Izrada videoigre utrkivanja u programskom alatu Unity

Karaula, Matija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Karaula

IZRADA VIDEOIGRE UTRKIVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Karaula

Matični broj: 0016148405

Studij: IPS – Umreženi sustavi i računalne igre

IZRADA VIDEOIGRE UTRKIVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, rujan 2023.

Matija Karaula

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad pruža pregled u Unity kao programski alat, razloge njegove popularnosti te neke od najpopularnijh primjera igara razvijenih u Unityju. Nakon opisa istih igara, rad će prikazati najistaknutije predstavnike žanra trkaćih igara. Srž i smisao ovog rada zapravo je prikaz procesa izrade jednostavne trkaće igre u Unityju. Igra je zamišljena kao time trial igra, što znači da je cilj postaviti što brži krug. Kroz sam rad bit će prikazana kompleksnost izrade realističnog sustava fizike te način programiranja logike same igre za ovakve trkaće igre. Rezultat, odnosno proizvod na kraju ovog procesa je jednostavan prototip trkaće igre s glavnim izbornikom, postavkama, menijem za pauzu i ljestvicom, a sama igra sadrži jedan model automobila te dvije različite staze između kojih korisnik može odabrati.

Ključne riječi: Unity; videoigra; igre utrkivanja; game development; programiranje; prototip

Sadržaj

1. Uvod

U ovom radu bit će opisan razvoj trkaće igre koristeći programski alat Unity. Unity je najpopularniji *game engine*, a uz to je besplatan za korištenje, čime pruža odlične temelje za game developere koji tek počinju učiti o razvoju igara. Međutim, Unity nije namijenjen samo početnicima, već je vrlo pogodan i za najiskusnije i najzahtjevnije korisnike te se radi o alatu koji je izuzetno kvalitetan i svestran.

Igre utrkivanja čine jedan od najpopularnijih žanrova videoigara, a njihov razvoj počinje gotovo na samom početku razvoja igara općenito. Igre utrkivanja poznate su po svojim vjernim zajednicama koje se oko njih formiraju i odaju im počast godinama. Dobar takav primjer je igra Auto Modellista, koja je izdana 2002., a zbog svog grafičkog stila i danas izgleda moderno i dobro. I danas igra ima zajednicu koja je okupljena oko nje i igra igru 21 godinu nakon što je izdana. Drugi primjer ovakve igre je Richard Burns Rally. Igra je izdana još 2004. za PlayStation 2 i Xbox, a nekoliko mjeseci kasnije i za Windows. Igra je i danas vrlo cijenjena i štovana u zajednici. Mnogi igrači od nje su odustali jer im je bila preteška, ali upravo zato igra i danas ima odanu zajednicu. Smatra se da ova igra ima najrealističniju fiziku automobila dosad, a također je cijenjena i zbog realističnosti modela staza. Upravo ova igra dokazuje da u ovom žanru grafička kvaliteta nije toliko presudna, nego je najbitnije kako je implementirano ponašanje automobila, a Richard Burns Rally vjerojatno je najnaprednija fizika automobila u usporedbi s bilo kojom igrom.

Smisao ovog rada je predstaviti programski alat Unity kao rješenje za razvoj igara, prezentirati predstavnike žanra trkaćih igara koji su obilježili zadnjih dvadesetak godina računalnih igara te prikazivanje procesa izrade igre u programskom alatu Unity na primjeru igre utrkivanja.

2. Metode i tehnike rada

Sama igra bit izrađena je u programskom alatu Unity. 3D modeli automobila, staze, ukrasa i sl. preuzeti su s Unity Asset Store trgovine. Vizual za glavni izbornik generiran je pomoću AI alata DALL-E 2.

3. Razrada teme

Izrada videoigara izuzetno je kompleksna grana informatike i informacijskih znanosti. Razvoj videoigre poznat je kao jako mukotrpan, kompleksan i detaljan posao. Za samostalnu izradu videoigre, čovjek mora imati znanja iz područja dizajna, zvuka, kinematografije, matematike, fizike, literarnog izražavanja (u smislu pričanja priče) te mnogih drugih područja. Koliko je kompleksan posao izrade videoigre govori i činjenica da je poznatu igru Grand Theft Auto V razvijalo više od tisuću ljudi.[1] Unatoč tome, danas je ipak moguće da pojedinac potpuno sam napravi svoju videoigru, čak iako nema sva potrebna znanja. Najbolji primjeri takvih igara vjerojatno su Minecraft, Spelunky i Undertale.[2] Izrada videoigara znatno je olakšana uvođenjem programskih alata za izradu igara, poput Unityja. Ovakva vrsta programskog alata naziva se i *game engine*. *Game enginei* znatno olakšavaju izradu videoigara, a najpopularniji, javno dostupni primjeri su Unity, Unreal te Godot. Zbog ovakvih programskih alata, manji timovi i pojedinci puno lakše mogu razviti videoigru, no oni ne služe samo njima. *Game engine* koriste i najveće kompanije za izradu najzahtjevnijih igara, baš zato jer znatno ubrzavaju i pojednostavljuju proces. *Game enginei* su toliko korisni jer u sebi sadrže već izrađenu programsku logiku za pojave koje su često potrebne u igrama. Na primjer, gravitaciju u programskom alatu Unity programer ne mora sam programirati i emulirati, već je ona ugrađena u sami *game engine*, u sklopu komponente RigidBody. Developer sebi tada samo treba postaviti podatke poput jačine gravitacije te mase tijela po svojoj potrebi i želji. Na ovaj način znatno je smanjeno vrijeme potrebno za razvoj videoigre i upravo zbog toga bilo tko može puno lakše razviti svoju videoigru.

3.1. Programski alat Unity

Kao što je i ranije navedeno, programski alat Unity vrsta je *game enginea*. Prema izvješćima, u 2022. godini najpopularniji *game engine* bio je Unity, gdje čak 38% game developera koji koriste *game engine* koriste Unity kao svoj primarni *game engine*. [3]

Činjenica da je Unity najpopularniji programski alat za izradu videoigara ne bi trebala izazivati veliko čuđenje. Prvi razlog tomu je taj da je Unity objavljen još 2005. g. Dakle, prije 18 godina. U tih 18 godina Unity je imao dovoljno vremena razviti svoj brend te je stalnim nadograđivanjem uvijek mogao, a može i danas, pratiti najnovije trendove i tehnologije, odnosno korisnicima daje priliku korištenja najnovijih tehnologija. Programski jezik koji Unity koristi je C#. C# kao programski jezik lakši je za ljude koji nemaju puno iskustva s programiranjem, za razliku od C++ programskog jezika koji se najčešće koristi u razvoju

igara i u drugim *game engineima*. C++ kao programski jezik niže je razine apstrakcije od C#-a pa je zbog toga C++ pogodniji za izradu igara, jer će programski kod biti bolje optimiziran. Samim time, Unity ne može prevladavati kao *game engine* u razvoju velikih i kompleksnijih PC igara, već u tom području značajniju ulogu ima Unreal engine, koji podržava C++ jezik. Budući da Unity zbog programskog jezika više apstrakcije nije toliko prilagođen za kompleksnije igre u usporedbi s Unrealom u području PC igara, Unity je postao bolje prilagođen izradi mobilnih igara. Mobilne igre trenutno su najpopularnije igre u svijetu, a Unity je jedan od najboljih alata za izradu mobilnih igara, prije svega jer je besplatan i korisnicima daje čak i mogućnost da prodaju svoju igru, a Unity alat koriste besplatno (dok god je prihod od prodanih igara manji od cifre zadane od strane Unityja). Također, Unity nudi i mogućnost uvođenja reklama u mobilne igre i korisnicima je taj postupak maksimalno olakšan, budući da Unity ima razvijenu uslugu Unity Ads te se na taj način vlasniku igre isplaćuje novac za svaku prikazanu reklamu. [4] Još jedna od prednosti Unityja je mogućnost razvoja igara za više platformi. Unity nudi podršku za sve najpopularnije platforme pa tako korisnik u Unityju može izrađivati igre za PC, mobilne uređaje, PlayStation, Linux i druge platforme. Zbog ovih karakteristika Unity se nameće kao najpopularniji *game engine*, iako mu Unreal Engine zbog svojih nedavnih poboljšanja počinje predstavljati ozbiljnu konkurenciju.

Unity je besplatan programski alat s mogućnošću pretplate na bolje razine usluge.[4] Korisnici mogu kreirati svoje projekte, a ono što im uvelike pomaže jest to da mogu izabrati neki od ponuđenih predložaka, na primjer 2D, 3D, Mobile, First-Person, Third-Person i slično. Unity na ovaj način pomaže korisniku jer mu daje prilagođene postavke i pakete za odabrani tip projekta. Naravno, odabir određenog predloška ne ograničava korisnika u onome što može napraviti u svom projektu. Na primjer, ako korisnik odabere predložak za 2D, i dalje može kreirati 3D igricu unutar tog projekta. Jedina razlika bila bi u tomu što bi korisnik dobio početne postavke i pakete bolje prilagođene 2D-u.

3.1.1. Primjeri igara razvijenih u Unityju

Najbolji dokaz uspješnosti Unityja kao *game enginea* upravo su proizvodi, odnosno igre koje su kreirane i razvijene u Unityju. Uspješni primjeri postoje gotovo za svaki žanr što je dodatni dokaz Unityjeve verzatilnosti.

3.1.1.1. Pokémon Go

Jedan od najboljih primjera igara razvijenih u Unityju je Pokémon Go. Pokémon Go revolucionarna je igra koja je potpuno promijenila konceptualni, do tada uobičajeni način igranja igara. Svojom inovativnošću, motivirala je igrače da uz užitak igranja igara ostanu i fizički aktivni. Ono što je inovativno u samoj igri jest korištenje AR sučelja koje do tada nije bilo uobičajeno. Cilj igre prije svega je uhvatiti Pokémone koji se *spawnaju* na različitim lokacijama na mapi, a mapa je zapravo cijeli svijet. Način kretanja po mapi je fizičko kretanje u stvarnom svijetu. Igra koristi lokaciju u stvarnosti kako bi igrača smjestila na mapu. Dakle, igrač treba sam otići do mjesta na kojem se nalazi Pokémon. Nakon toga, igrač treba upaliti kameru unutar igrice i pokrenuti AR sučelje te na taj način naći traženog Pokémona. Nakon toga treba ga i uloviti bacanjem Poké-lopte na njega.[5] Igrači su vrlo uzbuđeno i oduševljeno reagirali na ovakvu mehaniku igranja jer je bila vrlo bliska onome kako je to zamišljeno i u samoj seriji. Pokémon Go je zbog svoje inovativnosti i kreativnosti postao jedna od najpoznatijih igara u povijesti.

Slika 1: Princip rada Pokémon Go igre (izvor: [6])

3.1.1.2. Among Us

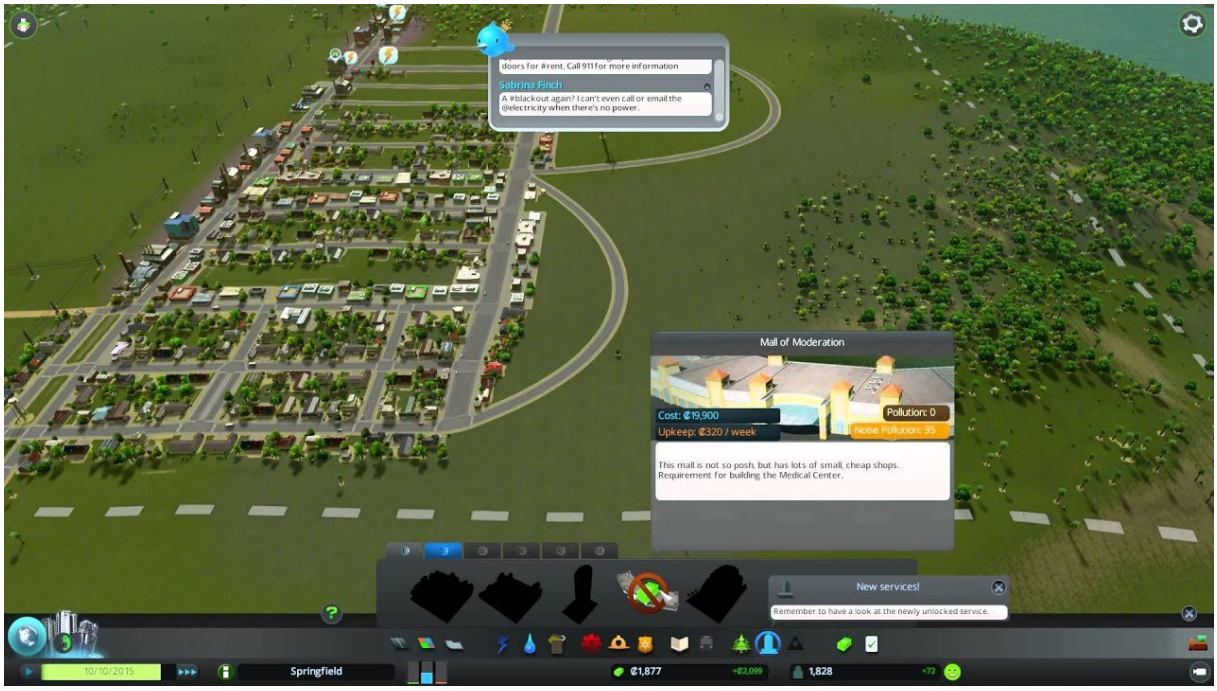
Among Us je još jedan dobar primjer igre razvijene u Unityju. Ova igra izdana je 2018., a sve donedavno bila je ekstremno popularna. U jednom trenutku imala je čak 248 milijuna aktivnih igrača. [7] Among Us je multiplayer društvena igra s malim brojem igrača u partiji. Cilj igre je izvršavati zadatke. Igrači su podijeljeni u dvije kategorije, članove posade i uljeze. Uljezi trebaju ubiti članove posade, a članovi posade trebaju izvršavati svoje zadatke te otkriti uljeze i izbaciti ih iz igre. [8] Igra je izdana na Android, iOS, Windows, PlayStation 4, PlayStation 5, Xbox One, Xbox Series X/S te Nintendo Switch, a 2022. godine izdana je i VR verzija. To govori o verzatilnosti Unityja kao *game enginea* kada su u pitanju platforme za koje se igra može izdati.

