Ajosimulaation luominen Unity-pelimoottorilla

Arttu Ojanen

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2024

Tiivistelmä

Tekijä(t)
Arttu Ojanen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Ajosimulaation luominen Unity-pelimoottorilla

Sivu- ja liitesivumäärä
25

Tämä opinnäytetyö käy läpi yksinkertaisen ajosimulaation luomisen. Ajosimulaatio tehdään Unity-pelimoottorilla. Opinnäytetyö koostuu teoriaosuudesta ja toiminnallisesta osuudesta. Teoriaosuuden tavoitteena on antaa alustava tietopohja toiminnallisen osuuden tueksi. Toiminnallisessa osuudessa käydään läpi ajosimulaation luonnin vaiheet. Simulaation tavoitteena on jäljentää auton realistisen käyttäytymisen kannalta keskeiset ominaisuudet, jotka ovat renkaat, moottori, vaihteisto ja pitovoima.

Teoriaosuuden ensimmäisessä pääluvussa kerrotaan ajosimulaatioista. Siinä selitetään, mitä ajosimulaatiot ovat ja mitkä ovat niiden käyttötarkoitukset. Luvussa kerrotaan myös historian ensimmäisestä ajosimulaatiosta, ja minkälaisia ajosimulaatiot ovat nykypäivänä.

Teoriaosuuden toisessa pääluvussa kerrotaan Unity-pelimoottorista. Tässä luvussa kerrotaan mikä Unity on, ja mitä pelimoottorit ovat. Luvussa käydään myös läpi, miten Unityn käyttöliittymä toimii, miltä se näyttää, sekä myös, mikä Unityn Asset Store on ja miten sitä käytetään.

Toiminnallisessa osuudessa käydään yksityiskohtaisesti läpi simulaation luomisen eri vaiheet. Osuus alkaa projektin luonnista, jonka jälkeen hankitaan testauksessa tarvittavat materiaalit, ja luodaan yksinkertainen testausympäristö. Seuraavaksi simulaation jokaisen ominaisuuden luominen käydään yksityiskohtaisesti läpi, kertomalla miten ominaisuudet tehdään C# -skriptillä, miten skriptit toimivat ja havainnollistamalla kuvien avulla.

Asiasanat
Simulaatio, Unity, pelimoottori, C#

Sisällys

1 Johdanto

Ajosimulaatiot ovat olleet keskeinen osa ammattiajajien ajotaitojen tutkimista ja kehittämistä jo jonkin aikaa. Teknologian kehittyessä ajosimulaatiot ovat tulleet saataville myös kuluttajille. Tänä päivänä kuka tahansa voi hankkia itselleen ajosimulaatiopelin muutamalla eurolla. Myös simulaatiotarvikkeiden kasvava ala tarjoaa nostetta virtuaalisen ajokokemuksen parantamiseksi. Mutta etsiessään itselle sopivaa ajosimulaatiota ostaja joutuukin valinnan eteen. Vaihtoehtoina on vain realistiseen ajokokemukseen keskittyviä simulaatioita, tai mukaansatempaavia, mutta epärealistisilla fysiikoilla varustettuja ajopelejä. Tästä johtuen minulle heräsi kysymys. Miksi olemassa ei ole sellaista ratkaisua, joka yhdistäisi simulaation aitouden ja pelin hauskuuden?

Vastauksen löytäminen herätti mielenkiinnon ajosimulaation luontiin. Opinnäytetyön tekeminen tästä projektista vaikutti hyvältä ratkaisulta, sillä työn dokumentointi sekä selittäminen vahvistaisivat omaa oppimista. Simulaatio tehdään Unity-pelimoottorilla aikaisemman Unity-kokemuksen vuoksi. Unityn käyttö on myös helposti opittavissa, joten kuka tahansa pystyy tekemään samanlaisen luomuksen seuraamalla tämän työn vaiheita. Projektin tekeminen Unitylla on veloituksetonta, kunhan projektia ei julkaista myytäväksi. Unityn valintaa voi perustella myös siitä saadulla kokemuksella. Videopeliala on suuri ja kasvava ala, joten kokemus Unitylla, joka on yksi käytetyimmistä pelimoottoreista, on vain hyödyksi.

Tämän opinnäytetyön tavoitteena on luoda yksinkertainen ajosimulaatio käyttäen Unity-pelimoottoria. Työ keskittyy realististen ajofysiikoiden jäljittelyyn ja tarkoituksena on saada auto käyttäytymään mahdollisimman aidon tuntuisesti. Tavoitteena on myös tehdä työstä helposti ymmärrettävä ja seurattava, jotta työtä voisi käyttää ohjeena tai mallina samanlaisen projektin tekemiseen. Työ voidaan jakaa neljään simuloitavaan ominaisuuteen, jotka ovat seuraavat:

- renkaat
- moottori
- vaihteisto
- pitovoima

Jokainen työn vaihe käydään läpi yksityiskohtaisesti omissa alaluvuissa. Työssä ei tulla keskittymään simulaation esteettisyyteen tai ulkonäköön. Simulaation testausympäristö kootaan Unityn valmiiksi tarjoamista objekteista. Auton kolmiulotteinen malli hankitaan Unity Asset Storesta. Poikkeuksena rajauksiin, ainoa visuaalinen ominaisuus tulee olemaan renkaiden animointi.

Keskeiset käsitteet

C#	C# on Microsoftin luoma olio-ohjelmointikieli (Pluralsight 2022).
Collider	Komponentti, joka tunnistaa peliobjektin törmäykset (Hilton 2021).
Float	Datatyyppi, joka määrittää desimaaliluvun (Bolton 2019).
For loop	Lauseke, joka määrittää kuinka monta kertaa jokin asia toistetaan (ByteHide 2023).
Int	Datatyyppi, joka määrittää kokonaisluvun (Bolton 2019).
Komponentti	Peliobjektin osa, joka määrittää peliobjektin toiminnallisuuden (Unity Technologies 2024).
Peliobjekti	Komponenteista koostuva objekti, joka muiden pelikomponenttien kanssa muodostaa pelin (Starloop 2021).
Rigidbody	Komponentti, joka mahdollistaa fysiikoiden vaikutuksen peliobjektiin (Jerga 2021).
Skene	Kokonaisuus, joka sisältää pelin ympäristön ja valikon. Skenejä voi olla useita (Occa Software 2023.)

2 Ajosimulaatiot

Ajosimulaatiot ovat virtuaalisia ympäristöjä, jotka pyrkivät jäljittelemään ajajan perspektiiviä. Yleensä ajosimulaatioissa käytetään ohjauspyörää sekä polkimia syötteen antamiseen, jotta ajaminen tuntuisi mahdollisimman todelta. (York driving simulators s.a.)

Simulaatiota, joissa tarvitaan ihmisen syötettä, kutsutaan termillä DIL (Driver-in-the-loop). DIL-simulaatiossa ajaja antaa syötteen simulaatiolle. Simulaatio vastaa ajajan antamaan syötteeseen, jolloin ajajan täytyy reagoida simulaation syötteeseen uusilla syötteillä. Tästä syntyy jatkuva silmukka, josta DIL saa nimensä. (Morse 2019.)

DIL simulaatiot voidaan jakaa kolmeen eri päätyyppiin. Nämä ovat viihde- DIL-simulaatiot, inhimilliset tekijät -DIL-simulaatiot ja suunnittelu- DIL-simulaatiot. Viihdesimulaatiot ovat nimensä mukaan viihdekäyttöön tarkoitettuja kaupallisia tuotteita. Ne eivät tyypillisesti sisällä työkaluja datan keräämiseen, joten ne eivät kelpaa ammattikäyttöön, mutta soveltuvat ajotaitojen parantamiseen. Inhimilliset tekijät DIL-simulaatiot muistuttavat hieman viihdesimulaatioita, mutta ne simuloivat paremmin aitojen kulkuneuvojen realistisia olosuhteita. Näitä simulaatioita käytetään usein ajajan ajotapojen mittaamiseen ja tutkimiseen. Suunnittelu- DIL-simulaatiot ovat samankaltaisia inhimilliset tekijät -DIL-simulaatioiden kanssa, mutta ajajan ajotapojen mittaamisen sijaan näitä käytetään kulkuneuvon, tai ajajan suorituskyvyn mittaamiseen. Suunnittelu-DIL-simulaatiot sopivat erinomaisesti kulkuneuvojen kehitystyöhön, sekä ammattiajajien taitojen kehittämiseen. (Morse 2015.)

