Haaga-Helia
ammattikorkeakoulu Oy

DreamHouse 2016

Jon Koivula

Opinnäytetyö

Tietojenkäsittely koulutusohjelma

2015

Haaga-Helia ammattikorkeakoulu Oy

Tiivis-telmä

Päiväys

Tekijä(t)
Jon Koivula

Koulutusohjelma
Tietojenkäsittely

Opinnäytetyön otsikko
Dreamhouse 2016

Sivu- ja liitesivumäärä
36 + 1

Opinnäytetyön otsikko englanniksi
Dreamhouse 2016

Tämän opinnäytetyön tehtävänä on luoda sovellusmainen peli, jossa käyttäjä voi asetella 3D-malleja virtuaalimaailmassa käyttäen *Oculus* Virtuaalilaseja. Peli tarjoaa muutamia valmiita malleja tätä varten, mutta mahdollistaa omien 3D-mallien lisäämisen peliin. Pelin tarkoitus on antaa käyttäjälle mahdollisuus sisustaa tyhjiä huoneita eri ympäristössä haluamallaan tavalla.

Opinnäytetyön tuloksena on peli, joka helpottaa ja opettaa käyttäjää sisustamaan huoneita. Peliä voi käyttää apuna hahmottamaan, miten huoneita voisi sisustaa eri huonekaluilla. Tätä voi hyödyntää siten, että käyttäjä voi ensiksi kalustaa pelissä ja käyttää samaa sisustus suunnitelmaa omassa elämässään. Tätä varten tarjotaan muutamia ympäristöjä, jotka ovat suuntaan antavia ajatuksia, miten kannattaa sisustaa huoneita.

Koska peli tarjoaa mahdollisuuden sijoittaa omia 3d malleja huoneisiin, voi käyttäjä luoda asunnosta juuri sellaisen kuin haluaa. Pelissä toimii mikä tahansa 3d malli tiedosto, kunhan ne löytyvät niille osoitetusta kansiosta ja pelaajalla on Unity asennettuna koneelle. Omien huoneiden luominen ei ole osa tätä projektia. Pelaaja on pelattava vain valitussapeliympäristöissä.

Dreamhouse 2016 peli tukee *Oculus Rift* virtuaalilaseja. Pääsääntöisesti tätä peliä suositellaan pelaamaan virtuaalilasien kanssa, tällöin hahmottaminen ja suunnittelu on helpompaa, kun käyttäjä tuntee olevansa itse huoneessa jota kalustaa. Jos käyttäjä ei omista *Oculus* virtuaalilaseja voi peliä pelata normaalisti näytöllä näppäimistön ja hiiren avulla.

Opinnäytetyön projektin tekemiseen käytetään *Unity* pelimoottoria ja ohjelmointikielenä C#. *Unity* pelimoottori tukee *Oculus* Virtuaalilaseja, tämä oli vahva syy miksi *Unity* valittiin tätä projektia varten.

Asiasanat
Unity, Oculus, C#, 3D mallit

Haaga-Helia
University of Applied Sciences

Abstract

Date

Author(s)
Indicate the author(s) here, first name before surname, alphabetized according to surname.

Degree programme
Business Information Technology

Report/thesis title
Dreamhouse 2016

Number of pages and appendix pages
36 + 1

The purpose of this thesis was to create a game, where the user is able to put 3D models in a virtual game world by using Oculus virtual glasses. The *Dreamhouse 2016* game offers a few pre added Models to be used in the game, but it also features an option for the user to add own models. The purpose of this game is to give an opportunity for players to decorate empty rooms with models as they wish.

The result of this work is a game, which teaches users to decorate rooms with their own imagination. This game can be used also to perceive how to decorate rooms. The benefit of it is that the user can first decorate a room in a game to see the results, and then use the same decoration ideas in reality. This game will be used as a demo by Timehouse in business meetings to sell architecture solutions in virtual reality environments. This demo will be shown with a new name called Tila3VR.

The game offers an opportunity for the user to put his/her own created 3D-models into virtual world. These 3D-model files should be placed in a certain folder, so it will be included in the game. Unity software is required in order to use own 3D-models within the game.

The Dreamhouse 2016 game supports *Oculus DK2* virtual glasses. Mainly this game is recommended to be played with *Oculus DK2*. The reason for this is that it's easier for the user to understand how to decorate the room, when it feels like he/she is inside that room. However, if the user does not own *Oculus DK2* VR glasses, the game can be played normally on screen using a mouse and keyboard.

This project was built by using *Unity game engine* and with a programming language called C#. Unity supports *Oculus* virtual glasses, which is why I chose this engine for my project.

The result of this project was a very interesting and working demo for marketing purposes at Timehouse.

Keywords
Unity, Oculus, C#, 3D Models

Sisällys

1 Johdanto

Tämän opinnäytetyön tavoite on luoda peli *Unity* pelimoottorilla, jossa käyttäjä pystyy asettamaan 3D malleja pelin sisällä haluallaan tavalla käyttäen *Oculus* virtuaalilaseja. Käyttäjä liikkuu tyhjissä huoneissa ja hänellä on eri huonekaluja käytettävissä, joilla sisustaa huoneita. Tämän projektin aikana luotua peliä voi käyttää apuna suunnittelemaan omia huoneita eri huonekaluilla, sekä omien 3D mallinnuksien tarkastelussa pelissä.

Käyttäjän pitää pystyä asettamaan useita eri huonekaluja huoneeseen oman suunnitelman mukaisesti *Oculus* virtuaalilaseilla. Virtuaalilasit mahdollistaa sen että käyttäjä näkee helpommin etäisyydet toisiin 3D malleihin, myös hahmottaminenkin on helpompaa, kun tuntee olevansa itse huoneen sisällä.

Tämän työn toimeksiantaja on *Timehouse Oy,* joka tulee käyttämään tätä työtä markkinointi tarkoituksiin muun muassa mainostamaan omia arkkitehtuurillisia töitä.

Tämän työn toteuttamiseen tarvitaan laajaa osaamista C#-ohjelmointikielestä, sekä paljon kokemusta *Unity* pelimoottorista. Jotta tämän opinnäytetyön tekeminen olisi mahdollista, opiskelin *Unity* pelimoottorin käyttöä laajalti. C#-ohjelmointikieltä ei minun tarvinnut opiskella, koska ammatikseni käytän päivittäin C#-ohjelmointikieltä

1.1 Projektin tausta

Opinnäytetyössä käytin *Unity 5.3* versiota pelimoottorista, koska se oli ainoa mikä tukee C#-ohjelmointikieltä. Koska teen työkseni ohjelmistoja ja web-sivustoja C#-ohjelmointikielellä, minun ei tarvinnut opiskella uutta ohjelmointikieltä, oli Unity pelimoottori selkeä valinta.

Jotta tämän työn tekeminen oli mahdollista, minun piti opiskella Unityn käyttöä yleisesti. Ostin www.udemy.com sivustosta kurssin, jonka avulla opiskelin Unityn käyttöä samalla kun tein tätä projektia. Koska en omistanut aikaisempaa merkittävää kokemusta Unitystä, tämän kurssi käyminen projektin aikana oli lähes pakko. Kurssin suorituksen loppu puolella olin jo saanut työni melkein valmiiksi, joten kaikkia asioita ei tarvinnut käydä läpi kurssilla.
Unitystä löytyy tuki Sonyn *Morpheus- ja Oculus* virtuaalilaseille, tämä vahvisti päätöstä tämän kyseisen pelimoottorin valitsemiseen, sillä itse omistan Oculus lasit, joita pystyin hyödyntämään peliä tehdessä. Unitystä voi laittaa asetuksen päälle, että pelin kamerat kytkeytyvät tietokoneeseen kytkettyihin virtuaalilaseihin jolloin testaaminen ja pelaaminen laseilla projektin aikana on mahdollista.

1.2 Projektin tavoite ja rajaus

Projektin tavoitteena on luoda peli, jossa käyttäjä liikkuu näppäimistön ja hiiren avulla virtuaalimaailmassa Oculus laseja käyttäen. Pelaaja tuntee olevansa itse pelin sisällä ja täten 3D mallien asettaminen tuntuu luontevammalta ja hauskemmalta. Käyttäjä pystyy myös vaihtamaan 3D mallien kokoa ja suuntaa ennen niiden asettamista.

Yksi työn tavoitteista on antaa käyttäjälle mahdollisuus ottaa peliin mukaan omia 3D mallinnuksia. Tämä on olennaista pelin käytettävyyden takia, koska on lähes mahdotonta laittaa peliin valmiiksi tuhansia eri malleja käytettäväksi. Tästä syystä tästä käyttäjä voi itse luoda haluamansa 3D mallit eri 3D mallinukseen soveltuvissa ohjelmistoissa kuten *Blender*, ja ottaa ne käyttöön itse peliin. Käyttäjä voi myös ostaa/ladata haluamiaan malleja erinäisistä sivustoista kuten *http://archive3d.net tai https://www.cgtrader.com/free-3d-models*

Projekti on rajattu siihen, ettei käyttäjällä ole mahdollisuutta rakentaa omaa taloa. Käyttäjä ei pysty myöskään asettamaan täten seiniä, ja siten muodostaa omia huoneita jo valmiiden huoneiden lisäksi. Tällaisen mahdollisuuden työstäminen projektiin olisi vaatinut paljon enemmän aikaa, todella hyvää matemaattista osaamista, että huomattavasti syvempää Unity pelimoottorin osaamista. Tästä syystä nämä ominaisuudet on rajattu pois tästä projektista ja käyttäjällä on vain rajattu määrä huoneita joita sisustaa.

1.3 Projektin vaatimukset

Projektin vaaditaan kokoemusta C#-ohjelmointikielestä, sekä olio-ohjelmoinnista. On myös hallittava perusasiat Unity pelimoottorista. On tiedettävä muun muassa, miten Unityssä voidaan luoda kohtauksia, tunnettava käyttöliittymää, ymmärrettävä objektien hallintaa ja hierarkiaa sekä niihin kytkettyjä komponentteja. On myös olennaista ymmärtää eri komponenttien toiminnallisuuksia. Tätä projektia varten on hyvä tietää yleistasolla, miten Fysiikkaan liittyvät komponentit toimivat. Tämän tietotaidon pystyy saamaan melkein mistä tahansa Unityn aloittelijakurssista. Ne tiedot ja taidot mitä ei saa, pitää itse opetella internetin avustuksella.

1.4 Käsitteet

Käsittelen tässä seuraavia käsitteitä, joita tulen käyttämään tässä raportissa.

C#	C# on Microsoftin luoma ohjelmointikieli .NET alustalle.

