Vuoropohjaisen nettipelin tekeminen Unity 3D:llä verkkoselaimelle käyttäen NativeWebSocket-kirjastoa

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus

Syksy 2023

Joona Öhberg

HAMK
Hämeen ammatti-
korkeakoulu

Tietojenkäsittelyn koulutus Tiivistelmä
Tekijä Joona Öhberg Vuosi 2023
Työn nimi Vuoropohjaisen nettipelin tekeminen Unity 3D:llä verkkoselaimelle käyttäen NativeWebSocket-kirjastoa

Ohjaaja Lasse Seppänen

Opinnäytetyön tarkoituksena oli toteuttaa netissä pelattava 3D-lautapeli Unity-pelimoottorilla. Peliin luodut säännöt saivat paljon inspiraatiota Lizzie Magien ja Charles Darrowin kehittämästä suositusta lautapelistä nimeltä Monopoli. Peli on suunniteltu toimivaksi WebGL-ohjelmointirajapintaa tukevissa nettiselaimissa ja on tarkoitettu 2–4 pelaajan pelattavaksi. Työssä tulee esille useampia osa-alueita pelikehityksestä, kuten 3D- ja 2D-grafiikan tuottaminen sekä erilaisten toiminnallisuuksien ja sääntöjen luominen käyttäjä- ja palvelinpuolelle C#-ohjelmointikielellä.

Pelin verkkoliikenteen protokollaksi valittiin NativeWebSocketin. Tämä mahdollistaa sujuvan kahdensuuntaisen yhteyden pelaajien ja palvelimen välillä, mikä on olennaista pelin sujuvuuden kannalta. NativeWebSocket-kirjaston käyttö tarjoaa myös pelille alustariippumattomuuden, mikä tarkoittaa sitä, että eri alustojen ja ohjelmistojen käyttäjät voivat avata saman sovelluksen ja yhdistää toisiinsa. Pelaajien välinen viestintä tapahtuu kevyinä JSON-paketteina. JSON-tiedon lähettäminen, käsittely ja muuntaminen on iso osa tätä projektia ja se määrää pelaajalle, että mitä tietoa pelaajan ruudulla näytetään. Pelin suorittamisen edellytyksenä on, että käytössäsi on riittävän moderni selain, joka tukee WebGL- ohjelmointirajapintaa, tietokone, joka pystyy suorittamaan alhaisen tason 3D-grafiikkaa, ja toimiva internetyhteys.

Graafisen pelin suorittaminen nettiselaimessa on huomattavasti vaativampaa verrattuna työpöytäsovelluksen käyttämiseen johtuen nettiselaimen omista rajoituksista. Tämä voi näkyä joillekin käyttäjille pätkimisenä. Nettiselaimessa pelaamisessa on kuitenkin monia etuja, kuten se, että peliä ei tarvitse asentaa erikseen, ja sitä voi suorittaa useilla eri laitteilla, laitteen rajoituksista tai säädöksistä huolimatta.

Avainsanat Unity, NativeWebSocket, WebGL, JSON, 2D, 3D, C#

HAMK
Hämeen ammatti-
korkeakoulu

Degree Programme in Business Information Technology
Author Joona Öhberg
Abstract
Year 2023
Subject Creating a turn-based online game with Unity 3D for a web browser using the NativeWebSocket library
Supervisors Lasse Seppänen

The purpose of the thesis was to implement an online 3D board game with the Unity game engine. The rules created for the game took a lot of inspiration from the popular board game called Monopoly, created by Lizzie Magie and Charles Darrow. The game is designed to work in web browsers that support WebGL technology and is intended for 2-4 players. In the work, several areas of game development will come up, such as producing 3D and -2D graphics and creating various functionalities and rules for the user and server side using the C# programming language.

NativeWebSocket was chosen as the game's network traffic protocol. This allows a smooth two-way connection between players and the server, which is essential for the smoothness of the game. Using the NativeWebSocket library also provides the game with platform independence, which means that users of different platforms and software can open the same application and connect to each other. Communication between players takes place in lightweight JSON packets. Sending, processing and converting JSON data is a big part of this project and it determines to the player what information is displayed on the player's screen. The prerequisite for playing the game is that you have a sufficiently modern browser that supports WebGL technology, a computer that can perform low-level 3D graphics, and a working internet connection.

Running a graphical game in a web browser is significantly more demanding compared to using a desktop application due to the web browser's own limitations. This may appear stuttering to some users. However, playing in a web browser has many advantages, such as the fact that the game does not need to be installed separately, and it can be played on several different devices, regardless of device restrictions or regulations.

Keywords Unity, NativeWebSocket, WebGL, JSON

Sanasto

Unity	Unity on Unity Technologiesin kehittämä monialustainen pelimoottori, jolla voidaan kehittää kaksi- ja kolmiulotteisia videopelejä useille eri alustoille.
WebGL	Verkkosivujen luomiseen tarkoitettu grafiikkarajapinta, joka mahdollistaa 2D- ja 3D grafiikan näyttämisen nettiselaimessa.
Palvelin	Tietokoneella oleva ohjelma, jonne muut käyttäjät yhdistävät ja voivat jakaa tietoa keskenään.
Websocket	Protokolla, joka mahdollistaa kahdensuuntaisen reaaliaikaisen tiedonsiirron.
Kahdensuuntainen (bi-directional)	Kahdensuuntaisella tarkoitetaan, että molemmat pelaaja, sekä palvelin pystyvät lähettämään toisillensa dataa pienellä vasteajalla ilman tarvetta pyyntö-vastaus-malliin mukaisesti.
Vasteaika	Pyynnön ja vastauksen välinen aika palvelimelta.
Skripti	Tekstitiedosto, joka sisältää listan komentoja tietokoneelle suoritettavaksi.
Alustariippumattomuus (Cross platform)	Tarkoittaa sitä, että ohjelma ei ole sidoksissa tiettyyn alustaan tai ohjelmistoon.

Sisällys

1 Johdanto 1

2 Nettipelien toimintaperiaate 3

3 Mikä on Websocket-protokolla? 4

4 Pelimoottorin ja nettikirjaston valinta 5

4.1 Unityn valinta pelimoottoriksi 6

4.2 NativeWebsocketin valinta pelin verkkoprotokollaksi? 7

5 Nettipeleissä huijaaminen 8

5.1 Suojautuminen prosessin nopeuttamisohjelmilta 8

5.2 Suojautuminen muistinmuokkausohjelmilta 9

6 Pelimaailman luominen ja grafiikka 10

6.1 Universal Render Pipeline (URP) ja High Definition Render Pipeline (HDRP) 11

6.2 Pelimaailman luominen 12

6.3 Gaia 14

6.3.1 Unityn oma Terrain-editori 15

6.3.2 Pelimaailman valaistus ja taivas Gaiaa käyttäen 16

6.3.3 Vedenpinnan luominen Gaialla 17

7 Pelin oleelliset luokat nettipeliä varten 18

7.1 WebsocketManager-luokka 18

7.1.1 Pyyntölista 19

7.1.2 HandleResponse-funktio 20

7.2 Room-luokka 20

7.3 Player-luokka 21

7.4 GameManager-skripti 22

7.4.1 Aula-silmukka 22

7.4.2 InGame-silmukka 23

7.5 PlayerPieceController-skripti 24

7.6 CameraRotate-skripti 25

8 Pelaajien välinen viestintä 27

8.1 Yhdistäminen ja tiedon lähettäminen 27

8.2 Pelaajien tunnistaminen 29

9 Pelin päätoimintojen sekä näkymien luominen 31

9.1 Tonttien kiinnitys ja avaamis- sekä talojen myynti ja osto -näkymä 31

9.1.1 Käyttöliittymäpuolen toiminnallisuus 32

9.1.2 Palvelinpuolen toiminnallisuus 34
9.2 Vaihtokauppa näkymä 35
10 HTTP-palvelin (XAMPP) 37
11 Testaaminen 38
12 Pelin käyttöliittymän toteutus ja näyttäminen 41
12.1 Canvas 41
12.2 Valikoiden näyttäminen eri pelin vaiheissa 43
13 Kehittämistyön tavoite ja tarkoitus 45
14 Projektin suunnittelu ja toteutus 47
15 Johtopäätökset ja pohdinta 48
16 Yhteenveto 49
Lähteet 50

1 Johdanto

Unity 3D on yksi suosituimmista pelintekotyökaluista, erityisesti indie-kehittäjien keskuudessa. Unity tarjoaa tehokkaat työkalut pelien kehittämiseen ja julkaisemiseen useille eri alustoille. Sen avulla kehittäjät voivat keskittyä pelin kehittämiseen, kun taas Unity hoitaa monimutkaisemmat tekniset asiat, kuten grafiikan ja fysiikan. Unityn laaja alustatuki, joka kattaa muun muassa PC:n, Macin, Linuxin, Androidin, iOS:n, WebGL:n, VR:n ja konsolit, tekee siitä houkuttelevan vaihtoehdon 2D- ja 3D-grafiikkaa hyödyntävien sovellusten kehittäjille. Endelin luoma NativeWebSockets-kirjasto tarjoaa helpon ratkaisun luoda kahdensuuntainen reaaliaikainen Websocket-yhteys pelaajien välille palvelimen avulla..