2.1 Ensimmäinen ajosimulaatio

Ensimmäisen ajosimulaation loi Iso-Britannialainen autovalmistaja Lotus 1960-luvulla. Lotus kilpaili Formula One sekä Indycar sarjoissa, ja menestyi hyvin molemmissa sarjoissa useiden innovaatioiden avulla. Yksi näistä innovaatioista oli kilpa-ajosimulaatio. Simulaatio tehtiin sen ajan lentokonesimulaatioiden pohjalta. Simulaatiossa käytettiin Lotus 31 kilpa-auton runkoa, josta poistettiin moottori ja vaihteisto. Runkoon lisättiin useita sensoreita mittaamaan ajajan syötteet ohjauspyörästä ja polkimista. Autoon yhdistettiin levy, jonka päälle oli mallinnettu haluttu kilparata. Levyn päällä oli kamera, joka kuvasi ajajan sijaintia radalla. Kameran kuva näkyi auton edessä olevalla valkokankaalla. Kun ajaja painoi kaasupoljinta, levy alkoi pyöriä, simuloiden radalla ajamista. Mitä enemmän ajaja painoi kaasupoljinta, sitä nopeammin levy pyöri, ja kun ajaja painoi jarrupoljinta, levyn pyörintä hidastui. Kun ajaja käänsi ohjauspyörää, levy siirtyi sivusuunnassa, simuloiden auton kääntymistä. Levyn yläpuolella oli myös kiinteä varsi, johon oli kiinnitetty lamppu, joka valaisi rataa. (Baxter 2022.)

2.2 Modernit simulaatiot

Teknologian kehittyessä ajosimulaatioista on tullut äärimmäisen realistisen tuntuisia. Viihdekäyttöön tarkoitetut pelit, kuten Assetto Corsa ja iRacing, simuloivat ajamiseen liittyviä fysiikoita, sekä auton käyttäytymistä erittäin tarkasti. Realismin rajoittavaksi tekijäksi jää usein simulaatiossa käytetyn laitteiston, kuten ohjauspyörän ja polkimien laatu, tai niiden puute.

IRacing on erityisesti ammattiajajien suosiossa. Formula One-ajaja ja tämän työn kirjoitusvaiheessa neljä maailmanmestaruutta voittanut Max Verstappen viettää paljon vapaa-aikaa iRacing:in parissa. Hän on kuulunut Team Redline simulaatiokilpa-ajo -organisaatioon vuodesta 2015 lähtien ja kilpailee virtuaalisen kilpa-ajon huipulla Formulan lisäksi. (Leporati 2024.) Myös Nascar-ajaja William Byron kertoi The Daily Downforcelle käyttävänsä iRacingiä kausien välissä ylläpitääkseen ajotaitojaan. (The Daily Downforce 2024.)

3 Unity-pelimoottori

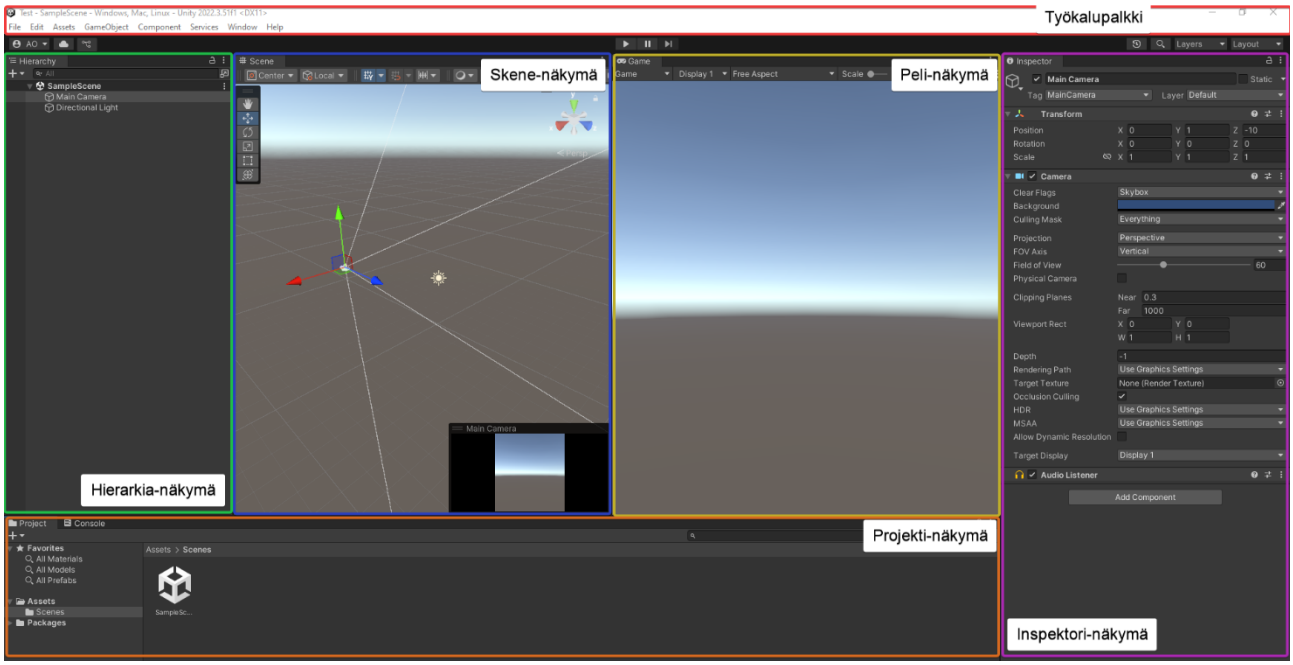
Unity on kaksi- sekä kolmiulotteisten sovellusten luontiin tarkoitettu pelimoottori, jonka ensimmäinen versio julkaistiin vuonna 2005. Unityn on kehittänyt Unity Technologies. Unity Technologies halusi luoda pelimoottorin, joka tarjoaisi helppokäyttöisiä kehitystyökaluja pelikehittäjille. Unity-pelimoottori on kehittynyt ja muuttunut paljon sen julkaisusta lähtien, mutta sen tavoite on pysynyt samana. Unity on helppokäyttöinen kehittäjän taitotasosta riippumatta. (Zenva 2024.)

3.1 Mikä on pelimoottori?

Pelimoottori on ohjelmisto, joka sisältää valmiiksi erilaisia järjestelmiä ja työkaluja pelikehitystä varten. Pelimoottorin ansiosta kehittäjän ei tarvitse luoda peliin jokaista järjestelmää täysin alusta asti. Eri pelimoottoreissa on erilaisia järjestelmiä, mutta yleisesti pelimoottori sisältää neljä keskeistä järjestelmää, jotka ovat grafiikoiden renderöintijärjestelmä, fysiikkamoottori, audiomoottori ja tekoälyjärjestelmä. Pelimoottorit ovat niin oleellinen osa pelikehitystä, että yleensä organisaatiot luovat oman pelimoottorinsa, jos mikään jo olemassa oleva pelimoottori ei sovellu organisaation tarpeisiin. (Crivello s.a.)

3.2 Unityn käyttöliittymä

Unityn käyttöliittymä koostuu kuudesta eri näkymästä. Nämä ovat työkalupalkki, Hierarkia-näkymä, Skene-näkymä, Peli-näkymä, Inspektori-näkymä ja Projekti-näkymä. Työkalupalkki on Unity Editorin ylälaidassa, ja se sisältää editorin yleiset asetukset. Hierarkia-näkymä sijaitsee editorin vasemmassa laidassa, ja sen sisällä on kaikki siinä skenessä käytetyt peliobjektit. Skene-näkymä näyttää valitun pelikohtauksen. Skene-näkymässä voi valita, asetella ja muokata peliobjekteja. Peli-näkymä näyttää pääkameran kuvakulman, ja sitä käytetään pelin testaamiseen. Inspektori-näkymässä näkyy valitun peliobjektin kaikki komponentit. Täällä voi myös muokata komponenttien asetuksia. Projekti-näkymästä löytyy kaikki projektin sisältämät tiedostot. Esimerkiksi, jos projektiin on tuotu kolmannen osapuolen peliobjekti, se ilmestyy Projekti-näkymään. Kuvassa 1 on havainnollistettu Unityn käyttöliittymän näkymät.



Kuva 1. Unityn käyttöliittymä

3.3 Unity Asset Store

Unity Asset Store on verkkokauppa, josta voi hankkia Unity Technologiesin, tai Unityn yhteisöön kuuluvien tekemiä tuotoksia. Tuotokset voivat olla esimerkiksi tekstuureja, malleja, animaatioita, projekteja, tai laajennuksia Unity Editoriin. Tuotteet voivat olla maksullisia tai ilmaisia. Asset Storen verkkosivuilla tuotteita voi etsiä kategorioittain, tai kirjoittamalla hakusanan hakukenttään. Verkkosivuilla on myös laaja suodatusvalikoima, jonka avulla voi kaventaa hakua. Asset Storeen voi kirjautua omalla Unity ID -käyttäjällä, joka mahdollistaa erilaisten ominaisuuksien käytön, kuten lataushistorian, tuotosten tykkäämisen, oman ostoskorin hallinnan, Asset Storen sovellusten käytön, sekä oman käyttäjätilin hallinnan. (Unity Technologies 2024.)