Unity	Unity on monialustainen pelimoottori pelien luomista varten.
3D	3D on lyhenne three dimension sanasta, joka tarkoittaa yleisellä tasolla kolmiulotteista.
Oculus/Oculus Rift	Oculus on virtuaalilasit, joka on amerikkalaisen Oculus VR yhtiön kehittämä.
Blender	Blender on avoimeen lähdekoodiin perustuva ohjelmisto, jolla luodaan 3D sisältöä.
Skripti	Unityssä ohjelmointikoodit kutsutaan usein termillä Skripti. Unityssä objekteihin liitetään ohjelmointikoodia, näitä koodi pätkiä kutsutaan Skripteiksi.
Gameobject	Tämä Unity pelimoottorissa yleinen käsite mistä tahansa objektista(peliobjekti). Tämä voi olla kuten hahmo, kamera, valon lähde, lattia tai tapahtumakomponentti.
Komponentti	Komponentti on toiminallisuus ja/tai datan tiedon säilytys paikka Unityn peliobjektista(gameobject). Esimerkiksi Unityssä peliobjekti voi omistaa komponentin ”Transform”, joka kertoaa missä päin 3D maailmaa tämä kyseinen objekti sijaitsee. Toinen esimerkki on, että komponentti voi olla toiminallisuus peliobjektissa, joka tuottaa ääntä.
UI-elementti	UI-elementti on Unityssä käyttöliittymää koskeva osa.
Event System	Event System on Unityssä gameobjekti, joka käsittelee pelissä tapahtumia Eventtejä eli tapahtumia.
Event	Event on tapahtuma tai toiminto joka suoritetaan tietyn tapahtuman jälkeen. Tämä voi olla kuten hiiren klikkaus nappulaan, joka suorittaa seuraavan tason lataamisen, kutsutaan Eventiksi. Event on mikä tahansa tapahtuma, joka tapahtuu tietyn toiminnon seurauksena.

SDK	SDK on lyhenne englanninkielisestä sanasta Software Development Kit. Se on toisin sanoen kirjasto, joka tarjoaa valmista koodia kehittämään sovellusta taikka peliä.
MonoDevelop	MonoDevelop on kääntäjä, joka tulee Unityn pelimoottorin mukana. Se tukee C#- ja javascript skirptien käännökset.
Visual Studio	Visual Studio on Microsoftin kehittämä tunnettu kääntäjä. Unity tarjoaa tuen kääntää ja kirjoittaa ohjelmointikieltä Visual studiossa, joka on integroitu Unityyn kehittämistä varten.
Olio-ohjelmointi	

2 Unity – pelimoottori

Unity on monialustainen pelimoottori, jota käytetään pelien luomiseen. Tämän pelimoottorin on kehittänyt *Unity Technologies*. Unityllä voi luoda video pelejä tietokoneelle, konsoleille, kännyköille ja web-sivustoille. Unity julistettiin vuonna 2005 vain OS X käyttöjärjestelmälle. Kovan suosion myötä vuosien jälkeen, Unityn tuki on laajennettu muillekin alustoille.

Unity on yleisesti tunnettu helposta käytettävyydestä, sekä laajasta monialustaisesta tuesta. Videopelien tekeminen yleisesti ottaen on todella haastavaa ja vaatii laajaa osaamista matematiikan ja fysiikan puolella, sekä ohjelmointikielessä. Unity pyrkii helpottamaan videopelien tekemistä eri alustoille. Unityä käyttää yli 1.3 miljoonaa kehittäjää pelien luomiseen.

Pelimoottori tukee seuraavia ohjelmointirajapintoja, jotta sen käytettävyys olisi eri alustoille mahdollista: Direct3D Windowsia ja Xboxia varten, OpenGL Mac ja myös Windowsille ja viimeisenä on OpenGL ES rajapinta, jota Unity käyttää, kun luodaan projekteja Androidille tai iOS käyttöjärjestelmille. Unity tukee myös muita ohjelmointirajapintoja, kun kyseessä on eri valmistajien videopeli konsoleita, kuten PlayStation 4, Nintedo ja Wii U.

2.1 Kehittämisen vaatimukset

Jotta pelimoottorista saa kaiken irti hyötyineen, on olennaista osata perusasiat C#-ohjelmointikielestä. Useat eri web-sivustot, kuten www.udemy.com tarjoavat laajaa opetusta aloittelijoille, joista pääsee nopeasti kiinni, miten pelejä kehitetään Unityssä.

Unityssä on useita eri käsitteitä kuten Rendering, Physics, Scripts, Prefabs, Camera, Image Effects, Terrain, Lights, Probes ja niin edelleen. Näiden tunteminen ja käyttöönoton osaaminen vaatii vuosien tuntemusta Unitystä, josta syystä ammattitason pelin tekemiseen vaaditaan vuosien kokemusta. Tosin nämä ovat helppoja käsitteitä, näiden opettelu vaatii enemmänkin aikaa kuin matemattista tai ohjelmoinnillista kokemusta.

2.2 Kehittäminen Unity ympäristössä

Unityn käytetyin ohjelmointikieli on C#. Unity käyttää .NET alustaa, joten on mahdollista käyttää samoja kirjastoja ja funktioita peliä kehittäessä, kuin kehittäisi Windows alustalle ohjelmistoa.

Unityllä kehittäminen on helppoa. Unityssä on käsite kohtaus (scene), jolla esimerkiksi pidetään valikko ja itse peli erikseen toisistaan projektissa. Otetaan esimerkiksi 2D tasohyppely peli johon kuuluu 4 eri tasoa ja pelin valikko. Unityssä oletettavasti luodaan viisi eri kohtausta. Yksi kohtaus sisältää pelin valikon ja neljä muuta edustavat eri tasoja pelissä. Yleisesti peliä kehitettäessä kehittäjiä löytyy enemmän kuin yksi. Yksi tapa on sopia työn prosessi siten, että jotkut kehittäjät keskittyvät tietyn kohtauksen tekemiseen ja toiset muihin. Näin vältytään myös vahingossa toisten kehitystyön sekoittamista.

Unity 5.3 pelimoottori toi uuden työkalun parantamaan käyttöliittymien tekemistä, jota voi käyttää pelivalikkojen luomiseen. Tämä sisältää useita eri valmiita UI-elementtejä kuten nappula, teksti ja piirtoalue.

Peleissä usein esiintyy samoja objekteja kuten itse pelin hahmo, tuoli, vihollinen tai ovi, näistä tehdään sellainen Prefab peliobjekti . Prefab on peliobjekti, joka sisältää tietyt informaatiot ja toiminnallisuudet valmiina. Täten jos jokin sama peliobjekti pitää laittaa peliin, esimerkiksi Ovi joka avautuu sisäänpäin, tästä voi tehdä kopion ja se sisältää samat tiedot. Koska kehittäessä useat eri tiedot ja toiminnallisuudet voivat muuttua tietyssä peliobjektista, on turhauttavaa ajatella, että tämä muutos pitää laittaa jokaiseen samanlaiseen gameobjektiin. Kuvitellaan vaikka tämä ovi joka avautuu sisäänpäin halutaankin avautuvan ulospäin. Jos tämän ovi gameobjekti on Prefab, riittää että Unityssä Prefab tietoja muuttaa, ja se välittyy jokaiselle kloonatulle Prefab gameobjektille.

3 Virtuaalitodellisuus

Virtuaalitodellisuudella tarkoitetaan yleiskielellä ympäristö, joka on luotu tietokoneella näyttämään todellisuudelta. Nämä ovat

3.1 Oculus virtuaalilasit

Oculus Rift virtuaalilasit on Oculus VR yrityksen kehittämät virtuaalilasit. Näitä voi käyttää peleissä että eri sovelluksissa. Oculus Riftistä ei ole vielä tuotu markkinoille kuluttuja versiota, mutta ne ovat tulossa 2016 alku vuodesta myyntiin. Oculus Riftistä on kuitenkin kehittäjäversio saatavilla, ja niitä voi tilata Oculuksen web-sivulta. Tätä projektia varten on käytössä kehittäjäversio Oculus Riftistä.

3.2 Hyödyt ja haitat

4 Työn vaatimukset ja tarpeet

Tämän opinnäytetyön vaatimuksiin kuuluu osata perusasioita matematiikasta, Unity pelimoottorista sekä hyvää kokemusta C# ohjelmoinnista, että olio-ohjelmoinnista. Näitä kaikkia taitoja tarvitaan työn suorittamiseen.

4.1 Ohjelmointi

Peli luomisessa käytetään C#-ohjelmointikieltä, jonka osaamisesta vaaditaan ainakin perusteet. Olio-ohjelmoinnista täytyy olla vähän enemmän tietoa, sillä ohjelmoinnin näkökulmasta kaikki taustalogiikat ovat olio-ohjelmointi pohjaisia ratkaisuja. On siis olennaista tietää mitä olio-ohjelmointi on. Ohjelmisto viitekehyksenä käytetään .NET 4.5 versiota, jossa on ohjelmointikieli versiona käytössä 5 C#. Unityn asennuksessa mukana tulee C#-ohjelmointikieltä varten kääntäjä MonoDevelop, tämä on Unityn oletus kääntäjä. Unity mahdollistaa kuitenkin myös pelin kääntämisen C#-skriptit Visual Studio 2013 ja 2015 versioissa. Projekti onkin toteutettu Visual Studio 2015 Community Edition kääntäjän avulla, jonka voi ladata erikseen Microsoftin sivuilta. Jotta lisäosan saa käyttöön on Unityn asetuksissa valittava kääntäjäksi Visual Studio.

4.2 Unity

Tässä projektissa käytetään Unity 5.3 versiota joka on uusin saatavilla oleva versio. Projektia varten pitää tuntea perusasiat, jotta työn tekeminen on mahdollista. Tähän kuuluu muun muassa, miten luodaan projekti, erilaisten kohtauksien (scene) luonti peliä varten ja miten lisätä gameobjekteja ja niihin komponentteja. On myös tärkeää ymmärtää miten hallitaan pelin taustalogiikkaa ohjelmoinnillisesti Unityssä.

4.3 Oculus virtuaalilasit osa Unity projektia

Projektia varten pitää olla kehittäjäversion Oculus virtuaalilasit. Oculus virtuaalilaseista on tehty kaksi eri versiota, Oculus SDK 1 ja SDK 2. Tässä projektissa käytämme Oculus SDK versiota 2. Tässä versiossa Oculus lasit näyttävät parempaa grafiikkaa, sekä kuvan tärähtely on merkittävästi pienempi laseja käytettäessä joka selventää kuvaa huomattavasi verrattuna aiempaan SDK 1 versioon. Oculus web-sivustosta pitää ladata myös Unityä varten erikseen kehittäjäpaketit, sekä ajurit. Käytämme ajuri versiota 0.8.

4.4 Tietokone komponenttien tarpeet

Jotta kehittäminen Oculus virtuaalilaseille on mahdollista, pitää omistaa tehokas tietokone. Oculus virtuaalilasit tarvitsevat muun muassa tehokkaan näytönohjaimen, jotta se jaksaa pyörittää kuvaa Oculus virtuaalilaseissa. Minimi vaatimukset tietokone komponenteille on vastaavat:

Näytönohjain: NVIDIA GeForce 760GTX
Prosessori: Intel Core 2 Quad 9660
Keskusmuisti: 2 Gt

Käyttöjärjestelmänä on käytettävä Windows 8.1 tai Windows 10 versioita. Käytössä voi olla mikä vain painos. (Home, Enterprise, Professional)

4.5 3D-mallit

Peli tukee monia eri tiedostoformaatissa olevia 3D-mallinnuksia. Esimerkiksi *Blenderistä* exporatut .fbx formaatissa olevat 3D-mallit, tai *3D Studio maxista* exportatut .3ds. 3D-mallit.
Käytännössä peli tukee kaikkea 3D-malleja, mitä Unity tukee.