Opinnäytetyö keskittyy yhdistämään Unity-pelimoottorin ja NativeWebSocket-kirjaston käytön keskenään. Niiden yhdistelmä mahdollistaa sen, että pelaajat voivat kommunikoida palvelimen kanssa aktiivisen WebSocket-yhteyden avulla. WebSocket on protokolla, joka mahdollistaa reaaliaikaisen kaksisuuntaisen viestinnän verkon yli. Se soveltuu erinomaisesti monenlaisiin sovelluksiin, kuten chat-sovelluksiin ja verkkopeleihin. NativeWebSocket-kirjaston käyttö mahdollistaa sen, että yhteys toimii myös selaimissakin ja täten mahdollistaa alustariippumattomuuden WebGL-ohjelmointirajapintaa tukevien laitteiden välillä. Opinnäytetyön isona osana oli myös pelissä olevien pääluokkien suunnittelu ja toteutus. Ne sisältävät kaiken tiedon, jota pelaaja lähettää ja saa palvelimelta vastauksena takaisin muunnettuna JSON-muotoon.

Opinnäytetyössä toteutin toimivan vuoropohjaisen selaimella pelattavan nettipelin. Pelaajien tehtävänä on ostaa kiinteistöjä, rakentaa taloja ja saada muut pelaajat ajautumaan konkurssiin. Peli tarjoaa myös mahdollisuuden tehdä kauppaa toisten pelaajien kanssa. Pelaajien liikkuminen pelissä perustuu siihen, minkä numeron he saavat heittämällä noppaa, ja tämä numero vaihtelee satunaisesti 2-12 väliltä. Pelissä voit myös kiinnittää ja tehdä kauppaa ostamillasi tonteilla. Peli sai paljon inspiraatiota suositusta lautapelistä nimeltä Monopoli, sääntöjen ja ulkoasun kannalta. Opinnäytetyössä käydään läpi myös Unityn oman Terrain-Editorin käyttö ja Procedural Worldsin kehittämä maaston generointiohjelma nimeltä Gaia. Pelissä tiedon välittäminen tapahtuu Websocket-protokollan avulla. Pelaajat voivat yhdistää toisiinsa ja lähettää sekä saada tietoa JSON-muodossa. Opinnäytetyö keskittyy toimivan vuoropohjaisen nettipelin tekemiseen selaimille, jotka tukevat WebGL-ohjelmointirajapintaa. Tähän liittyy paljon eri osa-alueita kuten, oikeiden käyttöliittymäelementtien näyttäminen palvelimelta saadun datan perusteella, pelaajien pelinappuloiden liikuttaminen oikeaan ruutuun sekä pelilaudan kiinteistöjen kiinnitykset ja talojen luominen / tuhoaminen pelilaudalla.

Tutkimuskysymykset:

Millaisia rajotteita saattaa ilmetä WebGL-ohjelmointirajapinnalle julkaistussa pelissä?

Mitä erilaisia tapoja pelaajien väliseen viestittelyyn löytyy?

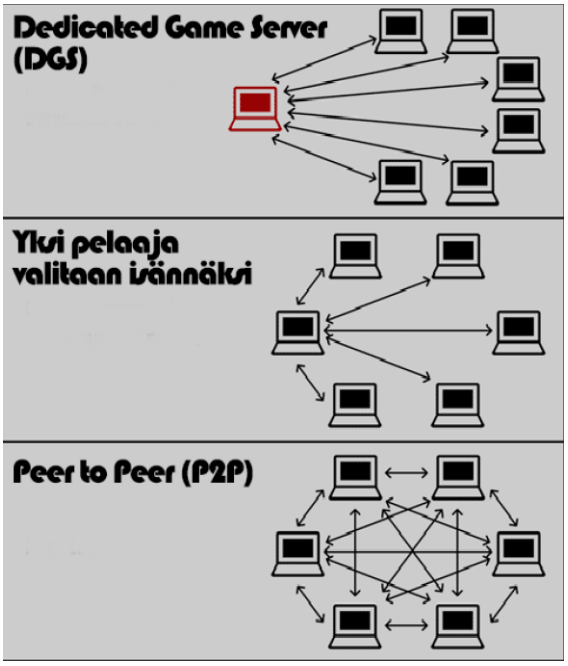
Miten pelaajien välinen viestittely tapahtuu Websocket-protokollalla?

2 Nettipelien toimintaperiaate

Nettipelien toimintaperiaate perustuu pelaajien väliseen tietojenvaihtoon, jossa kaksi tai useampi pelaaja jakavat tietoa toistensa tilasta. Tämä tapahtuu joko palvelimen, kuten omistetun pelipalvelimen (Dedicated Game Server, DGS), välityksellä, suorana yhteytenä ilman erillistä isäntää (Peer to Peer, P2P), tai yksi pelaajista toimii pelin isäntänä "Listen Server". Lisäksi on mahdollista yhdistää näitä teknologioita keskenään. (Kuva 1.)

P2P (Peer to Peer) on arkkitehtuuri, jossa jokainen pelaaja yhdistää toisiinsa ja välittää tietoa pelintilasta ja toiminnoista. Puhtaassa P2P-verkkotopologiassa ei ole erillistä isäntää vaan jokaisen pelaajan tehtävä on pitää huoli omasta tilastaan samalla, kun hän saa päivityksiä muiden pelaajien tiloista.DGS (Dedicated Game Server) menetelmällä nettipelien tekemisessä kehittäjän pitää pelaajanäkymän lisäksi myös rakentaa oma palvelin ja pitää huoli, että palvelin hyväksyy uusien pelaajien yhteydet tai käyttää valmista palvelinratkaisua. Tämä on yleensä turvallisempi ratkaisu peleille, jossa data on arkaa, jotta huijausmenetelmien kehittäminen olisi vaikeampaa. Palvelinratkaisuissa, jossa pelaajien välinen data kulkee pelkästään pelaajien välillä tai jossa yksi pelaaja toimii pelin isäntänä on paljon positiivisia puolia, mutta myös negatiivisia. Positiivisen pelin verkkoratkaisusta tekee sen, että siitä ei koidu erillisiä kustannuksia, mutta huijausmenetelmien tekeminen on paljon helpompaa, koska ei ole erillistä palvelinta, joka varmistaisi pelaajien lähettämän datan eheyden vaan pelaajilla ja varsinkin pelin isännällä olisi täysi hallinta pelaajien välisessä liikkuvassa datassa. (Hackernoon2023a)

Kuva 1 Verkkotopologiat (DGS, Hybridi, P2P)



3 Mikä on Websocket-protokolla?

Websocket on tietoliikenneprotokolla toiselta nimeltään RFC 6455, joka mahdollistaa kahdensuuntaisen reaaliaikaisen tiedonsiirron verkon yli. Se tarjoaa jatkuvan yhteyden pelaajan ja palvelimen välillä, mikä mahdollistaa reaaliaikaisen viestittelyn ilman tarvetta jatkuvasti lähettää pyyntöjä ja vastauksia, kuten perinteisessä http-protokollassa. (Mozilla2023a) Websocket yhtäläisyydet http-protokollaan ovat ne, että se tulkitsee käsittelyvaiheen HTTP-päivityspyyntöinä. (datatracker2023a)

Websocket perustuu TCP-protokollaan, eli se on vain kerros TCP-protokollan päällä, joka mahdollistaa tekstin tai binaaridatan lähettämisen pelaajan ja palvelimen välillä. Websocket yhteys alkaa URI muotoisella `ws://` tai `wss://` aloitteella. Ensimmäinen eli `ws://` tarkoittaa tässä tilanteessa normaalia ei salattua yhteysmuotoa ja oletusportti on sama kuin http-yhteyksissä käytettävä eli 80. Toinen `wss://` tarkoittaa salattua yhteyttä jonka oletusportti on 443, joka toimii myös HTTPS-protokollan oletusporttina. Se on tilallinen protokolla, mikä tarkoittaa sitä, että pelaajan ja palvelimen välinen yhteys säilyy samalla kanavalla niin kauan kunnes, jompikumpi osapuoli (pelaaja tai palvelin) sulkee sen. Sulkemisen yhteydessä yhteys katkeaa molemmilta osapuolilta. (GG2023a) Websockettien käyttöön liittyvissä asioissa, kuten versionhallinnassa. Websocket asiakas tai palvelimen ei välttämättä tarvitse olla samaa versioita tai suositelluin versio saadakseen viestin. Jos palvelin tukee pelaajan Websocket-versioita ja viesti on muuten oikein muotoiltu niin palvelin hyväksyy pelaajan lähettämän viestin, muulloin palvelin vastaa pelaajalle virheviestillä, jossa näkyvät kaikki versiot, joita palvelimen Websocket-versio tukee. (datatracker2023a)