4 Simulaatio

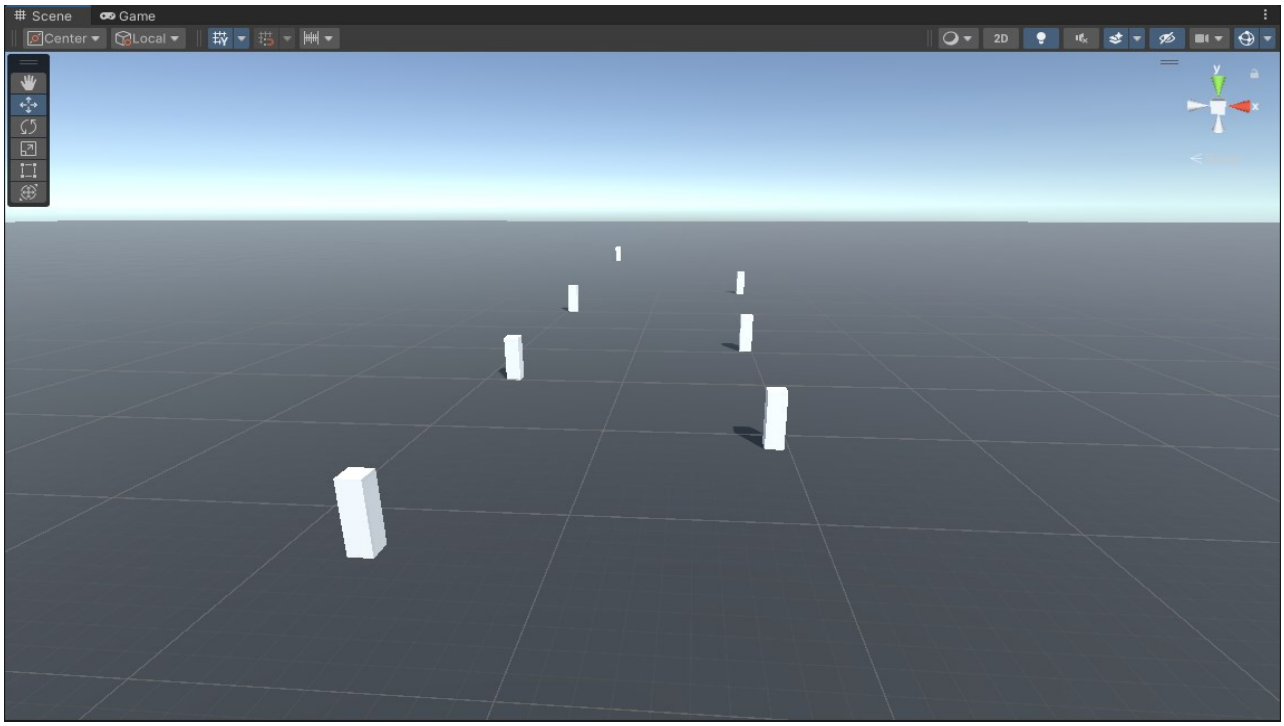
Tässä opinnäytetyössä luodaan yksinkertainen ajosimulaatio Unity-pelimoottorin avulla. Työssä pyritään jäljittelemään auton aidontuntuista käyttäytymistä simuloimalla auton oleellisia ominaisuuksia. Projektin jokainen vaihe on selitetty, jotta kuka tahansa ymmärtäisi työn kehityksen kulun, tai voisi luoda samanlaisen projektin itse.

4.1 Projektin luominen

Unity-projektin luominen aloitetaan Unity Hub -sovelluksesta. Unity Hubissa oikealla ylhäällä on sininen painike nimeltä New project. Sitä painamalla pääsee valitsemaan, minkälaisen projektin haluaa luoda. Tähän projektiin käytetään 3D (Built-in Render Pipeline) -mallia. Seuraavaksi projektille annetaan nimi kirjoittamalla se oikealla Project settings-otsikon alla olevaan Project name -kenttään. Location-kentästä voi muuttaa tiedoston sijaintia, jos haluaa. Unity Organization -kentästä valitaan oma käyttäjätunnus. Viimeiseksi painetaan Create project -painiketta, jolloin Unity aloittaa projektin rakentamisen.

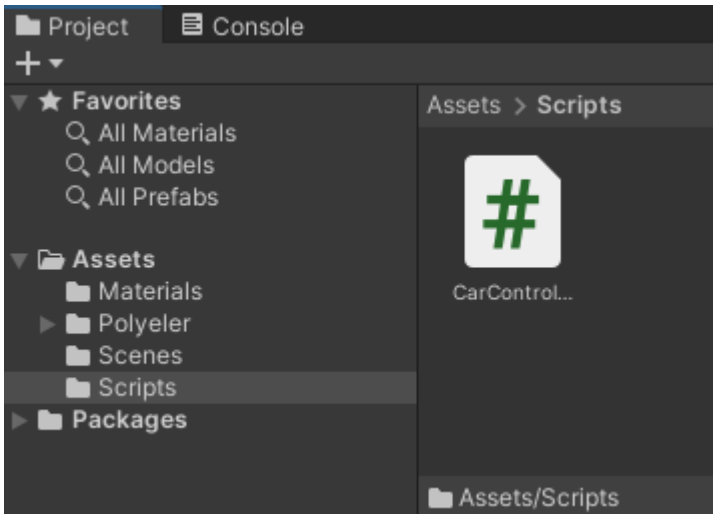
4.2 Projektin alustaminen

Ensimmäiseksi projekti tarvitsee ympäristön, missä voi testata simulaation ominaisuuksia. Tämän työn testiympäristö on todella yksinkertainen. Ajoalustana toimii plane -kolmiulotteinen peliobjekti, jonka saa luotua painamalla hiiren oikealla painikkeella Unity editorin Hierarkia-näkymää ja valitsemalla 3D Object ja Plane, eli taso. Peliobjektin kokoa voi muokata Inspektori-näkymästä kohdasta Scale. Tämän työn tason koon x ja y arvoiksi laitetaan 100, jotta tilaa olisi tarpeeksi. Tasolle kannattaa asetella muita kolmiulotteisia objekteja, jotta auton liikkeet on helpompi hahmottaa. (Kuva 2.)



Kuva 2. Simulaation testiympäristö

Simulaatio tarvitsee myös kolmiulotteisen automallin. Sellaisen voi joko tehdä itse, tai ladata Unity Asset Storesta. Tässä työssä käytetään ilmaista Simple Retro Car-mallia, jonka tekijä on Polyeler (Polyeler 2024). Kun malli on ladattu, sen voi tuoda omaan projektiin Package managerista. Package manageriin pääsee painamalla työkalupalkista ensin Window, ja sitten Package manager. Package managerissa valitaan Packages: My Assets, jonka jälkeen juuri ladatun paketin pitäisi tulla näkyviin. Seuraavaksi painetaan ensin paketin nimeä ja sitten Import painiketta managerin ylhäällä oikealla. Paketti löytyy nyt projekti-ikkunasta omasta kansiosta. Jotta mallin saa käyttöön, Projekti-näkymässä täytyy mennä Assets, jonka jälkeen Polyeler ja Prefabs. Malli tulee näkyviin Projekti-näkymään. Mallin voi raahata Skene-näkymään hiiren vasemmalla painikkeella. Seuraavaksi kamera kiinnitetään automalliin. Tämä tehdään Hierarkia-näkymässä raahaamalla Main Camera hiiren vasemmalla painikkeella Simple Retro Car:in sisälle. Näin kamera saadaan seuraamaan autoa ilman skriptaamista. Skriptien tekeminen on kuitenkin edellytys melkein kaikissa Unity-projekteissa. Skripteille on hyvä tapa tehdä oma kansio painamalla hiiren oikealla painikkeella Projekti-näkymässä Assets, Create ja Folder. Yleinen nimi skriptikansiolle on Scripts. Tässä työssä kansion sisään luodaan yksi C# skriptitiedosto nimeltä CarController, joka sisältää kaikki vaadittavaa skriptit. (Kuva 3.)



