Fysiikan toteuttaminen Unity-pelimoottorissa

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus

kevät 2022

Joona Kari

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus — Tiivistelmä

Tekijä Joona Kari
Vuosi 2022

Työn nimi Fysiikan toteuttaminen Unity-pelimoottorissa

Ohjaaja Tommi Lahti

Opinnäytetyön tarkoituksena oli kartoittaa Unity-pelimoottorin fysiikkaominaisuuksia ja selvittää, miten ne käytännössä toimivat työn tekoaikana tuoreimmassa Unity 2021.2 versiossa. Työn taustana toimii oma perustason osaaminen Unity-ympäristössä ja halu laajentaa sitä. Tavoitteena oli löytää pelimoottorin tärkeimmät fysiikkaominaisuudet ja selvittää miten niitä käytetään ja koota nämä tiedot yhteen.

Tämän opinnäytetyön kaksi keskeisintä käsitettä ovat peliobjektit ja komponentit. Unity-pelimoottorissa peleissä käytettävät elementit, kuten esimerkiksi hahmot ja esineet, esitetään peliobjekteina. Näihin peliobjekteihin voidaan liittää komponentteja, jotka puolestaan mahdollistavat objektille pelimoottorin eri ominaisuuksien käytön. Esimerkiksi grafiikka-, ääni ja fysiikkaominaisuudet saadaan käyttöön niihin liittyvien komponenttien kautta. Tässä työssä käydään fysiikkamoottorin tärkeimpiä ominaisuuksia ja niiden käyttöön tarvittavia komponentteja kattavasti läpi.

Opinnäytetyö on tyypiltään toiminnallinen ja käytännön osa muodostuu pienimuotoisesta Unity-peliprojektista. Peliprojekti sisältää muutamia käyttöesimerkkejä fysiikkajärjestelmän nivelkomponenteista, jotka yhdistävät toimiakseen useita fysiikkajärjestelmän eri ominaisuuksia.

Työn pohjalta johtopäätöksenä voidaan todeta, että Unity tarjoaa hyvin monipuolisia fysiikkaominaisuuksia sekä 2D- että 3D-ympäristössä. Työtä tehdessä löydettiin myös yksi täysin uusi fysiikkaominaisuus: CustomCollider2D -törmäytin ja sen manipuloimisen mahdollistava ohjelmointirajapinta. Myös fysiikkajärjestelmän tukeminen ohjelmoinnilla osoittautui mahdolliseksi, sillä fysiikkakomponenttien asetuksia voidaan muokata C#-ohjelmoinnin kautta, vaikka ohjelmointi jäikin työssä pieneen rooliin.

Avainsanat Pelinkehitys, Pelimoottori, Unity, Fysiikkasimulaatio

Sivut 38 sivua ja liitteitä 3 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Author Joona Kari
Subject Applying physics in Unity
Supervisors Tommi Lahti

Abstract
Year 2022

The purpose of the thesis was to map out the physics features of the Unity game engine and to find out how they work in practice in the Unity 2021.2 version which is the latest release at the time of writing. The thesis was inspired by the authors basic level competency in the Unity environment and willingness to improve it. The goal was to find the game engine's most important physics features and how they are used and gather all this information together.

Two key concepts for this thesis are "GameObjects" and components. In the Unity game engine, all elements that are used in a game environment, such as characters and items, are represented and referred to as "GameObjects". Components can be added to a GameObject to allow the GameObject to use the various features of the game engine. For example, graphics-, audio-, and physics features can be used via the relevant components. This thesis goes through the physics engine's key features and the components required to use those features extensively.

The thesis is practical, and the practical part comprises of a small-scale Unity game project. The game project includes a few practical use examples of the physics engine's joint components, which combine various features of the physics engine to function.

As a conclusion it can be stated that Unity offers a highly versatile selection of physics features for both 2D- and 3D-environments. While researching the topic, an entirely new 2D physics feature was found: CustomCollider2D -collider and the scripting API for manipulating it. Supporting the physics system via programming was also shown to be possible as the settings and states of the physics components can be edited via C#-programming, even though programming had a small role in the thesis.

Keywords Game Development, Game Engine, Unity, Physics Simulation

Pages 38 pages and appendices 3 pages

Sanasto

Peliobjekti:

Unity-pelimoottorissa kaikki peliympäristöön lisättävät asiat, esim. hahmot objektit ja ympäristöelementit esitetään Unityn hierarkiaikkunassa peliobjekteina. Peliobjektia kutsutaan englanniksi nimellä GameObject.

Komponentti :

Unityssä peliobjekteihin voidaan liittää erilaisia komponentteja. Komponentit mahdollistavat esimerkiksi ääni-, grafiikka ja fysiikkaominaisuuksien käytön

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

Nykyaikana videopelit ovat usein erittäin monimutkaisia, ja niistä löytyy toinen toistaan innovatiivisempia pelimekaniikkoja. Kaikkien mekaniikkojen pitää kuitenkin pohjautua kehitysympäristön tarjoamiin mahdollisuuksiin. Tämän takia on hyödyllistä perehtyä valitsemansa kehitysympäristön mahdollisuuksin joko ennen pelin kehitysprosessia tai sen aikana. Nykyiset videopelien kehitysympäristöt, kuten pelimoottorit tarjoavat kehittäjille monenlaisia valmiita järjestelmiä, joiden päälle itse peliä ja sen mekaniikkoja on helppo alkaa rakentaa. Tällaisia ovat esimerkiksi animaatio- ja tekoälyjärjestelmät. On kuitenkin yksi tällainen järjestelmä, joka on ehkä vielä olennaisempi kuin edellä mainitut; fysiikkajärjestelmä. Tässä opinnäytetyössä halutaankin tutkia erittäin laajassa käytössä olevan Unity-pelimoottorin sisäänrakennetun fysiikkamoottorin ominaisuuksia ja käyttömahdollisuuksia, sekä sitä miten fysiikkamoottoria käytännössä käytetään.

Lähden tutkimaan Unity-fysiikkaa pääasiassa 3D-ympäriston kautta, sillä sen fysiikkatoteutusmahdollisuudet ovat hiukan monipuolisempia kuin 2D-ympäristössä. 2D-fysiikka on kuitenkin oman vähäisen kokemukseni mukaan toiminnaltaan pitkälti 2D-fysiikkaa vastaavaa, ja jos 3D:n käyttö sujuu, 2D-ympäristö on monista samankaltaisuuksista johtuen helppo oppia. Työssä tutkitaan myös, onko uusimpaan Unity 2021.2-versioon lisätty uusia fysiikkaa koskevia ominaisuuksia, sekä pienissä määrin myös fysiikan tukemista ohjelmoinnilla. Tekijä on jo ennen työn aloittamista opiskellut Unityn käyttöä perustasolla, ja näin ollen lukijalta odotetaan perustason ymmärrystä siitä, miten Unity toimii.

Työ tulee muodostumaan hyvin käytännönläheisesti, joten Unityn olennaisimmat fysiikkaominaisuudet, kuten RigidBodyt, erilaiset Colliderit ja fysiikkamateriaalit sekä nivwlkomponentit, joita Unity-pelimoottorissa eniten hyödynnetään fysiikkaa toteuttaessa tulevat olemaan työn ytimessä. Tavoitteena on, että tämä työ antaa hyvän, ajantasaisen tietopaketin siitä, mikä Unityssä on mahdollista fysiikan simuloimiseksi tällä hetkellä ja antaa käytännön esimerkkejä siitä, miten fysiikka Unityssä toimii. Tutkimuskysymykset ovat:

- Millaisia toimintoja Unity tarjoaa fysiikan simuloimiseen?
- Miten Unityn fysiikkajärjestelmää voidaan tukea ohjelmoinnilla?
- Mitä uutta Unityn 2021.2 -versio tuo fysiikan toteuttamiseen?

2 Unityn fysiikkamoottori

Unity mahdollistaa fysiikan toteuttamisen peliprojekteissa niin, että pelien sisältämät objektit saadaan noudattamaan paino-, törmäys-, ja muita voimia halutulla tavalla. Tätä varten Unitystä löytyy useita eri fysiikkamoottoritoteutuksia. Näistä kaksi löytyy Unitystä sisäänrakennettuna. Ensimmäinen näistä on Unityn sisäänrakennettu 3D-fysiikkamoottori, joka pohjautuu Nvidian PhysX-teknologiaan. 2D-fysikkaa voidaan Unityssä toteuttaa sisäänrakennetulla 2D-mottorilla, joka puolestaan pohjautuu Box2D-moottoriin. (*Unity - Manual: Physics*, ei pvm.)

Unityn fysiikkaominaisuudet

Seuraavassa käymme läpi Unity-pelimoottorin tärkeimmät fysiikkaominaisuudet yleisellä tasolla. Tässä luvussa ei siis niinkään pyritä selittämän miten nämä ominaisuudet toimivat vaan pikemminkin millaisia mahdollisuuksia ne tarjoavat.

2.1 Erilaisten voimien kohdistaminen peliobjekteihin

Unity mahdollistaa monien erityyppisten fyysisten voimien kohdistamisen peliobjekteihin, kun objekti on asetettu käyttämään fysiikkamoottoria RigidBody-komponentin avulla. RigidBodyn ollessa käytössä voidaan objektiin kohdistaa liike- ja vääntövoimia sekä painovoimaa. Tämä avaa mahdollisuuden jäljentää esimerkiksi niveliin perustuvia liikkeitä ja realistisia törmäyksiä muiden objektien kanssa Nvidia PhysX:a hyödyntäen. (Unity Manual: RigidBody, ei pvm.)

Voimaa voidaan kohdistaa objekteihin lukuisin eri tavoin erilaisten voimatilojen (englanniksi ”force modes”) avulla. Unitystä löytyy neljä eri tilaa voimien kohdistamiseen objektiin; Oletuksena Unity käyttää Force-tilaa, joka kohdistaa voimaa vähitellen kohteen omaan massaan suhteutettuna. Muita tiloja ovat Acceleration, joka kohdistaa voimaa tasaiseen tahtiin, Impulse, joka kohdistaa välittömän voiman kohteeseen, eikä ajan kuluessa kuten

edellä mainitut ja VelocityChange joka tekee saman kuin Impulse, mutta ei ota kohteen massaa huomioon. (Intro to the Unity Physics Engine - 2019.3 - Unity Learn, ei pvm.)

On myös mahdollista yhdistää RigidBodyn sisältäviä objekteja toisiinsa käyttäen Unityn sisältämiä nivelkomponentteja. Esimerkiksi fysiikkapohjaisesti liikkuvan hahmon muodostaminen on mahdollista käyttämällä Character Joint -komponentteja. Toinen mahdollisuus on Configurable Joint, jonka liikkumasunnat ja -rajoitukset ovat täysin käyttäjän muokattavissa kuten nimestäkin saattaa päätellä. Muita mahdollisuuksia ovat Fixed Joint, jolla voidaan lukita yksi objekti seuraamaan toisen liikettä. Hinge Joint, jolla voidaan luoda yhdellä akselilla tapahtuvaa liikettä ja Spring Joint, joka mahdollistaa jousimaisen liikkeen kahden objektin välissä. (Unity - Manual: Joints, ei pvm.)

2.2 Osuman tunnistus

Hyvässä pelimottorissa täytyy tietenkin olla hyvät osumantunnistusmahdollisuudet, jotta esimerkiksi törmäysten ja osumien simulointi onnistuu. Unityssä tämä tehdään pääosin erilaisia collider-komponentteja käyttäen; collider (suomeksi ”törmäytin”) on objektin ympärille muodostuva näkymätön komponentti, joka tarkkailee, osutaanko kyseiseen objektiin. Törmäyttimen ei kuitenkaan tarvitse olla täsmälleen saman muotoinen kuin se objekti johon se liitetään, mikä mahdollistaa haluttaessa suorituskyvyllisesti kevyiden kuten laatikko-, ympyrä-, ja kapselitörmäyttimien käytön. Näitä voidaan myös yhdistellä yhdistelmätörmäyttimiksi. Näin voidaan saada aikaan monimutkainen törmäytin optimaalisesti käyttämällä yksinkertaisia muotoja, varsinkin jos törmäytintä käyttävän peliobjektin rakenne on huomattavan monimutkainen. Tietenkin Unityssa on saatavilla myös tarkempia törmäyttiimiä, kuten mesh-törmäytin, jota vastaamaan tarkalleen sen objektin muotoa, johon se liitetään. 2D-ympäristöstä löytyy myös polygonitörmäytin, joka ei täysin vastaa objektinsa muotoa, mutta sen yksityiskohtatasoa voidaan säädellä tämän saavuttamiseksi. Kaksi viimeksi mainittua törmäytintä ovat kuitenkin suorituskyvyllisesti huomattavasti muita edellä mainittuja vaativampia. (Unity - Manual: Colliders, ei pvm.)

2.3 Fysiikkamateriaalit

Unitystä löytyy myös fysiikkamateriaali-komponentti, jolla voidaan antaa peliobjektille lisää fyysisiä ominaisuuksia, jotka määrittävät miten objekti käyttäytyy suhteessa muuhun ympäristöön saadessaan osuman muokkaamalla sen kitkaa ja kimmoisuutta. Unity mahdollistaa myös erillisten kitka-arvojen asettamisen riippuen onko peliobjekti paikallaan vai liikkeessä ja on myös mahdollista säätää, sitä miten kahden kontaktissa olevan objektin keskinäiset kitka- ja kimmoisuusarvot lasketaan. (Unity - Manual: Physic Material, ei pvm.)

2.4 Kangasmateriaalin fysiikkasimulaatio

Kangasmateriaalin fysiikkasimulaatio on jossain määrin mahdollista Unityssä. Tosin on huomattavaa, että tämä ominaisuus on Unity-dokumentaation mukaan tarkoitettu nimenomaan pelihahmojen vaatetuksessa hyödynnettäväksi. Kangasmateriaaleja ei myöskään ole oletuksena tarkoitettu olemaan vuorovaikutuksessa muualla ympäristössä olevien törmäyttimien kanssa, vaan jos näin halutaan olevan, täytyy tämä säätää komponentin asetuksista tapahtumaan. Kangasmateriaalikomponentit voivat olla tekemisissä vain tietyn tyyppisten yksinkertaisten törmäytinten kanssa ja simuloida osumaa vain itseensä. Syynä tälle toiminnallisuuden rajallisuudelle on suorituskyvyn parantaminen. (Unity - Manual: Cloth, ei pvm.)

