila, pozicija kvačila i prijenosni omjer trenutnog stupnja prijenosa.

Input igrača obično se odnosi na papučicu gasa na ekranu, te je li igrač pritisnuo taj element. No, ukoliko je automobil u stupnju prijenosa s negativnim predznakom, input igrača odnosit će se na papučicu kočnice. To jest, ukoliko automobil ide unazad, igrač koči automobil koristeći papučicu gasa, a igrač dodaje snagu automobilu koristeći papučicu kočnice. Koristeći ovaj sustav, nije potrebno dodati još jedan element na ekran koji bi omogućio korištenje brzina za vožnju unatrag. Automobil će biti stavljen u brzinu za vožnju unatrag ukoliko trenutna brzina automobila iznosi 0, trenutni stupanj prijenosa iznosi 1, input za kočenje iznosi više od 0. Input za snagu, kao i input za kočnice može iznositi između 0 i 1 iz razloga što se ova igrica može upravljati i kontrolerom, što omogućuje i vrijednosti u racionalnim brojevima.

4.4. Kočenje

Kočenje se provodi po osovini. U stvarnim trkaćim automobilima, kočnice prednjeg i stražnjeg ovjesa ne moraju nužnu pružiti jednaku snagu. To je svojstvo modelirano i u ovoj video igri, u varijabli `front_brake_bias`. Svaki model auta ima svoj vlastiti `front_brake_bias`, a on iznosi od 0 do 1, gdje 0 znači da se prednje kočnice ne koriste, a 1 znači da se stražnje kočnice ne koriste.

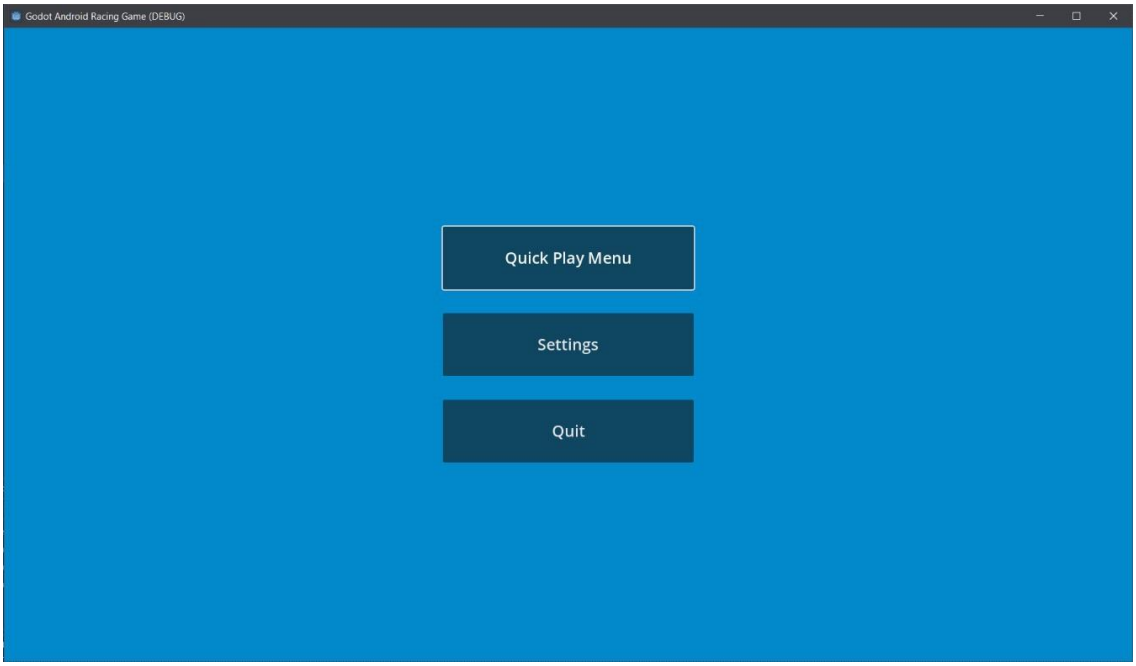
Uz `front_brake_bias`, snaga kočnica ovisit će i o maksimalnoj snazi kočnica te o jačini inputa za kočenje. Input za kočenje može biti između 0 i 1. Automobil će kočiti samo u slučajevima kada input za kočenje iznosi više od 0.