Kuva 3. Projektin kansiot

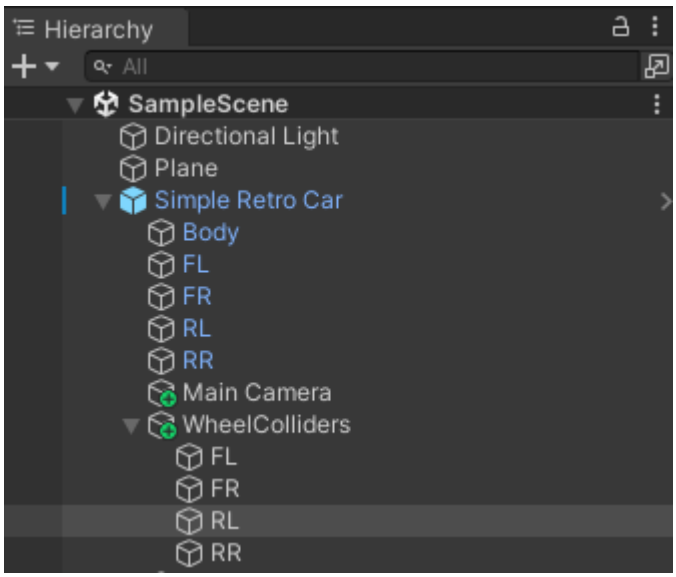
4.3 Renkaiden simulointi

Renkaiden realistinen toiminta on mahdollisesti tärkein ominaisuus ajosimulaatiossa. Ilman simuloituja renkaita autosta on hankala saada aidon tuntuista. Unitylla on realistisia renkaita varten luotu Wheel Collider-komponentti. Wheel Collider on käytännössä näkymätön renkaan muotoinen peliobjekti, mikä asetetaan jokaisen renkaan kohdalle. Wheel Colliderin avulla autoa voi liikuttaa pyörittämällä renkaita sen sijaan, että luotaisiin voima, joka työntäisi leijuvaa autoa, joka näyttäisi olevan maassa, eteenpäin. Wheel Collider saa myös auton kääntymään renkaiden kääntyessä.

Ennen kuin automallille voidaan antaa Wheel Colliderit, se tarvitsee Rigidbody-komponentin. Tämän tehdäkseen täytyy ensin painaa automallin nimeä Hierarkia-näkymässä, jonka jälkeen Add Component-painiketta Inspektori-näkymässä. Haku-kenttään kirjoitetaan Rigidbody ja valitaan se. Rigidbody antaa mallille fyysisen kehon, joka reagoi siihen kohdistettuihin voimiin. Rigidbodyn ensimmäinen asetus on Mass, eli massa. Tämä laskee kehon massan kilogrammoissa. Tässä työssä massan arvoksi annetaan 1500. 1500 kilogrammaa on suunnilleen keskivertoauton paino ja se toimii hyvin simulaation alkuvaiheen testaamisessa.

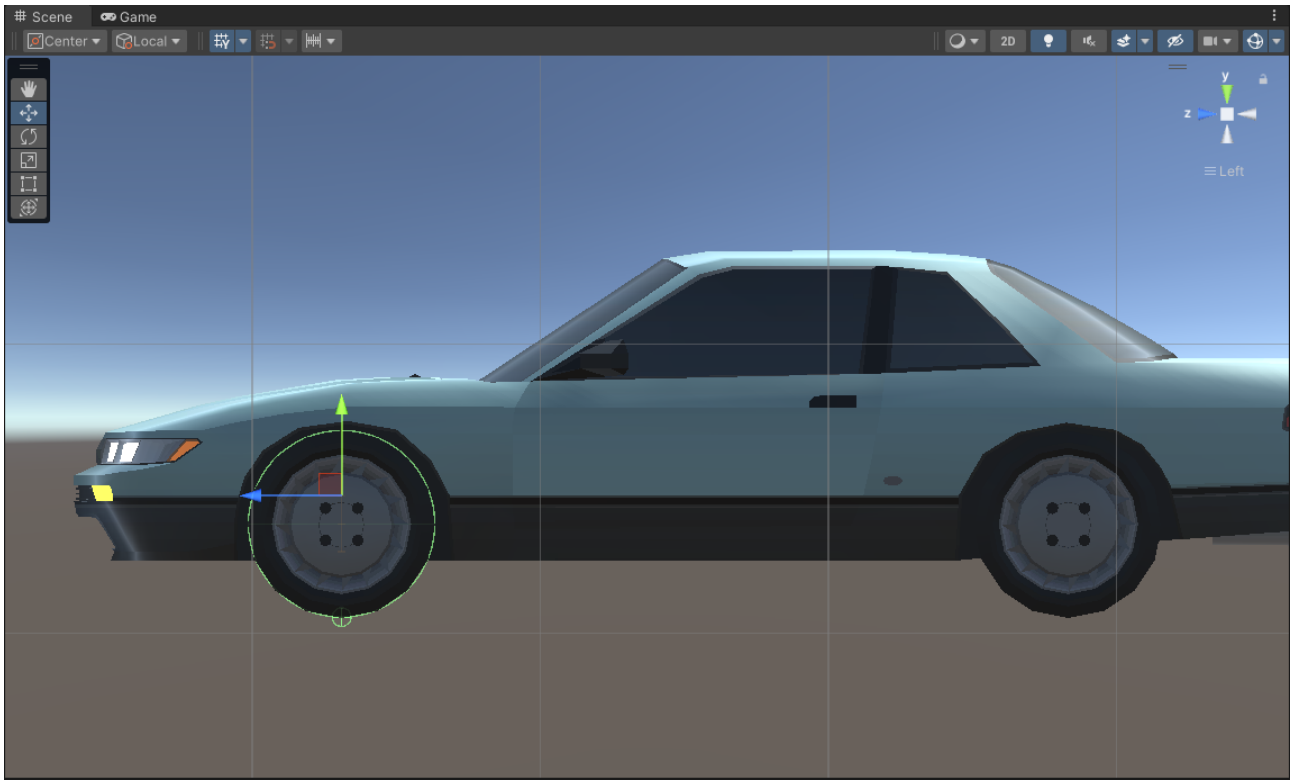
Simple Retro Car -malli sisältää tekstuurit erikseen auton keholle sekä renkaille. Tämä tekee Wheel Colliderien asettelemisesta helppoa. Ensin luodaan tyhjä peliobjekti mallin sisälle Hierarkia-näkymässä painamalla mallin nimeä hiiren oikealla painikkeella ja valitsemalla Create Empty. Tyhjän peliobjektin nimeksi tulee WheelColliders. Seuraavaksi valitaan renkaat pitämällä Shift-painiketta näppäimistössä, ja painamalla ensimmäistä ja viimeistä rengasta. Renkaiden nimet ovat tässä tapauksessa FL eli etuvasen, FR eli etuoikea, RL eli takavasen sekä RR eli takaoikea. Kun renkaat on valittu, ne kopioidaan painamalla Ctrl ja C painikkeet samaan aikaan. Sitten renkaat

liitetään juuri luotuun tyhjään peliobjektiin painamalla sitä, jonka jälkeen painetaan Ctrl- ja V-painikkeita. (Kuva 4.)



Kuva 4. Wheel Colliders-objektit

WheelColliders-objektin sisällä olevista renkaista täytyy poistaa tekstuurit, jotka tulivat kopioinnissa. Taas valitaan kaikki neljä rengasobjektia WheelColliders-objektin sisällä, jonka jälkeen objektien komponentit tulevat näkyviin Inspektori-näkymässä. Jokaisen komponentin oikeassa ylälaidassa on kolme pistettä. Pisteitä painamalla saa esille listan, jonka joukossa on Remove Component-valinta, joka poistaa kyseisen komponentin objektilta. Valituilta renkailta poistetaan kaikki komponentit, jonka jälkeen niille lisätään Wheel Collider-komponentit. Kaikki renkaat valittuina painetaan Inspektori-näkymässä Add Component painiketta ja kirjoitetaan hakukenttään Wheel Collider. Painamalla Wheel Collider -hakutulosta jokainen rengasobjekti saa Wheel Collider-komponentin. Wheel Colliderit täytyy vielä asettaa renkaiden kohdalle. Tämä vaihe täytyy tehdä jokaiselle renkaalle erikseen, jotta Wheel Colliderit saadaan juuri oikeaan kohtaan. Tässä vaiheessa kannattaa katsoa autoa suoraan sivusta painamalla Skene-näkymän oikeassa yläkulmassa olevaa Orientation-kuutioita ja kääntämällä kuvakulma auton sivulle. Skene-näkymän vasenyläkulmasta valitaan Move Tool, jolloin Wheel Collider voidaan asettaa haluttuun kohtaan. Samasta paikasta valitaan myös Scale Tool, jolla asetetaan Wheel Collider oikean kokoiseksi. Wheel Collider kannattaa asettaa keskelle myös sivusuunnassa, jotta auto toimii halutulla tavalla. (Kuva 5.)