2.5 Uudet fysiikkaominaisuudet Unity 2021.2 -versiossa

Unityn kehittäjät ovat lisänneet uusimpaan 2021.2 -versioon myös kaksi uutta tapaa käsitellä 2D-fysiikkamuotoja ja törmäyttimiä. Näistä ensimmäinen on CustomCollider2D -törmäytin, jolla voidaan suoraan luoda, muokata ja poistaa 2D-fysiikkamuotoja. Näin ollen tämä törmäytiin voi käyttää mitä tahansa yhdistelmää kaksiulotteisia perusmuotoisia fysiikkamuotoja, kuten ympyröitä ja kapseleita ja muodostaa näistä törmäyttimen. Koska CustomCollider2D on törmäytin, se voi myös käyttää kaikkia olemassa olevia ominaisuuksia kuten fysiikkamateriaaleja. (*Unity - Manual: New in Unity 2021.2*, ei pvm.)

Toinen uusi ominaisuus on PhysicsShapeGroup2D-rajapinta, joka mahdollistaa perusmuotoisten 2D-fysiikkamuotojen ryhmittelyn. PhysicsShapeGroup2D:n sisällä muotoja

voidaan lisätä, muokata tai poistaa. Useampi tällainen fysiikkamuotoryhmä voidaan myös yhdistää. Tämän komponentin idea onkin se että voidaan määrittää halutut muodot CustomCollider2D -törmäyttimen käyttöön. (*Unity - Manual: New in Unity 2021.2*, ei pvm.)

3 Toteutukseen tarvittavat komponentit ja niiden toiminta

Tässä luvussa käydään läpi Unityn fysiikkamoottorin käyttöön liittyvien komponenttien sisältämiä toimintoja ja niiden tarjoamia mahdollisuuksia.

3.1 RigidBody-komponentti

RigidBody-komponentin käyttöä varten sille pitää tietenkin määrittää joitain ominaisuuksia. Erittäin olennainen on ainakin massa, johon viitataan Unityssä nimellä Mass. Unityn painojärjestelmä käyttää oletuksena kiloja. Massan lisäksi RigidBody tarjoaa monia muita ominaisuuksia käyttäjälleen säädettäväksi. Esimerkiksi Gravity-asetuksella voidaan asettaa RigidBodyn sisältävä objekti noudattamaan painovoimaa, tai painovoima voidaan kytkeä se osalta pois päältä. Objekti voidaan myös asettaa RigidBodyssä kinemaattiseksi, jolloin se ei liiku muiden RigidBodyn sisältävien objektien osuessa siihen, Tämä voidaan tehdä asettamalla Is Kinematic -asetus päälle. (Unity Manual: RigidBody, ei pvm.)

RigidBody voidaan määrittää myös interpoloimaan (Interpolate) tai ekstrapoloimaan (extrapolate) omaa liikettään, jos sen liike vaikuttaa nykivältä. Interpolointi-asetuksella Unity pyrkii tasoittamaan RigidBodyn liikettä objektin sijainnin perusteella edellisen piirretyn framen aikana, ekstrapoloidessa Unity yrittää arvioimaan tätä sijaintia seuraavassa framessa. (Unity Manual: RigidBody, ei pvm.)

Myös osumantunnistustapaa voidaan tarvittaessa muuttaa. Oletusasetus on Discreet (suom. Hienovarainen), jolloin objekti käyttää hienovaraista osumantunnistusta muita pelissä olevia fysiikkaobjekteja kohtaan ja ne tekevät samoin. Continuous-asetuksella objekti käyttää hienovaraista osumantunnistusta sellaisia objekteja kohtaan, joilla on myös RiogidBody käytössä, paitsi jos vastaan tulee objekti, jonka RigidBody on määritetty käyttämään Continuous Dynamic -asetusta, jolloin käytetään jatkuvaa osumantunnistusta. Staattisia törmääjiä (eli sellaisia objekteja, jotka käyttävät törmäyttimiä mutta eivät RigidBody-

komponenttia) kohtaan käytetään pyyhkäisypohjaista jatkuvaa osumantunnistusta (englanniksi sweep-based continuous collision detection). On syytä tietää, että jatkuva osumantunnistus on suorituskyvyllisesti erittäin kallista, joten Continuous -asetusta kannattaakin käyttää vain silloin kun käsiteltävä objektin täytyy pystyä törmäämään muihin Continuous Dynamic-asetusta käyttäviin objekteihin. Continuous Dynamic -asetuksella käytetään pyyhkäisypohjaista jatkuvaa osumantunnistusta niitä objekteja kohtaan, jotka käyttävät jatkuvaa tai jatkuvaa dynaamista osumantunnistusta. Staattisia törmäyttimiä kohtaan hyödynnetään niin ikään jatkuvaa osumantunnistusta. Kaikkia muita törmäyttimiä kohtaan käytetään hienovaraista osumantunnistusta. Continuous Dynamic on tarkoitettu käytettäväksi sellaisille objekteille, jotka liikkuvat pelissä erittäin nopeasti, eli joille muiden osumantunnistuskeinojen tarkkuus ei ole riittävä. Sitten on vielä yksi osumantunnistusasetus: Continuous Speculative, joka pyrkii ennalta arvioimaan osumia objektien nopeuksiin perustuen. Tämä on ainoa osumantunnistusvaihtoehto, jota voidaan käyttää, kun objekti on asetettu kinemaattiseksi.

Lisäksi RigidBody-komponentissa voidaan määrittää liikkumisrajoituksia sitä käyttävälle objektille Constraints kohdan alaisista asetuksista. Näillä voidaan estää objektin liikkuminen (Freeze Position) tai kääntyminen (Freeze Rotation) tietyllä akselilla tai akseleilla. Unityn 3D-ympäristössä akselit ovat X (leveyssuunta), Y (korkeussuunta), ja Z (syvyyssuunta). Näiden lisäksi on mahdollista säätää, onko objektilla ilmanvastusta tavallisia ja vääntoimia kohtaan. Tätä säädetään Drag ja Angular Drag asetuksilla. Näiden arvojen ollessa nollassa (0), ei objektilla ole minkäänlaista ilmanvastusta. Tätä voidaan säätää aina loputtomaan asti, jolloin objekti pysähtyy välittömästi. (Unity Manual: RigidBody, ei pvm.)

3.2 Osumien tunnistus törmäyttimillä

Unity tarjoaa useita erilaisia vaihtoehtoja erimuotoisten- ja laatuisten peliobjektien saamien osumien tunnistamiseksi. Näistä käytetään englanniksi termiä ”collider”, vapaasti suomennettuna törmäytin. Seuraavassa käydään läpi erityyppisiä törmäyttimiä, joita pelimoottori tarjoaa.

3.2.1 Perusmuototörmäyttimet (”primitive colliders”)

Box Collider eli laatikkotörmäytin on suorakaiteen muotoinen törmäytin, jota voidaan hyödyntää yksinkertaisten objektien kuten laatikkojen, seinien ja lattioiden osumantunnistuksen toteuttamiseen. Jos törmäyttimen kokoa halutaan säätää, onnistuu tämä komponentin valikosta löytyvää Edit Collider -kuvaketta painamalla. Näin päästään muokkaustilaan, jossa törmäyttimen kokoa voidaan muuttaa vetämällä hiirtä halutun pinnan keskellä olevaan pistettä klikatessa. Tämä on hyödyllistä, jos törmäyttimestä halutaan tehdä pienempi tai suurempi kuin se oletuksena on. Muokkaustilasta pääsee pois painamalla samaa näppäintä uudestaan. Laatikkotörmäyttimen asetuksista voidaan säätää ko. törmäytin olemaan Trigger eli laukaisin, jolloin törmäytin havaitsee osumat, muttei toimi fysiikkamoottorin alaisen eli sen läpi voidaan esimerkiksi kulkea. Tämä voidaan tehdä asettamalla Is Trigger -asetus päälle. Lisäksi törmäyttimelle voidaan asettaa haluttu fysiikkamateriaali Material-kohdassa. Tarpeen tullen myös törmäyttimen sijaintia suhteessa sitä käyttävään objektiin voidaan muuttaa Unityn koordinaattijärjestelmän avulla muokkaamalla Center-kohdan alla olevia X, Y ja Z koordinaatteja. Haluttaessa myös törmäyttimen koko voidaan muuttaa numeroarvoja käyttäen Size-kohdan alla sijaitsevia arvoja säätämällä. Laatikkotörmäytin on yksi niistä perusmuototörmäyttimistä, joita voidaan käyttää yhdistelmätörmäyttimien muodostamiseen. Taulukossa 2 näkyy millaisilla asetuksilla törmäyttimet toimivat keskenään, toimivat yhdistelmät on merkitty rastilla. (Unity - Manual: Box Collider, ei pvm.)

Taulukko 1 Millaiset törmäyttimet toimivat fysiikkamoottorin kanssa oikein (*Unity - Manual: Physics*, ei pvm. ,muokattu)

Osuman tunnistus tapahtuu ja siitä lähetetään tieto pelimoottorille						
Törmäyttimen tila	Staattinen törmäytin	RigidBody törmäytin	Kinemaattinen RigidBody törmäytin	Staattinen Trigger Törmäytin	RigidBody Trigger törmäytin	Kinemaattinen RigidBody Trigger törmäytin
Staattinen törmäytin		X				
RigidBody törmäytin	X	X	X			
Kinemaattinen RigidBody törmäytin		X				
Staattinen Trigger Törmäytin						
RigidBody Trigger törmäytin						
Kinemaattinen RigidBody Trigger törmäytin						

Capsule Collider eli kapselitörmäytin on kapselinmuotoinen törmäytin, joka rakentuu sylinteristä keskellä ja kahdesta puoliympyrästä molemmissa päissä. Sillä on useita yhteisiä asetuksia laatikkotörmäyttimen kanssa; nämä ovat Edit Collider -toiminto, laukaisimeksi asettaminen (Is Trigger), fysiikkamateriaalin asettaminen (Material) ja törmäyttimen suhteellisen sijainnin asettaminen (Center). Kapselitörmäyttimen koon muokkaaminen komponenttiasetusten kautta poikkeaa kuitenkin edellisessä kappaleessa kuvatusta, sillä kapselilla ei ole sivuja samaan tapaan kuin suorakulmiolla. Kapselitörmäyttimen asetuksissa Radius-arvo viitta siihen miten suuria kapselin päissä olevien puolikaarien tulisi olla ja Height-arvo viittaa koko törmäyttimen korkeuteen. Direction-asetuksella voidaan muuttaa törmäyttimen pituussuuntaa suhteessa sitä käyttävään objektiin. Vaihtoehtoina ovat X- (pituus), Y- (korkeus) tai Z- (syvyys) akseli. Kapselitörmäytin on myös perusmuototörmäytin, joten sitäkin voidaan hyödyntää käyttää yhdistelmätörmäyttimien muodostamisessa. (Unity - Manual: Capsule Collider, ei pvm.)

Sphere Collider eli pallotörmäytin on pallon muotoinen törmäytin, joka mahdollistaa pyöreiden ja pyörivien objektien osumantunnistuksen. Pallotörmäytin on yksi yksinkertaisimmista törmäyttimistä ja jakaakin suurimman osan ominaisuuksistaan aiemmin esiteltyjen törmäyttimien kanssa. Törmäyttimen muodon muokkaaminen on mahdollista samalla tavalla kuin aiemmissa törmäyttimissä, kuten on laukaisijaksi asettaminen ja törmäyttimen suhteellisen sijainnin muuttaminenkin. Törmäyttimen koon muuttamiseen käytetään pallotörmäyttimen tapauksessa vain yhtä arvoa, joka on kapselitörmäyttimestäkin tuttu Radius-arvo. Myös pallotörmäyttimiä voidaan käyttää osana yhdistelmätörmäyttimiä. (Unity - Manual: Sphere Collider, ei pvm.)

3.2.2 Compound Collider eli yhdistelmätörmäytin

Kuten edellä todettiin, voidaan Unityn perusmuototörmäyttimiä yhdistämällä muodostaa yhdistelmätörmäyttimiä, joilla voidaan muodostaa erittäin suorituskykyisiä törmäyttimiä sellaisillekin peliobjekteille, joiden muoto ei täysin vastaa mitään perusmuotoa. Tallaisia törmäyttimiä voidaan muodostaa asettamalla yhdelle objektille useita laatikko-, kapseli- tai pallotörmäyttimiä, joita voi tarpeen mukaan olla käytössä kuinka paljon tahansa. Yhdistelmätörmäyttimellä voidaan siis saada aikaan törmäytin, joka vastaa suunnilleen sitä

käyttävän peliobjektin muotoa vähäisellä suoritustehon kulutuksella. (Unity - Manual: Colliders, ei pvm.)

3.2.3 Mesh Collider

Meshit ovat Unityssä 3D-objektin pintaa kuvaavia verkkopintamateriaaleja. Mesh Collider on nimenomaan erilaisten meshien pintakuvioiden tarkkaa hyödyntämistä varten kehitetty törmäytin, jonka erikoisominaisuus on, että se pystyy jäljentämään sitä käyttävän objektin muodon erittäin tarkasti. Kun Mesh Collider -komponentti otetaan käyttöön, törmätään sen asetuksissa ensimmäisenä Convex-asetukseen. Convex asetusta käyttävä Mesh collider voi koostua enintään 255 kolmiosta. Tämän asetuksen laittaminen päälle tekee sen, että kyseinen objekti voi ottaa osumaan muilta Mesh Collideride-komponenttia hyödyntäviltä objekteilta. Komponentista löytyy myös jo aiemmin esitelty Is Trigger -asetus. (Unity - Manual: Mesh Collider, ei pvm.)