5 Dreamhouse 2016 pelin toteutus

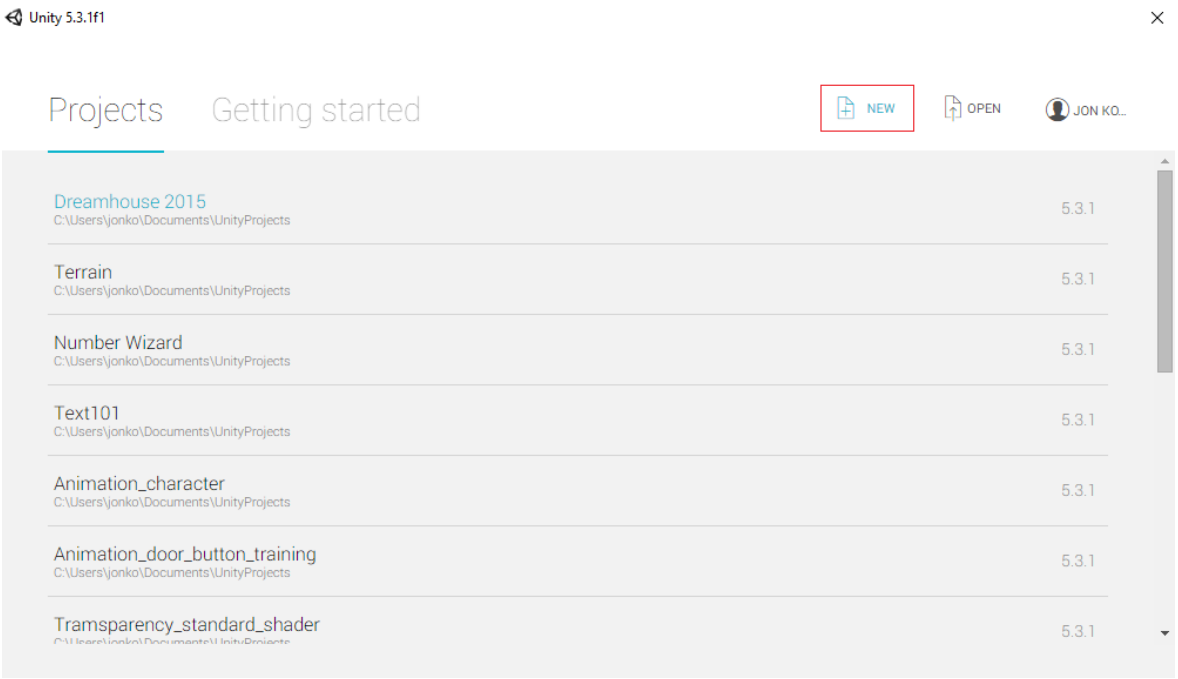
5.1 Asennukset ja ohjelmistot

Tätä työtä varten seuraavat ohjelmistot pitää asentaa työasemalle:

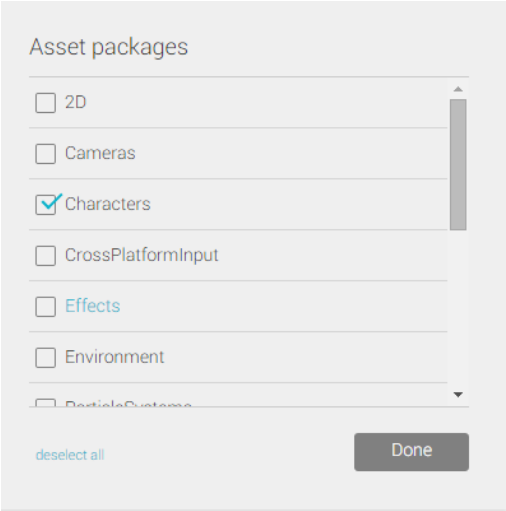
1. Unity versio 5.3<linkki>
2. Visual Studio 2015 Community Edition<linkki>
3. Oculus Runtime 0.8<linkki>

5.2 Projektin aloitus

Jotta kehittäminen voidaan aloittaa, täytyy luoda projekti Unityssä. Unityn käynnistäessä avautuu dialogi, joka pyytää käyttäjätunnuksia kirjautumiseen. Jos käyttäjätunnuksia ei ole, voi kehittämisen tarvittaessa aloittaa offline-tilassa. Tätä varten pitää klikata Work Offline -nappulaa. Uusi dialogi vaihtuu tilalle, jossa pyydetään avaamaan aiemmin luotu Unity projekti tai luomaan uusi. Tätä työtä varten meidän pitää luoda uusi projekti.



NEW-nappulaa klikatessa avautuu dialogi, joka kysyy projektin nimeä ja mitä paketteja otetaan projektiin mukaan. Nämä kyseiset paketit ovat Unityn asennuksen yhteydessä tulleita valmiita kehittämiseen luotuja tiedostoja. Nämä tiedostot ovat valmiita skriptejä, 3D-malleja, tekstuureita ja muita kehittämisen aloittamiseen tarkoitettua sisältöä. Tässä pelissä pelaajan pitää pystyä liikkumaan, ja koska emme luo omaa hahmoa peliin, käytämme valmista sisältöä tätä varten. Valitsemme projektiin valmiin paketin ”Characters”.



Ennen kuin luodaan projekti, on hyvä varmistaa, että 3D on asetettu päälle. Tämä laittaa Unity peliprojektille valmiiksi 3D asetukset.

Projektin aikana luodaan myös katseella aktivoituvia nappuloita. Tätä varten täytyy ladata erään kehittäjän luoma skripti. Tämä ladataan valmiiksi työasemalle ennen projektin aloittamista.

5.3 Projektin arkkitehtuuri ja organisointi

Pelin arkkitehtuuri sisältää kolme eri kohtausta, joista yksi on valikko ja loput pelin ympäristöjä. ”Valikko” kohtausta varten luodaan skripti, joka käsittelee käyttöliittämän tapahtumia, kuten nappulan painallusta. Tämä sama skripti myös vastaa logiikasta, mihin ympäristöön käyttäjä haluaa mennä pelaamaan napin painalluksesta. Ennen projektin aloittamista työasemalle ladattiin valmis skripti, joka aiheuttaa pelissä napin painalluksen tapahtuman pelkällä katseella, tämä valmis skripti otetaan valikko kohtaukseen mukaan käyttöön.

Pelistä löytyy kaksi eri kohtausta, jossa pelaaja pystyy pelaamaan. Molemmissa kohtauksissa on samat skriptit ja logiikat, paitsi ympäristö on erilainen. Toinen näistä ympäristöistä on rakennettu Unityssä luomalla iso tyhjä huone, joka koostuu primitiivisistä objekteista. Nämä primitiiviset objektit vastaavat mitä tahansa 3D-mallinus ohjelmassa luotuja modeleja Toisessa kohtauksessa taas on tämän työn toimeksiantajan yrityksessä toimivan työntekijän tekemä valmis huoneisto, jossa käyttäjä voi pelata.

Pelaamista varten täytyy luoda muutamia skriptejä. Yksi näistä skripteistä käsittelee logiikkaa, miten ladataan pelin sisälle käyttäjän laittamia 3D-mallinnus tiedostoja Resource -

kansioon. Toinen skripti joka luodaan, käsittelee pelaaja logiikkaa. Tämä sisältää toimintoja kuten, miten pelaaja pystyy asettamaan objekteja ja käsittelemään niitä. Objektien käsittelyä varten myös luodaan skripti, joka sisältää logiikan, miten saadaan keskitettyä tietty objekti jonka pelaaja voi halutessaan poistaa.

<kuva>

Projektin sisällöt organisoidaan eri kansioihin, jotta sisältöä on helpompi löytää ja jäsennellä. Unity projektia luodessa, projektiin tulee valmiiksi ”Assets” kansio, joka sisältää kaikki sisällöt, mitä peli käyttää. Tämän kansion sisälle luodaan alikansioita, jotka nimetään sisältöä kuvaavilla nimillä. Skriptejä varten Assets -kansion sisälle luodaan ”Scripts” kansio. 3D-malleja ja tekstuureita varten luodaan ”Resources” kansio, jonka sisälle luodaan kansio ”Models”.

Projektin luonnin yhteydessä valittiin mukaan ”Characters” paketti. Tämän paketin sisältö on automaattisesti ”Standard Assets” kansion sisällä, josta löytyy kaikki sisällöt, mitä Characters paketista löytyy.

Kohtauksia varten luodaan myös erikseen kansio Scenes, johon lisätään projektin aikana kaikki kohtaukset.

5.4 Valikko

5.4.1 Käyttöliittymä ja skriptit

Pelin valikkoa varten luodaan kohtaus nimeltään ”Gamemenu”, jota pidetään Scenes kansion sisällä. Tämä kyseinen kohtaus avataan auki Unityssä, mihin luodaan pelin valikko. Pelistä löytyy kaksi valikkoa, joista jompikumpi on aktiivinen eli esillä. Ensimmäisessä valikossa voidaan siirtyä valitsemaan peliympäristöä pelaamista varten tai poistua pelistä, kun taas toisesta valikosta valitaan kahdesta vaihtoehdosta, mitä peliympäristöä halutaan pelaamiseen.

Pelin ensimmäistä valikkoa varten lisätään kaksi nappulaa, joista yhdestä pääsee valitsemaan peliympäristön ennen pelaamista ja toisesta pääsee poistumaan pelistä. Nämä lisätään Unityssä GameObject → UI →Button nappulaa valitsemalla. Tämä toistetaan kaksi kertaan, että saadaan kaksi nappulaa. Nämä nappulat näkyvät Unityn hierarkia ikkunasta Canvas gameobjektin sisältä. Nämä asetellaan Unityssä keskelle canvasta, jotta valikko näyttää siistiltä. Tämä sama prosessi on tehty toiselle valikolle, joka löytyy toisen canvas objektin sisältä.

Tämä toinen valikko on pelin ympäristön valintaa varten. Tämä valikko laitetaan inaktiiviseksi valmiiksi, koska haluamme vain yhden valikon olevan esillä kerrallaan. Pelin ympäristön valikon nappuloiden yläpuolelta löytyy kuva, joka antaa käyttäjälle esikatselun pelinympäristöstä.

<kuva valikosta>

Valikkojen toiminallisuutta varten pitää luoda Event System, joka käsittelee Eventtejä, eli tapahtumia valikossa. Event Systemin voi lisätä valikosta GameObject → UI→ Event System. Tämä riittää, että saadaan tapahtumat toimimaan tässä kohtauksessa eli valikossa. Event System objektiin pitää vielä lisätä uusi komponentti, joka on skripti, joka käsittelee napin painalluksia katseella. Tämän skripti nimi on ”VRInputModule”, joka ladattiin työasemalle projektin aloitus vaiheessa.

Valikkoa varten myös tarvitaan toinen skripti, joka käsittelee napin painallusten aiheuttamat toiminnot, kuten pelistä poistuminen, peliympäristön valitseminen, valikon näyttäminen ja pelin aloittaminen. Projektiin on luotu skripti ”MenuLogic”, johon on kirjoitettu valikon näyttämisen ja kohtausten vaihtamiseen liittyvät toiminnallisuudet.