Tietoturva on yksi kysymys mikä tulee mieleen, kun puhutaan asiakkaiden välisestä kommunikaatiosta. Websocket-protokollan tietoturva riippuu myös asetetun webbipalvelimen tietoturvasta. Esimerkiksi Cross-Origin Resource-Sharing, on tekniikka, joka sallii verkkosivustojen tai websovellusten lähettää http-pyyntöjä toisille verkkosivustoille, jotka sijaitsevat eri verkkotunnuksissa tai protokollissa. Väärin asetettuna kyseinen tekniikka voi altistaa verkkosivun ristikkäissivustoskriptaukselle. Tämä käytännössä tarkoittaa sitä, että eteneemmällä käyttäjällä on mahdollisuus asettaa pelipalvelimelle oma haitallinen JavaScript-kooditiedosto, jolla hyökkääjä saa täyden hallinnan nettisivustolle millä hän pystyisi hallitsemaan ja keräämään käyttäjistä tietoa tai ohjaamaan käyttäjät kokonaan uudelle verkkosivustolle. Tämä on erityisesti ongelma sivustoilla, jossa käyttäjä voi syöttää omaa dataa nettisivulle, kuten palautelomakkeet, jossa tieto tallennetaan suoraan tietokantaan ja näytetään nettisivustolla. (owasp2023a)

4 Pelimoottorin ja nettikirjaston valinta

Nykypäivänä internetissä on paljon pelimoottoreita ja nettikirjastoja, joitten väliltä kehittäjä voi valita. Sen takia tässä kappaleessa otetaankin läheisempään vertailuun kaksi tunnettua pelimoottoria Unity ja Unreal Engine. Ennen pelimoottorin valintaa kehittäjien olisi hyvä pitää mielessä projektinsa lopulliset tavoitteet, kuten myös budjetti. Jokaisella pelimoottorilla on omat hyvät ja huonot puolensa. Toiset ovat kokonaan ilmaisia, jopa avoimen lähdekoodin tasolla, kun taas toiset ovat ilmaisia tiettyyn tulorajaan saakka. Tätä bisnesmallia käyttää esimerkiksi Unity, sillä pelimoottoreiden oppiminen ei ole aina mikään pieni askel ja suurin osa kehittäjistä aloittaakin pelien kehittämisen harrastuksena. Unity-pelimoottorin käyttäjä voi ansaita vuodessa enintään 100 000 dollaria pelillään, minkä jälkeen hänen on hankittava lisenssi. Lisenssin myötä tuloraja nousee: Plus-versiossa se on 200 000 dollaria ja Pro-versiossa rajaa ei ole. (UnityToS 6.11.2023a) Unreal Engine taas perii käyttäjältä 5% pelin tuloista, jos peli tienaa enemmän kuin 1 000 000$ vuodessa. (UnrealEngineToS 6.11.2023a).

Pelimoottoria ei kuitenkaan pitäisi valita yrityksen liiketoimintamallin mukaan. On hyvää pitää mielessä tämä, mutta se ei saisi olla syy lopulliseen päätökseen. Tärkeimpiä asioita pelimoottorin valinnassa projektillesi on, että mille alustalle projekti suunnitellaan ja, että kuinka kattava käyttäjäryhmä kyseisellä pelimoottorilla on. Mitä suositumpi pelimoottori niin todennäköisemmin löydät helpommin apua jokapäiväisiin ongelmiin, mitä pelikehitys tuo tullessaan. Myös aiempi ohjelmointiosaamisen kannattaa pitää mielessä pelimoottoria valitessa, esimerkiksi Unreal Engine käyttää c++:aa ohjelmointikielenä tai blueprint-systeemiä toimintojen tekemiseen. Unity taas C#:aa tai JavaScriptiä.

Pelimoottorin valinnassa seuraava merkittävä tekijä liittyy projektin kehityksen tehostamiseen, erityisesti sellaisiin osa-alueisiin, jotka nopeuttavat projektin valmistumista ja tarjoavat valmiita resursseja. Näitä voivat olla esimerkiksi valmiit 3D-objektit varjostimineen, animoidut hahmot sekä työkalut, joiden avulla voit luoda ja muokata geometriaa. Sekä Unity että Unreal Engine tarjoavat oman kauppanäkymänsä, josta kehittäjät voivat helposti etsiä ja integroida projektiinsa erilaisia toiminnallisuuksia ja objekteja.

4.1 Unityn valinta pelimoottoriksi

Unity tarjoaa vahvan tuen 2D- ja 3D-grafiikan näyttämiseen ja luomiseen, suorituskyvyn tarkkailuun ja maailman muokkaamiseen erilaisilla peliobjekteilla, joita voi ladata Unityn omasta Assets Storesta tai luoda 3D-grafiikan käsittelyohjelmilla, kuten Blender. Unity tarjoaa myös tuen partikkeliefektejen luomiseen ja näyttämiseen, jolla saa luotua eloa maailmaan. Projektissa kuitenkin täytyy muistaa se, että se on suunniteltu nettiselaimella pelattavaksi, joten liian raskas grafiikka ei ole vaihtoehto, kuten ylisuurien tekstuurien käyttäminen tai maastonmuokkauksessa liian tiheä ympäristöobjektejen käyttö. Unityllä pystyt rakentamaan pelin selaimelle WebGL-moduulilla. WebGL-ohjelmointirajapinta tukee useampia selaimia kuten, Google Chrome, Mozilla Firefox, Apple Safari ja Microsoft Edge. Joissakin näissä selaimissa on omia rajoituksia, kuten Safarin vanhemmat versiot (15 ja ennen) ei tue uudempaa WebGL 2-versioita, jolle peli on rakennettu. (Unity 2022.3a)

C#-ohjelmointikieli on suhteellisen helppo oppia ja päästä alkuun. C# tarjoaa vahvan tyypityksen, mikä helpottaa koodin luettavuutta ja täten myös tuotettavuutta. C# on hyvin dokumentoitu kieli, josta on paljon esimerkkejä Microsoftin ja Unityn omalla API-sivulla. Laaja yhteisö ja tuki tarjoaa runsaasti resursseja, kirjastoja ja apua omilla foorumeilla, josta saa tarvittaessa apua nopeasti. C# on suhteellisen tehokas kieli ja tarjoaa monia kehittäjää helpottavia ominaisuuksia kuten, muistinhallinnan automaation (Garbage Collection). Muistinhallinnan automaatio auttaa estämään monia yleisiä ohjelmointivirheitä, kuten puskurin ylivuotoja, joka johtaa muistin vioittumiseen ja lopulta ohjelman kaatumiseen. (Microsoft2023a)

Menestyvän pelin kannalta yksi keskeisimmistä tekijöistä on saavutettavuus. Peli suunnitellaan pelattavaksi nettiselaimessa, jotta mahdollisimman moni käyttäjä voisi pelata sitä, heikommallakin laitteella riippumatta laitteistosta tai laitteensa käyttöoikeuksista. Asiakkaiden tarvitsee vain käyttää laitetta, jossa on riittävän moderni käyttöjärjestelmä ja nettiselain, joka tukee WebGL 2 -ohjelmointirajapintaa, sekä toimiva nettiyhteys. Tällaiset selaimet ovat yleisiä nykyaikaisissa laitteissa, kuten älypuhelimissa ja tietokoneissa. Unity tarjoaa valmiin tuen WebGL-julkaisulle ilman erillisiä lisäosia.