- **IZBORNICI**

Video igre sadrže izbornike kako bi se spojile i organizirale različite funkcionalnosti igre. Ova igra ima četiri izbornika: glavni izbornik, izbornik za brzu igru, izbornik za postavke, izbornik koji se pojavljuje kada je igra pod pauzom i izbornik s rezultatima utrke. Kako bi napravio ove izbornike koristio sam različite verzije `Control` klase koju nudi Godot *game engine*. Ove klase omogućuju brzi i jednostavno stvaranje izbornika.

5.1. Glavni izbornik

Glavni izbornik je prva scene koja se pojavljuje kada se ova video igra otvori i učita. S glavnog izbornika, igrač može otići do izbornika za brzu igru ili izbornika za postavke. Glavni izbornik je također odrediste na koje će igrač doći jednom kada je utrka dovršena. Ukoliko je igra aktivirana na desktop platformi, poput Windows 10, na glavnom će se izborniku pojaviti i mogućnost da se igrica ugasi. Razlog zašto ove mogućnosti nema na mobitelima je ta što se na mobilnim platformama igrica u bilo kojem trenutku može ugasiti uobičajenim kontrolama na mobitelu, dok se na desktop platformama (ukoliko se koristi prikaz preko cijelog zaslona), igra ne može ugasiti s kontrolama koje su prikazane na ekranu.

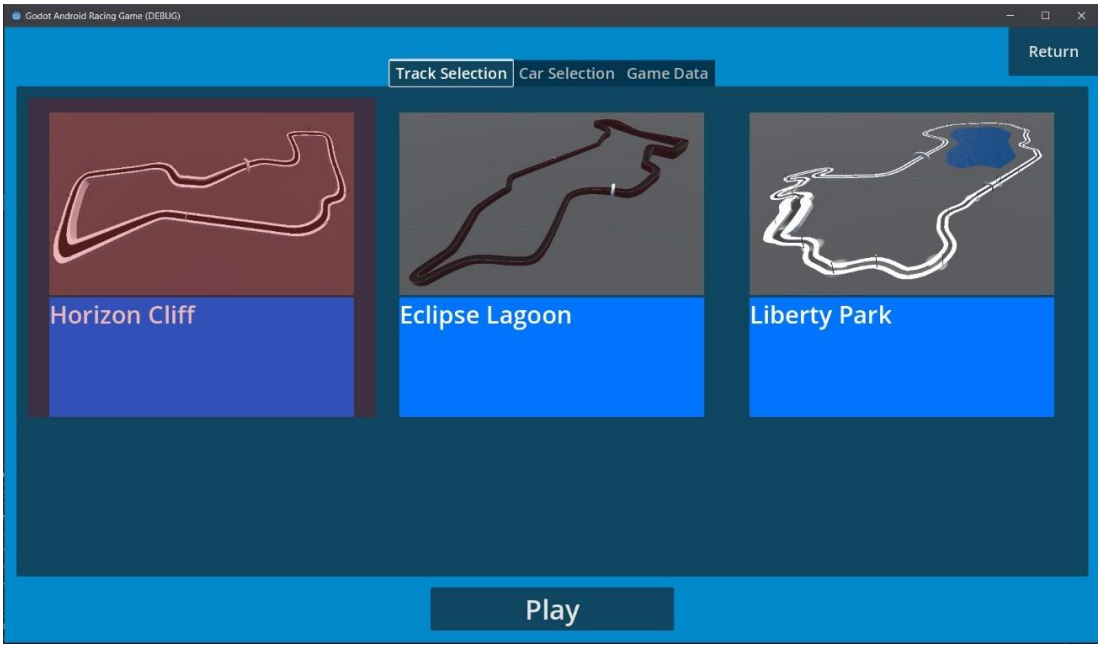


Slika 11. Prikaz glavnog izbornika

5.2. Izbornik za brzu igru

Izbornik za brzu igru dijeli se na 3 izbornika: izbornik za trkaće staze, izbornik za automobil kojeg će voziti igrač, te izbornik o podacima utrke na kojem se može podesiti broj krugova koji će se voziti u ovoj utrci. S ovog izbornika, igrač se može vratiti na glavni izbornik, ili otići u 3D prikaz te voziti utrku.