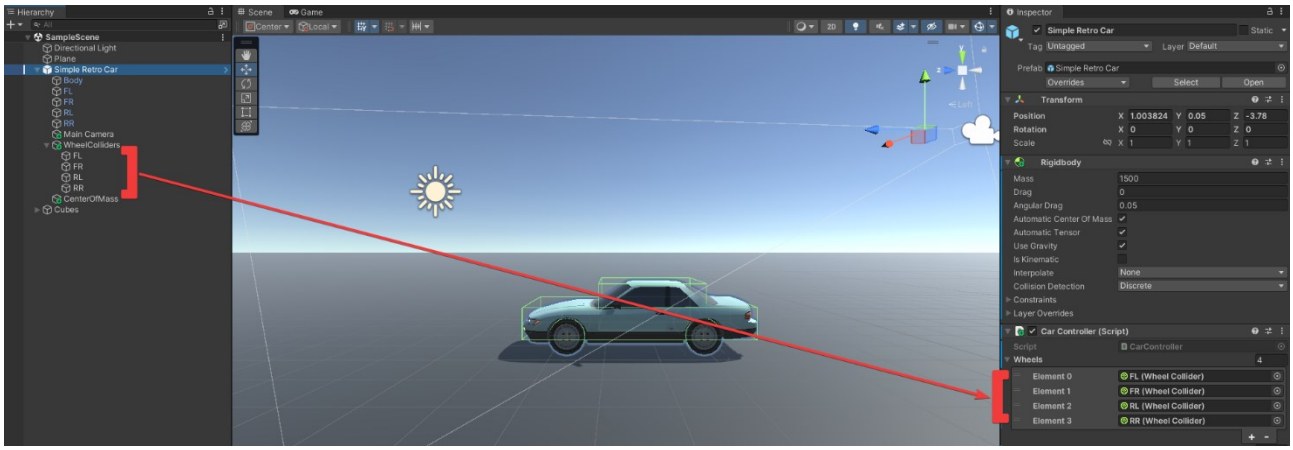
Kuva 5. Wheel Collider-objektin asettelu

Jotta Wheel Collidereita voisi käyttää skriptissä, niistä täytyy tehdä objekti aikaisemmin luotuun CarController -skriptiin. Objektista tehdään lista, jonka nimeksi tulee “wheels”. Listasta tehdään julkinen, jotta se näkyy Inspektori-näkymässä. Objektista voisi tehdä myös yksityisen ja kirjoittaa sen eteen “[SerializeField]”, jotta se näkyisi Inspektori-näkymässä. Tässä työssä tullaan käyttämään julkisia ja yksityisiä attribuutteja. (Kuva 6.)

```
public class CarController : MonoBehaviour
{

    14 references
    public WheelCollider[] wheels = new WheelCollider[4];
```

Kuva 6. WheelCollider-objekti

CarController ei silti vielä tunnista Wheel Collidereita. Hierarkia-näkymään tehdyt Wheel Collider-objektit täytyy raahata CarControllerin listaan Inspektori-näkymään. Objektit lisätään listaan yksitellen samassa järjestyksessä, missä ne näkyvät Hierarkia-näkymässä. (Kuva 7.)



Kuva 7. WheelCollider-objektien raahaaminen listaan

Kun Wheel Colliderit on alustettu, skriptiin voidaan luoda “Drive”- ja “Brake”-funktiot. Drive-funktio ottaa käyttäjän syötteen ja siirtää voiman renkaisiin. Myöhemmin voiman tuottaa simuloitu moottori, mutta tässä vaiheessa voima on pelkkä staattinen arvo. Brake-funktio säätää jarrutusvoimaa käyttäjän syötteen perusteella.

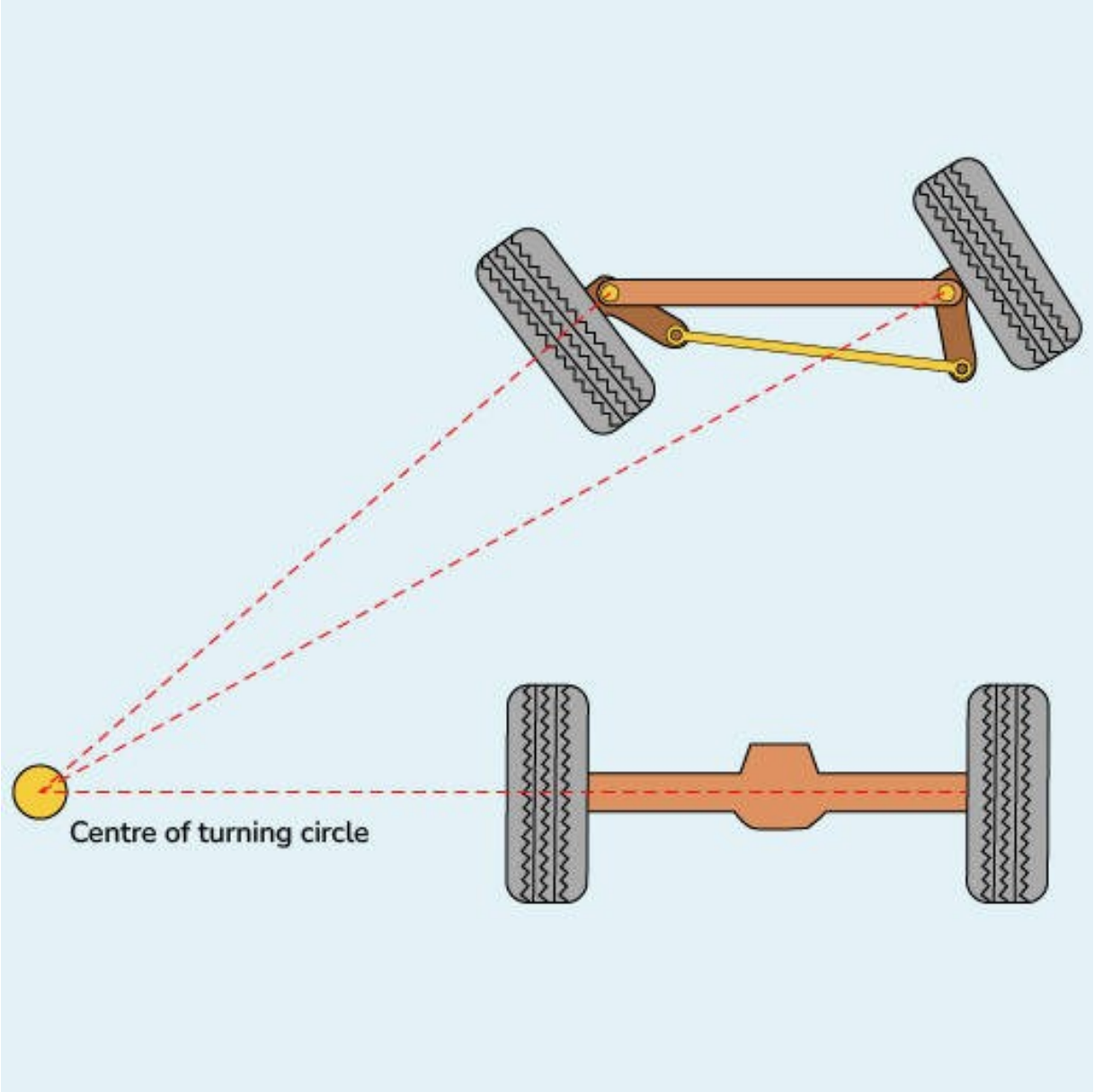
Näissä funktioissa käytetyt muuttujat täytyy ensin alustaa. Ensin luodaan public float “totalPower”, minkä arvoksi kannattaa antaa luku 200 ja 1000 välillä. Seuraavaksi luodaan private float “brakePower”, jolle ei tarvitse antaa arvoa. Käyttäjän syötteitä varten luodaan private float “vertical” ja “horizontal”. Horizontal-muuttujaa käytetään myöhemmin, kun luodaan funktio auton kääntymiselle. Drive-funktion alussa määritetään vertical-muuttuja ottamaan käyttäjän vertikaaliset syötteet. Tämä tarkoittaa näppäimistön w- ja s-painikkeita. Painamalla w-painiketta verticalin arvosta tulee positiivinen ja painamalla s-painiketta siitä tulee negatiivinen. Drive-funktio tarkistaa, onko verticalin arvo positiivinen. Jos on, se lähettää voiman renkaille for-loopin avulla. Tässä työssä autosta tehdään takavetoinen, joten for-loop laitetaan alkamaan numerosta kaksi. Näin voima siirretään vain listan kahdelle viimeiselle renkaalle. TotalPower-muuttuja jaetaan kahdella, jotta molemmat renkaat eivät saa täyttä voimaa. Drive-funktiossa jaetaan myös jarrutusvoima tasaisesti kaikille renkaille. Brake-funktio tarkistaa, onko vertical negatiivinen tai 0. Jos vertical on negatiivinen, eli käyttäjä painaa s-painiketta, lisätään paljon jarrutusvoimaa antamalla brakePowerin arvoksi 1500. Jos verticalin arvo on nolla, eli käyttäjä ei paina w- eikä s-painikkeita, annetaan brakePowerin arvoksi 500. Tämä simuloi moottorijarrutusta, eli kun ajaja ei paina mitään poljinta moottori hidastaa autoa hieman. Lopuksi määritetään jarrutusvoima muissa tilanteissa nollaksi, jotta auto ei jarruta, kun käyttäjä haluaa liikkua eteenpäin. Brake-funktiota kutsutaan Drive-funktion sisällä, joten Update-funktiossa tarvitsee kutsua vain Drive-funktiota. (Kuva 8.)

```
1 reference
private void Drive()
{
    Brake();
    vertical = Input.GetAxis("Vertical");

    if (vertical > 0)
    {
        for (int i = 2; i < wheels.Length; i++)
        {
            wheels[i].motorTorque = vertical * (totalPower / 2);
        }
    }
    for (int i = 0; i < wheels.Length; i++)
    {
        wheels[i].brakeTorque = brakePower;
    }
}

1 reference
private void Brake()
{
    if (vertical < 0)
    {
        brakePower = 1500;
    }
    else if (vertical == 0)
    {
        brakePower = 500;
    }
    else
    {
        brakePower = 0;
    }
}
```