4.2 NativeWebsocketin valinta pelin verkkoprotokollaksi?

Endelin luoma NativeWebsocketSharp-kirjasto perustui kehittäjän nimeltä Jiri Hybekin avoimen lähdekoodin projektiin unity-websocket-webgl. Ideana heillä oli luoda ratkaisu, jotta WebGL-ohjelmointirajapinta pystyisi kommunikoimaan palvelimen kanssa. Tämä sen takia, että WebGL ei tue Microsoftin System.Net-kirjastoja, johtuen selaimen tietoturvasta ja Unityn oma UnityWebRequest-kirjasto ei tarjoa samanaikaista yhteyttä pelaajan ja palvelimen välillä mikä on välttämätöntä nettipelissä, jossa pelaajan täytyy tietää muiden pelaajien tila säännöllisin väliajoin. Websocket-protokolla on kehitetty tarjoamaan samanaikaisen kahdensuuntaisen kommunikaation palvelimen ja pelaajan välillä pienellä vasteajalla, kun taas Unityn omaa WebRequest-kirjastoa käytetään enemmänkin satunnaisen tiedon hakemiseen tai lisäämiseen palvelimelle http-pyyntöjen avulla. Tämän takia Websocketit ovat parempi vaihtoehto vuoropohjaisen nettipelin toteuttamiseen pelissä olevien vaatimusten takia, kuten samanaikaisen yhteyden muodostamiseen pelaajien ja palvelimen välillä, jotta pelaajan näytölle päivittyisi muiden pelaajien tilat säännöllisin väliajoin. (Github 2023a)

Toinen etu Endelin luomassa NativeWebsocket-kirjastossa on, että palvelinpuoli toimii myös kokonaan C#-ohjelmointikielellä, joka on sama kieli mitä Unityssä käytetään. Se tarjoaa vahvan tyypityksen ja monia hyödyllisiä kirjastoja, kuten System.Linq, jotka auttavat listojen ja taulukoiden tiedon käsittelyssä ja muokkaamisessa. Tämä auttaa palvelinpuolen toimintojen rakentamisessa, kuten tiedontallennus- lähetys- systeemin luomisessa. Unityssä on useita eri verkkokirjastovaihtoehtoja, kuten PUN (Photon Unity Networking), mutta mikä tekee Websocket-protokollasta houkuttelevan vaihtoehdon on, että kehittäjä saa täyden hallinnan pelaajien välisestä viestinnästä. Oma palvelin infrastruktuuri on kuitenkin kustannustehokkaampi vaihtoehto ja helpommin skaalattava ilman erillisiä maksurajoituksia. Esimerkiksi ilmaisella PUN palvelinsuunnitelmalla pystyt maksimissaan pitämään vain 20 samanaikaista yhteyttä aktiivisena kerrallaan verrattuna Websocketteihin niin tämä luku on moninkertainen, riippuen tietenkin palvelinkoneen ja verkon tehoista, sekä palvelinohjelmiston tiedon käsittelyn tiheydestä ja vaativuudesta. (Photon2023a) Plussana palveluissa kuten PUN on tietenkin se, että näihin on jo valmiiksi rakennettu toiminnallisuudet pelaajien liittämisessä yhteen ja verkkopelaamisen tilan hallintaan sekä synkronointiin. Websockettien kanssa kehittäjä joutuu itse keksimään ratkaisut, että miten saada pelaajat yhdistämään ja viestittelemään toistensa kanssa.

5 Nettipeleissä huijaaminen

Nettipeleissä on tapana, että peli yritetään luoda siten, että siinä on mahdollisimman vaikea huijata. Huijaamisella pelimaailmassa tarkoitetaan pelissäolevien arvojen muuttamista ohjelmallisesti tilanteissa missä niitä ei pitäisi pystyä muokkaamaan. Pääperiaate huijaamisen estolle on se, että palvelimen ei ikinä pitäisi luottaa sokeasti pelaajien lähettämiin arvoihin. Tällä tarkoitetaan sitä, pelaajan ei pitäisi pystyä muokkaamaan omia tai muiden pelaajien arvoja ilman palvelimen suostumista.

Huijaaminen voi lähteä ihan pelin omasta viasta, jossa pelin tiettyä toiminnallisuutta voi käyttää hyväksi ei tarkoitetulla tavalla, antaen pelaajalle etuja, kuten rahaa pelissä. On olemassa myös ohjelmia, jolla käyttäjät pystyvät muokkaamaan pelin muistiosoitteita mikä vaikuttaa pelissä oleviin arvoihin. Nettipeleissä tämä ei ole yleensä ongelma, koska palvelimen pitää hyväksyä data ennen sen käsittelyä, mutta palvelimella, joka hyväksyy pelaajan lähettämän datan ilman erillisiä tarkistuksia niin se on ongelma. Tätä huijaus tapaa kutstutaan verkkoliikenteen väärentämiseksi, joka tarkoittaa, että käyttäjä muuttaa verkkopaketin arvoja juuri ennen sen lähettämistä palvelimelle. (Cheating in video games)

Tässä kaikessa pitää kuitenkin muistaa se, että pelin tekeminen on iso prosessi ja aloitteleville kehittäjillä on jo muutenkin tarpeeksi eri osa-alueita mihin heidän pitää keskittyä. Nettipelissä huijaamiselta suojautuminen pitää ottaa sitten vasta huomioon, kun sille on oikeasti tarve ja pelaajan data on arvokasta. Tässä projektissa esimerkiksi pelaajan dataa ei talleteta pelin jälkeen palvelimelle, joka tekee pelissä huijaamisesta hyödytöntä.

5.1 Suojautuminen prosessin nopeuttamisohjelmilta

Yksi yleisimmistä huijauskeinoista on prosessin nopeuttaminen. Tässä käyttäjä modifio pelin suoritusnopeutta, jolloin pelin toiminnot nopeutuvat pelaajan ruudussa. Pelissä, jossa pelaajan itse täytyy liikuttaa hahmoansa niin tämä toisi etua pelaajalle.. Vuoropainotteisessa pelissä sillä ei ole väliä, että kuinka nopeasti pelinappulat liikkuvat laudalla. Tähän voi kuitenkin luoda tarkistuksen pelin Update-silmukkaan, joka pitää silmällä käyttääkö pelaaja nopeuden muuttamista koodilla (Ohjelmakoodi 1.). (Protect you games from being hacked)

Ohjelmakoodi 1 Prosessin nopeuden muokkauksen havaitseminen.

```
systemTime = System data in seconds.
timer = systemTime + (time delta between this frame and last time).
If (timer - systemTime) > maximum_time_difference_allowed
PlayerUsesSpeedHack();
```

5.2 Suojautuminen muistinmuokkausohjelmilta

Muistinhallintaohjelmia on monia tarjolla arikipäivän käyttäjille. Tämä on yksi helpoimmista tavoista huijata pelissä, muuttamalla tai jäädyttämällä sovelluksen muistiosotteita käyttäjän haluamaansa arvoonsa. Nämä ohjelmat pystyvät skannaamaan jokaisen muistiosoitteen pelissä ja niiden peruskäyttö ei ole vaikeaa. Ohjelmien helpon saatavuuden takia peli on hyvä yrittää suojata mahdollisimman hyvin näiltä.

Se ei ole aina helppoa, koska isotkin AAA-peliyhtiöt eivät aina onnistu tässä. Tästä yksi hyvä esimerkki on Rockstar Gamesin luoma Grand Theft Auto V. Pelissä on nettipelimuoto ja mikromaksuja, jolla käyttäjä voi ostaa pelinsisäistä rahaa. Jos pelissä itsessään pystyy tekemään moninkertaisen määrän rahaa muistineditointiohjelmilla niin miksi käyttäjät ostaisivat yhtiön tarjoamia rahapaketteja? Tämä on iso menetys ollut Rockstaarille. Rockstar on yrittäynyt suojata peliänsä muutamilla keinoilla, kuten, jos pelaajan aika pelissä ei mene yhteen tämän saadun rahamärään kanssa niin tällöin on todella todennäköistä, että pelaaja on huijannut rahansa ja palvelin antaa automaattisesti pelaajalle porttikiellon. Rockstarin tekemä huijauksenestosysteemi toimii siten, että se pystyy kertomaan rehdisti ja huijaamisella hankitun rahan väliltä tai ainakin he väittävät näin. Totuus menee kuitenkin tästä pidemmälle. Käyttäjät pystyvät helposti huijaamaan nopeasti pieniä summia rahaa, käyttäen pelinsisäisiä elementtejä ja muuttaen näiden arvoja ohjelmallisesti itselle edukkaamaksi, kuten kasinoita ja muokkaamalla näiden arvoja, jotta pelaaja saa aina voittokertoimet. Tämä tekeekin Rockstaarin luomasta systeemistä ongelmallisen, vaikka pelaaja ei pysty huijaamaan isoja määriä rahaa kerralla niin pelaaja pystyy kuitenkin huijaamaan itselleen muutamia miljoonia pienessä ajassa varsinkin, jos pelaajalla on jo vanha käyttäjätili mille on kertynyt useita pelitunteja. Näitä muistineditointi-tiedostoja, jossa tiedoston tekijä on löytänyt muistisarakkeiden ’pointer’-osoitteet jaetaankin internetin foorumeilla ilmaiseksi. (Unknown Cheats)