Izbornik za automobil kojeg će voziti igrač dobit će podatke s lokalne json datoteke na kojoj su spremljeni podaci o svakom automobilu. Ti podaci su naziv samog automobila te slike automobila. Ovi će se podaci koristiti kako bi se kreirale kartice. Igrač će odabrati karticu s automobilom kojeg želi voziti, te će se u sam 3D svijet kreirati objekt odabranog automobila. Igrač može odabrati samo jedan automobil, a ukoliko korisnik nije odabrao vozilo, odabrat će se vozilo s vrha popisa iz json datoteke. Izbornik za trkaće staze funkcionira na sličan način. Odabrana staza ili automobil prepoznat će se po crvenoj boji tog grafičkog elementa.



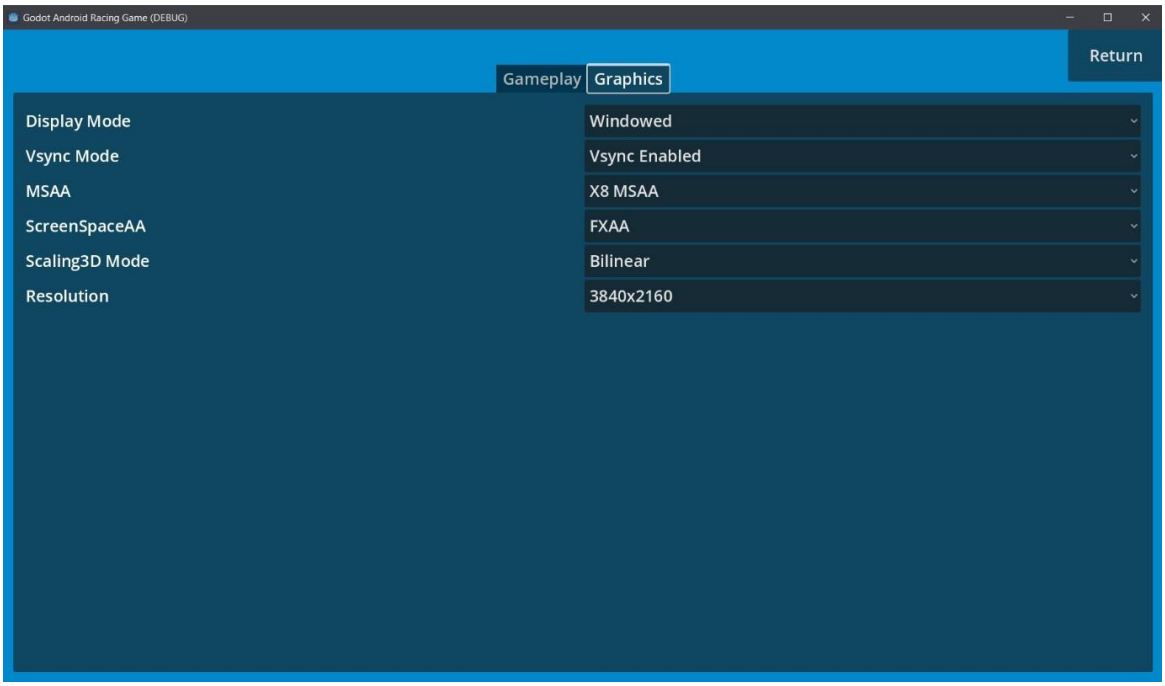
Slika 12. Prikaz izbornika za brzu igru

Jednom kada igrač pritisne element koji će započeti utrku, učitat će se scena koja sadrži svijet igre, zajedno sa svim trkaćim stazama, te će se pronaći odabrana trkaća staza. Učitat će se grid, te će se na željene *grid* pozicije postaviti dva automobila, jedan automobil kojeg će voziti računalo te drugi automobil kojeg je izabrao igrač. Zatim je potrebno automobilima dati

potrebne kontrolne skripte, te se kreira varijabla tipa dictionary u kojoj se zapisuju svi potrebni podaci o utrci, poput broja krugova u utrci, te trenutnom, najvećem broju kruga na kojem se neki automobil nalazi. Uz tu varijablu, potrebno je kreirati još jedan dictionary za svaki automobil, te će se u tu varijablu zapisati podaci o poziciji svakog automobila za vrijeme utrke. Ovi će se podaci koristiti za računanje razlike između automobila.