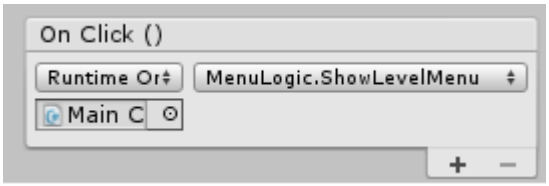
5.4.2 Valikon toiminnallisuudet

MenuLogic skriptiin on lisätty viittaus kohtauksessa oleviin kahteen eri valikkoon. Jotta napin painalluksista voi siirtyä toisiin valikkoihin ja takaisin, täytyy skriptillä löytyä viittaus näihin. Skripti sitten asettaa nämä valikot eli canvas objektit aktiiviseksi sitä mukaan, mitä valikkoa halutaan näyttää. Viittaukset asetetaan MenuLogic luokan sisällä julkisiksi objekti instansseiksi:

```
public class MenuLogic : MonoBehaviour {

    public Canvas MainMenu;
    public Canvas LevelMenu;
```

Tämä skripti on pää kameran komponentti. Pelaaja pitää pystyä katseellaan painaa nappuloita, tästä syystä tämä skripti on kytkettävä pelin kameraan. Event Systeemille annetaan viittaus napin painallusten jälkeisiin suoritettaviin funktioihin. Tämä viittaus annetaan kameran kautta.

Jokaiselle valikon nappulalle annetaan Eventti ”On Click()”, jotta voidaan suorittaa toiminto, kun käyttäjä on painanut nappulaa katseellaan. Tapahtumaan On Click täytyy vielä antaa viittaus Main Cameraan, jotta päästään käsiksi MenuLogic skriptin funktioihin. Alasveto valikosta voi valita MenuLogic skriptin ja sen funktion, mikä halutaan suorittaa napin painalluksesta.



Kuvasta näkee, miten On Click eventti on toteutettu PlayButton -nappulalle. PlayButton nappula on päävalikon ”Pelaa” nappula. Nappulan aktivoituessa näytetään toinen valikko, josta valitaan peliympäristö.

```
public void ShowLevelMenu()
{
    MainMenu.gameObject.SetActive(false);
    LevelMenu.gameObject.SetActive(true);
}
```

Kuvassa näkyy ShowLevelMenu funktion toiminallisuus. Skripti omistaa viittauksen pää valikkoon ja peliympäristön valinnan valikkoon. Käyttäjän aktivoidessaan Pelaa-nappulaa, asetetaan päävalikko inaktiiviseksi ja peliympäristö valinta valikko aktiiviseksi.

Kun käyttäjä on valinnut peliympäristön pelaamista varten, suoritetaan funktio, joka lataa kohtauksen pelaajalle pelaamista varten.

```
public void ChangeLevelToHouse()
{
    ChangeScene("House");
}

public void ChangeLevelToEmptyHouse()
{
    ChangeScene("EmptyHouse");
}

private void ChangeScene(string name)
{
    SceneManager.LoadScene(name);
}
```

MenuLogic skriptissä on vain yksi metodi joka vaihtaa kohtauksia pelissä. Tämä metodi saa parametrina halutun kohtauksen nimen, joka suorittaa funktion LoadScene, mikä avaa tietyn kohtauksen auki pelissä.

5.5 Pelin ympäristöjen toteutus

Tätä projektia varten on toteutettu kaksi erilaista ympäristöä, jossa pelaaja pystyy pelaamaan. Valikossa pelaaja pystyy valitsemaan, minkä ympäristön haluaa ottaa pelattavaksi.

Yksi näistä tasoista eli ympäristöistä on tyhjä iso huone. Ison tyhjän huoneen ajatuksena on toteuttaa pelaajalle mahdollisuus testata ja kokeilla vapaasti erilaisia objekteja eli huonekaluja, ja nähdä miltä ne näyttävät pelissä. Tämä mahdollisuus on suunniteltu sitä varten, että jos käyttäjä on luonut omia objekteja, on mahdollista testaa ja nähdä niitä isossa ympäristössä. On yleistä, että ei niin kokeneemmat 3D-mallinuksesta kiinnostuneet käyttäjät voivat vahingossa luoda liian suuria objekteja, jotka eivät sovi yhteen muiden kanssa.

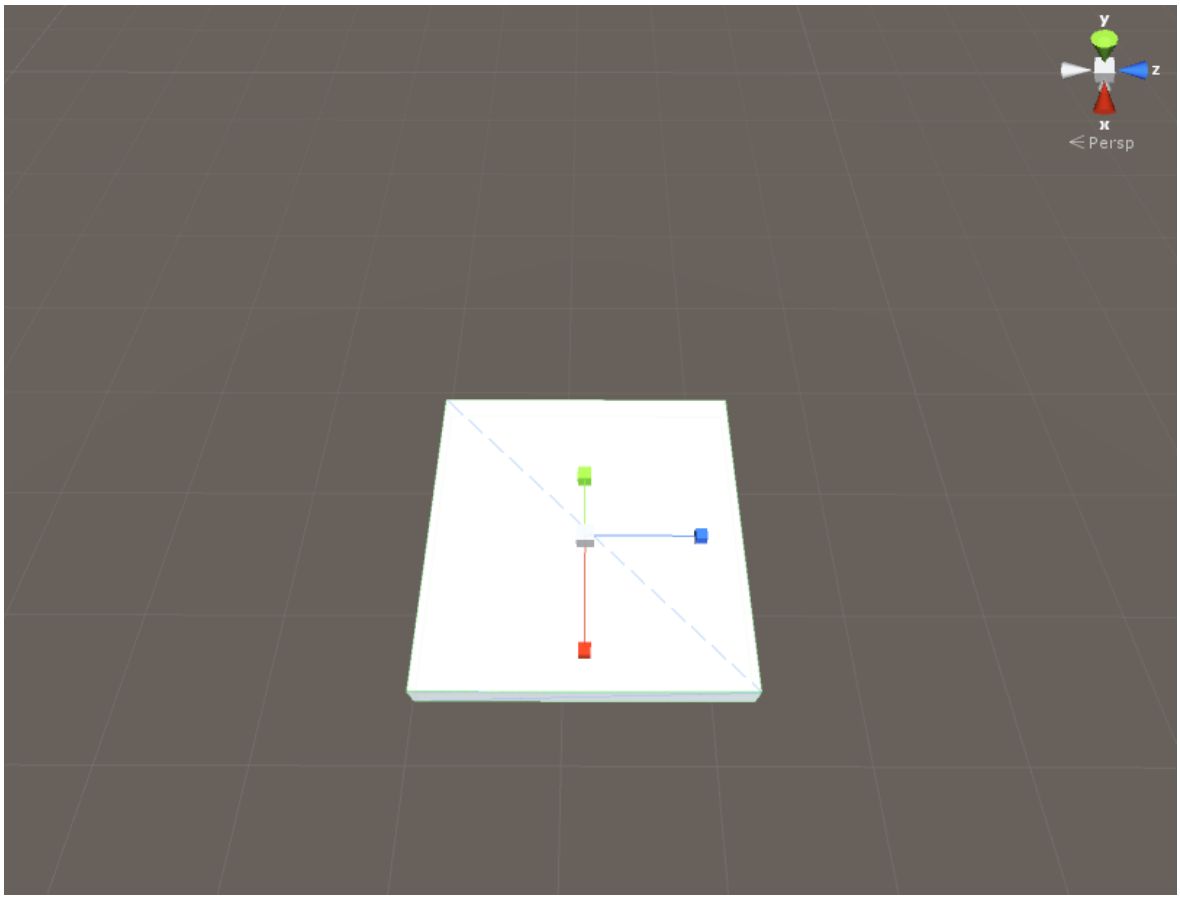
Kyseisen ison tyhjän huoneen idea ei ole myöskään pelkästään vain testata objekteja, vaan antaa käyttäjälle mahdollisuus niin sanotusti leikkiä eri huonekaluilla. Tämä ajatus toimii lapsille, jotka haluavat koristella ja asetella huonekaluja oman mielensä mukaan, minne haluaa rajoittamattomassa ympäristössä.

Toinen pelin ympäristöistä on iso kerrostaloasunto, josta yhteen huoneeseen, pelaaja on asetettu. Tämä on suljettu ympäristö, missä käyttäjä voi asetella huoneeseen huonekaluja. Tämä ympäristö eroaa edellisestä siten, että käyttäjän on mahdollista vaihtaa paikkaa eri huoneisiin, näin käyttäjä pystyy asettaa huonekaluja erilaisissa huoneisissa. Tämän ympäristön pitäisi antaa hyvän mahdollisuuden käyttäjän kokeilla objekteja erilaisissa huoneissa. Jos käyttäjällä on hallussa esimerkiksi useita valmiita 3D-malleja, on mahdollista sisustaa jokainen huone.

5.5.1 Tyhjä huone

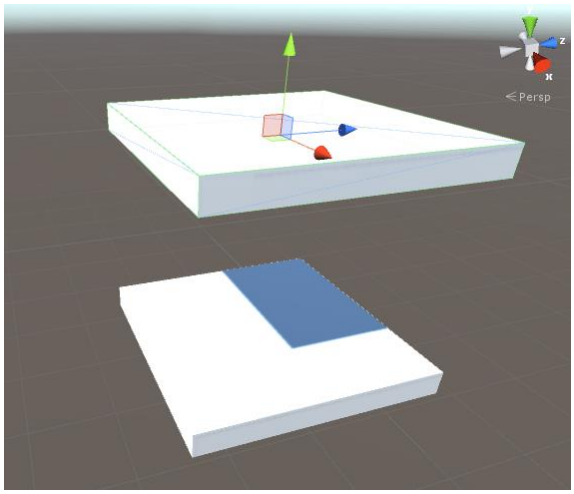
Tätä ympäristöä varten on luotu oma kohtaus, johon on rakennettu Unityn työkaluilla tyhjä huone. Tämä on tehty neliömäisillä 3D-objekteilla, joita on venytetty ja asetettu muistuttamaan tyhjää huonetta.

Luodaan siis yksi nelilömäinen 3D-primitiivinen objekti, eli Unityn oma valmis objekti, jonka on venytetty leveys suunnassa leveäksi, mutta matalaksi.