Ashish Gogna:n (Sovelluskehittäjä, Mobile Premier Leaguen) mukaan yksinkertaisin tapa suojautua muistineditointiohjelmilta tapahtuu siten, että jokainen arvo, joka tallennetaan muistiin salataan ennen sen tallettamista. Tämä tekee sen, että hakkerit eivät löydä muistiosotteita ainakaan niin helposti. Toinen tapa on luoda jokaiselle muuttujalle duplikaatti muuttuja, jonka arvo on `viimeinen arvo + arvottu numero`. Kun käyttäjä päivittää pelaajan aitoa dataa niin katsotaan ensin, että pelaajan aiemmin tallettama arvo, miinus arvottu numero, jos näin ei ole niin sitä on muokattu. (Protect your game from getting hacked)

6 Pelimaailman luominen ja grafiikka

Pelin graafinen toteutus on iso osa peliä ja sillä on merkittävä vaikutus peliin pelattavuuden kannalta. Grafiikka on yksi niistä tekijöistä, jotka voivat tehdä pelistä oikein tehtynä houkuttelevan ja immersiivisen pelaajille. Unityssä on työkaluja ja kolmannnen osapuolen lisäosia, jotka helpottavat ja nopeuttavat projektin tekemistä. Unity tarjoaa useita työkaluja jotka helpottavat maailman luomisessa, kuten Unityn oma Terrain Editor tai Probuilderin. Probuilder on suunniteltu muokkaamaan perus 3D-muotoja ja luomaan näihin geometriaa. Shader Graphin avulla voi myös käyttää Unityn tekemiä tai luoda omia varjostimia, jotka määräävät sen, että miltä objekti näyttää näkymässä olevan valaistuksen kanssa. Unitystä voit myös vaihtaa projektin rendaustyyliä URP:n ja HDRP:n väliltä. HDRP on fyysisesti perustuva renderöijä, joka käyttää valo yksiköitä. Se tarjoaa valmiita edistyneitä materiaaleja, valoja, pilviä, volyymiominaisuuksia, reaaliaikaista ja säteenseurantaa teräväpiirtovisuaalien saavuttamiseksi. URP taas puolestaan tarjoaa optimoidun grafiikkasuorituksen useille alustoille. URP tarjoaa kehittäjille käyttäjäystävällisiä työnkulkuja, joiden avulla voit luoda optimoitua grafiikkaa useille alustoille kuten mobiililaitteille, sekä huippuluokan tietokoneille ja konsoleille. (Unity 2023a)

Pelimaailman suunnittelussa pitää ottaa huomioon pelaajan kameran liikerata ja se, että mille alustalle peliä tehdään. Tarkoituksena oli luoda pelilautaa ympäröivä maasto ja muut objektit siten, että pelaajan silmä kuitenkin keskittyisi itse pelilautaan. Myös grafiikan tekstuurikuvien resoluutiot piti pitää mahdollisemman pienenä, jotta lopullisen rakennetun projektin kokonaiskoko olisi riittävän suppea selainpeliksi. Lopullisen rakennetun projektin koon pitämiseen kurissa auttoi se, että jokainen tekstuuri, joka asetetaan peliobjekteille piti olla potenssiin 2 samoilla leveys- ja korkeus arvoilla. Tämä sai aikaan sen, että tekstuurin kokonaiskoko on huomattavasti pienempi. Se johtuu pääasiassa siitä miten grafiikkalaitteisto käsittelee tekstuurikuvia. (Unity 2022.3b)

Kun puhutaan grafiikasta niin isossa osassa on pelaajan kamera. Kamera pitää sisällään kaiken sen datan miten grafiikka ilmaistaan pelaajalle. Kamerassa on useita eri asetuksia, joita säätelemällä kehittäjä voi valita miten haluaa näyttää pelimaailmansa pelaajalle. Yhdessä näkymässä voi olla useampi kamera aktiivisena samaan aikaan ja voit myös päättää, että mitä tasoja mikäkin kamera näyttää pelaajalle. Tässä projektissa kuitenkin on vain yksi kamera ja sen arvot ovat asetettu käytössä olevan URP-pipelinen arvojen mukaisesti.

6.1 Universal Render Pipeline (URP) ja High Definition Render Pipeline (HDRP)

Unityn oma Universal Render Pipeline on yksi Unity-pelimoottorin rendaustyökaluista, joka on suunniteltu tarjoamaan tehokkaan ja joustavan grafiikkarajapinnan 2D- ja 3D-projekteihin. Valitsin tähän projektiin URP:n, koska se tarjoaa enemmän suorituskykyä WebGL-peleihin HDRP:n sijasta. URP käyttää yksinkertaisempia rendaustekniikoita HDRP:hen verrattuna. Tämä johtuu siitä, että varjostimet, jotka ovat kirjoitettu käyttäen URP:tä ovat enemmän yksinkertaistettuja verrattuna HDRP kirjoitettuihin varjostimiin.

Molemmat renderöintiputket tarjoavat fyysikkaperusteisen (Physically Based Rendering) renderöinnin, jolla pyritään saamaan näyttävämpää grafiikka mallintamalla valon käyttäytymistä mahdollisimman tarkasti. Fysiikkaperusteisen renderöinnin tavoitteena on näyttää pelissä oleva grafiikka eri valaisuolosuhteissa pienemmällä laskentateholla. Menetelmän pääosa-alueita ovat heijastuminen, diffuusio, läpinäkyvyys ja läpikuultavuus, energian säilyvyys, metallisuus ja fresnel-kuvio. (Adobe2023a)

URP tukee viimeisimpiä rendaustekniikoita kuten, dynaamista valaistusta, GPU-instanssointia ja post-prosessointiefektejä. Unity on suunnitellut URP-renderöintiputken siten, että se tarjoaa suorituskykyä puhelimien väliltä korkean tason tietokoneisiin.. (Occasoftware 2023a)

URP tarjoaa laajan valikoiman erilaisia varjostimia kuten: Complex Lit, Lit, Simple Lit, Unlit, Terrain Lit, Particles Lit, Particles Simple Lit, Particles Unlit, SpeedTree, Decal & AutoDesk Interactive. Jokaisella näistä varjostimilla on omat matemaattiset kaavansa miten ne toimivat ja miten ne reagoivat valon kanssa. Varjostimien matemaattisissa laskukaavoissa, joissa käytetään exponenttia, logaritmia tai trigometrisiä luvun laskentafunktioita, kuten cos, sin ja tan tarvitsevat paljon enemmän laskentatehoa verrattuina varjostimiin, jotka käyttävät simppeleitä matemaattisia laskutoimituksia. (Unity 2022.3d)

6.2 Pelimaailman luominen

Pelimaailma koostuu useasta eri peliobjektista. Näitä ovat esimerkiksi ympäristöön liittyvät peliobjektit, kuten

Pelimaailman luomiseen käytettiin Unityn omaa maaston tekemis- ja muokkaustyökalua sekä Gaia-maaston, veden, valaistuksen ja sumun generointiohjelmaa. Maailma on suunniteltu siten, että pelilauta sijaitsee saarella joita ympyröi harjatyökalulla luodut vuoret. Saarta ympäröi meri, joka luotiin Gaian veden generointityökalulla. Pelinäkymässä olevan sumun arvot ovat asetettu siten, että sumu alkaa vasta vuorten takana sijaitsevasta rannasta ja tihenee mitä pidemmälle mennään. Tämä saa aikaan sen efektin, että pelimaailma olisi isompi mitä se oikeasti on ja pelaajan näkökenttä loppuisi tiheimpään sumuun. (Kuva 2.)