5.3. Izbornik za postavke

Izbornik za postavke nudi korisniku mogućnost promijeniti neke stavke ove video igre. Ovaj se izbornik sastoji od dva dijela: postavke za igru i postavke za grafiku. Postavke za igru omogućuju korisniku da promijeni jezik. Osim toga, ovdje se može i podesiti mjerna jedinica za brzinu. Opcije su u kilometrima na sat ili milje na sat. Postavke za grafiku korisniku omogućuju opciju promijene detalja o grafičkom prikazu igre. No, ove su mogućnosti dostupne samo na desktop platformama, jer su na mobilnim platformama nebitne ili pre opterećujuće za uređaje. Primjer nebitne postavke za mobilne uređaje je promjena rezolucije. Ukoliko se aplikacija izvodi na mobilnoj platformi, rezolucija će se postaviti na rezoluciju ekrana mobitela. S druge strane, korisnici s desktop platforma moći će postaviti rezoluciju po želji. Primjer postavke koja je pre opterećujuća za mobilne uređaje je MSAA. *Multisample Anti-Aliasing,* skr. MSAA, je tehnologija koja omogućuje izglađivanje rubova objekata. Korisnik s desktop platforme ovu opciju može ugasiti, ili izabrati između nekoliko jačina rada. Na mobilnim uređajima, ova je postavka ugašena. Slična tehnologija je *Fast Approximate Anti-Aliasing,* skr. FXAA, te ni ova tehnika nije prikladna za relativno slabe, mobilne uređaje, te je na njima onemogućena. Na desktop platformama dostupna je i opcija za 3d skaliranje. Ova opcija pruža bolje performanse zbog mogućnosti računanja grafike u manjoj rezoluciji od rezolucije igre. Također, moguće je omogućiti V-Sync tehnologiju. V- Sync uklanja rezanje na individualnim slikama, koja su posljedica ne sinkroniziranih slika za vrijeme izvođenja programa. Naposljetku, u grafičkim postavkama, igrač može promijeniti način prikaza prozora igre. Opcije su prozor, prozor bez granica te prikaz preko cijelog zaslona. Ovu opciju igrač može promijeniti u bilo kojem trenutku pritiskom na tipku F11.



Slika 13. Prikaz izbornika za postavke

Ove se postavke spremaju kada igrač odluči izaći iz izbornika za postavke. Kada se postavke spremaju, izvršava se sljedeća funkcija.

```
163 func save_settings() -> void:
164     var config_file : ConfigFile = ConfigFile.new()
165     config_file.load(Globals.save_path_config)
166     config_file.set_value("Gameplay", "Locale", TranslationServer.get_locale())
167     config_file.set_value("Gameplay", "SpeedUnit", speed_unit_check_button.button_pressed)
168     config_file.set_value("Graphics", "DisplayMode", DisplayServer.window_get_mode())
169     config_file.set_value("Graphics", "VSyncMode", DisplayServer.window_get_vsync_mode())
170     var viewport_width = ProjectSettings.get_setting("display/window/size/viewport_width")
171     var viewport_heght = ProjectSettings.get_setting("display/window/size/viewport_height")
172     var resolution : Vector2 = Vector2(int(viewport_width), int(viewport_heght))
173     config_file.set_value("Graphics", "Resolution", resolution)
174     config_file.set_value("Graphics", "MSAA", get_viewport().get_msaa_3d())
175     config_file.set_value("Graphics", "ScreenSpaceAA", get_viewport().get_screen_space_aa())
176     config_file.set_value("Graphics", "Scaling3DMode", get_viewport().get_scaling_3d_mode())
177     config_file.save(Globals.save_path_config)
```

Slika 14. Dio koda skripte Settings.gd koji sprema postavke

U ovoj je funkciji najprije se kreira nova datoteka tipa INI. Zatim se kreiraju kategorije postavka, te se po njima zapisuju podaci za svaku postavku. Na kraju se nova INI datoteka sprema u mapu gdje se nalaze podaci o aplikaciji. Na Windows platformama ta adresa te mape je AppData\Roaming\Godot\app_userdata\Godot Android Racing Game.