Slika 2: Prizor iz igre Among Us (izvor: [9])

3.1.1.3. Cities: Skylines

Cities: Skylines popularna je igra iz 2015. razvijena od strane Colossal Ordera i Paradox Interacitvea. Igra je zapravo simulacija izgradnje grada te je glavna konkurencija SimCityju, razvijenom od strane EA-a. Igra je uspjela postići solidne performanse pomoću Unity programskog alata. Naime, igra bez problema može prikazati i raditi s velikim geografskim područjima koja imaju stotinjak zgrada i čak do milijun stanovnika, bez kompromisa po pitanju grafičke kvalitete. Igra je izdana za PlayStation 4 i 5, Xbox One i

Xbox Series X/S, Windows, MacOS, Nintendo Switch, Linux, Xbox Cloud Gaming te Google Stadia platforme te još jednom pokazuje kako je Unity dobro prilagođen za velik broj platformi te za velik broj žanrova.



Slika 3: Prizor iz igre Cities: Skylines (izvor: [10])

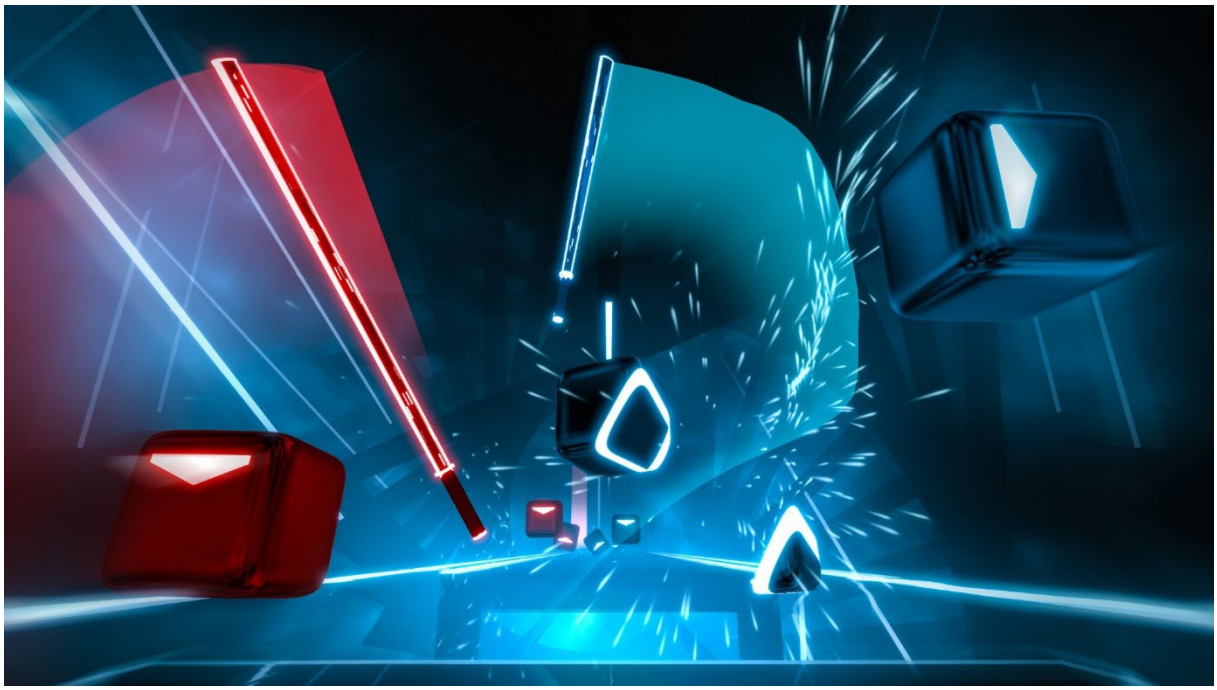
3.1.1.4. Rust

Rust je vrlo poznata i danas popularna shooter igra. Radi se o multiplayer igri preživljavanja gdje igrači moraju preživjeti koristeći materijale koje nađu u prirodi ili ukradu od drugih igrača. Igrači u igri grade svoje nastambe, ali i tamo im prijeti opasnost od drugih igrača. Igra ima i sustav gladi pa igrači moraju tražiti hranu u prirodi kako bi uopće preživjeli.[11] Igra je razvijena od strane Facepunch Studiosa i Double Elevena te je izdana 2018. godine za macOS i Windows te 2021. za PlayStation 4 i Xbox One.

Slika 4: Prizor iz igre Rust (izvor: [12])

3.1.1.5. Beat Saber

Beat Saber jedna je od najpoznatijih ritamskih igara. Radi se o VR igri koja je izdana 2019. godine za PlayStation 4 i PlayStation 5, Windows i Meta Quest, a od izdanja do danas stekla je veliku popularnost. Igra ima atraktivne vizuale i zanimljive mehanike igranja. Naime, u samoj igri, cilj je „presjeći“ kocke koje prema igraču lete u ritmu. Međutim, nije ih cilj samo „presjeći“, nego je to potrebno napraviti s određene strane i u određenom smjeru. Igra je i dalje ogroman uspjeh, čemu svjedoče i recenzije na Metacriticu, gdje je dobila rezultat 93/100 od strane kritičara te 8.5/10 od strane korisnika. [13]



Slika 5: Prizor iz igre Beat Saber (izvor: [14])

Sve gore navedene igre razvijene su koristeći Unity te služe kao izvrstan primjer onoga što se u Unityju može postići. Dakle, dobro je prilagođen za gotovo svaki žanr igara i vrlo je verzatilan. Iz primjera Cities: Skylinesa može se zaključiti da mu skalabilnost ne predstavlja veliki problem, a iz primjera Rusta i Among Usa očito je da je dobro prilagođen i za multiplayer način igranja. Također, vrlo je pogodan za izradu VR i AR igara, što je evidentno iz primjera Beat Sabera i Pokémon Go-a. Drugi programski alati poput RPG Makera možda su bolje prilagođeni za određen žanr ili određenu nišu u igrama, ali ono što Unity nudi jest kompatibilnost (igru je moguće izdati za gotovo sve platforme), skalabilnost (Unity nema problema s igrama velike mjere), verzatilnost (prilagođenost za gotovo sve žanrove) te tržišna pristupačnost (besplatan je).

3.1.2. Mogućnosti Unityja

Unity je tako dobro prilagođen svim vrstama projekata upravo zbog bogatog skupa mogućnosti i inovacija koje nudi. Na primjer, korisnicima nudi i nekoliko vrsta *pipelinea* za renderiranje: URP (*Universal Render Pipeline)* te HDRP (*High Definition Render Pipeline)*, a uz njih postoji i već ugrađeni *legacy pipeline*. Svrha *pipelinea* je transformacija 3D prostora, odnosno scene igre u 2D prikaz na ekranu. Zbog softverskih i hardverskih razlika između mobilnih uređaja i računala, gotovo je nemoguće razviti jedan univerzalan *pipeline* za sve

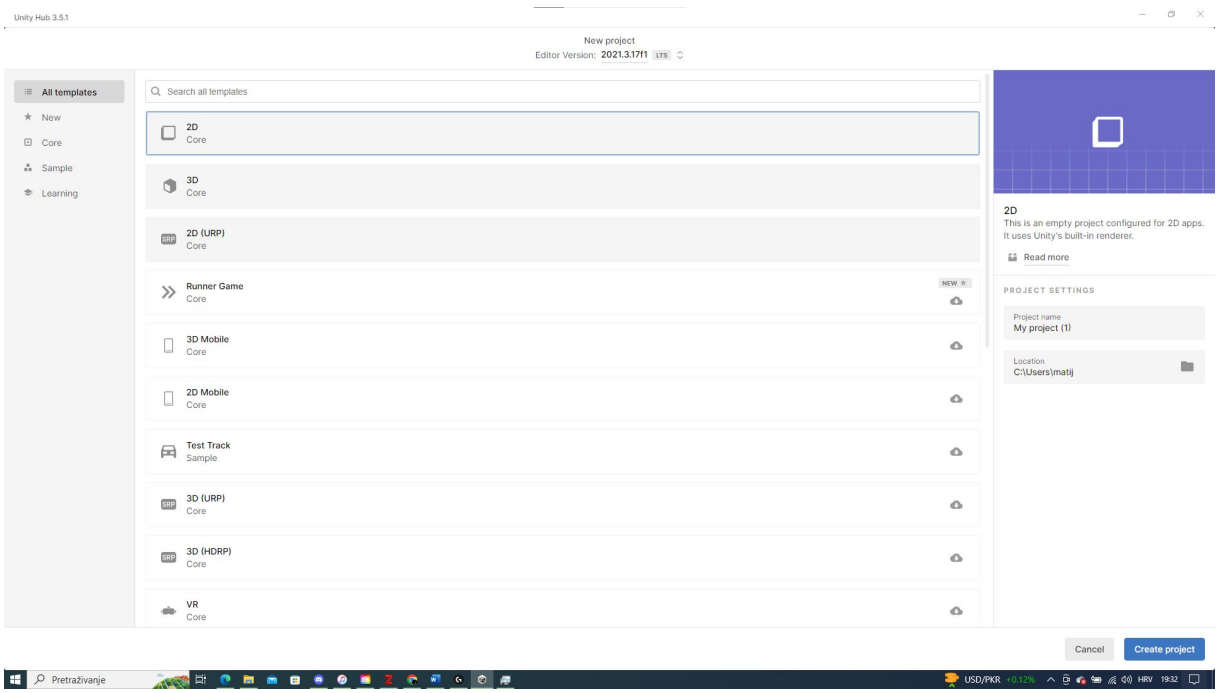
vrste uređaja, a usput osigurati optimalne performanse na svima. Stoga je Unity razvio tri različita. Unityjev ugrađeni *Render Pipeline* je zapravo *pipeline* koji se od početka koristi. Njegova prednost je ta što je svestran i može se koristiti za sve vrste projekata, ali ima poprilično ograničene mogućnosti prilagodbe i kustomizacije. *Universal Render Pipeline (URP)* je *pipeline* kojem su naglasak performanse. Iako koristi nove i kvalitetne metode renderiranja, poput dinamičkog osvjetljenja i naknadnog obrađivanja, URP je svejedno prilagođen davanju najboljih performansi, što ga čini pogodnim za mobilne i VR igre. *High Definition Render Pipeline (HDRP)* pak ima potpuno drugi prioritet. HDRP je osmišljen i kreiran za renderiranje što realističnije grafike pomoću ray-tracinga i volumetričnog osvjetljenja. Ovaj *pipeline* osmišljen je s ciljem prikazivanja najbolje moguće grafike, po cijenu performanse. Za pokretanje projekta koji koristi HDRP potrebno je imati dobar hardver. HDRP najbolje je prilagođen igrama za računala i konzole. Valja naglasiti da za korisnike Unityja ipak postoji još jedna opcija po pitanju izbora *pipelinea*. Naime, korisnik može kreirati sam svoj *pipeline* i koristiti ga u svom Unity projektu koristeći Unityjev Scriptable Render Pipeline API.[15]

Unity također nudi mogućnosti vizualnog programiranja. Vizualno programiranje odlično je za korisnike koji možda nisu toliko vješti u programiranju ili ne poznaju toliko dobro C# programski jezik, a imaju ideju koju žele realizirati. Također, ovo je korisna funkcionalnost za rad u timu s umjetnicima i dizajnerima jer se na taj način olakšava međusobna suradnja.[16]

Jedna od vrlo bitnih stvari i jedan od većih razloga zašto je Unity tako popularan je Unity Asset Store. Unity Asset Store je velika kolekcija, odnosno trgovina Asseta kreiranih od strane korisnika koji se mogu koristiti u Unity projektima. Korisnici mogu izraditi 3D dizajne likova, mapa ili pak mogu izraditi vlastite funkcionalnosti i kodove koji se mogu ugraditi u druge Unity projekte. Na samoj trgovini nalazi se mnoštvo besplatnih *asseta* pa svaki korisnik može za sebe pronaći ono što mu treba, čak i ako nema velik proračun. Uvoz *asseta* u Unity projekt iznimno je lagan. Sve što korisnik treba napraviti je prijaviti se u Asset Store svojim Unity ID-jem, odabrati paket koji želi i kliknuti Add to My Assets. Nakon toga, u Unity projektu treba otići na Packet Manager, odabrati svoj *asset*, kliknuti Download i nakon toga Import. Nakon toga, željeni *asset* uvezen je u Unity projekt. Ovo je izuzetno korisna funkcionalnost za sve korisnike, jer oni koji nemaju vremena ili znanja za, na primjer, dizajn likova, mogu samo otići na stranice trgovine i pronaći nešto za sebe, a drugi korisnici mogu imati neku korist od toga kroz prodavanje *asseta* u trgovini. Na ovaj način gradi se veliko i održivo društvo u Unityju koje svakog dana samo sebe čini boljim.