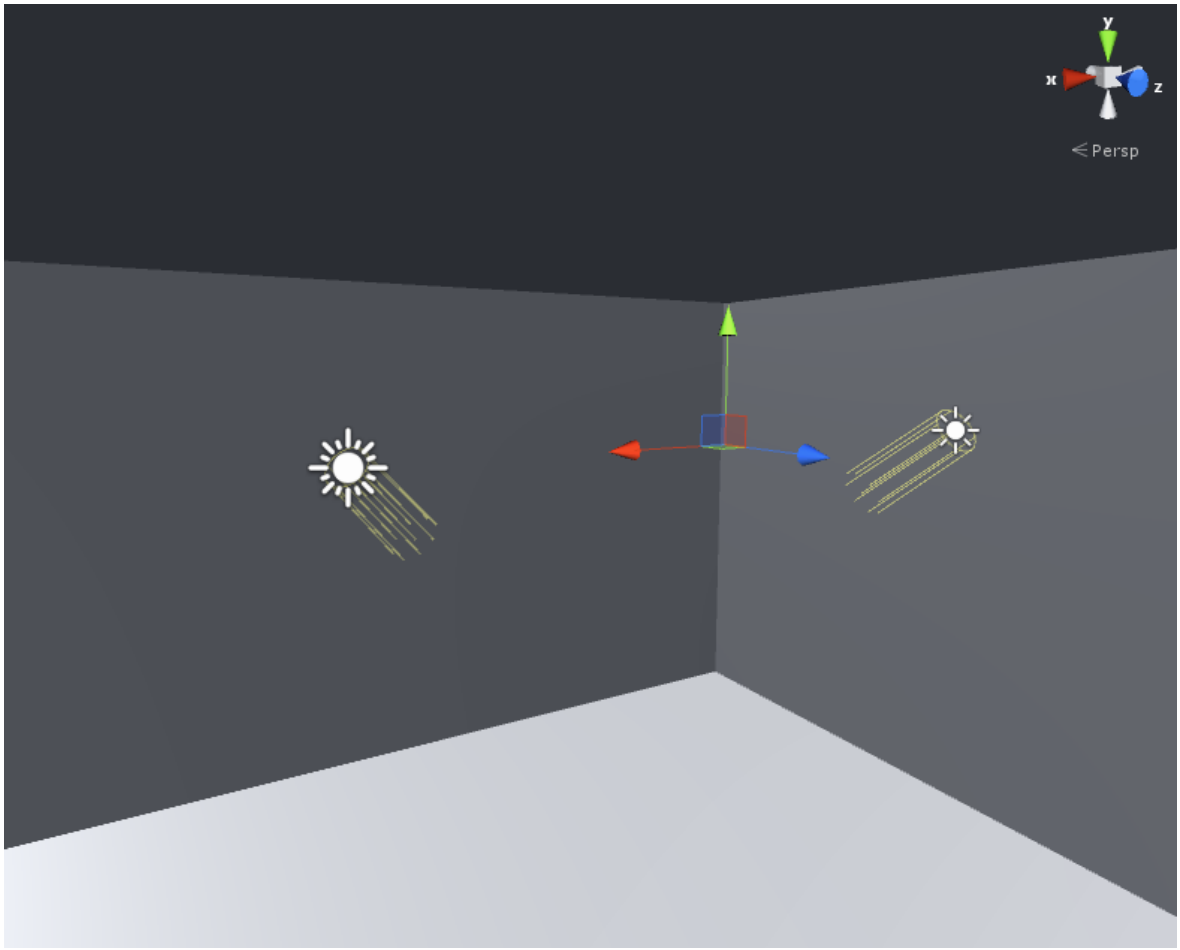
Unityssä saa valmiita 3D-primitiivisiä malleja, jotka ovat Unityn valmiita 3D-malleja. Tyhjän huoneen lattiaa varten valittiin neliö joka venytettiin muistuttamaan lattiaa. Tämän voi valita valikosta GameObject -> 3D Object -> Cube

Kopioidaan lattia ja siirretään se Y-akselia pitkin korkeammalle lattian yläpuolelle, jotta siitä saadaan helposti saman kokoinen katto huoneelle.

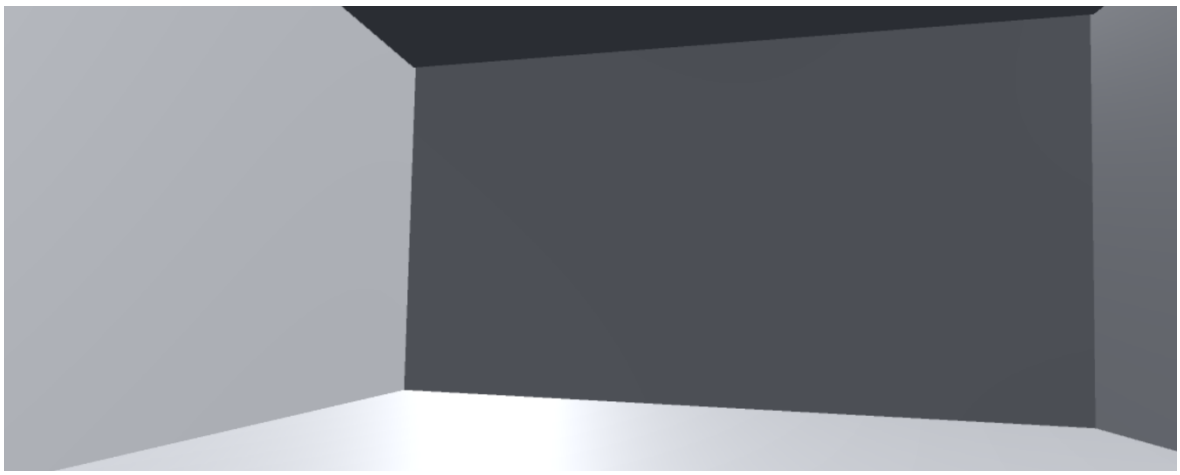


Seinät toteutetaan samalla tavalla, mutta leveys suunnassa ne ovat ohuempia, ja korkeus venytetään mahtumaan lattian ja katon väliin. Näin saadaan helposti toteutettua tyhjä

huone pelaajalle. Tyhjästä huoneesta enää puuttuu valot huoneen sisällä, ilman valoja ei pelaaja pystyy näkemään eteenpäin. Unity asettaa kohtauksen luonnin yhteydessä yhden valo objektin taivaalle, valo ei pääse kuitenkaan tyhjän huoneen sisälle. Asetetaan kaksi valo objektia huoneen sisälle, jotta tämä ongelma saataisiin ratkottua.



Kuvasta näkee, että kaksi valo objektia on asetettu tyhjään huoneen sisälle, jotta saadaan valaistus sopivaksi. Valo objektiksi valittiin Directional Light objekti, joka lähettää tiettyyn suuntaan valoa.



Tämä kuva näyttää, minkälainen tyhjä huone ja sen valoisuus näyttävät pelissä.

Tyhjän huoneen valmiiksi saattamiseksi pitää enää asettaa Unityn oma valmis yksinkertainen pelaaja objekti, joka toimii pelaajana tässä pelissä. Projektin luonnin yhteydessä otettiin Characters paketti mukaan projektiin, jonka mukana löytyy tällainen valmis Pelaaja objekti. Tämän tiedoston nimi on FPSController, joka löytyy Characters paketin sisällöstä olevista tiedostoista. Tämä objekti siirretään tyhjään huoneeseen lattian yläpuolelle, josta pelaaja aloittaa aina tässä ympäristössä pelattaessa.

5.5.2 Kerrostalo ympäristö

Kerrostaloa ympäristö on toteutukseltaan muutoin sama kuin tyhjän huoneen, mutta eroaa siten, että en luonut useita tyhjiä huoneita sitä varten. Toimeksiantajan Ville Rantasen antoi projektia varten valmiin ison 3D-mallin, joka vastaa isoa kerrostaloa.

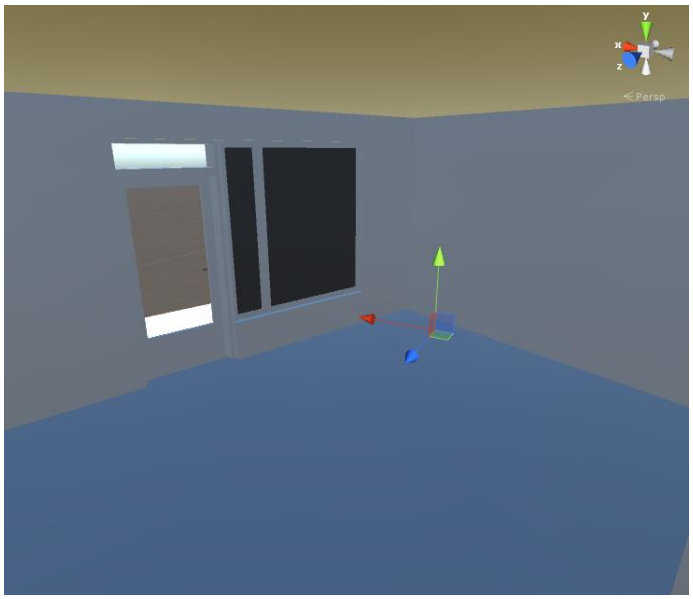
Kerrostalon ympäristöä varten tein uuden Scenen eli kohtauksen Unityssä. Tämän jälkeen otin projektiin mukaan uuden 3D-mallinnus tiedoston, joka on tämä kerrostalo. Asetin tämän keskelle maailmaa, jonka jälkeen lähdin toteuttamaan pelaajan paikat ympäristölle.

Ville Rantasen toteuttama 3D-malli tyhjä kerrostalo tulee toimimaan pelaajan ympäristönä. Pelaaja asetetaan yhteen näistä huoneista aloitettuaan pelaamisen tässä tasossa.

Valitaan yksi huoneisto, kymmenistä tyhjistä kerrostalon huoneista ja asetetaan FPSController pelaaja objekti huoneen sisälle. Kohdan on oltava sellainen josta pelaaja voi aloittaa heti tutkimisen ja huonekalujen asettelun.

Kuvan vasemmalla puolella näkyy Unityn kohtaus ikkuna, missä on pelaaja kamera ikoni. Oikealla puolella on kuva itse pelistä, mihin suuntaan pelaaja katsoo aloittaessaan tässä ympäristössä pelaamisen.

Tässä peliympäristössä on useita muitakin huoneita, mutta käyttäjä aloittaa vain yhdestä näistä. Pelaajalla on kuitenkin mahdollisuus käydä muissakin. Tätä varten on luotu tyhjä peliobjekti, ”teleportti”, joka on asetettu muihin huoneisiin. Teleportti objekteilla on xyz-koordinaatit. Pelaajan hahmon toiminnallisuuksiin kuuluu teleportaatio, jota käytetään nimenomaan tässä ympäristössä. Teleportaatio tapahtuu käytännössä siten, että pelaaja objektin xyz-koordinaatteihin vaihdetaan jonkin teleportti objektin koordinaatit kun pelaaja käyttää teleportaatio toiminnallisuutta. Tällöin pelaaja voi vapaasti liikkua eri huoneissa helposti.



Kuvassa on yksi kerrostalon huoneista, johon on asetettu tyhjä peliobjekti. Kun pelaaja haluaa, hän voi siirtyä tämän peliobjektin sijaintiin, milloin tahansa.

5.6 Pelaajan toiminnollisuuksien toteuttaminen

Jokaiseen ympäristöön, mitä on toteutettu pelaamista varten, on asetettu FPSController pelaajaobjekti.