Slika 15. Sastav datoteke u kojoj su spremljene postavke

Spremljeni se podaci moraju i učitati. Za učitavanje postavki potrebno je stvarno učitati i postaviti podatke u video igri, te postaviti podatke za grafičke elemente na izborniku za postavke. Postavke se učitaju pri otvaranje ove aplikacije, a grafički elementi na izborniku za postavke postavit će se tek onda kada korisnik otvori izbornik za postavke. Ukoliko se ovaj korak ne izvrši, primjerice, iako je korisnik odabrao kilometre na sat kao mjernu jedinicu brzine, kada korisnik zatvori izbornik, te ga onda ponovno otvori, u izborniku bi pisalo da se koriste milje na sat, no u igri bi se koristili kilometri na sat.

5.3.1 Promjena jezika

Kako bi se promijenio jezik, potrebno je imati sve potrebne podatke o svim string varijablama koje se koriste u igri prevedene u svaki jezik koji se može koristiti. Ti se podaci zapisuju u comma-separated values (skr. CSV) datoteku. U toj datoteci, u prvom se stupcu zapisuje identifikator za vrijednost, dok se u ostale stupce upisuju string vrijednosti riječi za taj identifikator. Svaki novi jezik je zada nozi stupac u CSV datoteci.

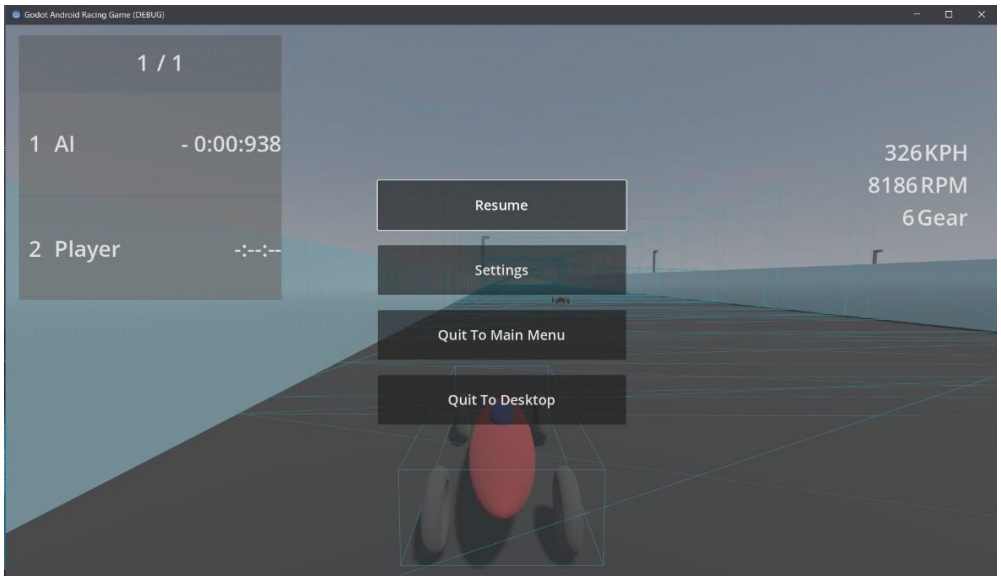
	A	B	C
1	Key	en_US	hr_HR
2	MAIN_MENU	Main Menu	Glavni izbornik
3	QUICK_PLAY_MENU	Quick Play Menu	Izbornik za brzu igru
4	SETTINGS	Settings	Postavke
5	QUIT	Quit	Ugasi igru
6	KPH	KPH	K/H
7	MPH	MPH	M/H
8	GEAR	Gear	Brzina
9	CURRENT_LAP	Current Lap	Trenutni krug

Slika 16. Primjer CSV datoteka za podacima za promjenu jezika

Nakon toga, potrebno je importirati CSV datoteku u Godot. Godot će od CSV datoteke automatski napraviti datoteke za svaki jezik, a datoteke imaju ekstenziju translation. Potrebno je otići u postavke Godot projekta, pod *localization* tab, te ovdje unijeti datoteke s ekstenzijom translation. Kada se jezik promijeni, potrebno je samo pozvati `set_locale` funkciju `TranslationServer` klase, te postaviti naziv novog jezika kao string parametar.