3.2. Kreiranje novog projekta u Unityju

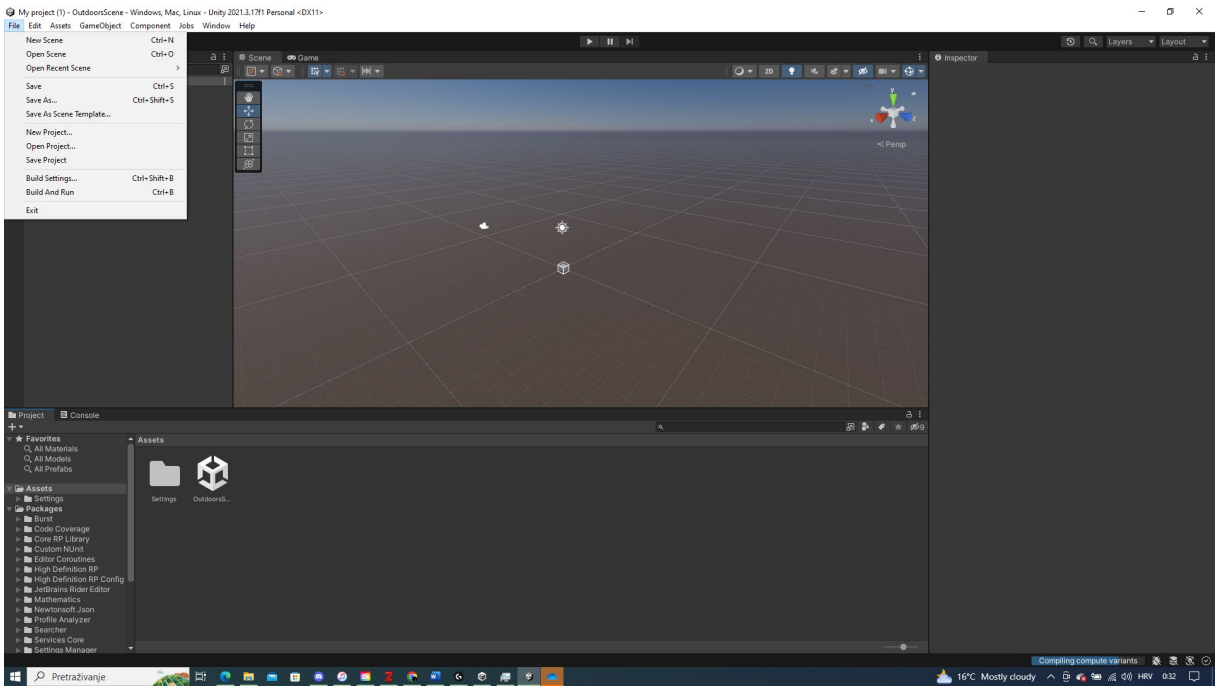
Kreiranje novog projekta u Unityju i početak rada izuzetno je jednostavan i brz proces. Za početak potrebno je preuzeti Unity Hub, instalirati ga na računalo te pokrenuti. Potom je potrebno prijaviti se svojim Unity korisničkim imenom. Ako ga korisnik nema, treba se prvo registrirati. Nakon toga sučelje izgleda ovako:



Slika 6: Početni zaslon Unity Huba

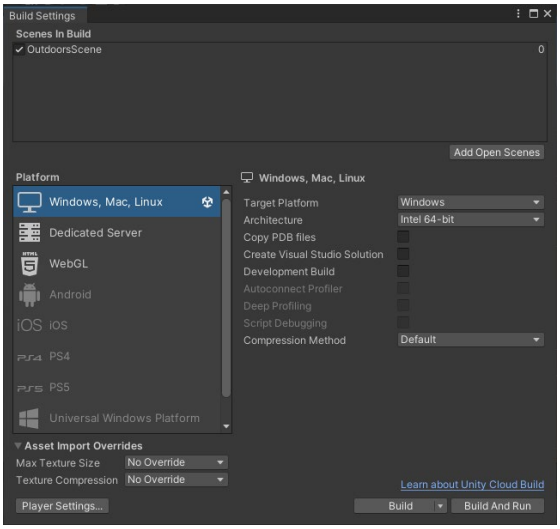
Na sučelju je potrebno izabrati željeni predložak. Predlošci omogućavaju kreiranje projekta s optimalnim postavkama za tu vrstu predloška. Nakon što je odabrao željeni predložak, korisnik ga još treba imenovati te odrediti gdje će ga spremiti na računalu. Nakon toga treba kliknuti na gumb Create project i projekt je kreiran i spreman za rad.

Nakon što je projekt gotov, vrlo ga je lagano kompajlirati i pokrenuti. Sve što je potrebno je na Build Settings u izborniku:



Slika 7: Početni zaslon Unityja te padajući izbornik s *Build Settings* opcijom

Nakon toga potrebno je prilagoditi željene postavke:



Slika 8: *Build Settings* izbornik

Nakon što je korisnik prilagodio postavke, potrebno je samo pritisnuti opciju Build ili Build and Run i nakon toga će moći pokrenuti svoju igru lokalno.

3.3. Predstavnici žanra trkaćih igara

Kroz godine, gotovo od samog početka videoigara trkaće igre bile su jedne od najpopularnijih. No, u zadnjih dvadesetak godina trkaće igre su se strmovito brzo počele razvijati i ubrzo je konkurencija postala ogromna. 2022. godine veličina tržišta trkaćih igara bila je oko 6.2 milijarde američkih dolara. [17] Trkaće igre razvijene su za gotovo sve platforme, a postoji puno zanimljivih i inovativnih primjera. 4 su glavna pod-žanra u trkaćih igara: arkadne, simulacijske, karting, i futurističke trkaće igre.

3.3.1. Arkadne trkaće igre

Arkadne trkaće igre jedna je od najpopularnijih grana trkaćih igara, primarno zbog svoje jednostavnosti i fokusa na ugodnost i zabavu. Arkadne igre inače imaju razvijenu fiziku automobila koja je polu-realistična. Automobili će najčešće imati više trenja i lakše će skretati nego što je to uobičajeno. Mehanička šteta najčešće je minimalna, a u mnogim slučajevima uopće ne postoji. Cilj ovih igara najčešće je štovanje automobilske i trkaće kulture na različite načine. Neke igre fokusirane su na *tuniranje* automobila i njihovo nadograđivanje za uličnе utrke, dok druge igre dodaju neke nove mehanike na klasično utrkivanje i na taj način odstupaju od realizma i dodaju zabavi. Primjer takve igre je Burnout, gdje je cilj naravno pobijediti u utrci, ali to se može postići uništavanjem suparničkih automobila na razne načine. Arkadne igre često su najpopularnije trkaće igre, upravo zbog jednostavne fizike i naglaska na zabavu. Najistaknutiji predstavnici arkadnih trkaćih igara su Need for Speed, Trackmania, Forza Horizon, Asphalt, Burnout i Sega Rally.

3.3.1.1. Need for Speed

Need for Speed jedna je od najpoznatijih franšiza igara u svijetu. Od 1994. godine sve do danas izdano je čak 25 igara. Need for Speed je arkadna trkaća igra koja koristi stvarne, licencirane modele automobila i mape koje su najčšće inspirirane najvećim svjetskim metropolama. Primarni fokus igre je zapravo ulično utrkivanje u *tuniranim* automobilima, iako su neke igre napravljene s fokusom na simulacijsko utrkivanje na pravim stazama. Svaka igra ima priču osmišljenu na svoj način, ne postoji jedan predložak priče koji se minimalno mijenja kroz svaku igru, nego je priča originalna za svaku igru. Igra je postala vrlo popularna jer je dobro uspjela obuhvatiti sve bitne elemente uličnog

utrkivanja: poznati sportski automobili, popularni svjetski gradovi, kustomizacija i nadograđivanje automobila.

Vjerojatno najpoznatija Need for Speed igra je upravo Need for Speed: Most Wanted. Need for Speed: Most Wanted deveta je igra u nizu u Need for Speed franšizi. Igra je izdana 2005. godine i razvijena je od strane EA-a, kao i sve ostale Need for Speed igre. Igra je izdana za PlayStation 2, Xbox, GameCube, Nintendo DS, Microsoft Windows, Game Boy Advance, Xbox 360 te PlayStation Portable. Tema igre je utrkivanje u ilegalnim uličnim utrkama, a vozači su se natjecali koji će vozač biti najtraženiji od strane policije. Igrač se mogao popeti na ljestvici vožnjom određenog broja utrka te nakon toga utrkivanje, s rivalom koji je iznad igrača na listi. Na primjer, utrkivanje protiv rivala koji je 6. na listi kako bi igrač zauzeo 6. mjesto na listi. Naravno, u igri postoji i mehanika policijske potjere u kojoj policija pokušava uhvatiti igrača, a cilj igrača je pobjeći od policije, koja postavlja blokove na cesti i bodlje za bušenje guma. Ukratko, igra utjelovljuje klasični stereotip ilegalnog uličnog utrkivanja i to radi na način koji je zabavan i nije naporan. Need for Speed: Most Wanted možda je najpopularniji Need for Speed jer su svi „sastojci“ na mjestu: zanimljiva priča, dizajn koji prati priču, atraktivna fizika, zanimljiv izbor automobila te puno sitnih elemenata koji čine cijelu igru boljom.

S tehničke strane, Need for Speed je dobar primjer arkadne igre zbog fizike automobila. Naime, u većini Need for Speed igara, mehanička šteta gotovo i ne postoji, osim bušenja guma. Dakle, gotovo je nemoguće oštetiti auto na način da se to osjeti u vožnji. Fizika automobila nije potpuno realistična i puno je tolerantnija. Automobili puno lakše mogu skretati, ne mogu prejako zakočiti i slično. Sve ranije navedeno su karakteristike arkadne igre.

Slika 9: Prizor iz igre Need for Speed Most Wanted (izvor: [18])

3.3.1.2. Trackmania

Trackmania je igra koja je stekla veliku popularnost posljednjih godina. Po svojoj prirodi, to je igra koja se suštinski gotovo u svemu razlikuje od Need for Speed igara, ali i jedna i druga spadaju u arkadne igre zbog svoje fizike. Trackmania također ima tolerantniju fiziku automobila bez mehaničkih oštećenja i pojednostavljeno ponašanje automobila.

Trackmania je, kao i Need for Speed, serija igara, Prva igra izašla je još 2003. godine, a posljednja do sada 2020. Trackmania je jedna od najdražih igara za tzv. s*peedrunnere*, kojima je u cilju postaviti što brže vrijeme na svim mapama i oko ove igre razvila se vrlo velika i aktivna zajednica.

Igra je koncipirana vrlo natjecateljski i vrlo je pogodna za to da postane e-sport svoje vrste. Postoje različiti tipovi automobila, ali oni se ne mogu nadograđivati i na takav način se ne može postići nikakva prednost. Tijekom utrka, nemoguće se sudariti s drugim igračem jer su svi drugi igrači implementirani kao *duhovi* pa je jedini način za pobijediti zapravo naći najbolje linije i najbrže odvoziti stazu. Upravo zbog ovog se razvila takva zajednica koja je razvila pravu ljubav prema ovoj igri, jer mogu na pravi način, bez prečaca prikazati svoje vještine.

Također, jedna od vrlo bitnih mehanika za ovu igru je izgradnja vlastite staze. Igračima je dana mogućnost izrade svojih staza, a to je također imalo velik utjecaj na razvoj

vjerne i odane zajednice. Igrači svoje dizajne mogu javno objaviti, a na njima svoja vremena mogu postavljati igrači iz cijelog svijeta.