Mesh Colliderista löytyy myös Cooking Options -asetukset. ”Cooking” tässä yhteydessä tarkoittaa prosessia, jossa Unity valmistelee tarvittavat meshit sellaisiksi, että Unityn fysiikkamoottori voi niitä hyödyntää. Tämä sisältää esimerkiksi tilahakurakenteiden valmistelemisen Raycast-funktioita varten. Cooking Options -kohdan kaksi ensimmäistä asetusta ovat melko selkeitä None (”Ei mitään”) ottaa kaikki listassa olevat optimisaatiot pois päältä, kun Everything (”Kaikki”) asetta ne käyttöön. Cook for Faster Simulation –asetuksen ollessa käytössä Unity tekee joitain asioita sen eteen, että tuloksena syntyvä mesh on mahdollisimman suorituskykyystävällinen peliä pelattaessa. Enable Mesh Cleaning -asetus sallii Unityn yrittää poistaa lopputuloksesta ylimääriä kolmioita ja mahdollisia geometriavirheitä, jotta meshin osumakohdat saadaan mahdollisimman tarkasti jäljennettyä. Weld Colocated Vertices -asetuksella Unity pyrkii tarkastamaan, onko meshissä samassa kohdassa olevia huippupisteitä (englanniksi ”vertices”), joita saattaa joskus jäädä meshiin 3D-mallinnuksen jäljiltä ja pyrkii yhdistämään ne. Tämä saattaa parantaa pelissä tapahtuvan osuman tunnistuksen tarkkuutta paljonkin. Use Fast Midphase -asetus laittaa Unityn käyttämään nopeinta mahdollista keskivaiheen kiihdytysrakennetta sekä algoritmia riippuen alustasta, jolla peliä pelataan. Jos tämän asetuksen päällä pitäminen aiheuttaa ongelmia Cooking-prosessissa joillain alustoilla, se kannattaa ottaa pois päältä, jolloin Unity käyttää vanhaa keskivaihealgoritmia. Lisäksi Mesh Colliderille voi määrittää halutun

fysiikkamateriaalin kuten muillekin törmäyttimille Material-kohdassa. Mesh-kohdassa komponentille kerrotaan mitä mesh-materiaalia sen kuuluu käyttää. (Unity - Manual: Mesh Collider, ei pvm.)

3.2.4 Wheel Collider eli rengastörmäytin

Unitystä löytyy myös nimenomaan renkaiden osumantunnistukseen tarkoitettu törmäytin auton ja muiden renkailla kulkevien ajoneuvojen tekemistä varten. Toisin kuin muille törmäyttimille, rengastörmäyttimelle voidaan määrittää suoraan massa komponentin Mass-kohdassa. Radius kohdassa voi määrittää törmäyttimelle myös koon. Rengastörmäyttimen iskunvaimennuksen määrää voidaan säätää muokkaamalla Wheel Damping Rate -arvoa. Rengastörmäytin mahdollistaa myös jousituksen simuloimisen. Suspension Distance -arvo määrittää, kuinka kauas rengasobjektin jousitus voi yltää maksimissaan. Tämä arvo lasketaan oletuksena metreissä ja se lasketaan aina pystysuunnassa alaspäin. (Unity - Manual: Wheel Collider, ei pvm.)

Force App Point Distance on arvo, joka määrittää mihin kohtaan renkaan voima kohdistuu, tarkoittaen siis matkaa metreinä rengastörmäyttimen pohjasta ylöspäin jousituksen suuntaisesti, eli esimerkiksi arvo 0 kohdistaa voiman suoraan renkaan pohjaan, jolloin jousitus ja iskunpehmennys jäävät erittäin heikoiksi. Komponentin asetusten Center-kohdassa voidaan muokata törmäyttimen keskikohtaa kuten monissa muissakin törmäyttimissä. (Unity - Manual: Wheel Collider, ei pvm.)

Suspension Spring kohdan alta löytyy useitta arvoja, joilla voidaan hienosäätää jousituksen toimintaa. Spring-arvoa muokkaamalla voidaan säätää, miten nopeasti jousen tuottama voima etenee kohteeseensa (Target Position) ja alkaa vaikuttaa. Damper-arvolla puolestaan säädetään, miten tehokas jousen iskunvaimennus on. Target Position kuvaa jousen lepopistettä pystysuunnassa niin että arvo 1 tarkoittaa täysin jännittynyttä jousta ja 0 tarkoittaa täysin sisään vetäytynyttä jousta. Target position on oletusarvoisesti 0,5, mikä vastaa kutakuinkin tavanomaisen auton jousituksen toimintaa. Lopuilla rengastörmäyttimen asetuksilla, Forward sekä Sideways Friction -kohtien alaisilla arvoilla säädellään simuloitavan renkaan kitka-arvoja eteenpäin- ja sivuliikkeessä. Jos haluat syventyä tarkemmin siihen, miten nämä arvot toimivat ja miten niitä muokataan, suosittelen lukemaan Unity-manuaalin

Wheel Collider artikkelin luvun Wheel Friction Curves. (Unity - Manual: Wheel Collider, ei pvm.)

3.2.5 Terrain Collider eli maastotörmäytin

Unityssä voidaan rakentaa 3D-ympäristöjä helposti Terrain-objekteja käyttäen. Tällaisia ympäristöjä varten on tietenkin olemassa oma törmäyttimensä fysiikan simuloimisen avuksi. Maastotörmäytin siis vastaa muodoltaan sitä Terrain-objektia, johon se liitetään. Muihin törmäyttimiin verrattuna maastotörmäyttimellä on erittäin vähän muokattavia asetuksia ja ominaisuuksia. Muiden törmäyttimien tavoin sille voidaan asettaa haluttu fysiikkamateriaali. Terrain Data kohdassa voidaan asettaa törmäyttimelle maastodata, jotta törmäytin tietää onko sen kohdeobjektissa esimerkiksi erityisen korkeita tai matalia kohtia, jotta nämä voidaan ottaa osumasimulaatiossa huomioon. Lisäksi on mahdollista ottaa maasto-objektiin tehdyt puut osumantunnistukseen mukaan Enable Tree Colliders -asetuksella. (Unity - Manual: Terrain Collider, ei pvm.)

3.2.6 Character Controller eli hahmon ohjain

Unityn fysiikkatyökaluista löytyy myös erityisesti yksinkertaisten pelaajahahmojen liikkeen ja osumantunnistuksen toteuttamiseen tarkoitettu Character Controller. Tämä komponentti asettaa hahmon ympärille kapselitörmäyttimen niin, että hahmon on mahdollista liikkua, mutta liikkuminen on kuitenkin ympäristöön osumisen perusteella rajattua, ja sille voidaan valmiiksi määrittää myös joitain liikkumista helpottavia ominaisuuksia, mikä vähentää tarvetta ylimääräisille komponenteille. (Unity - Manual: Character Controller, ei pvm.)

Slope Limit -arvolla voidaan määrittää asteluku, jota jyrkempiä reunoja hahmo ei voi kiivetä. Step Offset -arvolla voidaan määrittää miten korkeita askeleita hahmo voi ottaa; hahmo voi kiivetä vain sellaisia portaita, jotka ovat matalampia kuin tämä arvo. Skin Width määrittää, miten syvälle toinen törmäytin voi osua hahmon ohjaimeen. Liian pieni arvo voi aiheuttaa hahmon nykimistä tai jumiin jäämistä, joten tätä arvoa kannatta harkita tarkkaan. Virallinen Unity-ohjeistus suosittelee arvoksi yli kymmentä prosenttia komponentin Radius-arvosta (kapselitörmäyttimen leveys). Min Move Distance -arvo kertoo mikä matka hahmon pitää vähintään liikkua, jotta liike toteutetaan pelissä hahmo-ohjaimen toimesta, tällä voidaan

tarvittaessa vähentää nykimistä pieniä liikkeitä tehdessä. Center-kohtaan voidaan määrittään hahmo-ohjaimen kapselitörmäyttimen keskipiste suhteessa hahmon visuaaliseen malliin. Radius määrittää tuon kapsetörmäyttimen leveyden ja Height pituuden. (Unity - Manual: Character Controller, ei pvm.)

3.3 Fysiikkamateriaalit

Fysiikkamateriaaleja käytetään Unityssä määrittämään fysiikkamoottoria käyttäville peliobjekteille kitka- ja kimmoisuusominaisuuksia. Fysiikkamateriaalit eivät kuitenkaan ole objektiin liitettäviä komponentteja kuten RigidBody ja erilaiset törmäyttimet, vaan ne pikemminkin täydentävät kokonaisuutta osana näitä komponentteja. (*Unity - Manual: Physic Material*, ei pvm.)

Fysiikkamateriaalille voidaan määrittää kaksi eri kitka arvoa; dynaaminen kitka, jota käytetään objektin ollessa liikkeellä ja staattinen kitka, jota käytetään, kun objekti on liikkumaton. Näitä säädetään Dynamic Friction ja Static Friction -arvoilla, joita nostamalla esineen kohtaaman kitkan määrä nousee, kun arvon ollessa 0 objekti on jäämäisen liukas. Nämä arvot ovat yleensä jotain nollan (0) ja yhden (1) välillä. Fysiikkamateriaalin kimmoisuutta säädetään Bounciness -arvolla, välillä 0-1. 0-arvoisen fysiikkamateriaalin sisältävä objekti ei ole lainkaan kimmoisa. Fysiikkamateriaali, jonka kimmoisuusarvo on 1 puolestaan kimmottaa sen sisältävää objektia niin että se voi pomppia ja kimpoilla hyvinkin pitkään menettämättä liike-energiaansa. (*Unity - Manual: Physic Material*, ei pvm.)

Friction Combine kohdassa voidaan määrätä, miten kahden toisiinsa törmäävän fysiikkamateriaalin sisältävän objektin kitka-arvo lasketaan yhteen. Vaihtoehtoja on 4; Average (joka laskee molempien objektien kitka-arvojen keskimäärän), Minimum (jolloin pelimoottori käyttää kahdesta arvosta pienempää), Maximum (jolloin pelimoottori käyttää arvoista suurinta) ja Multiply (jossa kerrotaan molempien objektien arvot yhteen). Jos kuitenkin tulee tilanne, jossa kaksi objektia, joilla on kaksi eri fysiikkamateriaalia, joilla on molemmilla eri laskenta-asetukset käytössä, käytetään erityismenettelyä ratkaisemaan, millä laskenta-asetuksella kitka lasketaan. Tällöin vaihtoehdot laitetaan seuraavanlaiseen prioriteettijärjestykseen:

1. Maximum (suurin arvo)
2. Multiply (kerrottu arvo)
3. Minimum (pienin arvo)
4. Average (keskiarvo)

Tämän järjestyksen perusteella ylempiarvoista laskenta-asetusta käytetään tarvittavan arvon laskemiseen. Fysiikkamateriaalille voidaan määrittää myös vastaava asetus kimmoisuusarvon laskemiseksi, tämä on nimeltään Bounce Combine ja toimii samoilla asetuksilla ja logiikalla kuin Friction Combine. (*Unity - Manual: Physic Material*, ei pvm.)

3.4 Nivelkomponentit

Unityn fysiikkajärjestelmä sisältää useita erilaisia nivelkomponentteja, jotka mahdollistavat erilaisten nivelpohjaisten rakenteiden toteuttamisen.

3.4.1 Yleiskäyttöiset nivelkomponentit

Character jointit eli hahmonivelet ovat Unity-komponentteja, joita käytetään hahmojen kehojen tai räsynukkejen rakentamiseen fysiikkamoottoria käyttäen. Hahmoniveltä rakennettaessa sille määritetään yleensä RigidBody, jossa kyseinen nivel (ja sen varassa olevat mahdolliset raajat) ovat kiinni komponenttiasetusten Connected Body kohdassa. Tämä voidaan tosin haluttaessa jättää määrittämättä, jolloin nivel kiinnittyy suoraan peliympäristöön. Anchor kohdassa voidaan asettaa sijainti suhteessa kiinnityskohtaan, josta nivelen pyörimisliike lähtee. Näiden lisäksi voidaan määrittää myös kiinnike objektin (eli RigidBodyn tai itse pelimaailman) ankkuripiste. Tätä varten hahmonivelkomponenteilla on oletuksena käytössä Auto Configure Connected Anchor -asetus, jolla Unity laskee itse automaattisesti ankkuripisteelle sopivan sijainnin. Jos tämä kuitenkin halutaan poistaa käytöstä, voidaan ankkuripiste määrittää itse Connected Anchor -kohdassa. Swing Axis -kohdassa määritetään millä akselilla tai akseleilla nivel voi heilua. Low sekä High Twist Limit määrittelevät nivelen ala- ja ylätaipumisrajat, jotka lasketaan suhteellisina nivelen alkuasentoon. (Unity - Manual: Character Joint, ei pvm.)

Swing 1 Limit ja Swing 2 Limit puolestaan määrittävät millä välillä nivelen pitelemä objekti voi heilua. Nämä arvot ovat symmetrisiä, tarkoittaen että kumpikin arvo ulottuu vastaavaan arvoon myös miinuskulmassa, eli jos arvoksi asetetaan 40, komponentti sallii heilumisliikkeen 40 ja -40 asteen välillä. Lisäksi kaikille edellä manituille rajoituksille voidaan antaa omat kimmoisuus-, jousto- ja pehmennysarvot. Kimmoisuusarvo (Bounciness), asetetaan fysiikkamateriaalien tapaan välille 0–1 ja määrittää, kimpoileeko objekti raja-arvojen ylittyessä. Joustoarvo (Spring) määrittää miten voimakkaasti nivelrakenne joustaa yrittäessään pitää nivelen yhdistämät objektit yhdessä. Pehmennysarvo (Damper) pehmentää joustovoimaa halutusti. Näiden lisäksi voidaan nykimisen välttämiseksi määrittää Contact Distance -arvo, joka rajoittaa miten kauas rajakontaktit säilyvät. Jos nivelen halutaan mahdollisesti irrottavan kiinnittämänsä objektin, voidaan sille määrittää Break Force ja Break Torque -arvot, jotka siis kertovat miten paljon voimaa ja vääntöä vaaditaan siihen, että nivel irtoaa liitoksestaan. Enable Collision -asetuksella voidaan mahdollistaa nivelen yhdistämien objektien törmäykset keskenään ja ongelmatilanteissa Enable Preprocessing -asetuksen ottaminen pois käytöstä voi auttaa pelimoottoria prosessoimaan tilanteita, jossa vastaan tulee mahdottomia asentoja. (Unity - Manual: Character Joint, ei pvm.)