- **Izbornik koji se pojavljuje kada je igra pod pauzom**

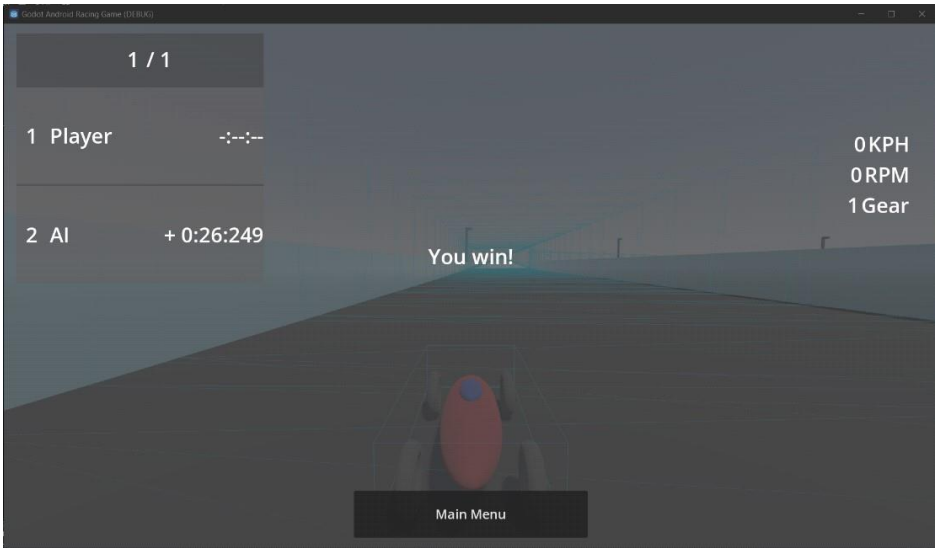
Izbornik koji se pojavljuje kada je igra pod pauzom može se aktivirati u bilo kojem trenutku dok korisnik vozi utrku. Ovaj izbornik omogućuje korisniku da za vrijeme utrke pristupi izborniku za postavke ili da napusti utrku. Ukoliko je ovaj izbornik aktivan, utrka i automobili koji se natječu biti će pod pauzom dok god korisnik ne obustavi pausu, ili dok korisnik ne izađe iz ove utrke. Kako bi korisnik jednostavnije primijetio kada je igra pod pauzom, pozadina ovog izbornika je zatamnjena.



Slika 17. Prikaz izbornika koji se pojavljuje kada je igra pod pauzom

5.5 Izbornik s rezultatima utrke

Ovaj izbornik pojavit će se kada igrač prijeđe ciljnu liniju na svojem zadnjem krugu, te na taj način dovrši utrku. Ovaj se izbornik sastoji od teksta i gumba za povratak na glavni izbornik. Tekst će ovisiti o rezultatu utrke. Ukoliko je igrač pobjednik, tekst će pisati "*You win!*", u suprotnom, tekst će biti "*You lost!*". Kako i kod prošlog izbornika, i ovaj izbornik ima zatamnjenu pozadinu kako bi se korisnik lakše snašao.



- Slika 18. Prikaz izbornika s rezultatima utrke

- ***HEADS-UP-DISPLAY***

Heads-up-display (skr. HUD) se sastoji od nekoliko dijelova: kontrole za zaslone osjetljivim na dodir, podaci o vozačevom automobili te podaci o utrci. Ovi elementi daju igraču bitne informacije o utrci, protivniku, stanju svoj automobila te omogućuju igraču upravljanje svojeg automobila na zaslonu osjetljivom na dodir.