5.6.1 FPSController

Tätä työtä varten tarvitsin objektin, joka on ohjelmoitu liikkumaan näppäimistä ja on kykenemään liikkumaan, hyppimään ja muita perusominaisuuksia, mitä yleisesti peleistä pelaaja hahmolta odotetaan. Aikaisemmin mainitun FPSController pelaajaobjekti, joka tulee Characters paketin mukana, vastaa minun tarpeitani. Tähän objektiin on liitetty First Person Controlller skripti, joka mahdollistaa objektin liikkumista näppäinten avulla. Myös skriptistä löytyy koodia, joka tuottaa ääniä pelaajan hyppiessä tai juostessaan. Tätä työtä varten tarvitsin pelaaja hahmon mahdollistaa käyttää Oculus virtuaalilaseja katsoessaan ympärille, että 3D-objektien piirtäminen siihen paikkaan, mihin käyttäjä katsoo. Tätä varten minun piti joko laajentaa valmista First Person Controller skriptiä, tai luon uuden ja kytken sen tähän FPSController objektiin. Päätin toteuttaa oman skriptin ja kytkeä se tähän objektiin, koska tulevat toiminnallisuudet eivät koske perus periaatteita liittyen liikkumiseen tai muuhun, on helpompi ja järkevämpää pitää nämä koodit erillään omissa tiedostoissaan.

5.6.2 Unity skriptit

Unityssä käsite Script eli Skripti on tiedosto, joka sisältää pätkän koodia. Unityssä pelien tekemisessä yleisesti luodaan useita skriptejä, jotka kytketään eri objekteihin. Nämä koodipätkät voivat olla lyhyitäkin saavuttaakseen halutun toiminnallisuuden. Eri toiminnallisuudet on yleensä jaoteltu eri skripteihin, tällä tavoin niiden ylläpitäminen ja laajentaminen on helpompaa.

Unityn skriptit sisältävät jo luodessa valmiiksi kaksi funktiota, Start ja Update. Kun peli käynnistyy, jokaisen skriptin Start funktio suoritetaan. Tästä syystä Start funktioon on hyvä asettaa kaikki alustukseen varten tarvittavat koodi pätkät kuten animaatioiden, arvojen asettaminen hahmolle, tiedon luku tai äänitiedostojen soittaminen. Esimerkiksi vaikka peliobjekti ”BackgroundMusic” johon on kytketty skripti, jonka Start funktion suoritettaessa se alkaa soittaa musiikkia.

Toinen merkittävä funktio skripteissä on Update. Tämä funktio suoritetaan koko ajan pelin ollessa käynnissä. Tässä funktiossa tyypillisesti tarkistetaan pelaajan käyttämiä näppäin painalluksia, tai onko pelaajaan osumassa jokin toinen objekti. Esimerkiksi Player Controller Scriptissä Update funktiossa tarkistetaan, kun pelaaja painaa W näppäintä, objektia liikutetaan siihen suuntaan, mihin objektiin kytketty kamera katsoo. Tai kun objekti törmää tai osuu toiseen objektiin, sen liikkuminen objektin sisään estetään.

5.6.3 PlayerScript

Luodaan uusi skripti nimeltään PlayerScript. Tähän skriptiin kirjoitetaan kaikki koodit koskien pelaajahahmon toiminnallisuuksia kuten 3D-mallien piirtämiseen katseen pääty pisteeseen, 3D-mallien asettaminen ja niiden poistaminen, sekä muut tarvittavat toiminnallisuudet. Tämä skripti lisätään osaksi FPSController objektia, jotta se tulee käyttöön pelatessa.

5.6.4 3D-mallien piirtäminen katseen mukaisesti

Unity tarjoaa kirjastoja, mistä löytyy jättimäisen paljon valmista funktiota apuna peliohjelmointia varten. Näiden funktioiden avulla käyttäjän on helppo lähteä rakentamaan omaa peliä. Unity myös tarjoaa manuaalin jokaista funktiota varten, jos tuntee epävarmuutta, miten sitä käytetään tai mikä sen tarkoitus on.

Seuraavaksi toteutetaan toiminnallisuus pelaajalle, jossa piirrettävä 3D objekti sijoitetaan pelaajan eteen siihen kohtaan mihin hän katsoo. Yksi Unityn valmiista funktioista tarjoaa ratkaisun tämän toiminnallisuuden toteuttamiseen. Funktion nimi on ”Raycast”, joka on osa Physics luokkaa. Funktio käytännössä ampuu näkymättömän säteen tiettyyn suuntaan ja palauttaa tiedon, kun se osuu johonkin. Tällä funktiolla pystytään siis pelissä saamaan tieto käyttäjän katsomasta kohteesta, olkoon se edessä oleva lattia, katto taikka seinä. Myös jo asetettujen objektien päälle on mahdollista piirtää uusi objekti. Funktion palauttaa tiedon objektista, esimerkiksi lattia, sekä osumapisteen paikan xyz-koordinaatistosta. Funktion palautettua tiedon pystytään siihen piirtämään 3D-objekti.

Jotta pelaajaobjekti ampuisi näitä näkymättömiä säteitä jatkuvasti pelaajan katseen suuntaan, on Update funktion sisällä kutsuttava Physics.Raycast funktiota. Tällöin aina kun funktio palauttaa tiedon osumapisteestä, siihen pisteeseen piirretään 3D-malli.

```
//Säteen ampuminen katseen suuntaan
if (Physics.Raycast(direction.position, direction.forward, out Hit))
{   //Asetetaan objektin position osumapisteen xyz koordinaattiin
    targetObject.transform.position = new Vector3(Hit.point.x, Hit.point.y, Hit.point.z);
}
```

Koska projektin aikana tullaan laajentamaan logiikkaa koskien 3D-mallien piirtämistä, asettelua ja poistoa, on järkevää kirjoittaa kaikki nämä logiikat yhden funktion sisälle. Tätä varten on tehty funktio ”ShootRayForObject”, johon kirjoitetaan kaikki aiemmin mainitut toiminnallisuudet. Tämä funktio käsittelee tällöin 3D-mallin piirtämislogiikkaa. Pitämällä funktioita loogisessa järjestyksessä helpotetaan kehittämistä.

PlayerScript Update funktion on laajennettu myös siten, että jos käyttäjä haluaa lopettaa 3D-mallin piirtämisen, käyttäjä voi painaa E-näppäintä. Saman napin painalluksesta piirtäminen alkaa uudestaan.

5.6.5 3D-mallin lähestymisen korjaaminen

Toteuttaessa perus toiminnallisuuksia koskien 3D-mallien piirtämistä törmäsin tämän työn aikana useisiin ongelmiin. Näiden ongelmien ratkaisujen löytäminen tuotti myös ongelmia, koska en löytänyt yhtään artikkelia, tai apua foorumeilta missä muut ohjelmoijat kysyvät neuvoja ja apuja. Muun muassa tästä syystä projektin aikataulu pidentyi huomattavasti.

Ensimmäinen ongelmani oli se, kun käyttäjä piirtää 3D-mallia johonkin paikkaan ja pitää päätä paikallaan, objekti alkoi lähestyä pelaajaa päin. Parin viikon jälkeen, kun palasin työn pariin, keksin ratkaisun ongelmaan. Ratkaisu oli lopulta todella yksinkertainen ja sain tuntea pitkästä aikaan tyhmyyden tunnettakin. Pelaajan pitäessä päätä paikallaan osumapiste oli aina melkein sama, paitsi se oli yhden yksikön verran pelaajaa päin.
Toisin sanoen tämä seuraava osumapiste olisi viime kerralla osutun pisteen viereinen piste pelaaja suunnassa. Tämä johti siihen, että 3D-malli piirrettiin seuraavaksi yhden yksikön verran lähemmäksi. Vähän ajan kuluessa 3D-malli pystyi saavuttamaan pelaajan, jos päätä oli pitänyt paikallaan koko ajan.

Ratkaisu tähän oli yksinkertainen. Säteen osuessa piti tarkistaa, oliko osuma piste eri kuin nykyisen 3D-objektin piste.

```
if (targetObject.transform.position != Hit.point)
```

Tarkistetaan if lauseella, onko osuma eri kuin piirretyn 3D-mallin positio. Tällä tavalla pystyy estämään 3D-mallin siirtymistä pelaajan suuntaan kun pää on paikallaan.

5.6.6 3D-objektin piirron esto toisen objektin sisään

Heti ratkaistuani edellisen ongelman, kohtasin uuden. Kaikki objektit, jotka pelaaja piirtää menevät toisten objektien sisälle. Esimerkiksi jos pelaaja katsoo huoneen seinää, pelaajan piirtämä objekti menee puoliksi seinän sisälle.

Tämä osoittautui todella haasteelliseksi ongelmaksi, koska ratkaisua ei löytynyt mistään Unityn omista harjoitusvideoista eikä foorumeistakaan löytynyt valmista ratkaisua. Lähdin itse Unityn virallisille foorumeille kyselemään, jos joku osaisi auttaa, mutta turhaan. Koitin kuukauden ajan kokeilla eri ratkaisuja. Koitin muun muassa selvittää, johtuuko ongelma siitä, että pelin seinästä puuttuu komponentteja joka estäisi objektia menemästä sisään, taikka siitä onko puutteellisia komponentteja 3D-mallissa, jota piirretään.

Lähestyin ensimmäisenä ongelmaan siten, että lisäsin jokaisen ympäristön objektiin komponentin Mesh Collider. Mesh Collider on sellainen komponentti, joka kykenee asettamaan monimutkaisille objekteille fyysiset rajat. Nämä rajat estävät minkä muun tahansa objektin tulemasta läpi, jos kyseisellä objektilla on tietynlaisia komponentteja. Yksi näistä komponenteista mitä objektilla pitää olla, jotta se ei mene toisen objektin sisään, oli Rigidbody ja jokin Collider komponentti.

Rigidbody komponentti edustaa kappaletta, jolla on massa ja siihen voi vaikuttaa painovoima. Tätä komponenttia käytetään jokaisessa pelin objekteissa, joiden halutaan olevan osa fyysistä maailmaa.

Collider komponentteja on paljon, kuten Box, Sphere, Circle ja Mesh Collider. Nämä komponentit toimivat siten, että ne estävät objekteja menemästä toistensa läpi.

Asetin Rigidbody ja MeshCollider komponentin piirretylle 3D-mallille Start funktiossa testatakseni, estääkö tämän objektin menemästä seinän läpi.

```
gameObject.AddComponent<MeshCollider>();
gameObject.AddComponent<Rigidbody>();
```

Testattaessa huomasin, että tämä toimi teoriassa, mutta ei vastannut haluttua lopputulosta. Tuoli pysyi kyllä paikallaan mutta pyöri itsensä ympäri salaman nopeasti. Tämän aiheutti pelaajan katseena toimiva Raycast funktio. Funktio ampuu jatkuvasti säteitä yhteen kohtaan, ja seinä työntää objektin aina itsestään poispäin, jotta se ei menisi seinän sisään. Tämä jatkuva toisto aiheutti objektin hallitsemattoman pyörimisen. Totesin, että tämä ratkaisu ei tule toimimaan lainkaan, ja jouduin etsimään toimivampaa ratkaisua.

Palasin takaisin alkupisteeseen, jossa objekti piirtyy seinän sisään. Huomasin, että objekti oli piirretty tasan puolet seinän sisälle. Päädyin johtopäätökseen, että tämän 3D-mallin keskipiste on keskellä mallia. Pohdin, että jos voisin saada selville seinän ja piirretyn 3D-mallin koon, voisin laskea kuinka paljon kappale menee seinän sisälle, ja piirtää se lähemmäksi pelaajaa.