Hinge Joint eli sarananivel-komponentilla voidaan yhdistää kaksi objektia niin että niiden välinen liike on saranamaista ja tätä komponenttia käytetäänkin usein esimerkiksi ovien saranarakenteen toteuttamiseen. Kuten hahmonivelellekin, saranivelelelle voidaan myös määrittää toinen RigidBodyn sisältävä objekti, johon se kiinnittyy Connected Body -kohdasssa. Niin ikään edellisessä kappaletta vastaavalla tavalla sarananivelen suhteellinen kiinnityskohta voidaan määrittää Anchor-kohdassa. Sama voidaan tehdä myös nivelen kiinnikeobjektille Connected Anchor -kohdassa tai jättää pelimoottorin hoidettavaksi Auto Configure Connected Anchor -asetusta käytttämällä. Axis-kohdassa voidaan määrittää, mitä akselia pitkin nivelen halutaan liikkuvan. Sarananiveltä voidaan käyttää kolmessa eri tilassa: jousitettuna (Spring), moottoroituna (Motor) tai rajoitettuna (Limits). Jokaista tilaa varten on omat asetuksensa. Spring-tilan asetukset ovat Spring, joka määrittää miten voimakas jousimekanismi on, Damper, joka kertoo, miten paljon jousiliikettä halutaan pehmennettävän ja Target Position, joka määrittää kulman johon jousi pyrkii ulottumaan. Jos jousimekanismitila halutaan komponentille käyttöön, tulee asettaa Use Spring -asetus myös käyttöön. (Unity - Manual: Hinge Joint, ei pvm.)

Motor-tilan asetukset ovat puolestaan seuraavat: Target Velocity, jolla määritellään, mihin nopeuteen moottori pyrkii kiihtymään, Force, eli kuinka paljon voimaa niveleen kohdistetaan nopeuden saavuttamiseksi ja Free Spin, jonka ollessa käytössä moottoria ei käytetä ollenkaan liikkeen hidastamisen vaan pelkästään sen kiihdyttämiseen. Jos moottoritila halutaan käyttöön, se voidaan tehdä aktivoimalla Use Motor asetus. Limits-tilassa määritetään minimi- ja maksimikulma (Min- ja Max -arvot), jonka välillä sarananivelen halutaan liikkuvan. Lisäksi voidaan määrittää Bounciness-arvolla, kimpoaako sarana takaisin rajakulmassa käytyään rajakulmassa ja Contact Distance -arvo, joka määrittää miten kauas rajakontaktit säilyvät. Sarananivelen rajoitustila saadaan päälle niin ikään ottamalla Enable Limits -asetus käyttöön. Hahmonivelen tavoin sarananivelelle voidaan myös määrittää halutut rikkoutumisolosuhteet Break Force ja Break Torque -arvoilla. Sarananivelen yhdistämien objektien yhteen törmääminen voidaan sallia Enable Collision asetuksella ja pelimoottorin esiprosessointi voidaan haluttaessa ottaa pois päältä ottamalla Enable Preprocessing asetus pois käytöstä. (Unity - Manual: Hinge Joint, ei pvm.)

Fixed Joint eli kiinteä nivel on yksinkertaisin Unity-pelimoottorin nivelkomponenteista. Sen tarkoitus on ainoastaan yhdistää yksi objekti toiseen objektiin tai pelimaailmaan ja näin ollen sillä ei ole minkäänlaisia liikkuvuusominaisuuksia. Komponentille kerrotaankin vain muutama asia: sille voi asettaa objektin, jossa sen halutaan olevan kiinni (tosin tämän voi myös jättää tekemättä, jolloin nivel tarttuu sijaintiinsa pelimaailmassa), sekä vaadittavat voima- ja vääntövoima-arvot, jos sen halutaan irtautuvan kiinnittämistään objekteista. Näiden lisäksi nivelen kiinnittämien objektien väliset kontaktit voidaan mahdollistaa ja pelimoottorin esiprosessointi poistaa käytöstä haluttaessa. (*Unity - Manual: Fixed Joint*, ei pvm.)

Spring Joint eli jousinivel-komponentti mahdollistaa objektien yhdistämisen joustavasti. Kuten muillekin nivelkomponenteille sille voidaan määrittää Connected Body eli toinen RigidBodyn sisältävä objekti johon nivel yhdistyy. Komponentti sisältää myös Anchor, Connected Anchor ja Auto Configure Connected Anchor -asetukset, jotka toimivat kuten aiemmin esitellyissä nivelkomponenteissakin. Spring-asetus määrittää miten vahva nivelen jousi on. Min - ja Max Distance -arvot määrittävät, millä välillä nivelen pitelemät objektit voivat liikkua itsenäisesti, ilman että jousi vetää niitä toisiaan kohti. Tolerance-arvolla säädetään miten paljon poikkeamaa edellä mainituista minimi ja maksimiarvoista jousi sietää. Lisäksi nivelelle voidaan määrittää rikkoutumisvoima ja -vääntö, kuten useissa

muissakin Unity-nivelkomponenteissa asettamalla Break Force ja Break Torque -arvot halutunlaisiksi. Enable Collision -asetuksella voidaan sallia niveleen kiinnitettyjen objektien osuminen toisiinsa. Enable Preprocessing -asetus määrää käyttääkö fysiikka moottori ennakkoprosessointia tätä niveltä kohtaan. Ennakkoprosessoinnin ottaminen pois käytöstä saattaa parantaa pelimoottorin vakautta tilanteissa, jossa komponentin asetuksia on mahdoton toteuttaa. (*Unity - Manual: Spring Joint*, ei pvm.)

3.4.2 Configurable Joint eli konfiguroitava nivel

Configurable Joint on kaikista saatavilla olevista nivelkomponenteista monipuolisin ja lähes mikä tahansa tähän komponenttiin vaikuttava tekijä on Unity-käyttäjän muokattavissa ja hallittavissa. Näin ollen Configurable Joint jakaa monia asetuksia edellä esiteltyjen nivelkomponenttien kanssa kuten Connected Body, Anchor, Connected Anchor ja sen automaattinen sijoittaminen ja Axis, jotka kaikki toimivat samalla tavalla kuin esimerkiksi hahmonivelen kanssa (ks. Luku 3.4.1). (*Unity - Manual: Configurable Joint*, ei pvm.)

Axis-asetuksen lisäksi, jolla siis asetetaan nivelelle kääntymisakseli, on tässä komponentissa mahdollista asettaa myös Secondary Axis, toissijainen kääntymisakseli, jolloin kolmas akseli on kohtisuorassa muihin akseleihin nähden, ja näin voidaan saada nivelen koordinaattijärjestelmä toimimaan juuri halutulla tavalla käyttäjän määrittämiä akseleita käyttäen. X,Y – ja Z Motion -asetukset määrittävät nivelelle ja sen kiinnittämille objekteille liikkumisrajoituksia yleisellä tasolla, nämä voidaan asettaa kolmeen eri tilaan: Locked, joka estää täysin nivelen liikuttamisen halutulla akselilla, Limited, joka sallii nivelen liikuttamisen halutulla akselilla muualla komponentin asetuksissa määritettyjen rajoitusten välillä ja Free joka sallii kaikenlaisen liikkumisen akselilla. Liikkumisen lisäksi voidaan rajoittaa erikseen nivelen kääntymistä Angular X, Y – ja Z Motion asetuksilla, joissa toimintavaihtoehdot ovat samat kuin edellä mainituissa sillä erolla, että nyt käsitellään siis liikkumisen sijaan kääntymisrajoituksia. (*Unity - Manual: Configurable Joint*, ei pvm.)

Sitten niihin tapoihin, joilla nivelen liikkumista voidaan säädellä ja rajoittaa komponenttiasetusten kautta. Linear Limit Spring -asetuksilla voidaan määrittää, kohdistetaanko niveleen joustoa, kun se ylittää lineaarisen liikkeen rajoitukset. Spring määrittää miten paljon, nivel joustaa eli, arvon ollessa 0 rajoitus on mahdotonta ylittää, ja

arvon ollessa tätä enemmän rajoituksesta tulee joustava. Damper -arvo puolestaan vaimentaa tätä joustovoimaa, jos se asetetaan olemaan suurempi kuin 0. Linear Limit -asetuksilla voidaan säätää tuota edellä mainittua lineaarista rajoitusta. Limit määrittää tämän rajoituksen Unityn mittayksikköjärjestelmän mukaan matkan nivelen alkuperäisestä sijainnista. Bounciness kertoo pelimoottorille, halutaanko nivelen pitelemän objektin kimpoavan rajoituksen ylittyessä. Contact Distance määrittää minimaalisen matkan nivelen sijainnin ja Limit kohdan määrittämän rajan välillä ennen kuin rajoitusta aletaan noudattaa, eli mitä isompi Contact Distance, sitä epätodennäköisemmin rajoitusta rikotaan. Suuri arvo saattaa kuitenkin vaikuttaa fysiikkamoottorin suorituskykyyn negatiivisesti. (*Unity - Manual: Configurable Joint*, ei pvm.)

Angular X Limit Spring -asetukset toimivat samalla tavoin kuin Linear Limit Spring, mutta koskevat nivelen kääntyvyyttä X-akselilla. Low Angular X Limit asetuksilla määritetään X-akselin kääntyvyyden alaraja samaan tapaan kuin Linear Limit asetuksissa. Koska kyseessä on kulmilla mitattava arvo, voidaan sille määrittää erikseen myös yläraja High Angular X Limit asetuksilla. Angular YZ Limit Spring asetuksilla asetetaan joustoarvot Y- ja Z-akselien kääntymiselle, samalla tavalla kuin Linear Limit Spring asetuksissa. Angular Y limit ja Angular Z limit -asetuksilla voidaan määrittää Y- ja Z akselien kääntymiselle raja-arvo. Komponentti tarjoaa myös muokkaustilan, jossa kääntymiskulmarajoja on mahdollista muokata visuaalisessa käyttöliittymässä samaan tapaan kuin esimerkiksi törmäyttimien osuma-alueita. Tila saadaan käyttöön klikkaamalla Edit Joint Angular Limits -näppäintä komponenttiasetuksissa. Jotta rajoituksia pääsee tässä tilassa muokkaamaan, täytyy Angular X, Y ja Z motion asetusten olla Limited -tilassa. (*Unity - Manual: Configurable Joint*, ei pvm.)

Projection Mode -asetus määrittää, miten nivel palautuu takaisin rajoitusten sallimaan sijaintiin sellaisissa tilanteissa, jossa fysiikkamoottori ei pysty operoimaan niiden sisällä. Vaihtoehdot tässä ovat None (ei mitään) tai Position and Rotation (sijainti ja kääntyminen). Jälkimmäistä käytettäessä voidaan määrittää myös Projection Distance ja Projection Angle, jotka määrittävät miten kauas rajojen yli nivelen täytyy liikkua ja/tai kääntyä ennen kuin fysiikkamoottori yrittää palauttaa sen paikoilleen. Configured in World Space -asetuksella voidaan vaihtaa kohde- ja drive-arvot laskettaviksi pelimaailman mittojen mukaan, kun ne oletuksellisesti lasketaan suhteessa nivelkomponentin sisältävään objektiin. Swap Bodies asetus määrittää missä järjestyksessä fysiikkamoottori käsittelee nivelen yhdistämät objektit.

Oletuksena ensin käsitellään se objekti, joka sisältää ko. nivelkomponentin ja sen jälkeen toinen nivelesssä kiinnioleva objekti (Connected Body), ja asetuksen ollessa päällä toimenpide suoritetaan päin vastoin. (*Unity - Manual: Configurable Joint*, ei pvm.)

Komponentille on mahdollista asettaa myös Break Force ja Break Torque jotka toimivat aiemmin kuvatulla tavalla, sillä poikkeuksella että Break Torque toimii vain jos luvun alussa mainitut yleiset liikeasetukset (X Movement jne.) ovat jossain muussa tilassa kuin Free. Enable collision asetus sallii nivelen yhdistämien objektien keskinäiset törmäykset. Disable Preprocessing saattaa myös tämän nivelen kanssa auttaa pelimoottorin vakautta mahdottomien asetusyhdistelmien kanssa operoidessa. Mass Scale ja Connected Mass Scale -arvoilla voidaan säätää nivelen yhdistämiin objekteihin kohdistuvan massan määrää, mikä voi olla hyödyllistä, jos nivelessä on kiinni kaksi objektia, joiden välinen painoero on suuri. (*Unity - Manual: Configurable Joint*, ei pvm.)

3.4.3 Drive-tila

Konfiguroitava nivel voidaan myös asettaa liikuttamaan tai kääntämään itseään antamalla sille itselleen liikevoimia, tätä tilaa kutsutaan nimellä drive. Jos näin halutaan tehdä, pitää nivelelle antaa jotain mitä kohti se pyrkii. Target Position ja Target Velocity kohtaan voidaan määrittä nivelelle sijainti ja nopeus, joihin se pyrkii drive-tilassa. XDrive -asetuksilla voidaan antaa nivelelle halutut voimat, joilla se liikkuu X-akselilla. Position Spring on vääntövoima, jolla nivelen halutaan liikkuvan kohti tavoiteltua sijaintia. Position Dampen on vastakkainen vääntövoima, joka pyrkii vaimentamaan Position Spring voimaa. Se kannattaa asetta nollaa suurempaan arvoon, jotta vältytään loputtomalta värähtelyliikkeeltä. Maximum Force määrittää ylärajan sille voimalle, mitä XDrive-asetuksella voidaan antaa. Se kannattaa asettaa niin korkeaksi, ettei ole todennäköistä että fysiikkamoottori päätyisi sitä rajoittamaan jotta drive-toiminnot toimivat halutusti. YDrive- ja ZDrive -kohtien asetukset toimivat vastaavasti Y- ja Z-akseleilla. (*Unity - Manual: Configurable Joint*, ei pvm.)