6.1. Kontrole za zaslone osjetljivim na dodir

Glavna svrha kontrole za zaslone osjetljivim na dodir je omogućiti igraču upravljanje svog automobila. Zbog toga se na ekranu pojavljuju 3 elementa: volan, papučica kočnica i papučica gasa. Uz te komponente, kontrole za zaslone osjetljivim na dodir sadrže i mogućnost pogleda u retrovizor te mogućnost pauziranja utrke. Ovdje je još bitno za napomenuti da se ove kontrole mogu prikazati i na računalu, ukoliko je monitor računala osjetljiv na dodir, te je tada moguće koristiti igru s tipkovnicom, kontrolerom ili sa zaslonom osjetljivim na dodir.

6.2. Podaci o vozačevom automobilu

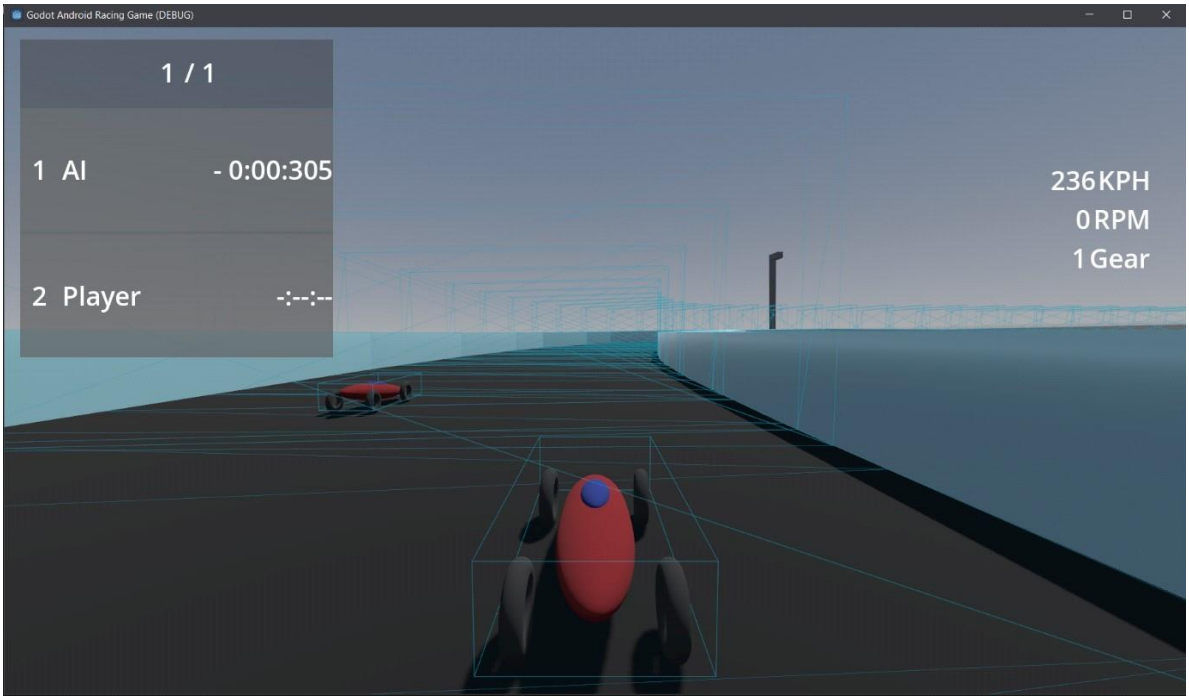
Podaci o vozačevom automobilu daju igraču informacije o 3 stvari: trenutnoj brzini automobila, trenutnom broju okretaja motora, te trenutnom stupnju prijenosa. Broj okretaja motora izražen je u okretaju po minuti, a trenutna brzina automobila može biti izražena u kilometrima na sat, ili miljama na sat, ovisno o postavci koju igrač može promijeniti.

6.3. Podaci o utrci

Podaci o utrci igraču daju informacije o 3 ključne stvari: broju kruga na kojem se nalazi igrač, broju krugova koje ova utrka sadrži, te razliku između igrača i auta kojeg kontrolira računalo. Ta razlika može biti izražena u vremenu ukoliko su oba auta na istom krugu, ili u razlici između krugova, ukoliko je jedan od automobila za čitav krug ispred ili iza drugoga. Ukoliko je razlika izražena u vremenu, tada je vrijeme formatirano na slijedeći način „minute :

sekunde : milisekunde“. U oba slučaja, ispred broja same razlike, pojavljuje se i predznak. Ukoliko je igrač iza auta kojeg kontrolira računalo, predznak je negativan, a ukoliko je igrač vodeći auto, predznak je pozitivan.