Unityllä on jokaiselle 3D-objektille luokka Bounds. Bounds luokalla on muuttuja size, joka on Vector3 tyyppiä. Tämä kertoo objektin koon. Tarkastelin objektien size arvoja pelatessani ja ne näyttivät nollaa. Otin vähän selvää mistä tämä johtui, ja selvisi, että objekti koostuu todellisuudessa useammasta objektista, esimerkiksi tuoli koostuu selkänojasta, jaloista ja istuimesta. Unity ei laskenut size arvoa koko objektille vaan pelkästään lapsi objekteille. Tämä ei siis ollut ratkaisu ongelmaan.

Jatkoin selvittämistä ja huomasin, että Bounds luokalla on kaksi ominaisuutta, max ja min. Nämä ovat tyyppiä myös Vector3, eli sisältävät xyz pisteet. Pohdin, että jos laskisin max ja min xyz-pisteiden erot, voisin saada selville objektin piirtorajat. Myös jokaisen objektin Bounds luokasta sai extend ja center ominaisuuksista tietoja, mikä on kappaleen keskipiste ja kuinka paljon se venyy z ja x akselissa. Näiden avulla pystyin selvittämään objektin piirtorajat. Jos siis vertailen objektien piirtorajoja ja minulla on myös tiedossa piirretyn 3D-objektin keskipisteen etäisyys reunaan, pystyn niitä vertailemalla laskea etäisyyden ja päätellä paljon x,y, tai z pisteesiin pitää lisätä arvoa, jotta objekti ei olisi enää toisen objektin sisällä.

Tarkasteltaessa arvoja, huomasin, että piirretyn 3D-mallin max ja min pisteet olivat arvoa nolla. Pohdin, että tämä saattaa mennä todella hankalaksi ja ihmettelin kovasti mistä tämä voi johtua. Otin asiasta selvää Unityn foorumeilta, ja löysin vastauksen. Piirretyn 3D-mallin, tässä tapauksessa tuoli objekti, koostui kahdesta lapsiobjektista. Minun pitää laskea näiden lapsiobjektien koot, saadakseni koko tuolin koon. Yksi näistä lapsiobjekteista edusti tuolin reunoja, kun toinen taas tuolin keskiosaa eli istuin aluetta.

```
Bounds GetObjectBounds(GameObject gameObject)
{
    var renderer = gameObject.GetComponent<Renderer>();
    if (renderer != null)
    {
        Bounds targetBounds = renderer.bounds;
        foreach (Transform child in targetObject.transform)
        {
            var childRenderer = child.gameObject.GetComponent<Renderer>();
            if (childRenderer != null)
                targetBounds.Encapsulate(childRenderer.bounds);
        }

        return targetBounds;
    }
    return new Bounds();

}
```

Loin funktion GetObjectBounds joka palauttaa tuolin koon tyyppinä Bounds. Paramterina otin piirretyn 3D-mallin peliobjektin ja hain Renderer komponentin. Jos sellainen löytyy, otan sen komponentin bounds tiedot. Seuraavaksi käyn läpi kaikki lapsiobjektit ja lisään niiden boundsin rajat Renderer komponentin bounds arvoihin. Palautan funktiosta rendererin boundsit tulevia laskuja varten.

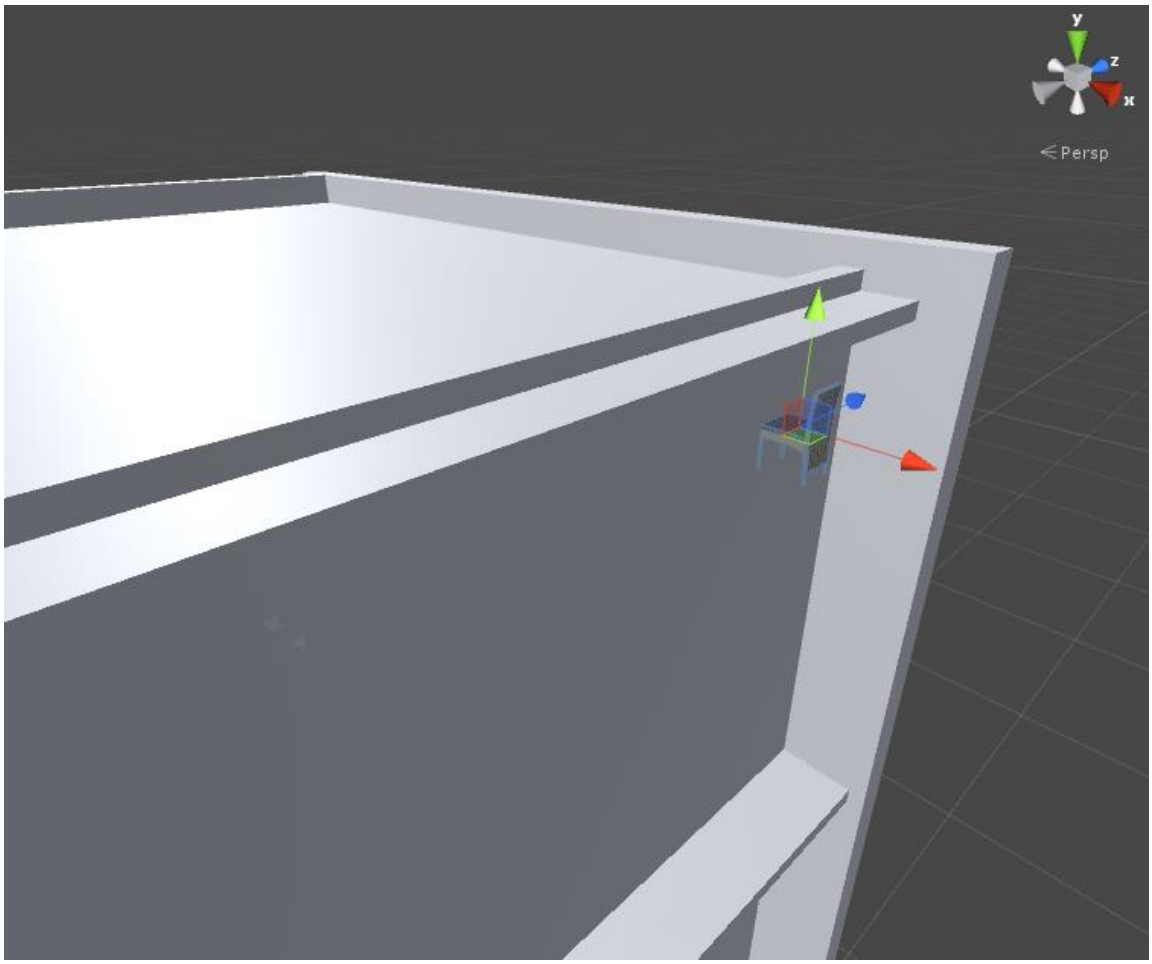
Nyt kun minulla oli hallussani piirretyn objektin raja-arvot, pystyin seuraavaksi lähteä laskemaan tämän ja toisen objektien välisiä etäisyyksiä toisistaan, jotta voisin estää piirretyn objektin piirtymistä toisen sisään.

```
var valueX = targetBounds.center.x + targetBounds.extents.x;
if (valueX > 0)
{
    xPoint = valueX - wallBounds.min.x;
}
else
{
    valueX = targetBounds.center.x - targetBounds.extents.x;
    xPoint = (valueX - wallBounds.max.x);
}

var valueY = targetBounds.center.y + targetBounds.extents.y;
if (valueY > 0)
{
    yPoint = valueY - wallBounds.min.y;
}
else
{
    valueY = targetBounds.center.y - targetBounds.extents.y;
    yPoint = (valueY - wallBounds.max.y);
}

var valueZ = targetBounds.center.z + targetBounds.extents.z;
if (valueZ > 0)
{
    zPoint = valueZ - wallBounds.min.z;
}
else
{
    valueZ = targetBounds.center.z - targetBounds.extents.z;
    zPoint = (valueZ - wallBounds.max.z);
}
```

Kuvassa lasketaan jokaisen xyz pisteen etäisyys keskipisteestä reunaan. Sitten verrataan, onko piste vastakkaisen objektin reuna pisteiden sisällä.

5.6.7 3D-Mallit, piirtäminen pelaajan suuntaan

Näiden edellisten ongelmien ratkettua sain ja melkein toimivan ratkaisun työhön. Olin testailussa kuitenkin löytänyt seuraavan ongelma koskien 3D-mallien piirtämistä. Jos katsoin esimerkiksi kattoon, 3D-malli ei piirtynyt suoraan eteen, vaan meni katon vasempaan reunaan huoneen ulkopuolelle.



Tuoli oli tyhjän huoneen ulkopuolella edellisen ongelman ratkonnan jälkeen. 3D-malliin lisättiin laskujen jälkeen x,y ja z pisteisiin arvoja, jotta objekti ei olisi toisen objektin sisältä.

Syyksi osoittautui se, että kun lasketaan x,y ja z suuntaiset erot piirretyn kappaleen ja vastakkaisen kappaleen välillä, nämä kaikki arvot lisättiin tuolin xyz-pisteeseen. Näin ei pitäisi olla, vaan pitäisi lisätä sen koordinaatio pisteet tuoliin, josta on lyhin matka ulos toisen objektin sisältä. Eli lopuksi pitäisi verrata, millä on pienin arvo, ja se lisätään tuolin koordinaatiopisteeseen.

```
// Millä akselilla on lyhyempi matka
if (positiveZ < positiveY && positiveZ < positiveX)
{
    yPoint = 0;
    xPoint = 0;
}
else if (positiveY < positiveX && positiveY < positiveZ)
{
    xPoint = 0;
    zPoint = 0;
}
else
{
    zPoint = 0;
    yPoint = 0;
}
```

Tässä koodi pätkässä verrataan, mitkä koordinaatiopisteet nollataan, eli lisäys ei aiheuta muutosta piirretyn 3D-mallin paikkaan. Halutaan vain yksi koordinaatiopisteiden lisäys tuolin piirtopisteeseen.

Tämä ei kuitenkaan vielä riittänyt, koska tietyissä tapauksissa kävi niin, että tuoli piirtyi toiselle puolelle objektia, eikä pelaajan suuntaan. Vielä piti laajentaa koodia siten, että otettiin selville mistä suunnasta pelaaja katsoo, eli onko katse kohdistettu akselin positiiviseen vai negatiiviseen suuntaan. Kun nämä tiedot otin vielä huomioon skriptissä, sain 3D-mallin piirto logiikan valmiiksi. *Näitä koodeja voi tarkemmin tarkastella liitteessä PlayerScript.cs*.

5.6.8 3D-mallien asettelu

Laajennetaan pelin toiminnallisuuksia. Pelissä pelaajan pitää pystyä asettamaan paikalleen kopio piirretystä 3D-mallista ympäristöön. Nyt pelaaja näkee koko ajan edessään reaaliajassa olevan 3D-mallin. Jotta käyttäjä voi jättää objektin paikalleen ja jatkaa toisesta objektista olisi hyvä saada jätettyä kopio itse 3D-objektista. Tällöin reaaliajassa oleva objekti jäisi pelaajan ”käsiin”.