Kuten todettua, drive-tilaa voidaan myös käyttää niveleen yhdistetyn objektin kääntämiseen. Tämä saadaan aikaan asettamalla haluttu asento Target Rotation -kohdassa, ja haluttu kääntymisnopeus Target Angular Velocity -kohdassa. Voimien jakautumista voidaan jakaa Angular X Drive- ja Angular YZDrive -kohdissa (huomaa että Y ja Z-akseleiden asetukset on

yhdistetty saman kohdan alle) ja nämä toimivat samoilla säädettävillä asetuksilla kuin edellisessä kappaleessa kuvatut liikevoimaa ohjaavat asetukset. (*Unity - Manual: Configurable Joint*, ei pvm.)

Unity tarjoaa kaksi eri drive-tilaa objektin kääntämiseen. Tuo toinen tila on Slerp Drive, jossa nivelen kaikille akseleille annetaan yksi ja sama kääntövoima samannimisessä komponenttiasetusten kohdassa. Tähän liittyvät säädöt ovat vastaavat kuin edellisissä kappaleissa. Drive-tilan kääntämistapaa voidaan muuttaa vaihtamalla Rotation Drive Mode asetusta X and YZ-tilan (jossa kääntövoimat voi säätää erikseen) ja Slerp-tilan välillä. (*Unity - Manual: Configurable Joint*, ei pvm.)

3.5 Cloth eli kangaskomponentti

Unity tarjoaa Cloth-komponentillaan mahdollisuuden simuloida kankaita fysiikkamoottorin alaisina objekteina. Kankaalle voidaan määrittää halutut venymis- ja taipumiskireydet Stretching Stiffness ja Bending Stiffness -arvoilla, mitä suurempi arvo, sitä kireämpi kangas. Use Tethers -asetuksella voidaan ottaa käyttöön rajoitukset, joilla liikkuvia kangaspartikkeleita estetään siirtymästä liian kauas paikallaan olevista partikkeleista. Tällä voidaan vähentää epärealistista taipumista. Use Gravity -asetus määrää kohdistetaanko kankaaseen painovoimaa. Damping-arvolla voidaan lisätä pehmennystä kangasmateriaalin liikkeeseen. External Acceleration -kohdassa voidaan määrätä, halutaanko kankaaseen kohdistaa jatkuvaa kiihtyvyyttä, ja mistä suunnista. Random Accelerationilla tehdään sama, mutta kiihtyvyys kohdistetaan satunnaisin aikavälein. World Velocity Scale määrittää paljonko kangasta käyttävän hahmon nopeus vaikuttaa kankaaseen. World Acceleration Scale määrittää saman asian mutta kiihtyvyyttä koskien. (*Unity - Manual: Cloth*, ei pvm.)

Friction-arvo säätää paljonko kitkaa kangas kokee osuessaan sitä käyttävään hahmoon. Collision Mass Scale puolestaan ohjaa paljonko painoa törmääville kangaspartikkeleille lisätään. Use Continuous Collision -asetus asettaa jatkuvan osumantunnistuksen päälle, mikä parantaa kankaan saamien osumien vakautta. Solver Frequency -asetus ohjaa kuinka monta kertaa sekunnissa fysiikkamoottori tarkistaa kankaan tilan. Sleep Treshold -arvo kertoo, kuinka nopeasti kangasmateriaali menee ”nukkumaan” eli pois päältä, mikäli siihen ei kohdistu osumia. Capsule Colliders ja Sphere Colliders -kohtiin voidaan määrittää minkä

kapseli- ja ympyrätörmäyttimien kanssa kankaan pitää törmätä, nämä ovat yleensä kangasta käyttävän hahmon törmäyttimet. (*Unity - Manual: Cloth*, ei pvm.)

3.6 Uudet 2D-fysiikkaominaisuudet Unity 2021.2 -versiossa

Unity tarjoaa 2021.2-versiossaan kaksi täysin uutta 2D-fysiikkaominaisuutta, jotka tarjoavat uusia tapoja manipuloida 2D-törmäyttimiä. Seuraavassa käydään nämä tuoreet ominaisuudet läpi tarkemmn

3.6.1 PhysicsShapeGroup2D-rajapinta

PhysicsShapeGroup2D on uusi ohjelmointirajapinta, jolla voidaan muodostaa useamman fysiikkamuodon (PhysicsShape2D) ryhmiä, johon voidaan tallentaa listamuodossa tiedot muotojen geometriasta ja kohopisteistä. Näitä listoja ryhmän voidaan luomisen jälkeen selata ja muokata itse ryhmään viittaamalla. Uuden fysiikkamuotoryhmän luominen onnistuu ohjelmoimalla, tapoja tähän on kolme. Yksi tapa on kutsua GetShapes metodia jonkin peliobjektin törmäytinkomponenttiin (Collider2D) viitaten. Tällöin ryhmä muodostetaan kyseisen törmäyttimen muodoista. Vastaavasti on mahdollista kutsua GetShapes-funktiota jonkin peliobjektin RigidBody2D komponenttiin kohdistuen, jolloin fysiikkamuotoryhmä muodostuu kaikkien kyseiseen RigidBodyyn liitettyjen törmäytinten muodoista. (*Unity - Scripting API: PhysicsShapeGroup2D*, ei pvm.)

PhysicsShapeGroup2D voidaan myös muodostaa perustamalla tyhjä fysiikkamuotoryhmä rajapinnan tarjoaman rakentimen (Constructor) avulla ja lisäämällä siihen sitten haluttuja muotoja rajapinnan tarjoamilla metodeilla, AddCircle, AddCapsule, AddPolygon, AddBox ja AddEdges. Rajapinnasta löytyy myös Add metodi, joka mahdollistaa jo olemassa olevan fysiikkamuotoryhmän sisällön kopioimiseen siihen fysiikkamuotoryhmään, johon viitaten metodia kutsutaan. Clear metodilla haluttu fysiikkamuotoryhmä voidaan tyhjentää tiedoista. Jos halutaan hakea ryhmästä tietty muoto, onnistuu tämä GetShape metodilla, jolle annetaan parametriksi halutun muodon paikkaindeksi. Yksittäinen muoto voidaan myös poistaa ryhmästä antamalla sen indeksi parametrinä DeleteShape metodille. Jos halutaan hakea kerralla kaikki fysiikkaryhmän tiedot onnistuu se GetShapeData metodilla, joka palautta muoto ja kohopistetiedot erillisinä listoina. GetShapeVertex metodilla voidaan

hakea yksittäinen kohopiste fysiikkamuotoryhmän listasta. Tämän listanpituus voidaan selvittää PhysicsShape2D:n vertexCount -ominaisuudesta. GetShapeVertices metodilla voidaan kopioida fysiikkamuotoryhmän kaikki kohopisteet. Jos halutaan määrittää tietyn muodon viereiset kohopisteet uudelleen, se on mahdollista SetShapeAdjacentVertices metodilla. Myös yksittäisen kohopisteen määrittämminen on mahdollista käyttäen SetShapeVertex metodia. Tietyn muodon säteen uudelleen määrittäminen on mahdollista SetShapeRadius metodia käyttäen. (*Unity - Scripting API: PhysicsShapeGroup2D*, ei pvm.)

3.6.2 CustomCollider2D

CustomCollider2D on uudenlainen 2D-törmäytin, joka mahdollistaa PhysicsShapeGroup2D-rajapinnan hyödyntämisen rajattoman määrän muotoja sisältävien, juuri halutunlaisen muotoisten törmäytinten rakentamisen Unity 2D-ympäristössä. Erityisen tästä törmäyttimestä tekee myös se, että sitä voidaan muokata muokkaustilan lisäksi myös pelin aikana, jolloin tehdyt muutokset eivät tosin tallennu pelin lopettamisen jälkeen. Törmäyttimelle itselleen voidaan määrittää oma painoarvo peliobjektin RigidBody2D:n painoarvon lisäksi Density-arvolla. Density voidaan määrittää vain, jos saman peliobjektin RigidBody käyttää myös Use Auto Mass-asetusta. Material-kohtaan törmäyttimelle voidaan määrittä haluttu fysiikkamateriaali. Is Trigger -asetus määrittää käytetäänkö törmäytintä laukaisimena, tämä toimii samoin kuin muissa tässä työssä aiemmin esitellyissä törmäyttimissä. Used by Effector määrittää käyttääkö mahdollisesti peliobjektiin liitetty Effector-komponentti tätä törmäytintä. Effector-komponentit ovat 2D-komponentteja, joilla voidaan kohdistaa peliympäristöön erilaisia fyysisiä voimia esimerkiksi tietyillä alueilla tai tietyissä paikoissa. Offset kohdassa voidaan asettaa sivuttaisuutta törmäyttimelle. Custom Shape Count ja Custom Vertex Count näyttävät, kuinka montaa fysiikkamuotoa ja kohopistettä törmäytin käyttää. (*Unity - Manual: Custom Collider 2D*, ei pvm.)

4 Fysiikan toteuttaminen

Fysikkamoottorin toiminnan demonstroimiseksi tässä opinnäytetyössä tehdään pienimuotoinen Unity-peliprojekti. Projektin tekoa varten tekijä on katsonut läpi useita asiaan liittyviä opetusvideoita. Esimerkit eivät kuitenkaan perustu suoraan mihinkään materiaaliin jollei toisin mainita.

4.1 Tarvittavat työkalut

Projekti tehdään Windows 10 -ympäristössä kirjoittamisaikana uusinta vakaata Unity 2021.2 -pelimoottoria käyttäen. Unity-pelimoottoria voidaan käyttää myös macOS- ja Linux-alustoilla.

4.2 Peliprojektin kuvaus

Peliprojekti tulee koostumaan pienehköistä yksittäisistä Unity-pelimoottorin fysiikkasimulaation esimerkeistä. Projektin päätarkoituksena on esitellä nivelkomponentteja, jotka tosin vaativat toimiakseen myös perusominaisuuksia kuten RigidBody- ja törmäytinkomponentteja.

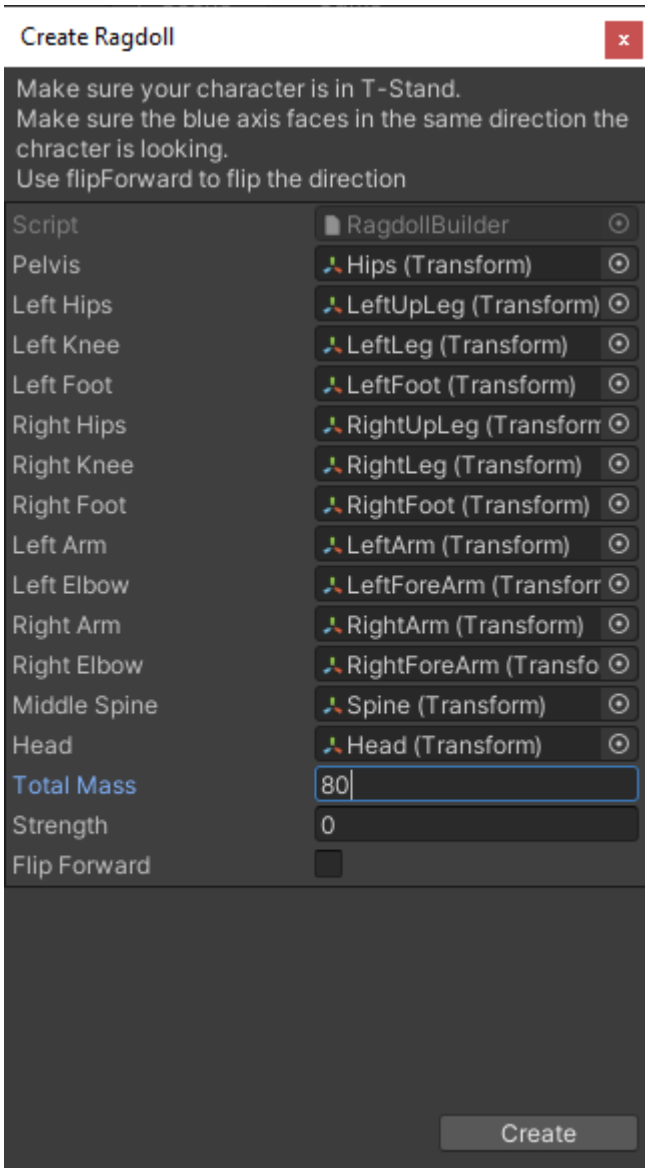
5 Peliprojektin sisällön läpikäynti

Tässä luvussa läpikäydään peliprojektin tuloksena syntyneet esimerkit, miten ne tehtiin ja mitä niillä voidaan demonstroida.