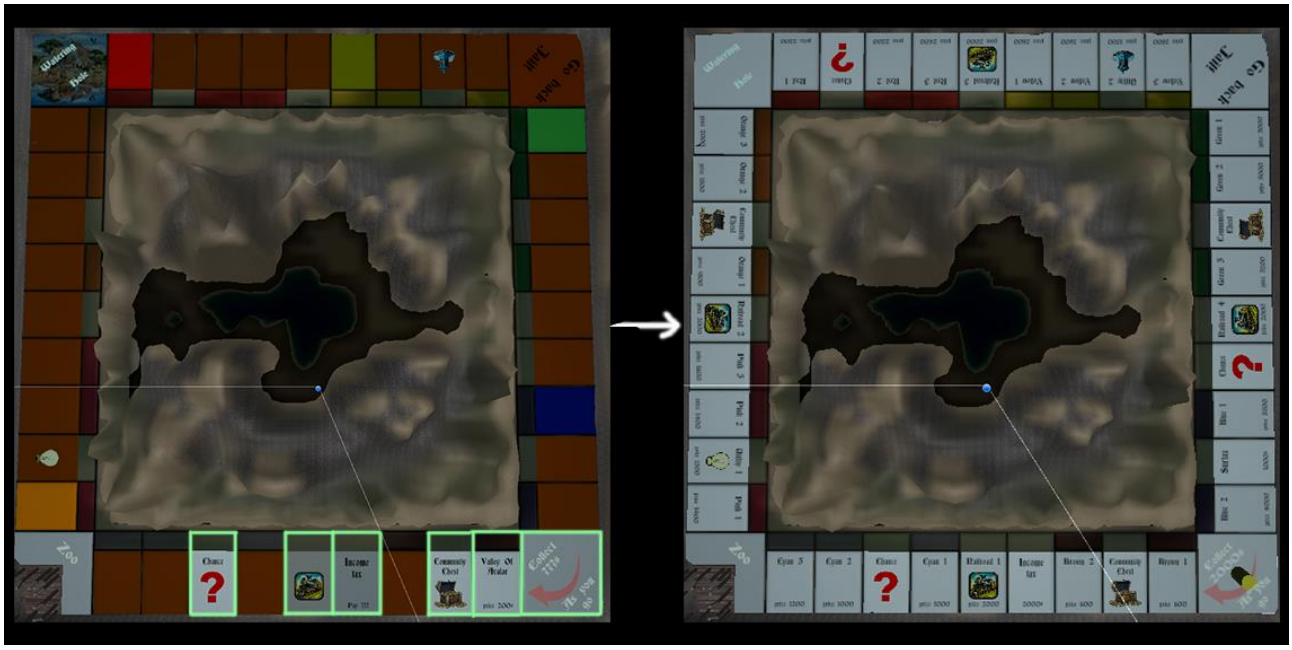
Kuva 2 Päävalikosta, jossa arvot ovat asetettu, siten, että maailma näyttää isommalta mtiä se oikeasti on.

Pelilaudan tekemisessä piti ottaa huomioon se, että pelissä liikutaan nopalla ja tämä oli ratkaiseva tekijä pelilaudan suunnittelussa. Pelilauta koostuu 40 + 1 ruudusta, josta ylimääräinen ruutu toimii eräänlaisena vankilaruutuna, johon ei normaalisti pelaaja kulje liikkuessaan laudalla. Joten tätä ruutua ei voinut sisällyttää normaaliin ruutulistaan pelaajan nappulan liikkumisalgoritmin takia. Pelilaudan ruutujen ulkoasu perustuu siihen, että jokaisessa ruudussa on joko TextMeshPro-tekstiobjekti tai plane-peliobjekti, joka toimii kuvan näyttämiseen 3D-maailmassa pelaajalle. Pelin Board.cs-skriptin Initialize-funktio hoitaa laudan tekstien ja kuvien alustuksen, joka suoritetaan

pelin GameManager-skriptin ConnectLoop-funktiossa pelin latausvaiheessa, sen jälkeen kun pelinarvot ovat ladattu palvelimelta.

Funktion tehtävä on listata kaikki pelissä olevat ruudut ja palauttaa ne, jossa on ”Square”-nimessä. Tämä saa aikaa sen, että kehittäjän pitää vain alustaa arvot jokaiseen uniikkiin ruutuun kerran, jolloin skripti silmukoi ottaa tästä ruudusta teksti- ja kuvaobjektin ja duplikoi sen muihin ruutuihin. (Kuva 3.)

Kuva 3 Vasemmalla alustamaton pelilauta. Oikealla alustettu pelilauta.



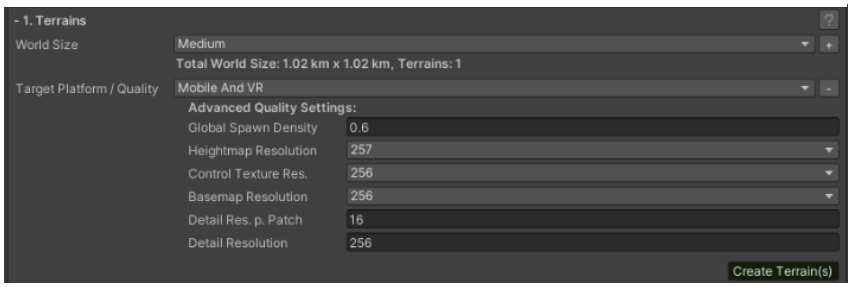
"Board.cs"-nimisessä skriptissä sijaitsevat kaikki funktiot, joita käytetään laudan hallintaan. Pelissä voi esiintyä tilanteita, joissa pelaaja saattaa kiinnittää yhden tai useamman tontin. Tällöin skriptin "UpdateMortgageSquares"-nimisen funktion tulee suorittaa silmukka ja tarkistaa, onko pelaajien tonttilistojen "mortgaged"-arvoissa tapahtunut muutoksia. Jos muutoksia havaitaan, metodin tehtävänä on muuttaa ruutujen värit sen mukaan, onko pelaajan tontti kiinnitetty vai ei. Jokaisella pelissä olevalla tontilla on oma indeksiarvo, jonka avulla pelissä voidaan tunnistaa, mikä tontti on kyseessä. Board-luokassa on metodi, joka ottaa parametrina pelaajan tontin ja palauttaa pelaajalle ruutulistasta ruudun indeksiarvon. Tämän arvon avulla pelilaudalla pystytään hallitsemaan tonttien materiaalien ja ikonien kiinnitysvärejä.

6.3 Gaia

Gaia on Procedural Worlds:in luoma lisäosa Unityyn, jolla pystyy generoimaan kokonaisia maailmoja käyttäjän antamien parametrien perusteella. Gaia tarjoaa myös monia työkaluja realistisen ja elävän maailman luomiseen, kuten reaaliaikainen valaistus usealla eri asetuksella, jolla saa luotua helposti haluamansa tunnelman pelimaailmaan, kasvillisuuden generointi, toimiva vesijärjestelmä ja tausta äänet Gaia tukee myös molempia grafiikkarajapintoja URP:ta ja HDRP:tä. (Procedural-Worlds 2023a)

Tässä projektissa kuitenkaan kokonainen pelattava maailma ei ole oleellinen osa, joten vain visuaalinen ulkoasu riittää ilman erillisiä objektien collidereita. Joten suurin osa Gaia:n toiminnallisuudesta, kuten kylien generointi jaa käyttämättä suorituskyvyn kannalta. Pelissä on muutenkin pieni alue, jonka pelaaja näkee niin tämän takia on fiksumpaa asettaa haluttavat ympäristöobjektit käsin tai Unityn omalla terrain-editorilla. Jos pelialue olisi tästä suurempi niin tällöin Gaian generoimat ympäristöobjektit olisivat käteviä.nopeuttamaan suunnitteluprosessia. Alla näet valikon, jossa asetetaan generoidun maaston parametrit, jossa määritetään kuinka paljon maastossa tulee olemaan yksityiskohtia (Kuva 4.). Koska peli suunnitellaan selaimella suoritettavaksi, niin yksityiskohdista pitää karsia pois, jotta suorituskyky pysyisi kohtuullisena.

Kuva 4 Gaia-maaston generointityökalusta kuva.



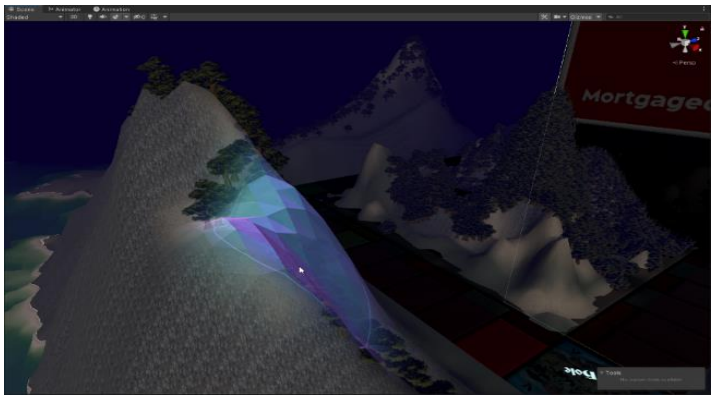
Heightmap, control ja basemap-tekstuuri resoluutioksi valitsin mobiili- ja selainpeleille suositellut arvot, jotta pelin suorituskyky pysyisi kohtalaisena heikoimmillakin laitteilla. Mitä pienempi resoluutio niin sitä vähemmän yksityiskohtaisempaa pelin maaston kivi ja hiekka tekstuurit ovat, mutta suorituskyky paranee. Mikä Gaian automaattisesti generoiduista maastoista tekee käytännöllisen tähän projektiin, on se, että niitä pystyy muokkaamaan jälkeenpäin Unityn omalla maastonmuokkaus työkalulla.