Laajennetaan PlayerScript tiedoston Update funktiota siten, että lisätään logiikka F-näppäin painallukselle. F-napin painalluksella pelaajan pitäisi siis pystyä jättämään 3D-objektin siihen paikkaan, mihin pelaajan katse osuu parhaillaan.

Jotta pelaaja pystyy asettaa 3D-objekteja paikoilleen, pitää luoda aina uusi GameObject tyyppinen olio ja antaa sille lähes samat arvot ja komponentit, mitä jo piirretyllä peliobjektilla on. Luodaan uusi funktio PlaceObject, joka ottaa vastaan piirretyn objektin GameObject tyyppisenä oliona. Funktion sisällä lisätään nykyiseen piirrettävään 3D-objektin

peliobjektiin yhden uuden komponentin ja sen tagille arvo. Myös kyseisen objektin lapsiobjekteille lisätään komponentti MeshCollider

```
private GameObject PlaceObject(GameObject gameObject)
{
    gameObject.tag = "Furniture";
    gameObject.AddComponent<ChangeMaterialScript>();

    foreach (Transform child in gameObject.transform)
    {
        child.gameObject.AddComponent<MeshCollider>();
    }

    return gameObject;
}
```

Funktio lisää nykyiseen piirrettyyn 3D-objektiin uuden skriptin ja tagille arvon, ennen kuin se asetetaan kopioiksi uudelle GameObject tyyppiselle oliolle. Tämä uusi GameObject olio tulee edustamana paikallaan olevia objekeja peliympäristössä. Funktiossa käydään myös kaikki lapsiobjektit läpi, joille annetaan MeshCollider komponentti. Ilman kyseistä komponenttia on mahdotonta piirtää uutta 3D-mallia edellisen viereen.

Funktion palautus asetetaan uuden GameObject olion arvoksi, jonka jälkeen peliympäristöön ilmestyy kopio piirretystä GameObjektista. Unityssä aina luotu uusi GameObject vastaa samaa asiaa, kuin peliin ilmestyisi uusi elementti kuten valo, musiikki tai objekti. Uudelle objektille asetetaan myös tag arvoksi ”Furniture”.

Uuden 3D-objektin luonnin jälkeen näytetään seuraavaa 3D-mallia. Funktio InitializeGameObject lukee seuraavan 3D-mallin ja piirtää sitä seuraavaksi. Tämä aiheuttaa myös sen, että kaikki viimeisen piirron aloittamisen jälkeen piirretyn 3D-objektin lisätyt komponentit ja muutokset lähtevät pois. Jos InitializeGameObjectia ei kutsuttaisi, PlaceObject funktion sisällä lisätyt komponetit säilyisi piirretyssä 3D-objektissa.

5.6.9 3D-mallien suurentaminen ja suunnan vaihtaminen

Laajennetaan Update funktio ottamaan myös huomioon muita napin painalluksia ja suorittamaan toiminnollisuuksia sitä kautta. Yksi näistä toiminnollisuuksista on vaihtaa kappaleen koko pienemmäksi ja suuremmaksi. Käyttäjän painaessa F1- tai F2-näppäintä, objektia skaalataan 0.002 arvon verran pienemmäksi tai isommaksi.

```
if (Input.GetKeyDown(KeyCode.F1))
{
    ChangeObjectSize(new Vector3() { x = -0.002f, y = -0.002f, z = -0.002f });
}

if (Input.GetKeyDown(KeyCode.F2))
{
    ChangeObjectSize(new Vector3() { x = 0.002f, y = 0.002f, z = 0.002f });
}
```

Molemmat If-lauseet suorittavat ChangeObjectSize funktion, joka ottaa vastaan Vector3 olion. Piirretyn 3D-mallin kokoa skaalataan 0.002 yksikön verran.

5.6.10 3D-mallin pyörittäminen

Toinen toiminnallisuus mitä pelaaja voi tehdä on pyörittää piirrettävää 3D-mallia myötä -tai vastapäivään. F3 ja F4 näppäimet vaihtavat *Quaternion* tyyppistä olion arvoa peliobjektista, joka kertoo mihin suuntaan kappale osoittaa.

```
if (Input.GetKeyDown(KeyCode.F3))
{
    RotateObject(0, 10, 0);
}

if (Input.GetKeyDown(KeyCode.F4))
{
    RotateObject(0, -10, 0);
}
```

RotateObject vaihtaa piirretyn 3D-mallin suuntaan tietyn akselin suuntaan. Kuvassa peliobjektia pyöritetään Y-akselin suunnan mukaisesti.

5.6.11 Teleportaatio

Pelaaja pystyy kerrostalo ympäristössä vaihtaa huonetta teleportaatiolla. Näppäin Z ja X vaihtava pelaajan paikkaan kerrostalon sisällä. Näin pelaaja voi halutessaan katsoa muita huoneita ja asettaa 3D-malleja. *PlayerScript* TeleportLocations taulukkoon on asetettu kerrostalo kohtauksessa useita eri peliobjektien (teleport) paikkoja talteen. Skripti siirtää pelaajan näiden peliobjektien paikalle.

```
private void Teleport(int index)
{
    if (index == TeleportLocations.Length)
    {
        TeleportIndex = TeleportLocations.Length - 1;
        index = TeleportIndex;
    }
    else if(index == -1)
    {
        TeleportIndex = 0;
        index = 0;
    }


    Player.transform.position = TeleportLocations[TeleportIndex].transform.position;
}
```

Jos indeksi ylittää Teleport paikkojen määrän, asetetaan indeksi teleporttien paikkojen lukumäärän.

5.6.12 3D-mallien fokusointi

Seuraava toiminnallisuus mahdollistaa sen, että käyttäjä voi fokusoida jollakin tavoin jo asetettua peliobjektia. Tämä tarkoittaa sitä, että kun käyttäjä katsoo pelissä tiettyä jo asetettua 3D-objektia, tämän objektin väri vaihtuu indikoiden, että sinulla on fokus tähän objektiin. Silloin jos käyttäjä painaa T-näppäintä, peliobjekti poistetaan pelistä.

Laajennetaan siis nykyistä *ShootRayForObject* funktiota siten, että tarkastellaan säteen osuessa, onko osumapiste jokin asetetuista huonekaluista. Tämä otettiin jo osittain huomioon aikaisemmin koodissa, kun käyttäjä asettaa 3D-mallin ympäristöön. *PlaceObject* funktio asettaa peliobjektille tagin arvoksi ”Furniture”, joka kertoo, että kyseinen objekti on huonekalu. Säteen ampuma logiikka on laajennettu niin, että IF-lauseessa tarkastetaan, onko osumapisteen peliobjektin tagin arvo ”Furniture”. Jos IF-lauseen ehto on tosi, vaihdetaan kohdeobjektin väriä.

```
//Onko osumapisteen objektin tagi Furniture
if (Hit.transform.gameObject.tag == "Furniture")
{
    GameObject furnitureObject = Hit.transform.gameObject;
    SetFocusToObject(furnitureObject, true); // Vaihdetaan kohde objektin väriä
}
```

IF-lause tarkistaa onko säteen osumapisteen peliobjektin tagin arvo Furniture. Ehdon olleen tosi, noukitaan ylös tämä peliobjekti ja annetaan se parametrina funktiolle SetFocusToObject. Toinen parametri kertoo funktiolle annetaanko, vai otetaanko focus objektilta.

Otetaan myös huomioon, kun käyttäjä ei katso huonekalu objektia. Tällöin pitää kutsua *SetFocusToObject* funktiota uudelleen, ja antaa parametri arvona false. Toisen parametrin arvon ollessa false, se indikoi Fokus funktiolle ottamaan kyseisestä peliobjektista fokus pois, eli palauttaa kaikki edelliset värit takaisin paikalleen.

SetFocusToObject funktiolla on vastuu asettaa fokus peliobjektille. Tämä tarkoittaa sitä, että funktio pitää tiedossa yhtä fokusoitua objektia kerrallaan muistissa. Funktio kutsuu annetun peliobjektin komponenttia *ChangeMaterialScript*, ja kutsuu metodia skriptistä, joka vaihtaa objektin väriä.

5.6.13 ChangeMaterialScript

ChangeMaterialScript on skripti, jonka rooli on vaihtaa väriä asetetulle peliobjektille. Skriptin Start funktiossa kerätään ylös skriptiin linkitetyn peliobjektin materiaalit muistiin. Kun peliobjektia ei enää fokusoida, on hyvä tietää mitkä materiaalit on sitä ennen ollut käytössä ja asettaa ne takaisin peliobjektiin.

Kun peliobjektia fokusoidaan ja sen värit vaihdetaan toiseksi, se tarkoittaa sitä, että sen materiaalia vaihdetaan toiseksi. Projektissa on materiaali nimeltään *Maali-05*, joka edustaa vaaleanpunaista väriä. Peliobjektia fokusoidessa, sen materiaali vaihdetaan materiaaliin Maali-05. ChangeMaterialScript *Fokus* funktio toteuttaa peliobjektin materiaalin vaihdoin ja takaisin asettamisen.

```
void Start()
{
    ObjectMaterials = new Dictionary<string, Material>();
    foreach (Transform child in this.transform)
    {
        string objectName = child.name;
        Material mat = child.GetComponent<MeshRenderer>().material;
        ObjectMaterials.Add(objectName, mat);
    }
}
```

Käydään peliobjektin kaikki lapsiobjektit läpi ja laitetaan Dictionary listaan muistiin avaimena objektin nimi ja sen arvoksi objektin materiaali.

```
public void Focus(bool focus)
{
    foreach (var dict in ObjectMaterials)
    {
        foreach (Transform child in this.transform)
        {
            if (child.name == dict.Key)
            {
                if(focus)
                {
                    child.gameObject.GetComponent<Renderer>().material = Resources.Load("Maali-16") as Material;
                }
                else
                {
                    child.gameObject.GetComponent<Renderer>().material = dict.Value;
                }

            }

        }

    }
}
```

Jos metodi saa parametrina arvon true, jokaiselle lapsiobjektille asetetaan materiaaliksi ”Maali-05”. Muussa tapauksessa asetetaan materiaalin arvoksi alkuperäinen materiaali.

5.6.14 3D-mallien poistaminen

SetFocustoObject funktio asettaa globaalisen olion muistiin, mikä huonekalu on aina fokuksessa. Tämän 3D-mallin poistamiseen ei tarvita muuta kuin laajentaa Update funktiota tunnistamaan T-näppäimen painalluksen ja tarkistaa onko mitään objektia fokuksessa. Jos jonkin peliobjekti on fokuksessa, kutsutaan funktiota Destroy, joka kertoo Unitylle, että tämä peliobjekti halutaan poistaa pelistä.

```
if (Input.GetKeyDown(KeyCode.T))
{
    if (FocusedObject != null)
    {
        Destroy(FocusedObject);
    }
}
```

Jos If-lause on tosi, jokin objekti on nyt fokuksessa ja se poistetaan pelistä.

5.6.15 Käyttäjän omat 3D-mallit osaksi peliä

6 Johtopäätökset

6.1 Tulokset

6.2 Kohdatut haasteet

6.3 Jatkokehitysideat

7 Lähteet