5.1 Räsynukke portaissa

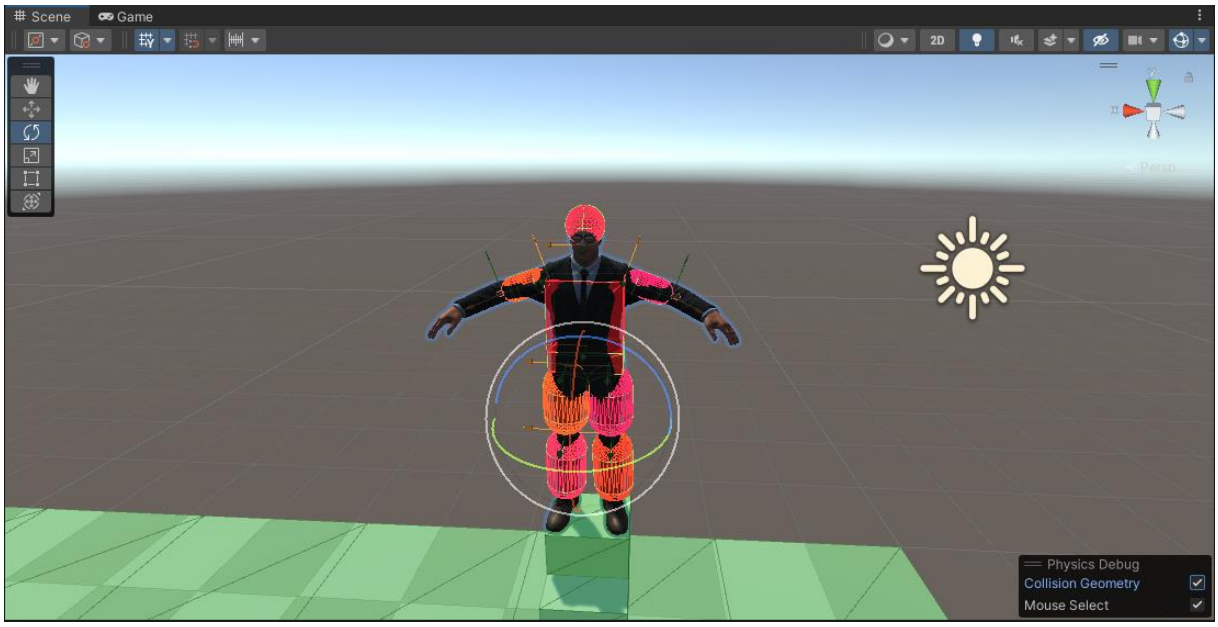
Tämä esimerkki pohjautuu Brackeys -YouTube-kanavan Ragdolls in Unity -videoon. Tässä esimerkissä tehtiin räsynukke Character Joint-, RigidBody ja Collider komponentteja käyttäen. Räsynukke asetettiin putoamaan alas portaita. Työ aloitettiin rakentamalla pienen porrasympäristön, ja asettamalla hahmon portaiden yläpäähän. Tämän jälkeen käytettiin pelimoottorista löytyvää Ragdoll-työkalua (löytyy Unityssä GameObject>3D Object>Ragdoll kohdasta) Kuva 1 Ragdoll-työkalun käyttöliittymä. Tällä saadaan hahmomallista tarpeelliset ruumiinosat valitsemalla luotua nivelet (Character Joint), RigidBodyt sekä perustörmäyttimet (Box-, Sphere-, ja Capsule Collider) suurimalle osalle hahmomallin ruumista oletusasetuksineen.

Kuva 1 Ragdoll-työkalun käyttöliittymä



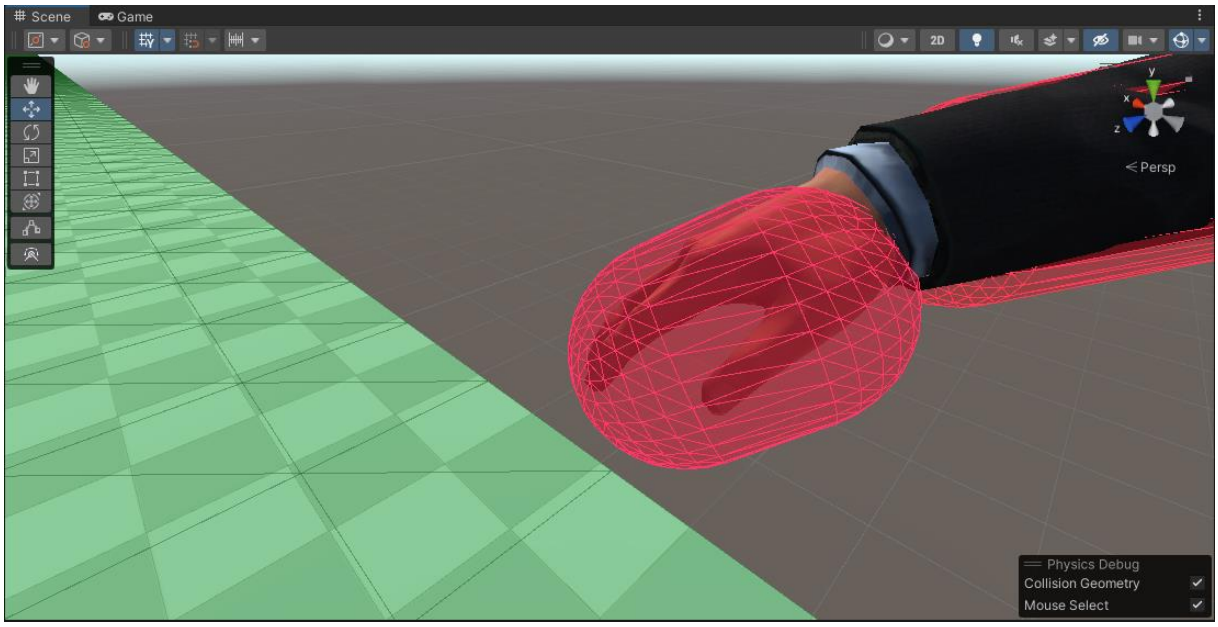
Ragdoll-työkalu mahdollistaa myös kokonaispainon määrittämisen räsynukelle, joten tämä asetettiin olemaan 80 Unity-painojärjestelmän yksikköä, joka vastaa 80 kiloa. Tässä kohtaa ottettiin käyttöön Physics Debug -työkalun, jolla voidaan analysoida fysiikkajärjestelmää (löytyy **Window>Analysis Physics>Debug** kohdasta). Tällä työkalulla voidaan nähdä esimerkiksi kaikki pelissä olevat törmäyttimet yhtä aikaa visualisoituna. Näin havaittiin, että useat äsken luodut törmäyttimet olivat jääneet liian suuriksi, alempien käsivarsien törmäyttimet liian pieniksi ja kämmenissä tai jalkaterissä ei ollut törmäyttimiä ollenkaan.

Kuva 2 Tältä hahmo näytti Physics Debug -näkymässä heti Ragdoll-työkalun käytön jälkeen.



Korjaaminen aloitettiin alemmista käsivarsista, joiden kapselitörmäyttimien suunta käännettiin X-akselin suuntaiseksi, jotta se vastaa hahmomallin käden suuntaa ja liikerataa. Seuraavaksi nollattiin törmäyttimen Center-arvot joilla, säädetään törmäyttimen sijaintia. Nämä arvot päädyttiin asettamaan seuraavasti: X:0.12, Y:0 ja Z:0.01 oikealla puolella, nämä arvot kopioitiin sitten vasemmalle puolen ja muutettiin käänteisiksi. Törmäyttimen Radius arvoksi asetettiin tässä tapauksessa 0.064 ja Height-arvoksi 0.3, niin että törmäytin vastaa kooltaan kutakuinkin olkavarren törmäytintä. Törmäyttimiä itse toteuttaessa kannatta muistaa, että niiden koko riippuu totta kai käytettävien mallien mitoista ja pelintekijöiden omista tarkkuusvaatimuksista. Pään alueella sijaitsevaa pallotörmäytintä siirrettiin hiukan alaspäin Y-akselilla, sillä se sijaitsi liian korkealla. Kaikkien jalkojen alueella sijaitsevien törmäyttimien paksuuteen vaikuttavia -Radius-arvoja pienennettiin myös hiukan, sillä ne olivat hiukan ylileveitä hahmon mittoihin nähden. Myös keskivartalo- ja lannealueiden laatikkotörmäyttimiä muokattiin hiukan suuremmiksi Z-akselilla, jotta ne ottavat iskut vastaan myös hahmon etupuolella. Tässä vaiheessa merkittävimmät virheet räsynuken luonnissa oli saatu korjattua. Hahmon liikkumamahdollisuuksia haluttiin kuitenkin vielä hiukan laajentaa, joten kämmeniin lisättiin kapselitörmäytin, RigidBody ja hahmonivelkomponentti. Tämä tehtiin ensin vasempaan kämmeneen, minkä jälkeen komponentit kopioitiin oikealle.

Kuva 3 Hahmon vasempaan käteen luotu törmäytin



Vastakkaisia komponentteja kopioidessa kannattaa pitää mielessä, että sijaintiarvot pitää muuttaa vastakkaisiksi (esimerkiksi jos arvo on 1 oikeanpuoleisessa komponentissa pitää sen olla -1 vasemmalla), jotta esimerkiksi törmäyttimet ja muut tarkkaa sijoittamista vaativat toiminnot toimivat oikein. Hahmonivelen tapauksessa Axis-arvo pitää muuttaa halutun liikkumisakseli kohdalla vastakkaiseksi. Kopioitaessa myös nivelen Connected Body jää olemaan sama kuin alkuperäisessä nivelessä, joten jos tätä on tarvetta muuttaa, täytyy tämä tehdä erikseen.

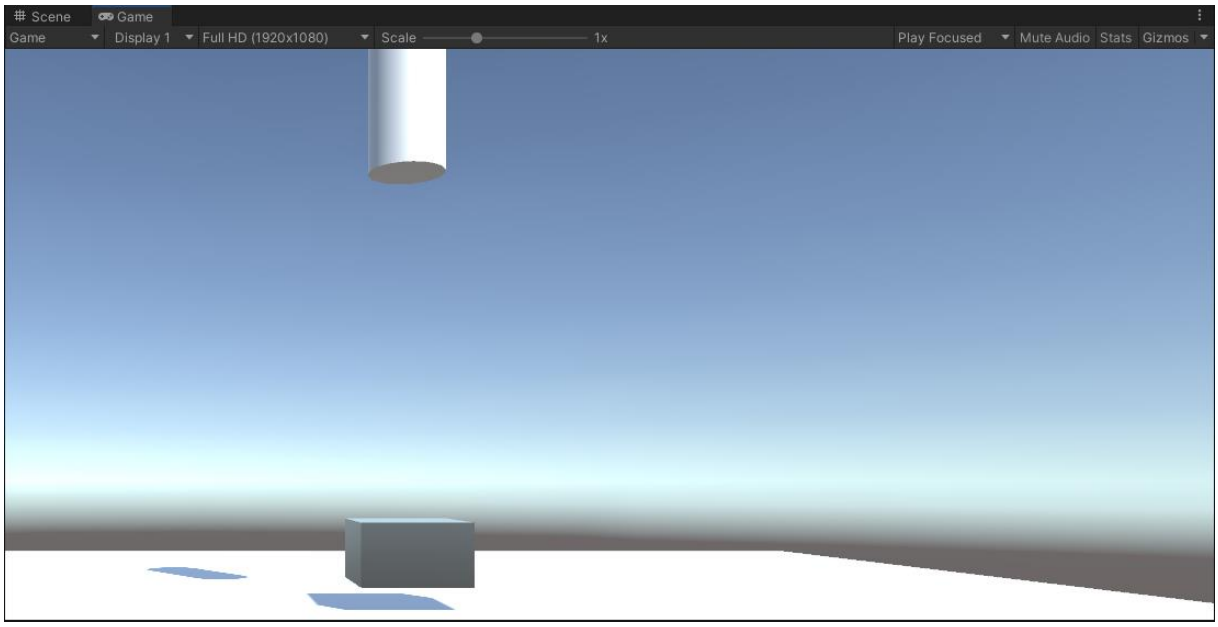
Kuva 4 Tältä hahmon osumantunnistusmalli näyttää edellä mainittujen muokkausten jälkeen.

Hahmon työntämiseksi portaita alas tehtiin myös lyhyt skripti, joka asetettiin hahmon selkärankaa mallintavan ruumiinosan komponentiksi, jotta voima kohdistuu hahmon selkään. Skriptin avulla on mahdollista myös säätä hahmoon kohdistuvaa voimaa pelimoottorin Inspector-ikkunassa. Skripti on nähtävissä kokonaan liitteessä 2.

5.2 Laatikko jousen varassa

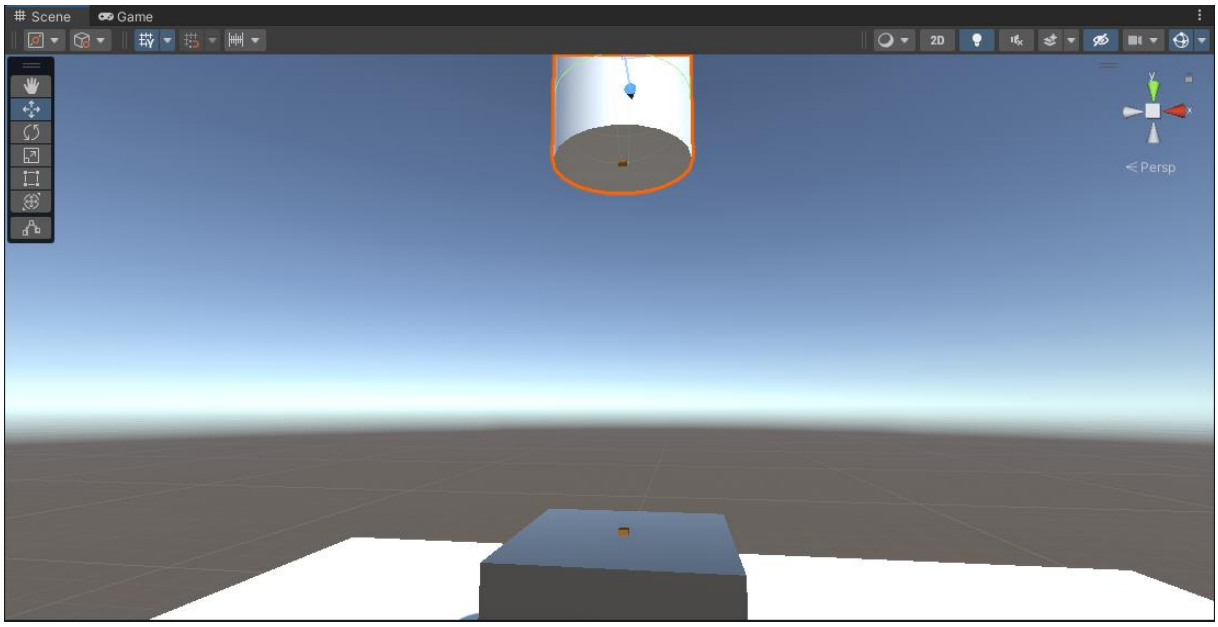
Tässä esimerkissä tehtiin jousinivelellä toteutetun nostomekanismin varassa liikuteltavissa oleva laatikko. Tämä esimerkki on rakenteeltaan erittäin yksinkertainen: se koostuu kahdesta peliobjektista; kapselipeliobjektista ja laatikko peliobjektista. Puolen metrin kokoinen kapseliobjekti asetettiin ilmaan 4,5 metrin korkeuteen, minkä jälkeen siihen liitettiin jousinivel (Spring Joint)-komponentti. Pelimoottori lisäsi tässä kohtaa myös RigidBody-komponentin, joka vaaditaan jousinivelen käyttöä varten. Tämä RigidBody asetettiin kuitenkin kinemaattiseksi, sillä kapselin on tarkoitus säilyttää paikkansa liikkumattomana toimiakseen nivelen kiinnityskohtana. Seuraavaksi peliin tuotiin kuutio-objekti, joka muokattiin laatikon muotoiseksi antamalla sille seuraavat mitat: X:0,7, Y:0.4 ja Z:0,7, minkä jälkeen laatikolle annettiin RigidBody-komponentti, joka mahdollistaa sille tässä esimerkissä kaksi olennaista asiaa, painovoiman käytön ja jousinivelen osaksi liittämisen.