6.3.1 Unityn oma Terrain-editori

Terrain-editori tarjoaa myös työkalut nopean kasvillisuuden lisäämiseen ja muiden ympäristöobjektien, kuten kivien, puiden, puskien talojen asettamiseen. Näissä pitää kuitenkin pitää mielessä se, että mille alustalle peliä luodaan, jotta suorituskyky ei kärsisi liikaa. Tähän projektiin ei kuitenkaan sisällytetty ympäristöobjekteja, koska tarkoitus oli saada peli pyörimään heikommillakin laitteilla mahdollisimman sujuvasti. Mahdollisuus kuitenkin olisi lisätä asetusvalikko, josta pelaaja voi asettaa grafiikka-asetuksensa oman laitteensa vaatimustensa mukaisesti. Pienemmällä asetuksella nämä ympäristöobjektit katoaisivat kokonaan, kun taas korkeimmilla niitä olisi enemmän näkyvissä. (Unity 2022.3c)

Ideana peliin oli luoda vuoret pelilaudan ympärille, jotka ohjaavat pelaajan silmän keskittymään oikeaan paikkaan, eli pelilautaan. Tämä onnistui helposti ottamalla kiinni maastosta ja vetämällä tätä ylöspäin Terrain editorin-harjatyökalulla. Myös pelilaudan alaosa on tasoitettu mahdollisimman lähelle pelilautaa, jotta se antaisi pelaajalle vaikutelman, että lauta olisi osa maastoa. Vuoret eivät kuitenkaan saaneet olla liian korkeita tai liian lähellä keskustaa, jotteivat ne tulisi pelaajan kameran liikeradan tielle. (Kuva 5.)

Kuva 5 Harjatyökalusta Unity-Editorista. jolla kohotetaan vuoria ja lisätään kasvustoa.



6.3.2 Pelimaailman valaistus ja taivas Gaiaa käyttäen

Gaia tarjoaa käyttäjälle työkalut luodakseen peliin reaaliaikaisen valaistuksen. Valoasetusten mukana tuli myös sumunasetukset, jolla voi lisätä maailmaan sumua, muuttamalla sumun alkamis- ja lopetus arvoa.. Pelin sumukerros on asetettu siten, että se alkaa pelaajanäkymän vuorien takaa ja tihenee mitä kauemmas pelaajan kamerasta mennään. Tämä saa aikaiseksi sen efektin, että pelaajan pelimaailma olisi isompi mitä se oikeasti on.

Tavoitteenani oli luoda valaistus, joka saa pelilaudan erottumaan muusta maastosta ja tuomaan ruutujen tekstit mahdollisimman selkeästi esiin. Tämä tapahtui muuttamalla auringonvalon asetuksia lähelle valkoista, kääntämällä ”sun rotation”-slideriä, joka saa valon tulemaan oikeasta kulmasta vuorien takaa, sekä laittamalla auringon intentiseetti arvoksi 1.6. Tämä tuo pelimaailmaan kirkkautta tarpeeksi, jotta pelilaudalla olevat elementit, jotka heijastavat valoa, kuten tonttien nimet, värit ja kuvat näkyisivät selkeästi, mutta ei kuitenkaan liikaa, jotta laudan ja maaston yksityiskohdat eivät katoaisi. (Kuva 6.)

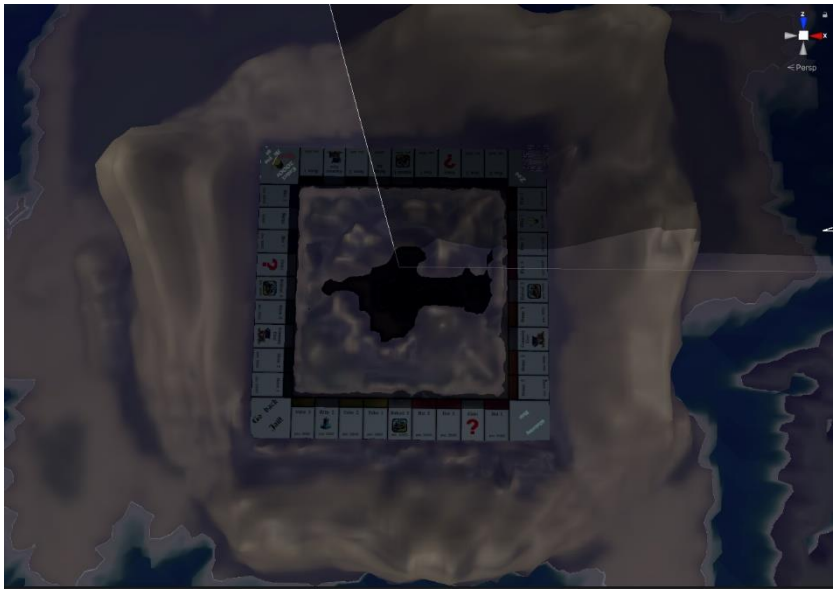
Kuva 6 Gaia-valaistus komponentin parametrit vasemmalla ja lopputulos oikealla.

Lopputuloksena on selkeä valaistus, joka tuo esiin laudalta ne elementit joihin käyttäjän pitää kiinnittää huomioita, kuten tonttien nimet, hinnat ja muut kuvakkeet. Kuvassa (Kuva 6.) näkyy myös aiemmin lisäämä sumu. Sumu peittää näköyhteyden vuorien taakse siististi lineaarisella menetelmällä. Jatkokehitystä ajatellen GAIA:n luomaan valaistus komponenttiin voisi esimerkiksi luoda helposti oman hallintaskriptin, joka hallinnoisi käytettävän valaistuksen arvoja ja muuttaisi ne pelaajan kellonajan mukaan. Tämä tarkottaisi sitä, että jos oikeassa elämässä on päivä niin tällöin myös pelissäkin olisi päivä.

6.3.3 Vedenpinnan luominen Gaialla

Vaikka vesi ei ole pelissä tärkeä elementti niin se on silti miellyttävä lisä, joka luo tunnelmaa. Veden generoiminen tapahtuu Gaian omalla vesiobjektissa olevalla Gaia Scene Water-komponentilla. Komponentti tarjoaa laajan valikoiman muuttujia veden ulkoasun sekä veden sisäisten efektien muokkaamiseen, kuten sukeltamiseen. Maasto on tehty vettä silmällä pitäen siten, että keski- ja sivuosissa maastoa on alennettu vedenpinnan alapuolelle asti, jotta vesi saadaan näkyviin. Vesiobjekti on tässä tilanteessa litteä planeobjekti, jossa on kiinni varjostin, joka saa aikaan veden ulkoasun. Gaian vesikomponentissa pystyy laajasti muokkaamaan veden toimintaa, kuten aaltojen korkeutta, määrää ja nopeutta, kuten myös veden kolmioiden määrää, jota lisäämällä saa vedestä yksityiskohtaisempaa. Mitä korkeamman laatuista vettä haluaa niin sitä enemmän vesikomponentti maksaa suorituskykyä. Vesikomponentin asetukset ovat asetettu suorituskykyä suosiviksi pienentämällä planeobjektin geometriaa ja tekemällä vesialueesta pienempi. (Kuva 7.)

Kuva 7 Maailma kuvattu ylhäältä päin. Keskellä ja sivuilla näkyvät vesialueet.



7 Pelin oleelliset luokat nettipeliä varten

Tämä luku käsittelee, miten luokkarakenteet on toteutettu projektiin, jotta saadaan aikaiseksi toimiva nettipeli. Newtonsoft-kirjasto sisältää useita hyödyllisiä funktioita JSON-viestien arvojen muuttamiseen ja tallettamiseen C#-attribuuteille ja kokonaisille luokille. Ennen kuin sukelletaan liian syvälle yksittäisiin luokkiin, niin on hyvä tietää miten tiedon lähettäminen ja saaminen tapahtuu. Peli käyttää Endelin luomaa NativeWebSocket-kirjastoa yhdistääkseen pelin palvelimeen. NativeWebSocket-kirjasto käytännössä toimii siten, että kun pelin käynnistää Unity kutsuu WebsocketManager.cs-skriptissä olevan Start-funktion, jossa luodaan yhteys Websocket-muuttujaan. Websocket-ws-muuttujan avulla voi lähettää viestejä ja saada niitä palvelimelta. Tämä tapahtuu siten, että websocket-muuttujalle luodaan tapahtumakuuntelija ’onMessage’. Se aktivoituu, joka kerta kuin palvelin vastaa pelaajan lähettämään pyyntöön.