Kuva 8. Drive- ja Brake-funktiot

4.3.1 Ackermannin ohjausjärjestelmä

Auton kääntyminen on olennainen osa simulaatiota ja mitä paremmin se toimii, sitä paremmalta ajaminen tuntuu. Kääntymisen voisi ohjelmoida siten, että renkaat kääntyvät yhtä suuressa kulmassa, mutta tämä tuottaisi ongelmia. Kun auto kääntyy, sisempi rengas kulkee lyhyemmän matkan kuin ulompi rengas. Jos molemmat renkaat kääntyvät samassa kulmassa, ne yrittävät kulkea yhtä pitkän matkan, mikä ei ole mahdollista, joten toinen rengas menettää pidon. Ackermannin ohjausjärjestelmä tarjoaa ratkaisun tähän ongelmaan. Ackermannin ohjausjärjestelmä toimii siten, että kääntyessään auton renkaan sivut osoittavat samaan kuvitteelliseen pisteeseen. Tällöin sisempi rengas pystyy kulkemaan pienemmän matkan kuin ulompi rengas ja renkaat kykenevät pitämään pidon. (Kuva 9.) (Skill-lync 2024.)



Kuva 9. Havainnekuva Ackermannin ohjausjärjestelmästä (Skill-lync 2024)

Seuraavaksi tehdään Steer-funktio, jonka tehtävä on kääntää autoa hyödyntäen Ackermannin ohjausjärjestelmää. Funktion alussa määritetään muuttuja “horizontal”, joka lukee käyttäjän horisontaalisella akselilla annetun syötteen. Sitten tehdään ehtorakenne, joka tarkastaa, onko horisontaalinen arvo positiivinen vai negatiivinen ja sen perusteella kääntää renkaita, käyttäen Ackermannin ohjausjärjestelmä kaavaa. Jos horisontaalinen arvo on nolla, funktio pitää renkaat suorassa. (Kuva 10.)

```
private void Steer()
{
    horizontal = Input.GetAxis("Horizontal");

    if (horizontal > 0)
    {
        wheels[0].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (radius + (1.5f / 2))) * horizontal;
        wheels[1].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (radius - (1.5f / 2))) * horizontal;
    }
    else if (horizontal < 0)
    {
        wheels[0].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (radius - (1.5f / 2))) * horizontal;
        wheels[1].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (radius + (1.5f / 2))) * horizontal;
    }
    else
    {
        wheels[0].steerAngle = 0;
        wheels[1].steerAngle = 0;
    }
}
```

Kuva 10. Ackermannin ohjausjärjestelmän kaava

4.3.2 Luiston lisääminen

Renkaiden kyky pysyä kiinni tiessä ei ole ääretön. Realismin lisäämiseksi auton täytyy pystyä menettämään pito, jos se kääntyy sivuttain tarpeeksi suurissa nopeuksissa. Tätä varten luodaan addSlip-funktio. Ensin täytyy määrittää muuttuja slip. Muuttujasta tehdään julkinen lista, jotta se pystyy lisäämään luiston kaikkiin renkaisiin. Slip-lista tulee näkymään Inspektori-näkymässä, ja siitä näkee, kuinka paljon renkaille lisätään luistoa. (Kuva 11.)

```
public float[] slip = new float[4];
```

Kuva 11. Slip-muuttujan alustaminen

Funktio käy for loopilla renkaat läpi ja tarkistaa renkaiden osuman maan kanssa. Jos renkaat ovat sivusuunnassa auton menosuuntaan kohden, funktio lisää luistoa renkaisiin, jolloin auto menettää pidon. (Kuva 12.)

```
private void addSlip()
{
    for (int i = 0; i < wheels.Length; i++)
    {
        WheelHit wheelHit;
        wheels[i].GetGroundHit(out wheelHit);

        slip[i] = wheelHit.sidewaysSlip;
    }
}
```

Kuva 12. Luiston lisääminen

4.3.3 Renkaiden animointi

Työn ainoa esteettinen ominaisuus on renkaiden animointi. Animoidut renkaat helpottavat testaamista, sillä renkaiden liikkeistä näkee, liikkuuko auto halutulla tavalla. Valmiiksi animoidut renkaat helpottavat myös jatkokehitystä. Renkaiden animointi vaatii mallinnetut renkaat. Työssä käytetyssä automallissa on valmiiksi eritelty renkaat. Jotta CarController-skripti pystyy tunnistamaan renkaat, on luotava samankaltainen lista, kuin wheel collidereille. Sen sijaan, että uusi lista olisi julkinen wheel collider, siitä tehdään julkinen peliobjekti, jonka nimeksi tulee wheelMesh. (Kuva 13.)

```
public GameObject[] wheelMesh = new GameObject[4];
```

Kuva 13. WheelMesh-objekti

Inspektori-näkymässä CarControllerin alla näkyy nyt Wheel Mesh-lista, mihin valmiiksi mallinnetut renkaat raahataan samalla tavalla, kuin aikaisemmin wheel colliderien kanssa. Nyt CarController tunnistaa renkaat, joten voidaan luoda AnimateWheels niminen funktio, joka hoitaa renkaiden animoinnin. (Kuva 14.)

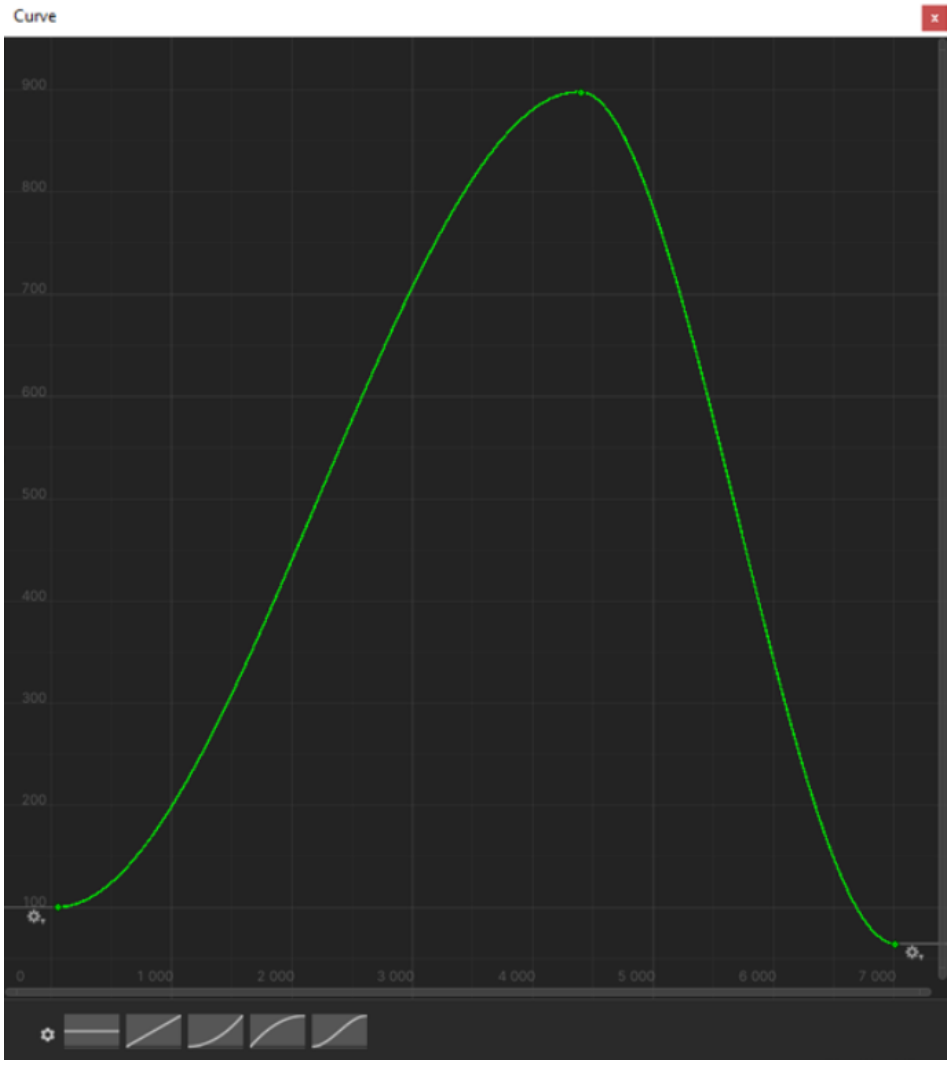
```
private void AnimateWheels()
{
    Vector3 wheelPosition = Vector3.zero;
    Quaternion wheelRotation = Quaternion.identity;

    for (int i = 0; i < 4; i++)
    {
        wheels[i].GetWorldPose(out wheelPosition, out wheelRotation);
        wheelMesh[i].transform.position = wheelPosition;
        wheelMesh[i].transform.rotation = wheelRotation;
    }
}
```