Slika 10: Prizor iz igre Trackmania (izvor: [19])

3.3.2. Simulacijske trkaće igre

Simulacijske trkaće igre uvelike se razlikuju od arkadnih igara. Naime, simulacijske trkaće igre imaju samo jedan glavni cilj: što bolje i što preciznije prikazati stvarnost te na najrealističniji mogući način simulirati iskustvo utrkivanja. Ovdje naglasak nije na uličnim utrkama, nego na stvarnim prvenstvima i klasama automobila koje se natječu u stvarnosti. Ovdje fizika automobila nije osmišljena da bi bila ugodna i zabavna, nego stvarna, a iz tog razloga su takve igre često istaknute kao izrazito teške. U simulacijskim igrama naglasak nije na grafičkoj ljepoti i na priči, već isključivo na iskustvu utrkivanja, iako postoje simulacijske igre koje imaju i dobro razvijenu priču, a većina ih danas ima i grafiku na zavidnoj razini, u ovim igrama priča nije presudna i smatra se dodatkom. Najistaknutiji predstavnici simulacijskih trkaćih igara su Gran Turismo, Forza Motorsport, Assetto Corsa, Automobilista 2 te F1. Najčešći problemi kod simulacijskih igara za nove korisnike su česta pretjerana kompliciranost igara (previše postavki koje su često samo nabacane na sučelje u kojem se teško snaći) te potreba za skupom hardverskom opremom. Pravo iskustvo igranja

simulacijskih igara može se doživjeti tek sa skupom opremom, poput *force feedback* upravljača.

3.3.2.1. Gran Turismo

Jedna od općenito najpoznatijih igara u povijesti definitivno je Gran Turismo. Gran Turismo je japanska franšiza trkaćih igara i jedan od predstavnika i pionira simulacijskih trkaćih igara. Prva igra serijala izdana je 1997. od strane Sonyja te Polyphonyja, a najnovija igra izdana je 2022. godine, Gran Turismo 7. Gran Turismo vrlo je prepoznatljiva igra zbog svojih istaknutih i jedinstvenih sučelja te zanimljivog izbora glazbe. Na primjer, Gran Turismo u svojoj playlisti sadrži veliku količinu miksane i editirane klasične glazbe.

Gran Turismo sadrži veliku kolekciju licenciranih automobila, od onih svakodnevnih i uobičajenih, poput Forda Focusa, sve do GT automobila namijenjenim utrkama, a igra sadrži i zanimljive koncepte. Osim licenciranih automobila, u igri su dostupni i izmišljeni modeli licenciranih marki. Također, igra sadrži i veliku kolekciju staza koje su spoj pravih i izmišljenih.

Igra nudi puno mogućnosti uređivanja automobila i vođenja brige o njima. Na primjer, u najnovijem Gran Turismu 7, igrač može mijenjati boju automobila, kreirati vlastiti dizajn za taj model, može kupiti nove dijelove te urediti tijelo svog automobila, a može voditi i brigu o automobilu tako da ga opere, zamijeni mu ulje i slično.

Gran Turismo nudi i puno mogućnosti online igre pa tako postoje dnevne utrke, *time trial* događaji, u kojima je cilj postaviti što bolje vrijeme u danom automobilu i na danoj stazi i biti što više na globalnoj ljestvici te postoje i prvenstva u kojima se igrači utrkuju tijekom sezone.

U samoj igri, implementirana je vrlo realistična fizika automobila, koja uistinu dolazi do izražaja korištenjem *force feedback* upravljača. Vožnjom automobila moguće je prejako zakočiti i proklizati. Gume se troše vožnjom i tijekom utrke potrebno ih je mijenjati pa je potrebno i upravljati strategijom. Automobili se mogu oštetiti, a tijekom utrke se troši i gorivo koje također utječe na samu strategiju u utrci. Ovakve igre u usporedbi s arkadnim igrama imaju dodatne slojeve kompleksnosti koje arkadne igre nemaju. U arkadnim igrama nije moguće pregrijati gume ili potrošiti gorivo, za razliku od simulacijskih igara.

Slika 11: Prizor iz igre Gran Turismo (izvor: [20])

3.3.3. Karting igre

Karting trkaće igre se razlikuju od arkadnih i simulacijskih igara i puno je lakše uvidjeti razliku između ovog i ostalih pod-žanrova. Prva velika razlika je u modelima automobila. U karting igricama, automobili su, naravno, dizajnirani po uzoru na go-kartove. Također, jedna velika razlika u odnosu na ostale podžanrove jest važnost mehanike drifta. Naime, u karting igricama je česta pojava da driftanje kartom daje ubrzanje igraču i na taj način je brži. Također, jedna od stvari koja je poprilično česta pojava u karting igrama su *power-upovi* u obliku kutija iznenađenja. Oni daju prednost igraču na različite načine. Najistaknutiji predstavnici karting igara su Mario Kart i Crash Team Racing.

3.3.3.1. Crash Team Racing

Crash Team Racing igra je s bogatom poviješću i odanom zajednicom i jedna je od najpoznatijih karting igara. Izdana je 1999. godine od strane Naughty Doga za PlayStation, a igra je ponovno izdana 2019. godine pod naslovom Crash team Racing Nitro Fueled kao *remaster* za PlayStation 4, Nintendo Switch te Xbox One.

Igra ima sve tipične sastojke karting igre, jednostavnu i nenamteljivu priču, nekoliko različitih, ali generalno sličnih dizajna automobila, tematske staze (zimske, pustinjske, staze na plaži, svemirske, ...), *boss fightove*, kutije iznenađenja i ubrzavanje pomoću drifta.

Igra također sadrži nekoliko različitih modova igranja osim tipičnih utrka, a dobro su prilagođeni za online multiplayer, na primjer *capture the flag*. Crash Team Racing oko sebe je uspio okupiti veliku zajednicu što je tipično za karting igre, najviše zbog online poredaka. Igrači diljem svijeta uvijek se natječu za postavljanje što boljeg rezultata na svim stazama i traže mjesta na kojima se dizajn staze može iskoristiti i pronaći prečace. Zbog toga ovakve igre igrači često znaju igrati godinama, iako u igru ne dolaze i ne uvode se nikakve posebne novosti.

Karting igre, pa tako i Crash Team Racing obično nemaju mehaničku štetu na automobilima, a fizika automobila nije realistična, nego je napravljena da prati taj poseban stil gdje je brže driftati nego 'normalno' voziti.

Slika 12: Prizor iz igre Crash Team Racing (izvor: [21])

3.3.4. Futurističke trkaće igre

Futurističke trkaće igre još jedan su pod-žanr trkaćih igara, a kao takve, vjerojatno se i najviše razlikuju od ostalih pod-žanrova, budući da obično imaju najmanje doticaja sa stvarnošću i najmanje pokušavaju imitirati stvarnost. Futurističke igre stoga su najčešće i najkreativniji proizvodi ovog žanra jer u velikom broju igara čak ni fizika vozila nije ni blizu ničemu razvijenom u stvarnosti. Osim znanstveno-fantastičnih vozila, za futurističke igre uobičajeni su i elementi borbe, odnosno često je moguće i pucati oružjem na svoje suparnike. Futurističke trkaće igre postale su popularne još 80-ih godina, kada su izdane igre poput Star Ridera i Cosmos Circuita. Najveći predstavnik ovog pod-žanra je Wipeout.

3.3.4.1. Wipeout

Wipeout je serija futurističkih trkaćih igara. Prva igra izdana je 1995. godine za PlayStation, DOS, Microsoft Windows te Sega Saturn platforme, dok je najnovija igra izdana 2022. godine za iOS i Android. Wipeout 2097 mnogi smatraju jednom od najboljih igara za PlayStation.

Igre su jako brzog tempa te ih karakterizira elektornička glazba koja svira u pozadini. U Wipeoutu vozila kojima se igrači utrkuju su zapravo anti-gravitacijski brodovi s naoružanjem. Igra također sadrži i *power-up* elemente, a oni mogu biti za napadačku ili obrambenu svrhu, na primjer rakete, strojnice, mine i štitovi te turbo. Svaki brod ima svoje karakteristike, poput brzine, upravljanja i jačine štita. Svaki brod ima svoj štit, čija je svrha upijanje štete. Tijekom utrke šteta se dobiva sudarima broda ili primanjem metaka, a ako se štit uništi te brod zadobije još određenu količinu štete, uništit će se te je igrač nakon toga eliminiran iz utrke. Igra sadrži različite modove igranja, na primjer *zone* mod igranja, gdje je cilj samo što drulje preživjeti dok brod sve više i više ubrzava do ogromnih brzina. Međutim, kao i u svakoj igri, najbitniji su osnovni načini igranja i najbitnije je da oni budu dobro razvijeni. U Wipeoutu tako postoje tri načina igranja, jedna utrka, *time trial* te turnir.

Posljednji Wipeout koji nije razvijen za mobilne uređaje nego konzole je Wipeout: Omega Collection, a izdan je 2017. Na Metacriticu ova igra ima ocjenu 85/100, dok od korisnika ima ocjenu 8/10, što je odlično. Već na prvi pogled, očito je kako se ovakve igre dramatično razlikuju od ostalih igara iz drugih pod-žanrova.

Slika 13: Prizor iz igre Wipeout 2097 (izvor: [22])

3.4. Razvoj videoigre utrkivanja u Unityju

Kako bi se što bolje prikazale mogućnosti Unityja kao programskog alata, rad u njemu potrebno je pokazati na primjeru. Tako će u ovom primjeru biti razvijen prototip trkaće igre u Unityju koristeći mnoge njegove mogućnosti.

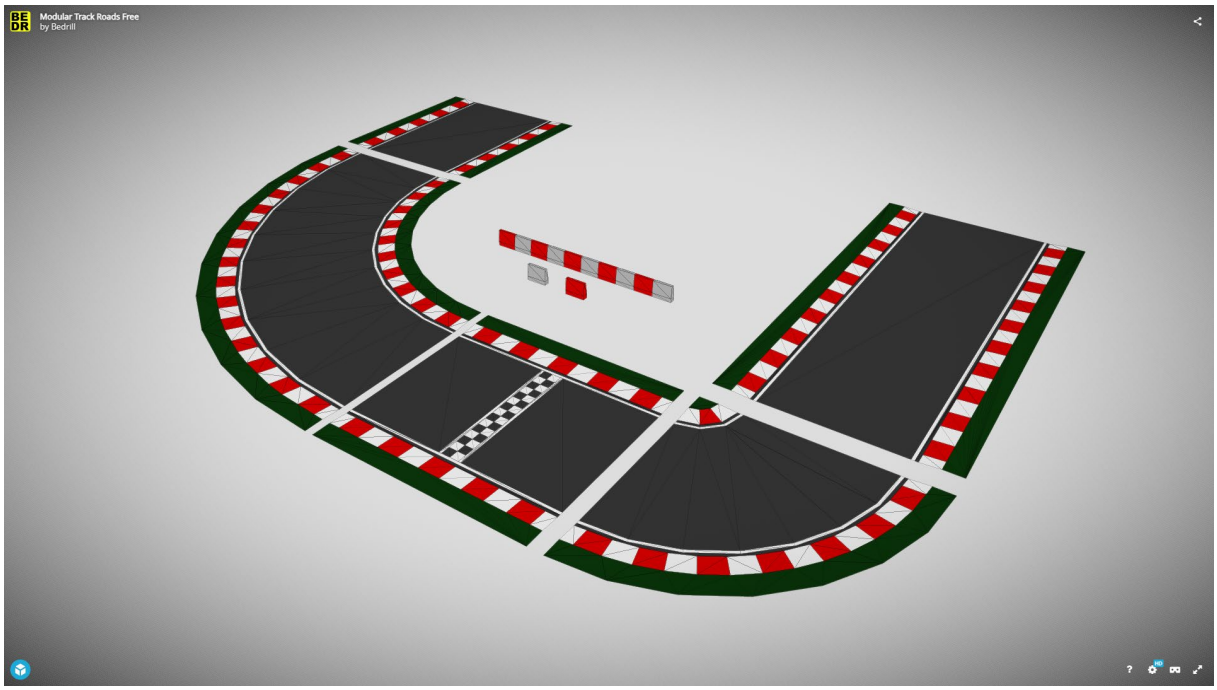
3.4.1. Dizajn igre

Prije kreiranja same igre, potrebno je osmisliti i dizajnirati samu igru. Tek nakon donošenja ideje o samoj igri može početi razvoj iste. Naslov igre je Burn the Rubber, a smisao igre je sličan kao u Trackmaniji, postavljanje najbržeg kruga. Igra je zamišljena kao igra s jednim igračem i bez umjetne inteligencije. Pred igračem bit će samo jedan zadatak, postaviti što je brži krug moguće. Igra će stilski pratiti duh igre. Budući da se radi o trkaćoj igri, a za trkaće igre karakteristična je brzina i energičnost, smisleno je u dizajnu koristiti crvenu i žutu boju. Crvena boja dobra je za isticanje, upadljiva je i snažna što se dobro uklapa s motivima trkaće igre. Stoga će crvena boja biti korištena u dizajnu sučelja, na primjer za tekst, kako bi se u igraču pobudili slični osjećaji. Žuta boja je energična i kontrastna boja koju bi isto bilo dobro u ovom kontekstu iskoristiti. Može biti dominantna u, na primjer, pozadinskoj slici na glavnom izborniku i može se koristiti pri dizajnu automobila.