Kad se ovi sustavi spoje, rezultat je na sljedećoj slici. Da se napomenuti, da je ova verzija igre na računalu te da je za vrijeme igra korišten kontroler, stoga se na zaslonu ne nalaze kontrole za zaslone osjetljivim na dodir.



Slika 19. Prikaz dovršenog HUD sustava

Sa ovim sustavima igra je dovršena. Igrač za vrijeme vožnje ima sve potrebne podatke, kvalitetan model automobila, svijet, postavke, niz izbornika, te protivnika kojeg kontrolira računalo.

• ZAKLJUČAK

Završena igra sastoji se od svijeta i trkaćih staza, automobila, izbornika, heads-up-display elemenata i niza skripti. Igrač može odabrati između nekoliko trkaćih staza, odabrati koji će auto voziti te postaviti broj krugova utrke. U utrci igrač će se natjecati s računalnim protivnik te će se za vrijeme utrke pokazati razlika između igrača i protivniku u vremenskom obliku ili u broju krugova između njih. Također, igrač ima nekoliko postavki koje može promijeniti u bilo kojem trenutku, a postavke će se spremiti te učitati, jednom kada se igra ponovno otvori. Automobili su relativno realistični. Imaju složen sustav prijenosa snage, te prilično dobar kod za skretanje baziran na Ackermannovom modelu. Računalni protivnik je poprilično jednostavan, no dovoljno je dobar da bi se zadovoljio cilj rada. Igra je funkcionalna. Svaka trkaća staza neki svoj teži dio. Od tri staze, dvije su relativno široke, no igraču mogu biti teške zbog promjene u visini koji može proizvesti gubitak kontrole nat automobil. Treća staze je poprilično ravna, no zbog svoje užine zahtjeva visoke sposobnosti kontrole automobila s velikom preciznošću na velikim brzinama.

Ovaj projekt i dalje razvijam, te sam odlučio promijeniti koncept računalnog protivnika kako bi isti postao brži i realističniji. Koncept kojeg planiram mogao bi biti reaktivan, odnosno, mogao bi se braniti i napadati protivnike poput stvarnih trkaćih vozača. Ovaj bi koncept tada zaista davao inpute skriptama automobila, ovisno o dijelu staze na kojem se nalazi te drugim relevantnim podacima. Za budući napredak ovog projekta, planiram dodati i tip igre u kojem se igrač bori za prvenstvo kroz niz utrka na većem broju staza, taka da igrač ima mogućnost igrati brzu igru, ili sezonsku. Komponente automobila već sada funkcioniraju drugačije ovisno o njihovoj temperaturi, no u ovoj je izvedbi ta temperatura uvijek stalna te nudi najbolje performanse. U budućnosti, cilj mi se napraviti sustav koji će omogućiti promjenu temperature na realističan način, tako da se komponente mogu pregrijati, što bi rezultiralo lošijim performansama.

LITERATURA

https://docs.godotengine.org/en/stable/about/introduction.html

https://thatonegamedev.com/cpp/what-are-shaders/

Adrian Newey. *How To Build A Car*

https://bimmertips.com/dogleg-gearbox-advantages/

https://www. HYPERLINK "https://www.racecar-engineering.com/articles/tech-explained-ackermann-steering-geometry/"racecar-engineering.com/articles/tech-explained-ackermann-steering-geometry/

https://www.formula1.com/en/latest/article/f1-explains-how-f1-lap-times-are-recorded-in-a-world-of-hundredths-and.5qzycCnnJg5zXJXDrWXJpd

https://phrase.com/blog/posts/godot-game-local HYPERLINK "https://phrase.com/blog/posts/godot-game-localization/"ization/

https://cyberglads.com/making-cyberglads-3-head-up-display.html

PRILOZI

Popis slika