Kuva 14. Renkaiden animointi

Ensin funktiossa määritetään Vector3 ja Quaternion. Vector3.zero määrittää renkaan keskipisteen, jonka ympäri se pyörii. Quaternion.identity määrittää, että rengas pyörii suhteessa sen vanhempaan, eli hierarkiassa ylempänä olevaan objektiin verrattuna. Funktio käy renkaat läpi for loopilla ja pyörittää, sekä kääntää renkaita riippuen auton sen hetken liikkeestä.

4.4 Moottorin simulointi

Tässä vaiheessa auto saa voiman aikaisemmin luodusta staattisesta muuttujasta. Tarkoituksena on simuloida moottori, joka tuottaa voimaa dynaamisesti riippuen moottorin voimakurvista. Moottorin voimakurvi näyttää, kuinka paljon vääntöä se tuottaa moottorin sen hetken kierroksilla. Unityssa voimakurvin voi luoda animaatiokurvin avulla. CarControllerin alkuun määritetään julkinen AnimationCurve EnginePower, jolloin kurvi tulee näkyviin Inspektori-näkymään. (Kuva 15.)



Kuva 15. Animaatiokurvin avulla simuloitu moottorin voimakurvi

Aikaisemmin luodusta totalPower-muuttujasta voi ottaa staattisen luvun pois, sillä animaatiokurvi tulee määrittämään voiman määrän. TotalPower-muuttuja kuitenkin säästetään ja pidetään julkisena, sillä se näyttää voiman määrän Inspektori-näkymässä testauksen aikana.

Moottorin kierrosten laskemiseksi täytyy ensin laskea renkaiden kierrokset wheelRPM-funktiolla. Funktiota varten määritetään ensin yksityinen float wheelsRPM. Funktio toimii siten, että for loopin sisällä renkaiden kierrokset lisätään sum-muuttujaan. For loopin kierrosten määrä tallennetaan R-muuttujaan. Jos R:n arvo on nolla, renkaiden kierrosluku on myös nolla, eikä summaa jaeta nollalla. (Kuva 16.)

```
private void wheelRPM()
{
    float sum = 0;
    int R = 0;
    for (int i = 0; i < 4; i++)
    {
        sum += wheels[i].rpm;
        R++;
    }
    wheelsRPM = (R != 0) ? sum / R : 0;
}
```

Kuva 16. Renkaiden kierrosten laskeminen

Nyt voidaan luoda funktio CalculateEnginePower, joka laskee moottorin voimansyötön sekä kierrokset. Kuten kuvasta 16 näkee, moottorin simuloinnissa käytetään gearShifter-funktiota, eli vaihteistoa. Vaihteiston simulointi käsitellään seuraavassa alaluvussa. CalculateEnginePower-funktiota varten täytyy määritellä yksityiset muuttujat engineRPM ja smoothTime. SmoothTime-muuttujan arvoksi annetaan 0.09f. Funktion sisällä määritetään float velocity, minkä arvoksi annetaan nolla. Funktio laskee moottorin kierrosmäärän käyttäen wheelsRPM-funktiota, velocity ja smoothTime muuttujia, sekä sen hetkistä vaihdetta. Moottorin voima lasketaan aikaisemmin luodun animaatiokurvin, moottorin kierrosten, vaihteiston, sekä käyttäjän syötteen avulla. WheelRPM, Drive sekä gearShifter-funktioita kutsutaan CalculateEnginePower-funktion sisällä. (Kuva 17.)

```
private void CalculateEnginePower()
{
    wheelRPM();
    Drive();
    gearShifter();

    totalPower = EnginePower.Evaluate(engineRPM) * (gears[gearNum]) * vertical;
    float velocity = 0.0f;
    engineRPM = Mathf.SmoothDamp(engineRPM, 1000 + (Mathf.Abs(wheelsRPM) * 3.6f * (gears[gearNum])), ref velocity, smoothTime);
}
```

Kuva 17. Moottorin simulointi

4.5 Vaihteiston simulointi

Kuten aikaisemmassa alaluvussa mainittiin, moottorin simulointi vaatii toimiakseen vaihteiston. Tätä varten luodaan yksinkertainen gearShifter-funktio. Vaihteisto vaatii myös listan, joka sisältää kaikki vaihteet, sekä muuttujan aktiiviselle vaihteelle. (Kuva 18.)

```
public float[] gears;
6 references
public int gearNum = 0;
3 references
```

Kuva 18. Vaihteiston muuttujat

Gears-lista näkyy nyt Inspektori-näkymässä. Ensimmäiseen kenttään annettava luku määrää vaihteiden määrän, tässä työssä vaihteiden määräksi annetaan viisi. Kun vaihteiden määrä on annettu, listaan ilmestyvät elementit, jotka kuvaavat vaihteita. Jokaiselle vaihteelle on annettava arvo, joka kuvaa vaihdesuhdetta. Vaihdesuhde tarkoittaa kuinka monta kierrosta vaihderatas pyörii oman akselin ympäri siinä ajassa, missä seuraava vaihderatas pyörähtää kerran itsensä ympäri. Tässä työssä käytettävät vaihdesuhteet näkyvät tässä kuvassa. (Kuva 19.)

Kuva 19. Vaihteet ja vaihdesuhteet

Nyt kun vaihteistolle on tehty peliobjekti, voidaan tehdä gearShifter-funktio, joka vaihtaa vaihteita käyttäjän syötteen perusteella. Funktio koostuu kahdesta ehtolauseesta. Ensimmäinen ehtolause vaihtaa isommalle vaihteelle, jos käyttäjä painaa E-painiketta ja jos sen hetkinen vaihde ei ole suurin vaihde. Toinen ehtolause vaihtaa vaihdetta pienemmälle, jos käyttäjä painaa Q-painiketta ja jos sen hetkinen vaihde ei ole pienin vaihde. (Kuva 20.)

```
private void gearShifter()
{
    if (Input.GetKeyDown(KeyCode.E) && gearNum < gears.Length - 1)
    {
        gearNum++;
    }
    if (Input.GetKeyDown(KeyCode.Q) && gearNum > 0)
    {
        gearNum--;
    }
}
```

Kuva 20. Vaihteiden vaihtaminen

4.6 Pitovoiman simulointi

Viimeisenä ominaisuutena simulaatioon lisätään pitovoima. Pitovoima on vertikaalinen voima, joka työntää autoa alaspäin. Kun auto liikkuu tarpeeksi nopeasti ilman läpi, ilmavirta kulkee auton virtaviivaisten muotojen yli työntäen autoa alaspäin. Pitovoiman ansiosta auton renkaat työntyvät vahvemmin tiehen, jolloin renkaiden pito on myös vahvempi, varsinkin kääntyessä. (Mercedesamgf1)

Pitovoiman luomiseksi autolle täytyy luoda massankeskipiste. Hierarkia-näkymässä Simple Retro Carin sisälle luodaan uusi tyhjä peliobjekti, jonka nimeksi annetaan CenterOfMass. Uusi peliobjekti asetetaan mahdollisimman matalalle ja keskelle autoa. Seuraavaksi CarController skriptissä määritetään uusi rigidbody-objekti sekä centerOfMass-peliobjekti. (Kuva 21.)

```
private Rigidbody rigidbody;
2 references
private GameObject centerOfMass;
```