Potrebno je osmisliti dizajn automobila koji bi odgovarao dizajnu igre. Dakle, potrebno je da automobil bude sportski i atraktivnog dizajna. Besplatni 3D model automobila preuzet je s Unityjevog Asset Storea[23], a izgleda ovako:

Slika 14: 3D model automobila za igru (Izvor: [23])

Može se reći da dizajn automobila odgovara ovakvom tipu igre. Nakon što je pronađen zadovoljavajući dizajn automobila, potrebno je osmisliti mape, odnosno staze na kojima će taj automobil igrač i voziti. Prva staza zamišljena je kao klasična trkaća staza s karakterističnim rubnicima, tribinama i ogradama. Druga staza bila bi drugačija od prve. Bila bi smještena u prirodi, u planinskom području s puno promjena visine. Prva staza bit će izgrađena pomoću priloga iz Asset Storea koji sadrži dijelove staze[24], a u ovom primjeru od tih dijelova bit će složena nova staza. Dijelovi staze izgledaju ovako:



Slika 15: 3D model dijelova prve staze (Izvor: [24])

Druga, planinska staza bit će preuzeta gotova, kao Terrain modul koji će onda po potrebi biti preuređen i preoblikovan[25]. Druga staza izgleda ovako:

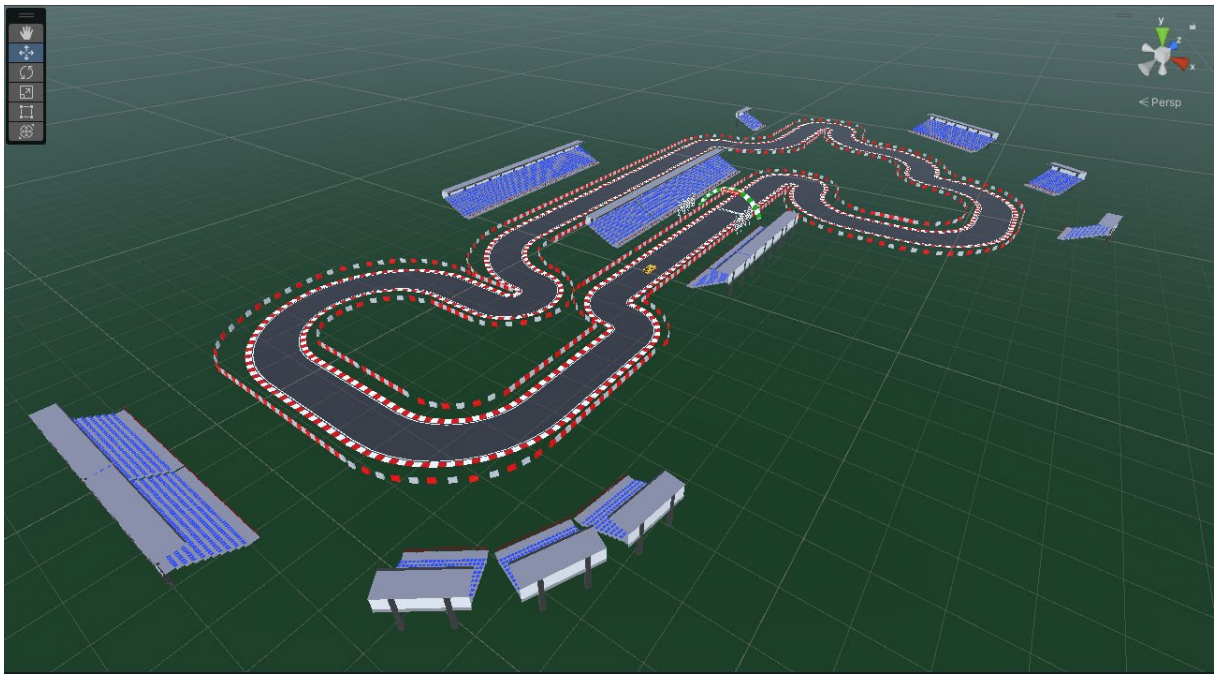
Slika 16: 3D model druge staze (Izvor: [25])

Sad kada su pripremljeni modeli automobila i staze, može se reći da je prikupljeno dovoljno materijala za početak izrade igre, budući da su minimalne komponente, automobil i staza, prikupljene.

3.4.2. Razvoj igre

3.4.2.1. Postavljanje staze

Kako bi se što prije dobio minimalni proizvod dovoljan za testiranje rezultata, najbolje je prije svega kreirati početnu stazu po kojoj će se onda automobil kretati. Kao što je već ranije rečeno, staza će biti složena od više različitih 3D modela staze. Prije svega, preuzeti asset potrebno je uvesti u Unity kroz Package Manager. Nakon što je na web-stranici Unity Asset Storea dodan asset, u projektu je potrebno otvoriti Package Manager, odabrati dodani paket te nakon toga ga preuzeti klikom na gumb Download i uvesti u projekt klikom na Import. Nakon toga, paket je dodan u Unity projekt te se može koristiti. Dijelove staze na scenu je vrlo jednostavno dodati – koristeći drag and drop funkcionalnost. Dakle, nakon pronalaska željenog dijela staze, potrebno ga je odabrati mišem i povući na scenu te je taj dio staze sada postao dijelom scene, a samim time i dijelom igre. Na taj način bit će izgrađena prva staza. Kako bi staza bila potpunija kozmetički, bit će dodani i određeni ukrasi poput tribina i zastava na stazu, kako bi sama staza izgledala ljepše. Asset s ukrasima za stazu također je preuzet s Unity Asset Storea.[26] Na isti način, drag and dropom bit će dodan i automobil na scenu. Nakon „slaganja“ staze, ona izgleda ovako:



Slika 17: Konačni izgled prve staze

Bitno je obratiti pozornost na collidere. Collideri su komponente koje se mogu dodati na komponente u igri. Oni služe za simulaciju fizike, pogotovo kod kontakta i interakcije između dvaju tijela. Collideri prepoznaju sudare između dva objekta te na temelju tih sudara simuliraju ponašanje objekata. Zbog toga je bitno da svi dijelovi mape imaju collider, kako bi automobil mogao interagirati sa stazom, odnosno kako automobil djelovanjem gravitacije ne bi samo „propao“ kroz stazu. Dakle, i automobil i svi dijelovi staze moraju imati collidere. Za dijelove staze najbolje je koristiti Mesh Collidere jer oni prate oblik mesha. Automobil ima specifičan set collidera. Naime, kod 3D modela, u ovom slučaju, bitno je da model automobila ima odvojeno tijelo od kotača, a da je svaki kotač zasebni model, odnosno mesh. Na sam automobil, odnosno na tijelo automobila bit će dodan Box Collider koji je u obliku kvadra. U ovom slučaju nije potreban Mesh Collider jer je oblik Box Collidera dovoljan za dobru interakciju, a Box Collider je mnogo jeftiniji po pitanju računalnih resursa od Mesh Collidera zbog svog jednostavnog oblika. Na kotače automobila bit će dodani Wheel Collideri. Radi se o posebnim colliderima koji su prilagođeni za kotače vozila. Okruglog su oblika i lako se postavljaju na kotače, a ti Wheel Collideri izrazito su bitni jer u sebi imaju svojstva i mogućnosti bitna za upravljanje ponašanjem i fizikom automobila. Nakon što su svi collideri dodani, može se reći da je scena pripremljena te se može prijeći na razvoj logike automobila.

3.4.2.2. Razvoj fizike automobila

Fizika automobila najkompleksnija je komponenta cijelog projekta i cijele igre zbog svoje matematičke složenosti te zbog svoje važnosti. Ona je jedan od najvažnijih čimbenika koji utječu na ugodnost i igrivost same igre. Stoga je potrebno obratiti posebnu pozornost na razvoj ovog dijela igre te na dobar način pristupiti rješavanju problema.

Za početak, potrebno je kreirati novu komponentu na objektu automobila. Ta komponenta zapravo je C# skripta pomoću koje će se automobil moći kretati. Kreirana skripta zove se CarController. Prije svega potrebno je odrediti varijable preko kojih će se moći lagano mijenjati ponašanje automobila te određene javne varijable za elemente poput collidera kako bi se ispravni collideri mogli dodijeliti na pravo mjesto.

```
public int maxSpeed = 250;
public int maxReverseSpeed = 45;
public int accelerationMultiplier = 2;
```

```
public int maxSteeringAngle = 27;

public float steeringSpeed = 0.5f;
public int brakeForce = 600;
public int decelerationMultiplier = 2;
public int handbrakeDriftMultiplier = 5;
public Vector3 bodyMassCenter;


public GameObject frontLeftMesh;
public WheelCollider frontLeftCollider;
public GameObject frontRightMesh;
public WheelCollider frontRightCollider;
public GameObject rearLeftMesh;
public WheelCollider rearLeftCollider;
public GameObject rearRightMesh;
public WheelCollider rearRightCollider;

public bool useEffects = false;

public ParticleSystem RLWParticleSystem;
public ParticleSystem RRWParticleSystem;

public TrailRenderer RLWTireSkid;
public TrailRenderer RRWTireSkid;


public bool useUI = false;
public TMPro.TMP_Text carSpeedText;
public bool useSounds = false;
public AudioSource carEngineSound;
public AudioSource tireScreechSound;
float initialCarEngineSoundPitch;
```

Prve varijable, `maxSpeed`, `maxReverseSpeed`, `accelerationMultiplier` i slične varijable sa zadanim vrijednostima služe za lakše mijenjanje vrijednosti koje utječu na ponašanje automobila i lakše uštimavanje osjećaja vožnje. Stoga su te varijable javne i preko Inspectora se mogu vrlo jednostavno mijenjati. Nakon toga zadane su javne varijable `frontRightMesh`, `frontRightCollider` i tako dalje. Preko njih će se lagano moći

dodijeliti odgovarajući meshevi i collideri za svaki kotač automobila. Bitno je da su meshevi i collideri u različitim objektima. Varijable `useUI`, `useEffects` i `useSounds` služe kako bi se u Inspectoru lagano moglo uključiti i isključiti zvukove, efekte za dim i tragove guma te prikaz brzine. U početnoj fazi razvoja često se dogodi da nisu u potpunosti razvijeni još svi detalji poput dima pa ih je potrebno isključiti kako se pri izvedbi ne bi događala pogreška. Shodno tome, varijable poput `RLWParticleSystem`, `RLWTireSkid`, `carSpeedText`, `carEngineSound` i `tireScreechSound` služe za dodjeljivanje određenih objekata skripti koja onda upravlja tim zvukovima i efektima. Zadana je još i `initialCarEngineSoundPitch` varijabla, a ona služi za određivanje početne frekvencije zvuka motora, kako bi se lakše mogao prilagoditi zvuk motora, tako da odgovara modelu automobila.

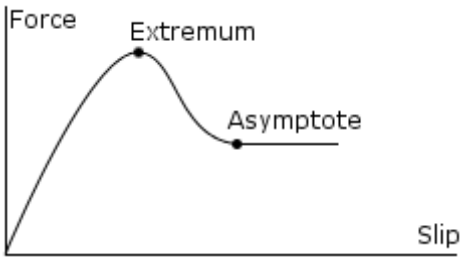
Nakon što su zadane varijable, potrebno je i isprogramirati logiku kretanja automobila. Zbog duljine same skripte neće biti pokriveni svi njeni dijelovi, nego samo oni najvažniji. Prije svega, potrebno je odrediti neke dodatne privatne varijable za rad skripte:

```
Rigidbody carRigidbody;
float steeringAxis;
float throttleAxis;
float driftingAxis;
float localVelocityZ;
float localVelocityX;
bool deceleratingCar;

WheelFrictionCurve FLwheelFriction;
float FLWextremumSlip;
WheelFrictionCurve FRwheelFriction;
float FRWextremumSlip;
WheelFrictionCurve RLwheelFriction;
float RLWextremumSlip;
WheelFrictionCurve RRwheelFriction;
float RRWextremumSlip;
```