Kuva 5 Jousinivel äärimmäisessä ala-asennossa.



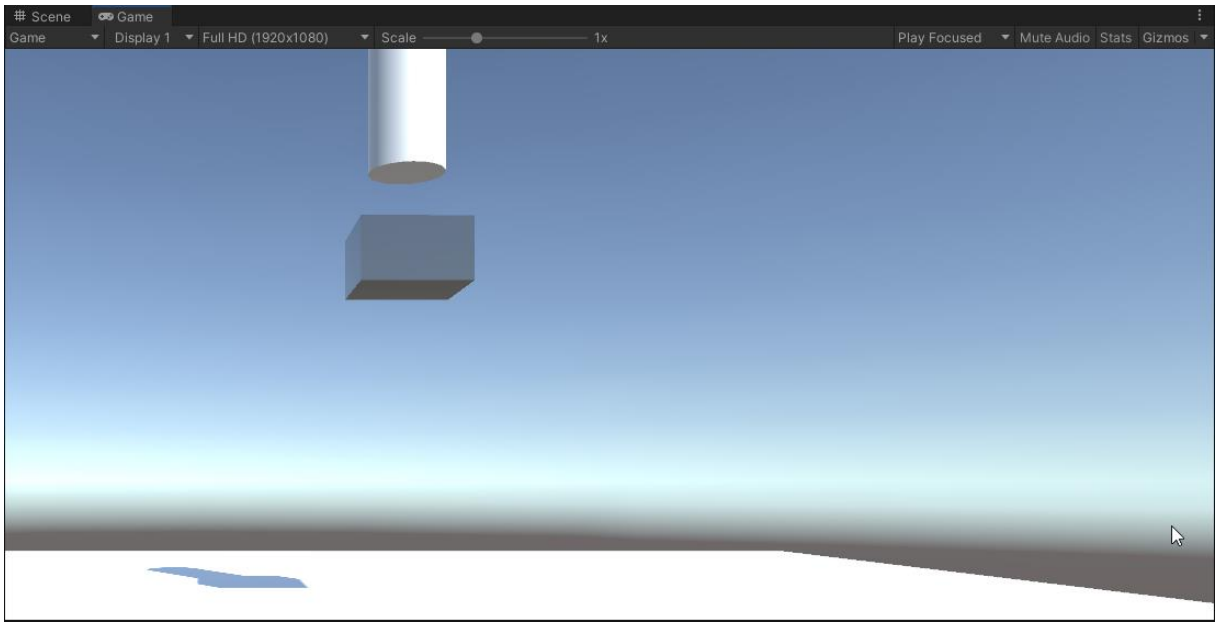
Laatikko objekti asetettiin kolmen metrin korkeuteen ilmaan, jotta päästään demonstroimaan jousinivelen kykyä pitää siihen yhdistettyä objektia ilmassa. Tämän jälkeen laatiikko-objekti yhdistettiin jousiniveleen komponentin Connected Body -kohdassa. Unityn sisäänrakennetut jousinivelen visualisaatiot ovat hyvin rajalliset, Unity visualisoi ainoastaan nivelen ankkuripisteet sekä objektissa, jonka osana nivel on, että toisessa niveleen yhdistetyssä objektissa. Näin ollen jousinivelen säätäminen tarkalleen sellaiseksi, kuten sen halutaan toimivan, on hyvin aikaa vievää. Testauksen jälkeen päädyttiin ankkuripisteet asettamaan seuraavasti: kapseliobjektin ankkuripiste: X:0 Y:0,98 ja Z:0 ja laatikko-objektin ankkuripiste: X:0, Y:0,5 ja Z:0 (nämä arvot ovat suhteellisia molempien objektien sijainteihin). Näillä arvoilla ankkuripisteet saatiin objektien ala- ja yläpäihin, joten nivel saatiin asetettua tarkalleen objektien väliin. Ankkuripisteet näkyvät kullankeltaisina kuvassa 6.

Kuva 6 Jousinivelen ankkuripisteet aseteltuna siihen liitettyjen objektien päihin



Jousinivelen jousiarvoksi asetettiin 15 ja pehmennysarvoksi (Damper) 3. Tämä vastaa myös nivelen oletusarvojen välistä 20 prosentin suhdetta, mutta juuri nuo edellä mainitut arvot valittiin siksi että ne mahdollistavat objektin liikuttelun jousinivelen varassa mitä haluttiin demonstroida, sekä sen että nivel jaksaa pitää laatikkoa ilmassa. Tämä ei onnistunut esimerkiksi oletusarvoilla 1 (jousto) ja 0.2 (pehmennys), koska jousto voima oli niin pieni, ettei se riittänyt pitämään laatikkoa ilmassa. Toisaalta jos joustoarvo on liian suuri, ja pehmennysarvo liian pieni, lähtee jousen varassa oleva objekti liikkumaan hallitsemattomasti kun siihen kohdistetaan liikevoimaa. Nivelen minimi- ja maksimietäisyydet asetettiin 0,5 ja 1,5 arvoihin niin että kapselin, maan ja laatikon väliin jää äärimmäisissä ylä- ja ala-asennoissa aina noin puoli metriä. Kuvat 5 ja 7 esittävät nivelen edellä mainitut ääriasennot.

Kuva 7 Jousinivel äärimmäisessä yläasennossa.



Laatikon liikuttelemista varten tehtiin skripti, joka nostaa laatikkoa fysiikkapohjaisesti hiiren vasenta nappia painettaessa, ja laskee sitä hiiren oikealla näppäimellä. Tämä skripti yhdistettiin sitten laatikko-objektiin. Skriptin avulla on mahdollista myös säätä laatikkoon kohdistuvaa voimaa pelimoottorin Inspector-ikkunassa. Testattaessa voima-arvoksi asetettiin 500, jolla varmistetaan, että laatikkoa jaksetaan nostaa. Skripti on nähtävissä kokonaan liitteessä 3.

5.3 Automaattisesti liikkuvat sylinterit

Tässä esimerkissä asetettiin joukko sylintereitä liikkumaan halutuilla tavoilla konfiguroitavia niveliä (Configurable Joint) ja RigidBodyja käyttäen. Tarkoituksena oli siis hyödyntää konfiguroitavien nivelten eri drive-tiloja. Tätä varten luotiin kolme 1x1x1 metrin mittaista sylinteripeliobjektia. Ensimmäinen sylinteri asetettiin peliympäristön maanpintaan kiinni. Toinen sylinteri asetetiin 4 yksikön korkeudelle ilmaan Y-akselilla, minkä lisäksi se käännettiin 90 asteen kulmaan X-akselilla, jolloin se saatiin käännettyä poikittain. Kolmas objekti asetettiin 4 yksikön korkeuteen, jotta äsken mainittu objekti mahtuu pyörimään sen ja alimmaisen objektin välissä. Kun sylinterit oli saatu aseteltua, lisättiin alimpaan sylinteriin Configurable Joint -komponentti. Samalla Unity lisäsi objektiin myös RigidBodyn ja kapselitörmäyttimen, jotka ovat olennaisia nivelen toiminnan kannalta. Sitten keskimmäinen sylinteri joka haluttiin saada pyörimään ympyrää sivuttaissuunnassa kahden muun objektin

välissä, liitettiin niveleen sen toisena osana. Nivelen pääakseliksi asetettiin X-akseli arvolla ja toissajaiseksi akseliksi Y-akseli niin ikään arvolla 1. X, Y ja Z akselien liike lukittiin, sillä objektien ei tässä testitapauksessa haluttu liikkuvan, ainoastaan kääntyvän. Halutun kiertoliikkeen saavuttamiseksi nivelle sallittiin vapaa kiertoliike Y-akselilla ja muut kiertoakselit lukittiin. Tässä konfiguraatiossa Y-akseli on siis se akseli, jonka kautta sylinterien on mahdollista pyöriä itsensä ympäri. Koska nivelen haluttiin pyörivän, asetettiin komponentin Target Rotation -asetuksissa halutuksi kiertonopeudeksi (Target Angular Velocity) 5 Y-akselilla. Näin ollen nivel pyrkii saavuttamaan viiden kierroksen sekuntinopeuden. Tämän jälkeen Angular YZ Drive -asetuksissa annettiin Position Spring -arvoksi 2, mikä antaa nivelelle alkuliikenopeudeksi kaksi kierrosta sekunnissa. Position Damper -arvoksi annettiin yksi, jolloin nivelen liikevoimaa vaiennetaan puolella suhteessa sen alkunopeuteen.

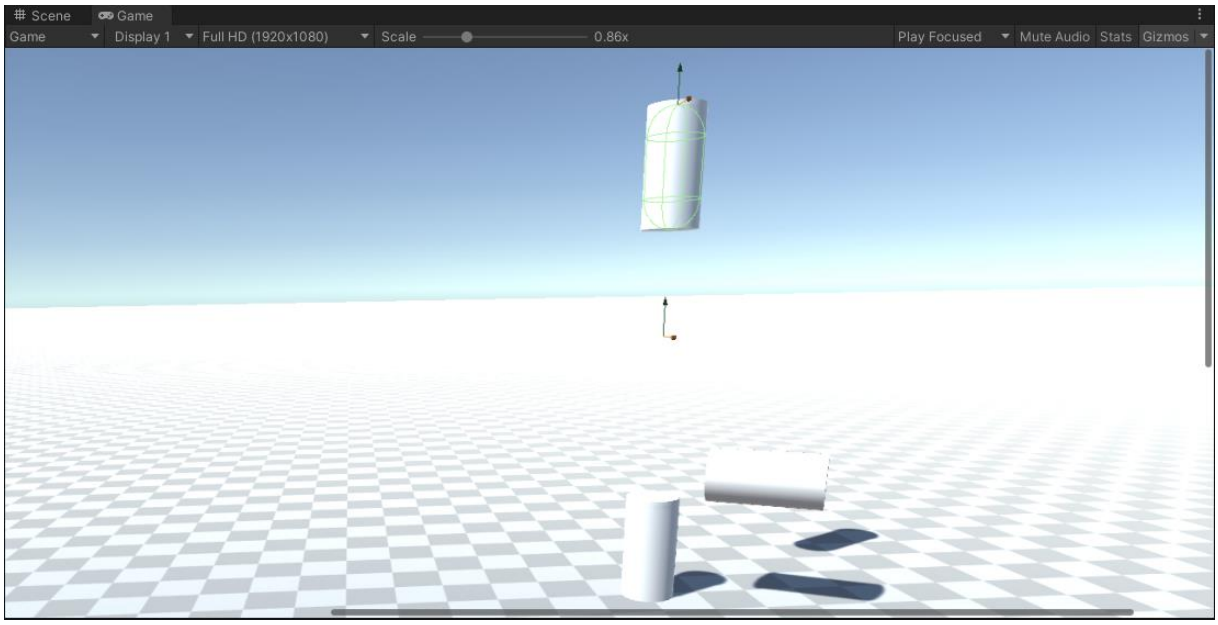
Näillä asetuksilla voitiin aloittaa pyörimismekanismin testaus. Testausvaiheessa ongelma na ilmeni kuitenkin, että nivelen pyörittäessä itseään ja molempia siihen kiinnitettyjä objekteja, nopeuden noustessa alkunopeutta korkeammaksi lähtee pohjimmainen objekti kaatumaan aiheuttaen, sen että keskimmäinen objekti osuu sivusuunnassa pyöriessä ylimpään objektiin. Tämä johtaa loppujen lopuksi koko rakennelman kaatumiseen ja liikkeen pysähtymiseen. Tämä ongelma ratkaistiin asettamalla alimman sylinterin RigidBody kinemaattiseksi. Näillä asetuksilla keskimmäinen sylinteri voi pyöriä myötäpäivään menettämättä nopeuttaan.