Kuva 21. Massankeskipisteen alustaminen

Start-funktiossa määritetään juuri luotujen peliobjektien tehtävät. Rigidbody määrittää auton rungon. CenterOfMass etsii aikaisemmin luodun tyhjän peliobjektin. Näiden peliobjektien avulla skripti pystyy aina seuraamaan missä massankeskipiste on. Start-funktiota kutsutaan ensimmäisenä ohjelman käynnistyksessä, joten skripti löytää massankeskipisteen heti. (Kuva 22.)

```
void Start()
{
    rigidbody = GetComponent<Rigidbody>();
    centerOfMass = GameObject.Find("CenterOfMass");
    rigidbody.centerOfMass = centerOfMass.transform.localPosition;
}
```

Kuva 22. Massankeskipisteen määrittäminen

Kun massankeskipiste on määritetty, voidaan autolle lisätä pitovoimaa. Pitovoimalle täytyy luoda julkinen float downForce, joka määrittää pitovoiman määrän. Tässä työssä pitovoiman määräksi annetaan 50. Seuraavaksi luodaan addDownForce-funktio, joka lisää pitovoimaa auton massankeskipisteeseen. Funktio kasvattaa pitovoimaa, mitä nopeammin auto kulkee. (Kuva 23.)

```
private void addDownForce()
{
    rigidbody.AddForce(-transform.up * downForce * rigidbody.velocity.magnitude);
}
```

Kuva 23. Pitovoiman lisääminen

5 Pohdinta

Opinnäytetyön tuloksena valmistui ajosimulaatio, joka sisältää suunnitteluvaiheessa määritetyt ominaisuudet. Auto käyttäytyy halutulla tavalla, imitoiden karkeasti aidon auton käyttäytymistä. Joissain tilanteissa auton käyttäytyminen tuntuu kuitenkin oudolle. Vaikka autolle annettiin melko paljon voimaa, se kiihtyy yllättävän hitaasti ensimmäisellä vaihteella. Asian korjaamiseksi moottorin voimakäyrän, sekä vaihteiston arvoja täytyisi muuttaa, kunnes kiihtyvyys tuntuisi hyvältä. Näppäimistön käyttäminen testaamisessa toimi rajoittavana tekijänä, sillä autoa pystyi kääntämään joko täysillä, tai ei ollenkaan. Tämä johti auton pidon hetkelliseen menettämiseen korkeissa nopeuksissa kääntyessä. Reaktio itsessään on täysin realistinen, mutta simulaation kehittämisen kannalta tarkkoihin syötteisiin kykenevä laitteisto olisi kannattavaa. Toinen yllättävä asia oli, kuinka pitovoiman lisääminen sai auton hieman heilumaan korkeissa nopeuksissa. Heilunta ei ollut suunniteltu, mutta se sattumalta simuloi pitovoiman vaikutusta paremmin, kuin oli tarkoitus. Kun pitovoima työntää autoa alaspäin, jouset taistelevat tätä voimaa vastaan, mikä aiheuttaa auton heilunnan.

Työn lopputuloksessa on kaikki simulaation perusominaisuudet, mutta se on vielä raaka tuotos. Se vaatii jatkokehittämistä ja hienosäätöä, jos siitä haluaa julkaisukykyisen. Jatkokehittämistä miettiessä työstä löytyy epäkohta. Vaihteistossa ei ole neutraalia-, eikä peruutusvaihdetta. Seuraavat kehityskohdat olisivat siis vaihteiston korjaaminen ja moottorin voimakäyrän hienosäätäminen. Lisäyksenä simulaatio kaipaisi kameraskiptiä kokemuksen parantamiseksi. Myös skriptit ja peliobjektit tulisi tehdä siten, että niitä voisi käyttää useammassa eri autossa, jotta jokaiselle uudelle autolle ei tarvitsisi luoda kaikkea alusta asti. Tuotos on kuitenkin hyvä pohja täyden ajosimulaation tekemiseen, mikä voisi yhdistää realistisen ajotuntuman ja videopelin hauskuuden.

Opinnäytetyön tekeminen on ollut oppimiskokemuksena antoisa. Vaikka omasin perustason tietämyksen työn aiheesta ennen sen aloittamista, työn tekemisen aikana päädyin tutkimaan aihealuetta tarkemmin, syventäen omaa osaamista. Projektin tekeminen Unitylla on vahvistanut aikaisempaa osaamista kyseisellä pelimoottorilla. Kokemus pelimoottorin käytöstä voi johtaa moneen suuntaan uralla ja vaikka ei johtaisikaan, siitä tulee vähintään antoisa harrastus.

Lähteet

Baxter, R. 22.12.2022. The First Racing Simulator Ever Built. Luettavissa: https://simracingcockpit.gg/the-first-racing-simulator-ever-built/. Luettu 27.11.2024.

Bolton, D. 7.1.2019. Definition of Int in C, C++ and C#. Luettavissa: https://www.thoughtco.com/definition-of-int-958297. Luettu 5.12.2024.

Bolton, D. 3.5.2019. Definition of Float in C, C++ and C#. Luettavissa: https://www.thoughtco.com/definition-of-float-958293. Luettu 5.12.2024.

ByteHide 28.12.2023. For Loop in C#: Concept and Implementation. Luettavissa: https://www.bytehide.com/blog/for-loop-csharp. Luettu 5.12.2024.

Crivello, A. s.a. Game Engines. Luettavissa: https://www.g2.com/glossary/game-engines-definition. Luettu 4.12.2024.

Hilton, C. 16.6.2021. Collider's and Triggers in Unity – Understanding the Basics. Luettavissa: https://christopherhilton88.medium.com/colliders-and-triggers-in-unity-understanding-the-basics-7192714f3440. Luettu 5.12.2024.

Jerga, F. 31.5.2021. Unity Rigidbody Explained. Luettavissa: https://medium.com/eincode/unity-rigidbody-explained-fb208d0f97f3. Luettu 5.12.2024.

Leporati, G. 16.8.2024. What it's like to sim race against Max Verstappen. Luettavissa: https://www.motorsport.com/f1/news/sim-racing-against-max-verstappen/10643407/. Luettu 27.11.2024.

Mercedesamgf1 s.a. Feature: Downforce in Formula One, Explained. Luettavissa: https://www.mercedesamgf1.com/news/feature-downforce-in-formula-one-explained. Luettu 25.11.2024.

Morse, P. 30.3.2015. Basic Facts about the 3 Main Types of Driving Simulators. Luettavissa: https://www.ansiblemotion.com/automotive-driver-in-the-loop-simulation-articles/3-main-types-of-driving-simulators. Luettu 27.11.2024.

Morse, P. 28.2.2019. Driver-in-the-loop Simulators: Who's the Driver? Luettavissa: https://www.ansiblemotion.com/automotive-driver-in-the-loop-simulation-articles/who-drives-driver-in-the-loop-simulators. Luettu 27.11.2024.

Occa Software 27.12.2023. Unity: What Is a Scene. Luettavissa: https://www.occasoftware.com/blog/unity-what-is-a-scene. Luettu 5.12.2024.

Pluralsight 13.11.2022. What is C# Programming? A Beginners Guide. Luettavissa: https://www.pluralsight.com/resources/blog/software-development/everything-you-need-to-know-about-c-. Luettu 5.12.2024.

Polyeler 7.8.2024. Simple Retro Car. Luettavissa: https://assetstore.unity.com/packages/3d/vehicles/simple-retro-car-291522. Luettu 11.11.2024.

Skill-lync 2024. Ackermann's Principle of Steering: Working & Applications. Luettavissa: https://skill-lync.com/blogs/ackermans-principle-of-steering-working-and-applications. Luettu 15.11.2024.

Starloop 2021. What is a Unity GameObject, and How Do You Fit It Into Your Game? Luettavissa: https://starloopstudios.com/what-is-a-unity-gameobject-and-how-do-you-fit-it-into-your-game/. Luettu 5.12.2024.

The Daily Downforce 13.10.2024. How real Nascar drivers use sim racing. Luettavissa: https://dailydownforce.com/how-real-nascar-drivers-use-sim-racing/. Luettu 27.11.2024.

Unity Technologies 2024. Unity's Asset Store. Luettavissa: https://docs.unity3d.com/Manual/AssetStore.html. Luettu 4.12.2024.

Unity Technologies 2024. Introduction to components. Luettavissa: https://docs.unity3d.com/Manual/Components.html. Luettu 5.12.2024.

York driving simulators s.a. What is a driving simulator? Luettavissa: https://yctsims.com/what-is-a-driving-simulator. Luettu 27.11.2024.

Zenva 19.9.2024. What is Unity? A Top Game Engine for Video Games. Luettavissa: https://gamedevacademy.org/what-is-unity/. Luettu 3.12.2024.