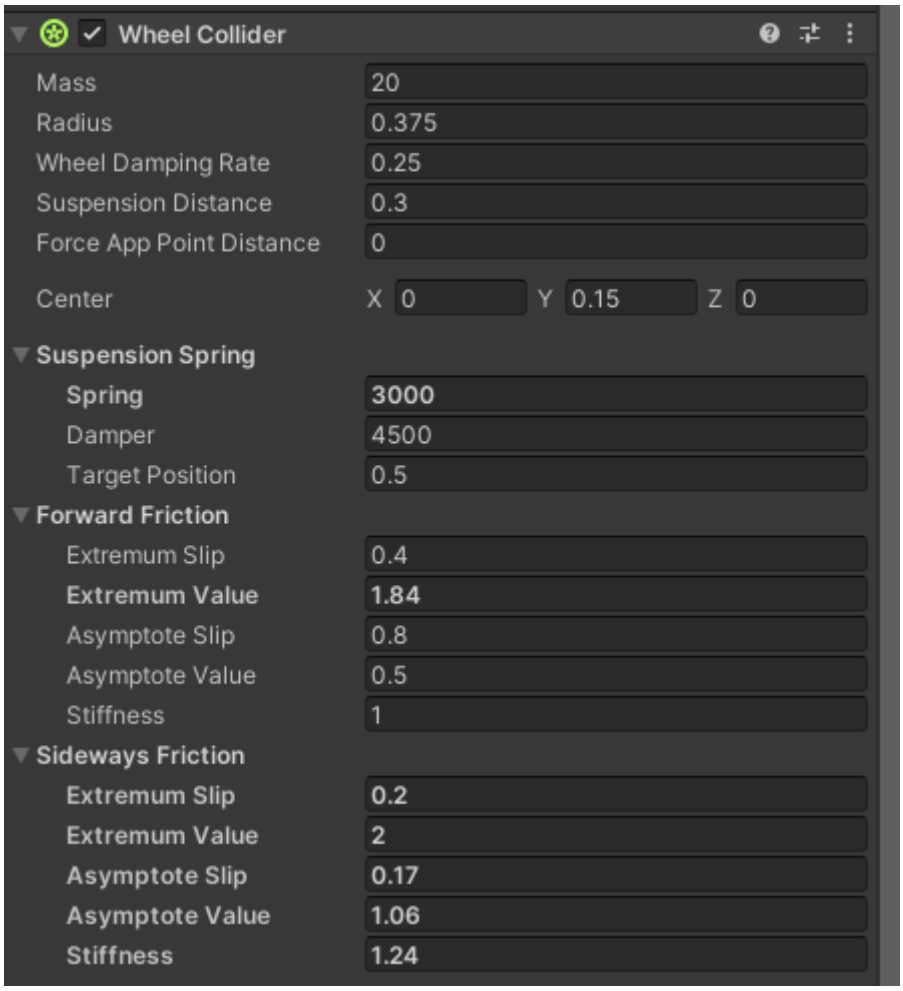
U ovom dijelu zadane su varijable koje spremaju brzinu po lokalnim osima x i z (`localVelocityZ`, `localVelocityX`), varijable koje spremaju podatak je li pritisnut gas i koliko, je li dana uputa za skretanje te drifta li automobil, a pomoću njih se poziva funkcija za skretanje, kretanje naprijed ili nazad te drift. Varijable `FLwheelFriction`,

`FLWextremumSlip` itd. Služe za određivanje količine trenja pomoću fizike samih WheelCollidera. Naime, u WheelCollider komponenti postoji razvijena logika trenja kotača. Trenje se u toj komponenti izračunava po posebnoj jednadžbi, a graf same jednadžbe izgleda ovako:



Slika 18: Graf trenja u WheelCollider komponenti (izvor: [27])

Mijenjanjem vrijednosti za „extremum slip“ i „asymptote“ može se dobiti različito ponašanje automobila jer se na taj način mijenja količina trenja te kako se trenje mijenja s dodanom silom. U ovom primjeru, na samim WheelColliderima zadane su sljedeće vrijednosti:



Slika 19: Zadavanje željenih vrijednosti trenja u WheelCollider komponenti

Nakon zadavanja potrebnih varijabli, sljedeći korak je programiranje metode `Start()`:

```
carRigidbody = gameObject.GetComponent<Rigidbody>();
carRigidbody.centerOfMass = bodyMassCenter;

FLwheelFriction = new WheelFrictionCurve ();
FLwheelFriction.extremumSlip =
frontLeftCollider.sidewaysFriction.extremumSlip;
FLWextremumSlip = frontLeftCollider.sidewaysFriction.extremumSlip;
FLwheelFriction.extremumValue =
frontLeftCollider.sidewaysFriction.extremumValue;
```

```
FLwheelFriction.asymptoteSlip =
frontLeftCollider.sidewaysFriction.asymptoteSlip;
FLwheelFriction.asymptoteValue =
frontLeftCollider.sidewaysFriction.asymptoteValue;
FLwheelFriction.stiffness = frontLeftCollider.sidewaysFriction.stiffness;
```

U početnom dijelu se u `carRigidbody` varijablu sprema rigidbody automobila, preko kojeg se dobivaju podatci o lokalnim brzinama i sličnim vrijednostima. U sljedećem dijelu `Start()` metode prepisuju se zadane vrijednosti o trenju koje su ranije spomenute iz WheelCollidera u lokalne varijable te se taj dio koda ponavlja za sve ostale kotače. Ovaj postupak prepisivanja vrijednosti u lokalne varijable ponavlja se za sve druge ranije navedene varijable te time završava `Start()` metoda. Sada je potrebno razviti `Update()` metodu, koja je zapravo u ovom slučaju najbitnija. Prije svega, treba odrediti kako će se u varijable spremati podatak o brzini automobila te o brzinama po lokalnim osima. To je zapravo vrlo jednostavno za ostvariti, a postiže se na sljedeći način:

```
carSpeed = (2 * Mathf.PI * frontLeftCollider.radius * frontLeftCollider.rpm
* 60) / 1000;
localVelocityX =
transform.InverseTransformDirection(carRigidbody.velocity).x;
localVelocityZ =
transform.InverseTransformDirection(carRigidbody.velocity).z;
```

Dakle, u ranije deklarirane varijable spremaju se podatci o brzini tijekom svakog *framea*. Brzina samog automobila (`carSpeed`) računa se iz brzine okretanja kotača, a onda će se dodatno skalirati radi postizanja stabilnijeg i realističnijeg prikaza brzine na ekranu.

Kako bi automobil bio upravljiv, potrebno je zadati kontrole te funkcije koje će se pozivati prilikom pritiska određene tipke. Tako su zadane sljedeće kontrole sa sljedećim funkcijama:

```
if(Input.GetKey(KeyCode.W)){
  CancelInvoke("DecelerateCar");
  deceleratingCar = false;
  GoForward();
}
if(Input.GetKey(KeyCode.S)){
```

```
            CancelInvoke("DecelerateCar");
            deceleratingCar = false;
            GoReverse();
          }

          if(Input.GetKey(KeyCode.A)){
            TurnLeft();
          }
          if(Input.GetKey(KeyCode.D)){
            TurnRight();
          }
          if(Input.GetKey(KeyCode.Space)){
            CancelInvoke("DecelerateCar");
            deceleratingCar = false;
            Handbrake();
          }
          if(Input.GetKeyUp(KeyCode.Space)){
            RecoverTraction();
          }
          if((!Input.GetKey(KeyCode.S) && !Input.GetKey(KeyCode.W))){
            ThrottleOff();
          }
          if((!Input.GetKey(KeyCode.S) && !Input.GetKey(KeyCode.W)) &&
!Input.GetKey(KeyCode.Space) && !deceleratingCar){
            InvokeRepeating("DecelerateCar", 0f, 0.1f);
            deceleratingCar = true;
          }
          if(!Input.GetKey(KeyCode.A) && !Input.GetKey(KeyCode.D) &&
steeringAxis != 0f){
            ResetSteeringAngle();
          }
        AnimateWheelMeshes();
```

Kao što je i vidljivo iz priloženog koda, za svaku kontrolu pokreće se određena funkcija. Također, potrebno je dodati i određena ponašanja kad se automobilom ne upravlja pa je tako osigurano da automobil usporava ako korisnik ne pritišće tipku W, S ili razmaknicu. Osim toga, kada korisnik prestane držati tipku za skretanje, budući da se igrom upravlja na tipkovnici, potrebno je osigurati da se kut skretanja vrati na početni. Također, uvijek će se izvoditi funkcija `AnimateWheelMeshes()`, pomoću koje se osigurava to da se

kotači, odnosno 3D meshevi kotača rotiraju i mijenjaju poziciju zajedno s njihovim colliderima.

Nakon što je završena `Update()` metoda, potrebno je implementirati sve ostale metode koje upravljaju ponašanjem automobila. Prva takva metoda je metoda za skretanje, odnosno metoda `TurnLeft()`:

```
    public void TurnLeft(){
      steeringAxis = steeringAxis - (Time.deltaTime * 10f * steeringSpeed);
      if(steeringAxis < -1f){
        steeringAxis = -1f;
      }
      var steeringAngle = steeringAxis * maxSteeringAngle;
      frontLeftCollider.steerAngle =
Mathf.Lerp(frontLeftCollider.steerAngle, steeringAngle, steeringSpeed);
      frontRightCollider.steerAngle =
Mathf.Lerp(frontRightCollider.steerAngle, steeringAngle, steeringSpeed);
    }
```

Kao što je i ranije u kodu navedeno, pritiskom na tipku A poziva se ova metoda. Pritiskom tipke A tako se smanjuje vrijednost osi skretanja, ovdje varijabla `steeringAxis`. U ovom slučaju predznak označuje smjer osi skretanja pa ako je ona negativna, to označava smjer ulijevo, a ako je pozitivna to znači da automobil skreće udesno. Sljedeći uvjet ograničava os skretanja na -1f koja je maksimalna vrijednost osi u ovom slučaju. Kut skretanja, odnosno varijabla `steeringAngle` dobiva se množenjem vrijednosti osi skretanja i ranije određenog maksimalnog kuta skretanja, što je javna varijabla koja se u Inspectoru može namjestiti na željenu vrijednost. Prema tome, kod skretanja ulijevo, kut skretanja u nekom trenutku može biti vrijednost između 0 i ranije određenog maksimalnog kuta skretanja. Nakon toga se na colliderima za prednji lijevi i prednji desni kotač postavlja vrijednost kuta skretanja u varijabli `frontLeftCollider.steerAngle`, odnosno `frontRightCollider.steerAngle`, a postavlja se pomoću matematičke metode `Lerp()`, koja linearno interpolira vrijednosti između dvije točke (prva dva argumenta) zadanom brzinom (treći argument). Pomoću metode `Lerp()` osigurava se postupna i kontrolirana promjena vrijednosti. Na isti način programira se i skretanje udesno, s tim da je jedina razlika u tome što se kod skretanja udesno vrijednost `steeringAxis` varijable povećava, a maksimalna vrijednost je 1f. Time je osigurano skretanje automobila. Također, potrebno se pobrinuti i za situaciju kada nije pritisnuta nijedna kontrola za skretanje. To je

ostvareno istom ovom logikom, ali u tom slučaju potrebno je vrijednost osi skretanja postaviti na 0f.

Budući da se i sama brzina računa iz brzine rotacije kotača, a i zbog osiguravanja određene doze realizma, potrebno je implementirati i rotaciju samih kotača. Rotacija je implementirana u metodi `AnimateWheelMeshes()` na sljedeći način:

```
void AnimateWheelMeshes(){
  try{
    Quaternion FLWRotation;
    Vector3 FLWPosition;
    frontLeftCollider.GetWorldPose(out FLWPosition, out FLWRotation);
    frontLeftMesh.transform.position = FLWPosition;
    frontLeftMesh.transform.rotation = FLWRotation;
  }catch(Exception ex){
    Debug.LogWarning(ex);
  }
}
```

Radi izbjegavanja redundantnosti, prikazana je implementacija rotacije samo za jedan kotač. Za sve ostale kotače, implementacija je jednaka, osim razlike u imenu varijabli te u varijablama kojima se mijenja sama pozicija i rotacija. Podatci o rotaciji i poziciji odgovarajućeg WheelCollidera spremaju se u `Quaternion` (struktura podataka za rotaciju) `FLWRotation` (odnosno `FRWRotation`, `RLWRotation` ili `RRWRotaion`) te u `Vector3` (struktura podataka za predstavljanje vektora i pozicija u 3D prostoru) `FLWPosition` (odnosno `FRWPosition`, `RLWPosition` ili `RRWPosition`) pomoću metode `GetWorldPose()` unutar samog collidera. Nakon toga, te vrijednosti spremaju se u varijable za poziciju i rotaciju samih mesheva. Nakon što je implementirano skretanje, potrebno je implementirati i kretanje unaprijed, a to je implementirano metodom `GoForward()`, koja se poziva pritiskom tipke W.

```
public void GoForward(){
  if(Mathf.Abs(localVelocityX) > 3f){
    isDrifting = true;
    DriftCarPS();
  }else{
    isDrifting = false;
    DriftCarPS();
```

```
        }
        throttleAxis = throttleAxis + (Time.deltaTime * 3f);
        if(throttleAxis > 1f){
          throttleAxis = 1f;
        }
        if(localVelocityZ < -1f){
          Brakes();
        }else{
          if(Mathf.RoundToInt(carSpeed) < maxSpeed){
            frontLeftCollider.brakeTorque = 0;
            frontLeftCollider.motorTorque = (accelerationMultiplier * 75f) *
throttleAxis;
            frontRightCollider.brakeTorque = 0;
            frontRightCollider.motorTorque = (accelerationMultiplier * 75f) *
throttleAxis;
            rearLeftCollider.brakeTorque = 0;
            rearLeftCollider.motorTorque = (accelerationMultiplier * 75f) *
throttleAxis;
            rearRightCollider.brakeTorque = 0;
            rearRightCollider.motorTorque = (accelerationMultiplier * 75f) *
throttleAxis;
          }else {
                    frontLeftCollider.motorTorque = 0;
                    frontRightCollider.motorTorque = 0;
            rearLeftCollider.motorTorque = 0;
                    rearRightCollider.motorTorque = 0;
              }
        }
    }
```