Seuraavaksi siirryttiin muokkaamaan ylintä sylinteriobjektia, joka haluttiin saada liikkumaan pystysuunnassa. Näin ollen siihenkin liitettiin konfiguroitava nivel. Tämän olisi voitu suorittaa myös yhdistämällä pohjimmainen sylinteriobjekti nivelellä ylimpään objektiin, mutta havaittiin, että yhteen objektiin on mahdotonta liittää useampaa samanlaista nivelkomponenttia eri toiminnoilla. Tällä kertaa nivelelle ei annettukaan Connected Anchor -kiinnityskohtaa, jolloin nivel kiinnittyy toisesta päästä johonkin paikkaan pelimaailmassa. Tämä kohta annettiin Unityn automaattisesti määritettäväksi ja päätyi olemaan Unity-koordinaatein X: -5, Y:5, Z:-10. Tässä kohtaa lienee hyvä mainita että itse sylinterin sijainti on X:-5, Y:4, Z:-10, eli nivelen toinen, pelimaailmaan kiinnittyvä pää sijaitsee siis hiukan sylinterin yläpuolella. Tästä johtuen pystyliikkeen mahdollistamiseksi Target Positionin Y koordinaatiksi päädyttiin asettamaan -1. Normaaliolosuhteissa tuo arvo voisi siis olla positiivinen mutta koska nivel on pelimoottorin logiikan mukaan ylösalaisin, täytyy arvon olla

negatiivinen. Nopeudeksi pystyliikkeelle annettiin 5 YDrive -kohdan Position Spring arvolla, Position Damper arvo jätettiin tässä tapauksessa nollaan, sillä sen nostamisen havaittiin rajoittavan nivelen liikerataa merkittävästi, johtaen lopulta liikkeen pysähtymiseen. Nopeuden menetystä on komponentin kautta mahdotonta eliminoida täysin pystysuuntaisessa liikkeessä, mutta asiaa tutkiessa havaittiin, että liike-etäisyyden pidentäminen ja korkea YDrive liikenopeus auttavat niveltä ylläpitämään liikkeen pidempään. Tämän vuoksi Target Position arvo nostettiinkin -2:een, jotta nivelellä on enemmän liikkumavaraa.

Niin ikään asiaa tutkittaessa Unityn gizmos-työkalulla, havaittiin että nivelen ankkuripisteet olivat oletuksena aseteltu erittäin kauas nivelen tarkoituksenmukaisesta sijainnista, mikä osaltaan vaikutti ylimmän sylinteriobjektin epäloogiseen hidastumiseen. Kuva 8 näyttää miten ankkuripisteet oli oletuksena Unityn toimesta aseteltu.

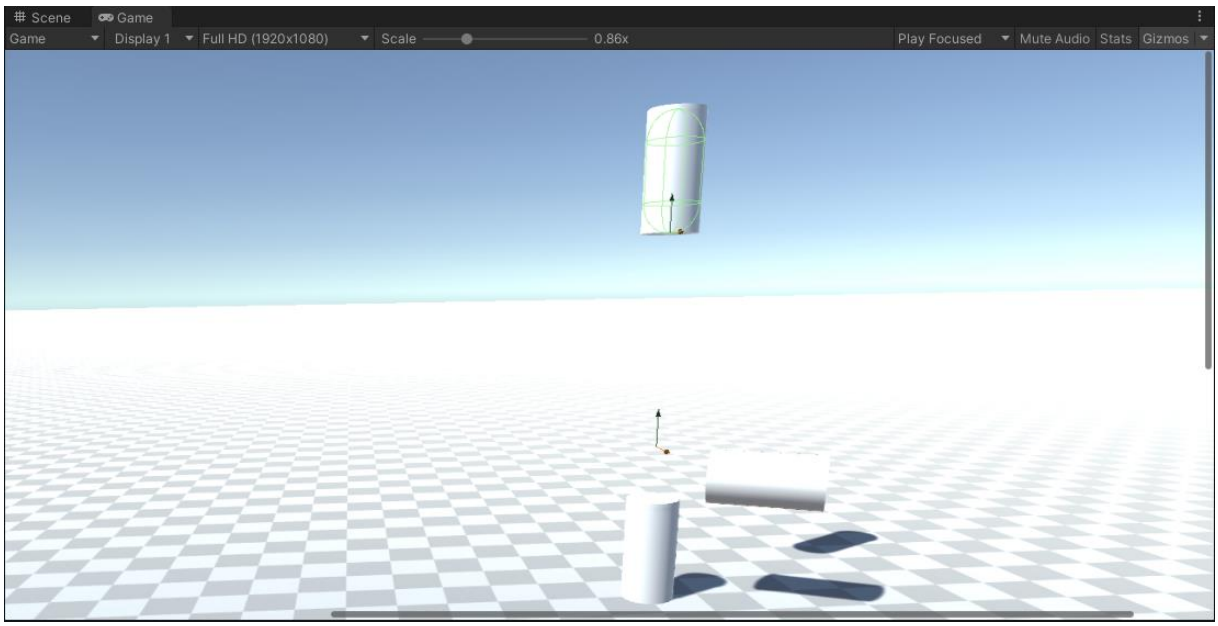
Kuva 8: Nivelen oletuskiinnityskohdat nuolilla esitettyinä



Tämä aiheutti sen, että pysähtyessään sylinteri jäi paljon haluttua ylemmäs ilmaan. Loogisemman liikeradan saavuttamiseksi objektissa oleva kiinityspiste (Anchor), korjattiin suhteellisesta sijainnista X:0, Y:1, Z:0 sijaintiin X:0, Y:-1, Z:0, jolloin se siirtyi sylinteriobjektin ylimmästä kohdasta sylinterin alimpaan kohtaan. Myös pelimaailmaan kiinnittyvä ankkuri siirrettiin manuaalisesti kohtaan X:-5, Y:3, Z:-10, joten se on nyt sijoitettu olemaan scenen alkaessa ylimmän sylinterin pohjaa vastaavassa kohdassa. Näillä muutoksilla ankkurit saatiin

sijoiteltua täsmälleen niin kuin ne olikin loogisesti tarkoitus asettaa. Muutokset näkyvät kuvassa 9.

Kuva 9 Nivelen kiinnityskohdat korjausten jälkeen



Tämä ei ratkaissut aiemmin mainittua ongelmaa liikenopeuden menetyksen kanssa, mutta nyt nivelen liikerata vastaa haluttua lopputulosta.

6 Tulokset ja yhteenveto

Unity tarjoaa sisäänrakennettuna monipuolisen fysiikkajärjestelmän sekä 2D- että 3D-ympäristöihin. Pelin toteutuksen kannalta keskeisimmät fysiikkaominaisuudet ovat nk. RigidBodyt, törmäyttimet sekä fysiikkamateriaalit. Muita ominaisuuksia ovat nivelrakenteiden simulointi sekä kangasmateriaalin fysiikkasimulaatio. 2D- ja 3D-pelien fysiikkakomponentit ovat toiminnaltaan pitkälti samankaltaisia, joten esimerkiksi 3D-fysiikasta 2D-fysiikkaan siirtyminen on suhteellisen helppoa.

Unityn fysiikkajärjestelmää voidaan tukea C#-ohjelmoinnilla, jonka kautta päästään muokkaamaan fysiikkakomponenttien asetuksia. Aiheesta löytyy lisätietoa esimerkiksi Unity-manuaalin scripting -osasta, jota ei tässä työssä käsitelty syvemmin, jotta pystyttiin keskittymään enemmän itse työlle olennaisiin komponentteihin. Ohjelmointiin tutustuminen on kuitenkin hyödyllistä sillä se mahdollistaa esimerkiksi fysiikkaolosuhteiden muokkaamisen pelitapahtumiin perustuen.

Unity 2021.2 -versioon on lisätty uusi PhysicsShapeGroup2D-rajapinta sekä sitä hyödyntävä CustomCollider2D törmäytin, jotka parantavat ja monipuolistavat 2D-törmäytinten luomista merkittävästi. Huomattavaa on myös, että CustomCollider2D on aiemmista törmäyttimistä poiketen täysin muokattavissa myös pelin ollessa käynnissä.

Opin tätä opinnäytetyötä tehdessä paljon uutta Unityn fysiikkamoottorin ominaisuuksista ja toiminnasta. Ennen tätä osasin käyttää RigidBodyjä, törmäyttimiä ja fysiikkamateriaaleja perustasolla, mutta etenkin törmäyttimistä löytyi useita itselleni vähemmän tuttuja hyvin spesifin käyttötapauksen mahdollistavia komponentteja kuten rengastörmäytin. Myös etenkin nivelkomponenteista opin paljon, olin niistä tietoinen vain ominaisuuden tasolla, enkä ollut käyttänyt niitä kertaakaan ennen tämän työn käytännön osaa. Löysin 2D-fysiikasta myös uuden ominaisuuden, CustomCollider2D:n, joka monipuolistaa entisestään 2D-törmäyttimien käyttömahdollisuuksia, varsinkin kun sitä voidaan muokata jopa pelin aikana PhysicsShapeGroup2D-rajapinnan avulla.

Unityn roadmap-dokumentaatiosta selviää, että Unityn kehittäjät haluavat uudistaa pelimoottorin ydintoimintoja lähitulevaisuudessa radikaalisti DOTS (Data-Oriented

Technology Stack) teknologiapaketilla, joka sisältää mm. kokonaan uuden entiteettipohjaisen renderöintijärjestelmän. DOTSin toivotaan parantavan suuresti pelimoottorin skaalattavuutta, suorituskykyä ja moninpelimahdollisuuksia sekä myös mahdollistavan tulevaisuuden pelialustojen sujuvan tukemisen. DOTSin tarjoamilla laajamittaisilla parannuksilla on varmasti myötävaikutuksia myös fysiikkasimulaatioon ja sen toteutusmahdollisuuksiin. Myös tämänhetkisen fysiikkamoottorin pohjana toimiva Nvidia PhysX on vanheneva teknologia, ja onkin todennäköistä, että Unity Technologies suunnittelee laajempaa fysiikkamoottorin päivitystä tulevaisuudessa.

Lähteet

Intro to the Unity Physics Engine—2019.3—Unity Learn. (ei pvm.). Noudettu 20. tammikuuta 2022, osoitteesta https://learn.unity.com/tutorial/intro-to-the-unity-physics-engine-2019-3?tab=overview

Unity Manual: RigidBody. (ei pvm.). Noudettu 18. tammikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-Rigidbody.html

Unity—Manual: Box Collider. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-BoxCollider.html

Unity—Manual: Capsule Collider. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-CapsuleCollider.html

Unity—Manual: Character Controller. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-CharacterController.html

Unity—Manual: Character Joint. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-CharacterJoint.html

Unity—Manual: Cloth. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/Manual/class-Cloth.html

Unity—Manual: Colliders. (ei pvm.). Noudettu 19. tammikuuta 2022, osoitteesta https://docs.unity3d.com/Manual/CollidersOverview.html

Unity—Manual: Configurable Joint. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-ConfigurableJoint.html

Unity—Manual: Custom Collider 2D. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-CustomCollider2D.html

Unity—Manual: Fixed Joint. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-FixedJoint.html

Unity—Manual: Hinge Joint. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-HingeJoint.html

Unity—Manual: Joints. (ei pvm.). Noudettu 21. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2020.2/Documentation/Manual/Joints.html

Unity—Manual: Mesh Collider. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-MeshCollider.html

Unity—Manual: New in Unity 2021.2. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/WhatsNew20212.html

Unity—Manual: Physic Material. (ei pvm.). Noudettu 20. tammikuuta 2022, osoitteesta https://docs.unity3d.com/Manual/class-PhysicMaterial.html

Unity—Manual: Physics. (ei pvm.). Noudettu 18. tammikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/PhysicsSection.html

Unity—Manual: Sphere Collider. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-SphereCollider.html

Unity—Manual: Spring Joint. (ei pvm.).

Unity—Manual: Terrain Collider. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-TerrainCollider.html

Unity—Manual: Wheel Collider. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/Manual/class-WheelCollider.html

Unity—Scripting API: PhysicsShapeGroup2D. (ei pvm.). Noudettu 3. helmikuuta 2022, osoitteesta https://docs.unity3d.com/2021.2/Documentation/ScriptReference/PhysicsShapeGroup2D.html

Liite 1: Aineistonhallintasuunnitelma

Työn tuloksena syntyvää peliprojektia säilytetään tekijän tietokoneen F-levyllä, ja siitä tehdään jokaisen muutoskerran jälkeen pakattu varmuuskopio tekijän HAMK-OneDriveen. Jos projekti todetaan toimivaksi muutoskerran jälkeen, edellinen varmuuskopio poistetaan uuden tieltä. Päiväkirjaa säilytetään F-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta.

Peliprojektin lisäksi työssä ei synny muuta olennaista aineistoa.

Työn aineiston ja tulokset omistaa tekijä, mutta niiden uudelleenkäyttö on sallittua.

Liite 2: Objektin työntämisen mahdollistava ohjelma, C#-kieli

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

public class AddForce : MonoBehaviour
{
    //Luodaan käyttäjän määriteltävissä oleva voima-arvo.
    [SerializeField] float force;
    //Luodaan viittaus ohjelman kohdeobjektin RigidBodyyn.
    Rigidbody rb;

    void Start()
    {
        //Haetaan kohdeobjektin RigidBody käyttöön.
        rb = this.GetComponent<Rigidbody>();
        //Kohdistetaan RigidBodyyn voimaa eteen päin.
        rb.AddForce(Vector3.forward*force,ForceMode.Force);
    }

    void Update()
    {

    }
}
```

Liite 3: Objektin nostamisen ja laskemisen mahdollistava ohjelma, C#-kieli

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

public class Lift : MonoBehaviour
{
    //Luodaan käyttäjän määriteltävissä oleva nopeusarvo.
    [SerializeField] float speed;
    //Luodaan viittaus ohjelman kohde-objektin RigidBodyyn.
    private Rigidbody rb;
    void Start()
    {
        //Haetaan kohdeobjektin RigidBody käyttöön.
        rb = this.GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void Update()
    {
        //Määritellään kohdeobjekti liikkumaan ylös- tai alaspäin hiiren näppäimiä
painettaessa.
        if(Input.GetKey(KeyCode.Mouse0))
        {
            rb.AddForce(Vector3.up * speed*Time.deltaTime);
        }
        else if (Input.GetKey(KeyCode.Mouse1))
        {
            rb.AddForce(Vector3.down * speed*Time.deltaTime);
        }
    }
}
```