Na početku same metode provjerava se brzina automobila po osi x. Postavljeno je da ako je ta vrijednost veća od 3f automobil počne driftati, a to je osigurano postavljanjem varijable isDrifting na true te pozivanjem metode `DriftCarPS()` koja realizira emitiranje čestica dima i traga guma. Ovaj uvjet potreban je zbog dobivanja realističnije fizike automobila te simulira stvarna vozila koja također počinju proklizavati pri prejakom skretanju pri određenoj brzini. Dodavanje brzine implementirano je na isti način kao i kod skretanja, ali se ovdje povećava vrijednost varijable `throttleAxis`. Osim toga provjerava se i kreće li se automobil u trenutku davanja gasa unazad. Ako se automobil već kreće unazad, potrebno je prvo zakočiti, a onda kada brzina postigne vrijednost 0, automobil se

može početi kretati unaprijed. Kočenje je implementirano pozivanjem metode `Brakes()`. Povećavanje brzine automobila implementirano je povećanjem okretnog momenta motora te postavljanjem vrijednosti okretnog momenta kočnice (`brakeTorque`) na 0f. Okretni moment motora lako se može postaviti na željenu vrijednost jer mu se može pristupiti preko vaijable `frontLeftCollider.motorTorque` (te za ostale kotače respektivno). Okretni moment motora opet se postavlja na 0f kad je postignuta maksimalna zadana brzina. Na isti način implementirana je i metoda za kretanje unazad, `GoReverse()`. Razlika je u tome što se vrijednost varijable throttleAxis postavlja na -1f, umjesto 1f kod kretanja unaprijed. Također, ovdje je potrebno prvo primijeniti kočenje ako se automobil kreće unaprijed, a tek onda se dodaje negativni okretni moment motoru. Metoda `Brakes()` zapravo je vrlo jednostavna, a implementirana je tako da povećava vrijednosto kretnog momenta kočnice te smanjuje okretni moment motora:

```
public void Brakes(){
  frontLeftCollider.brakeTorque = brakeForce * 1500f;
  frontRightCollider.brakeTorque = brakeForce * 1500f;
  rearLeftCollider.brakeTorque = brakeForce * 1500f;
  rearRightCollider.brakeTorque = brakeForce * 1500f;
  frontLeftCollider.motorTorque = -700f;
  frontRightCollider.motorTorque = -700f;
  rearLeftCollider.motorTorque = -700f;
  rearRightCollider.motorTorque = -700f;
}
```

3.4.2.3. Razvoj sučelja

Nakon implementacije ponašanja automobila, potrebno je implementirati i ostale elemente igre, poput sučelja. Jedan od elemenata sučelja koji je gotovo neizostavan u svim današnjim igrama utrkivanja je *minimap*, koji predstavlja umanjenu kartu staze na kojoj se utrkuje te prikazuje poziciju vozila na istoj. *Minimap* je i ovdje implementiran na jednostavan način. Na scenu je dodan velik broj praznih objekata, otprilike na svakih 5 metara staze. Ti prazni objekti predstavljaju točke koje iscrtavaju put kojim staza ide. Kroz te točke onda će se provući linije koje predstavljaju oblik staze. Linije se iscrtavaju pomoću Unityjeve komponente `LineRenderer` koja će se dodati na novi objekt, ovdje nazvan `TrackPath`. Svi objekti točke zapravo su objekti dijete objekta `TrackPath`. Na `TrackPath` objekt, osim

`LineRenderer` komponente, dodana je i još jedna skripta pomoću koje je implementirano crtanje staze.

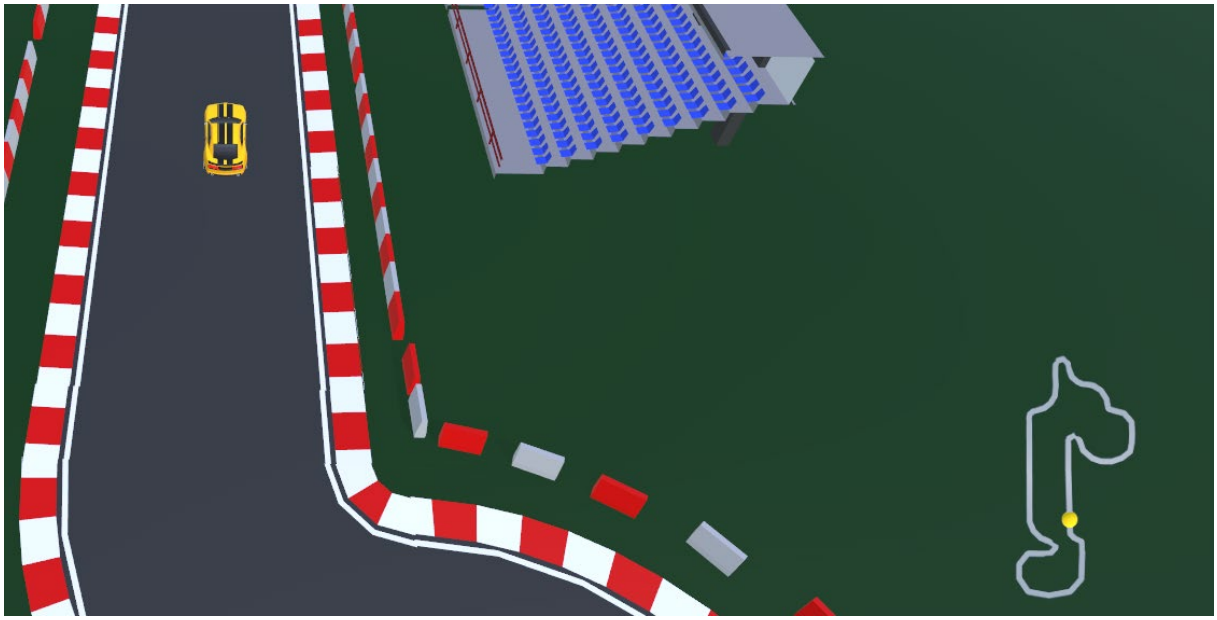
```
private LineRenderer lineRenderer;
private GameObject trackPath;
void Start()
    {
        lineRenderer = GetComponent<LineRenderer>();
        trackPath = this.gameObject;

        int positionCount = trackPath.transform.childCount;
        lineRenderer.positionCount= positionCount;

        for(int i=0; i<positionCount; i++)
        {
            lineRenderer.SetPosition(i, new
Vector3(TrackPath.transform.GetChild(i).transform.position.x,
                4, TrackPath.transform.GetChild(i).transform.position.z));
        }
        lineRenderer.SetPosition(positionCount-1,
lineRenderer.GetPosition(0));
        lineRenderer.startWidth = 7f;
        lineRenderer.endWidth = 7f;
    }
```

U ovoj skripti implementirana je samo `Start()` metoda jer je ona jedina potrebna. Skripta radi na način da se u `lineRenderer` varijablu spremi sama istoimena komponenta dodana na taj objekt, a sami objekt pozivat će se preko `trackPath` varijable. U varijablu `positionCount` sprema se broj točaka na stazi kroz koje će proći linija. Nakon toga, kod prelazi na for-petlju u kojoj se pomoću vektora određuje sljedeća pozicija kroz koju linija treba proći, odnosno prepisuje se pozicija sljedeće točke, odnosno sljedećeg djeteta u hijerarhiji. Nakon for-petlje, iscrtat će se linija, ali neće biti zatvorena poput staze pa je stoga poziciju `lineRenderera` potrebno opet postaviti na početnu točku kako bi se krug zatvorio. Sama linija na sljedećem pokretanju igre postat će vidljiva. Međutim, to nije ono što je potrebno. Zato je u Unityju potrebno napraviti novi sloj (layer) te dodijeliti liniju tom sloju te u *Inspectoru* glavne kamere pod postavkom *Culling Mask* odznačiti sloj u kojem se nalazi ta linija. Nakon toga potrebno je kreirati novi objekt vrste *Canvas*. *Canvas* je tip

objekta u Unityju koji služi za prikazivanje sučelja. Na novokreirani *Canvas* potrebno je dodati još jednu kameru. Kameru je nakon toga potrebno postaviti iznad staze te joj dodati praznu pozadinu i za *Culling Mask* odabrati samo taj sloj na kojem se nalazi linija. Nakon toga, na tijelo automobila potrebno je dodati novi objekt. Na primjer, jednu kuglu (*Sphere*). Taj objekt kretat će se zajedno s automobilom, a kako se ne bi vidio u igri potrebno ga je dodati u isti sloj kao i liniju. Tako će biti vidljiv na mapi, ali ne u samoj igri. Rezultat je sljedeći:



Slika 20: Prikaz mape na sučelju

Kao što je vidljivo i na samoj slici, u igru je dodana mapa bez pozadine koja prikazuje oblik staze te gdje se igrač nalazi na stazi. Na isti *Canvas* bit će potrebno je dodati i nove tekstualne objekte (*TextMeshPro* komponenta) koji će prikazivati brzinu automobila te vrijeme kruga. Prikaz brzine implementiran je u `CarController` skripti:

```
public void CarSpeedUI()
    {
        if (useUI)
        {
            try
            {
                Vector3 carForward = transform.forward;
```

```
                Vector3 velocityDirection =
carRigidbody.velocity.normalized;

                float scaledSpeed = carRigidbody.velocity.magnitude *
scalingFactor;
                int roundedSpeed = Mathf.RoundToInt(scaledSpeed);


                    if (Vector3.Dot(velocityDirection, carForward) >= 0f)
                    {
                        carSpeedText.text = roundedSpeed.ToString();
                    }
                    else
                    {
                        carSpeedText.text = "- " + roundedSpeed.ToString();
                    }
            }
            catch (Exception ex)
            {
                Debug.LogWarning(ex);
            }
        }
    }

    public float scalingFactor = 6f;
```

Ova metoda osigurava da se brzina prikazuje na realističan način te da prijelaz između vrijednosti nije prenagao što se može lako dogoditi ako se koristi samo vrijednost brzine preko `RigidBody` komponente automobila.

Osim sučelja u igri (HUD-a), potrebno je razraditi i ostale elemente sučelja, poput glavnog izbornika, izbornika za pauzu, ekrana za kraj igre, postavki te ljestvice. Svi ovi elementi izrađeni su na već opisani način pomoću *Canvas* objekta te dodavanjem gumba na njega. Logika gumba može se vrlo jednostavno postaviti u samom *Inspectoru* te se preko tih postavki može pozvati bilo koja javna metoda te se može jednostavno upravljati sučeljem.

Glavni izbornik prvi je ekran pri pokretanju igre. To sučelje sadrži pozadinu koja je kreirana pomoću DALL-E 2 AI alata [28], naslov igre te 3 gumba: *time trial*, opcije i izlaz. Također, na početnom izborniku u pozadini je puštena pjesma koja je omogućena pomoću Unityjeve *Audio Mixer* komponente. Glavni izbornik izgleda ovako:

Slika 21: Izgled početnog izbornika

Klikom na gumb Options pokreće se ekran s postavkama:

Slika 22: Izgled izbornika s postavkama

U postavkama je korisniku omogućeno upravljanje glasnoćom igre, korisnik može odabrati rezoluciju igre te hoće li igra biti u prozoru ili preko cijelog ekrana te može odabrati

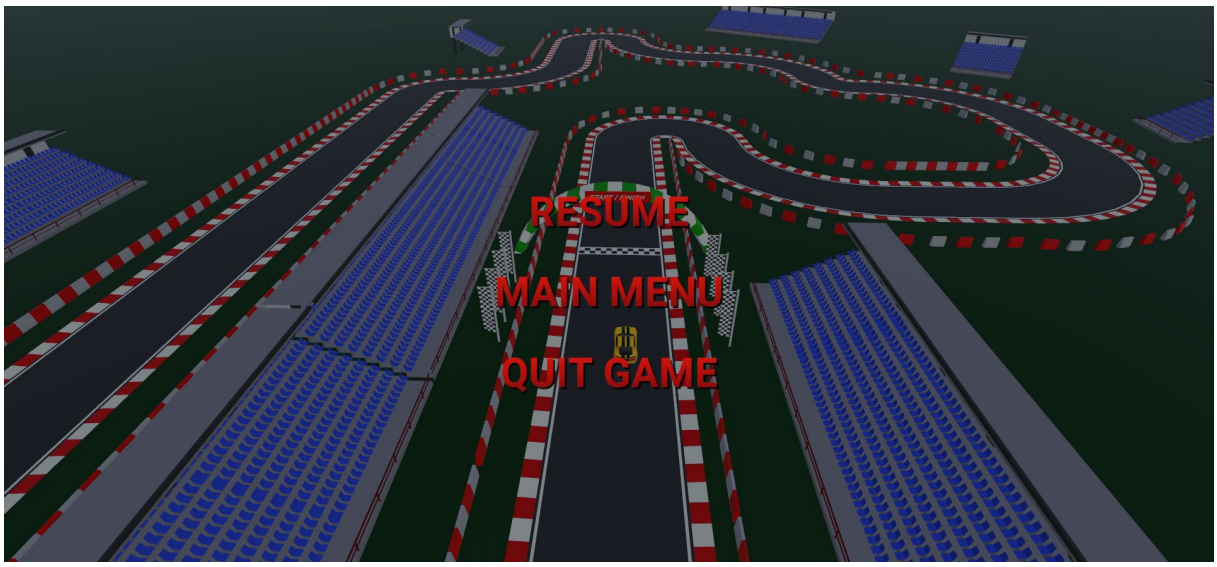
kvalitetu prikaza grafike. Ako pak korisnik odabere Time Trial opciju na glavnom izborniku, prikazat će mu se sljedeći ekran:

Slika 23: Izbor staze

Dakle, korisniku se nudi izbor staze na kojoj želi voziti. Sučelje u samoj igri izgleda ovako:

Slika 24: Izgled sučelja tijekom igranja igre

Kao što je vidljivo, u donjem desnom kutu nalazi se mapa koja prikazuje stazu te poziciju igrača na njoj. U gornjem desnom kutu nalazi se vrijeme kruga, a u donjem desnom kutu brzina kojom se automobil trenutno kreće.



Slika 25: Prikaz izbornika kod zaustavljanja igre

Pritiskom na tipku Escape, korisnik zaustavlja igru i pokreće izbornik tijekom igre. Na njemu su ponuđene opcije nastavka igre, povratak na glavni izbornik te izlaz iz igre. Ponovnim pritiskom na tipku Escape, igra se također nastavlja. Nakon završetka kruga, korisniku se prikazuje ekran na kojem je prikazano vrijeme kruga te opcije povratka u glavni izbornik, ponovnog pokretanja razine te odlaska na ljestvicu:

Slika 26: Sučelje za prikaz rezultata

Klikom na ljestvicu, korisniku se prikazuje poredak deset najboljih vremena te datum i vrijeme kada su postavljeni. Podatci o vremenima kruga te o datumu i vremenu spremaju se u `PlayerPrefs` komponentu Unityja koja omogućuje lokalno pisanje i čitanje podataka. Podatci o vremenima tako se spremaju u JSON obliku prilikom završetka kruga. Pokretanjem ekrana s ljestvicom dohvaćaju se podatci koji su lokalno spremljeni. Dohvaćaju se u JSON obliku te se onda spremaju u dinamičku listu, sortiraju, parsiraju i prikazuju na samoj ljestvici:

Slika 27: Prikaz ljestvice

3.4.2.4. Razvoj logike igre

Bez razrađene logike, igra nema smisla. Stoga je potrebno odrediti kada je cilj igre zadovoljen te kako se igra ponaša. S tim ciljem, kreirana je nova skripta, `TimeTrial`. Prije svega, potrebno je zadati varijable koje će se koristiti:

```
public TMPro.TMP_Text TimeText;
public TMPro.TMP_Text ResultText;
private bool not_finished = false;
private bool started = false;
public List<GameObject> waypointObjects;
public GameObject StartFinishLine;
private float startTime;
private bool lapFinished;
public AudioSource CarSound;
public AudioSource TireSound;
public GameObject ResultUI;
public GameObject HUDUI;
public TimeSpan timeSpan;
public TimeSpan finalTime;
public Leaderboard leaderboard;
```

U samu igru potrebno je dodati nove objekte, odnosno kontrolne točke, kako bi se moglo provjeriti kada je krug započeo te kada je završio. Kontrolne točke zapravo su objekti s `BoxCollider` komponentom. Bitno je da te komponente imaju označeno svojstvo *Is Trigger*. Na taj način osigurava se da se automobil u kontaktu s kontrolnom točkom ne zabije u nju, nego da prođe kroz nju kao da i ne postoji, a može se pratiti je li igrač prošao kroz kontrolnu točku. Smatra se da je krug započeo prvim prolaskom kroz startnu, odnosno ciljnu liniju na stazi, a krug je završio ako je igrač već jednom prošao početnu liniju, prošao kroz sve kontrolne točke na krugu i opet dodirnuo početnu liniju. Nakon toga vrijeme se zaustavlja te se pokreće sučelje s rezultatom. To je sve implementirano u sljedećem kodu:

```
public void Start()
{
    GameObject[] allWaypoints =
  GameObject.FindGameObjectsWithTag("Waypoint");
    waypointObjects.AddRange(allWaypoints);
```

```
        StartFinishLine = GameObject.FindGameObjectWithTag("StartFinish");
        ResultUI.SetActive(false);
        HUDUI.SetActive(true);
        Time.timeScale = 1f;



    }

    void Update()
    {
        StartFinish StartFinishObject =
StartFinishLine.GetComponent<StartFinish>();

        if (!started)
        {
            TimeText.text = "0:00.000";
        }

        if (StartFinishObject.StartLineHit && started == false)
        {
            started = true;
            not_finished = true;
            StartFinishObject.StartLineHit = false;
            startTime = Time.time;
        }

        if (started && not_finished)
        {
            timeSpan = TimeSpan.FromSeconds(Time.time - startTime);
            TimeText.text = string.Format("{0:00}:{1:00}.{2:000}",
timeSpan.Minutes, timeSpan.Seconds, timeSpan.Milliseconds);
        }

        if (StartFinishObject.StartLineHit && started &&
CheckAllWaypoints(waypointObjects) && not_finished)
        {
            not_finished = false;
            lapFinished = true;

        }
```

```
    if (lapFinished == true)
    {
        Time.timeScale = 0f;
        HUDUI.SetActive(false);
        ResultUI.SetActive(true);
        ResultText.text = TimeText.text;
        CarSound.Pause();
        TireSound.Pause();
        lapFinished = false;
        finalTime = timeSpan;
        leaderboard.lapTime = finalTime.TotalMilliseconds;
        leaderboard.dateTime = DateTime.Now.ToString();
    }
}

bool CheckAllWaypoints(List<GameObject> waypointObjects)
{
    foreach (GameObject waypoint in waypointObjects)
    {
        Waypoint waypointObject = waypoint.GetComponent<Waypoint>();
        bool waypointHit = waypointObject.WaypointHit;
        if (waypointHit == false)
        {
            return false;

        }
    }
    return true;
}

public void Retry()
{
    SceneManager.LoadScene("Game");
}
```

4. Zaključak

Ovim radom opisan je programski alat Unity kao pouzdan i svestran alat za razvoj igara te su navedene i opisane jedne od najpoznatijih i najpopularnijih igara koje su razvijene u Unityju, s ciljem dokazivanja verzatilnosti Unityja kao programskog alata za izradu igara te s ciljem dokazivanja njegove popularnosti, čak i u izradi najpopularnijih i najvećih (tzv. AAA) igara.

Također, opisane su i glavne igre predstavnice žanra utrkivanja te su opisane njihove karakteristike i kako se pod-žanrovi međusobno razlikuju te po čemu su prepoznatljivi.

Cilj ovog rada bio je prikazati proces razvoja igre utrkivanja u programskom alatu Unity i na taj način pokazati kako se u Unityju može razviti videoigra i koje Unityjeve karakteristike i na koji način olakšavaju i ubrzavaju sam razvoj igara. Rezultat ovog rada je razvijen novi funkcionalni prototip igre utrkivanja. Razvijena igra ima implementirano sučelje tijekom igre na kojem se pokazuju podatci o vremenu kruga, brzini automobila te mapa, a razvijena su i osnovna sučelja izbornika, prikaz rezultata na kraju igre te ljestvica najboljih rezultata. Ova igra, nastala kao rezultat ovog rada, sadrži sve osnovne stvari koje sve igre utrkivanja imaju te kao takva pruži dobar temelj za dodatnu nadogradnju.

Popis literature

[1] E. Makuch, „Rockstar: More than 1,000 people made GTAV“, *GameSpot*, listopada 2013. https://www.gamespot.com/articles/rockstar-more-than-1000-people-made-gtav/1100-6415330/ (pristupljeno 29. kolovoza 2023.).

[2] J. Ramée, „The 14 Best Games Developed By Only One Person“, *GameSpot*. https://www.gamespot.com/gallery/the-14-best-games-developed-by-only-one-person/2900-2172/#15 (pristupljeno 29. kolovoza 2023.).

[3] E. Fanouraki, „Did you know that 60% of game developers use game engines?“, *SlashData*, prosinca 2022. https://www.slashdata.co/blog/did-you-know-that-60-of-game-developers-use-game-engines (pristupljeno 30. kolovoza 2023.).

[4] Unity Technologies, „Compare plans“, *Choose the Plan that is Right for You*. https://unity.com/compare-plans (pristupljeno 1. rujna 2023.).

[5] „Pokémon GO“, *Pokémon*. https://www.pokemon.com/us/app/pokemon-go/ (pristupljeno 5. rujna 2023.).

[6] M. Padilla, „Pika-Who? How Pokémon Go Confused the Canadian Military“, *The New York Times*, 1. siječnja 2020. Pristupljeno: 12. rujna 2023. [Na internetu]. Dostupno na: https://www.nytimes.com/2020/01/01/world/canada/pokemon-go-canada-military.html

[7] P. Susic, „21+ Among Us Statistics: How Many People Play It?“, *HeadphonesAddict*, srpanj 2023. https://headphonesaddict.com/among-us-statistics/ (pristupljeno 5. rujna 2023.).

[8] „Among Us“, *Innersloth*. https://www.innersloth.com/games/among-us/ (pristupljeno 5. rujna 2023.).

[9] M. Craig, „The viral game that made a comeback “Among Us”“, *The Bird Feed*, 11. studenog 2020. https://thebirdfeed.org/14387/showcase/the-viral-game-that-made-a-comeback-among-us/ (pristupljeno 12. rujna 2023.).

[10] *Let’s Try Cities: Skylines! [PDXCon 2015 Gameplay]*, (2015.). Pristupljeno: 12. rujna 2023. [Video]. Dostupno na: https://www.youtube.com/watch?v=VMf6EO73ryk

[11] „Rust“, *Rust*. https://rust.facepunch.com/ (pristupljeno 5. rujna 2023.).

[12] L. Winkie, „Rust review“, *PC Gamer*, 19. veljače 2018. https://www.pcgamer.com/rust-review/ (pristupljeno 13. rujna 2023.).

[13] „Beat Saber“, *Metacritic*. https://www.metacritic.com/game/pc/beat-saber (pristupljeno 5. rujna 2023.).

[14] „Beat Saber on Steam“, *Steam*. https://store.steampowered.com/app/620980/Beat_Saber/ (pristupljeno 13. rujna 2023.).

[15] „Unity: Understanding URP, HDRP, and Built-In Render Pipeline“, *Occa Software*. https://www.occasoftware.com/blog/unity-understanding-urp-hdrp-built-in (pristupljeno 5. rujna 2023.).

[16] Unity Technologies, „Unity Visual Scripting“, *Unity*. https://unity.com/features/unity-visual-scripting (pristupljeno 12. rujna 2023.).

[17] A. Dhapte, „Racing Games Market Overview“, *Market Research Future*. https://www.marketresearchfuture.com/reports/racing-games-market-9560 (pristupljeno 5. rujna 2023.).

[18] „Need for Speed Most Wanted Xbox 360 Gameplay - Bump ‚n‘ Grind“, *IGN*, 3. studenog 2005. https://www.ign.com/videos/need-for-speed-most-wanted-xbox-360-gameplay-bump-n-grind (pristupljeno 13. rujna 2023.).

[19] A. S. Maria, „Trackmania Review“, *IGN*, 30. lipnja 2020. https://www.ign.com/articles/trackmania-review-2 (pristupljeno 13. rujna 2023.).

[20] „Gran Turismo 7“, *Gran Turismo*. https://www.gran-turismo.com/us/products/gt7/ (pristupljeno 13. rujna 2023.).

[21] „Crash Team Racing“, *IMDb*, https://www.imdb.com/title/tt0260831/ (pristupljeno 13. rujna 2023.)

[22] „WipEout 2097, PlayStation“, *The King of Grabs*, 15. ožujka 2023. https://thekingofgrabs.com/2023/03/15/wipeout-2097-playstation/ (pristupljeno 13. rujna 2023.).

[23] Mena, „ARCADE: FREE Racing Car“, *Unity Asset Store*. https://assetstore.unity.com/packages/3d/vehicles/land/arcade-free-racing-car-161085 (pristupljeno 9. rujna 2023.).

[24] Bedrill, „Modular Lowpoly Track Roads FREE“, *Unity Asset Store*. https://assetstore.unity.com/packages/3d/environments/roadways/modular-lowpoly-track-roads-free-205188 (pristupljeno 9. rujna 2023.).

[25] Crafted Digitals, „Mountain race tracks“, *Unity Asset Store*. https://assetstore.unity.com/packages/3d/environments/landscapes/mountain-race-tracks-110408 (pristupljeno 9. rujna 2023.).

[26] Bedrill, „Props for Track Environment Lowpoly Free“, *Unity Asset Store*. https://assetstore.unity.com/packages/3d/props/props-for-track-environment-lowpoly-free-211494 (pristupljeno 10. rujna 2023.).

[27] Unity Technologies, „Unity - Manual: Wheel Collider component reference“, *Unity Documentation*. https://docs.unity3d.com/Manual/class-WheelCollider.html (pristupljeno 12. rujna 2023.).

[28] OpenAI, „DALL·E 2“, *OpenAI*. https://openai.com/dall-e-2 (pristupljeno 12. rujna 2023.).

Popis slika