7.1 WebsocketManager-luokka

WebsocketManager-luokan tehtävä on luoda yhteys palvelimelle ja pitää kirjaa aiemmista yhteydenotoista ja palvelimen vastauksista. Viestit tulevat palvelimelta JSON-muodossa mikä mahdollistaa helpon muuntamisen ja tallentamisen omiin luokkiin käyttäen Newtonsoftin JSON-kirjastoja. Websocket ws-muuttujan tapahtumakuuntelijan ’onMessage’ -sisällä on funktio HandleResponse-jota varten muunnetaan saadut tavutaulukon tavut string-muotoon ja nämä muunnetut tavut string-muodossa syötetään HandleResponse-funktioon, joka lopulta käsittelee ne.

Pelaajan ja palvelimen välinen kommunikaatio tapahtuu ws-socketin välityksellä, jossa pelaaja lähettää pelaajaluokan palvelimelle JSON-muodossa ja palvelin vastaa tähän pelaajan lähettämän string muotoisen connectAction-muuttujan mukaan..

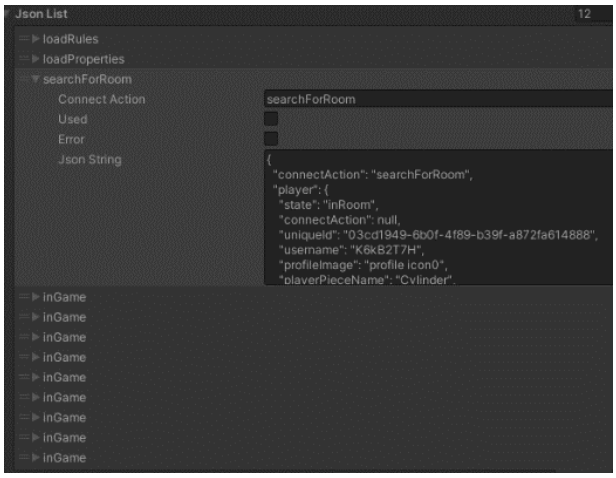
Pelaajaluokan rakenteesta löytyy connectAction-niminen string-muuttuja. Tämän muuttujan avulla palvelin tietää, että mitä sen pitää tehdä kunkin viestin perusteella. Näitä tilanteita voi olla esimerkiksi heitä noppaa, kiinnitä tontti, aloita peli, lopeta vuoro. Komentoja pelistä löytyy 34 + kehittäjä komennot, pelaajan ja palvelimen välisestä viestittelystä. Tätä pelaajan ja palvelimen välistä viestittely tapaa voi ajatella kuin chat-sovelluksena, mutta valmiiksi määritetyillä viesteillä. Tässä tapauksessa viestit ovat Newtonsoft-kirjaston string-muodoksi muuntamia pelaajaluokka-viestejä, josta jokaisesta connectAction-arvosta tapahtuu oma toimintoketju palvelinpuolella.

7.1.1 Pyyntölista

Pelin toiminta perustuu JSON-viestien käsittelyyn ja elementtien näyttämiseen saadun JSON-viestien perusteella. Tämän takia projektiin täytyi kehittää menetelmä pitää kirjaa pyynnöistä. Jokaisella pyynnöllä pelissä on oma ”connectAction”-nimike, jolla identifioidaan pyynnöt toisistaan. Nimike ’connectAction’ on kovakoodattu arvo jokaiselle palvelimella olevalle vastausfunktiolle, jonka palvelin palauttaa pelaajalle. Tämä on sen takia, että pyynnöt eivät sotkeutuisi keskenään, koska pyyntöjen sotkeutumisesta voisi seurata ongelmia pelin toiminnallisuuden kannalta missä oletetaan, että pyyntö ’X’ sisältäisi arvon ’B’. Jos jostakin syystä tapahtuu, että pyynnössä ’X* ei ole arvoa ’B’ niin tällöin pelaajan käyttöliittymässä toiminnot eivät päivity ja jotkut pelin toiminnot saattavat hajota.

Pyyntölistan (Kuva 8.) tehtävä on pitää kirjaa pelaajan ja palvelimen välisestä viesittelystä. Pelaajan pitää olla yhteydessä palvelimeen tasaisin väliajoin, jotta pelaajan ruudulla päivittyisi muiden pelaajien tekemät liikkeet, joka johtaa siihen, että pyyntöjä kertyy useita kymmeniä minuutissa ja tämä hidastaa pelin toimivuutta. Tämän takia pyynnön lisäämisfunktiossa ’JsonListAdd’ on tarkistus, jos pelaajan lähettämää pyyntöä ’X’ on 10 tai enemmän. Mikäli tämä käy toteen niin pyyntölistasta poistetaan kaikki ’X’ pyynnöt kyseisellä ’connectAction’-nimikkeellä ja vain tuorein, eli viimeinen pyyntö jätetään.

Kuva 8 Unityn hierarkiasta kuva pyyntölistasta



Pyyntölistalla on useampi tarkoitus pelissä toimivuuden kannalta. Vaikka aiemmin käsitellyssä HandleResponse-funktiossa talletetaan arvot suoraan pelaaja ja huonemuuttujaan JSON-skriptissä niin joissakin tapauksissa arvo joudutaankin etsiä suoraan JSON-viesteistä, koska pelissä on muutamia tapahtumia liittyen nopanheiton jälkeiseen tonttiin saamiseen. Tätä tontti muuttujaa ei talleteta HandleResponse-funktiossa vaan se etsitään suoraan pelaajan saaduista JSON-

viesteistä. Pyyntölista myös auttaa ja nopeuttaa sovelluksen kehittämisprosessia. Pyyntölistan JSON-arvot näkyvät suoraan Unityn omassa hierarkiassa WebsocketManager-peliobjektin komponentissa. Tästä oli suuri apu pelin kehittämisvaiheessa, koska mikäli ongelmia ilmeni eikä pelin käyttöliittymä päivittynyt oikeaan arvoon ws-pyynnön jälkeen, niin tällöin pyyntölistasta pystyi tarkistamaan, mikä arvon pitäisi olla missäkin päivitys vaiheessa.

7.1.2 HandleResponse-funktio

HandleResponse-funktion tehtävä on suoriutua jokaisen palvelimelta saadun vastauksen jälkeen ja käsitellä data Newtonsoft-kirjastoa käyttäen. Funktio käsittelee jokaisen vastauksen vastauksessa olevan connectAction-arvon mukaan ja tekee pelaajanäkymään muutokset. Nämä muutokset sitten tallennetaan JSON-luokassa olevaan pelaaja ja huoneobjektiin. Tämä tapahtuu siten, että pyyntölistan viimeisimmästä pyynnöstä katsotaan room-muuttujan arvo. Mikäli room-muuttuja on olemassa ja se sisältää pelaajia niin silloin funktio käy läpi jokaisen pyyntölistasta tulleen pelaajan ja päivittää pelaajapuolella muiden pelaajien arvot. Näitä päivitettyjä arvoja voidaan sitten helposti vertailla että mitä elementtejä pelaajan näytöllä pitäisi olla missäkin tilanteessa ja tehdä tarvittavat muutokset.

7.2 Room-luokka

Tässä vaiheessa on hyvä tietää, miten pelin palvelin toimii. Pelissä yksi palvelin voi sisältää monia huoneita. Nämä huoneet taas sisältävät pelaajia väliltä 1 – huoneeseen asetetun maksimi pelaajamärään mukaisesti. Huoneen tehtävä on pitää kirjaa pelaajista, huoneen tilasta, huoneen indeksi ja vuoronumerosta. Huoneluokassa (Ohjelmakoodi 2 Huone-luokka, joka sisältää pelin jokaisessa huoneessa olevan tiedon. on int-listat molemmille korteille, jotka sisältävät indeksi numerot käytetyistä korteista, jotta useampi pelaaja ei voi saada samoja kortteja. Ranking-listan tarkoitus on pitää kirjaa pelaajien sijoituksista. Nämä sijoitukset sitten näytetään pelaajalle sen mentyä konkurssiin kaavalla `Constants.MAX_PLAYERS – (i = paikallisen pelaajan indeksi)` Eli mitä aiempi indeksi taulukossa niin sitä huonompi sijoitus loppuruudussa.

Ohjelmakoodi 2 Huone-luokka, joka sisältää pelin jokaisessa huoneessa olevan tiedon.

```
public class Room
{
    public int maxPlayers = Constants.MAX_PLAYERS;
    public enum State { none, lobby, gameStarting, inGame };
    public string state;
    public int roomIndex;
    public int turnNumber; // index of current players turn in "players" list
    public List<int> usedCommunityChestCards = new List<int>();
    public List<int> usedChanceCards = new List<int>();
    public List<string> ranking = new List<string>();
    public List<Player> players = new List<Player>();
    …
}
```

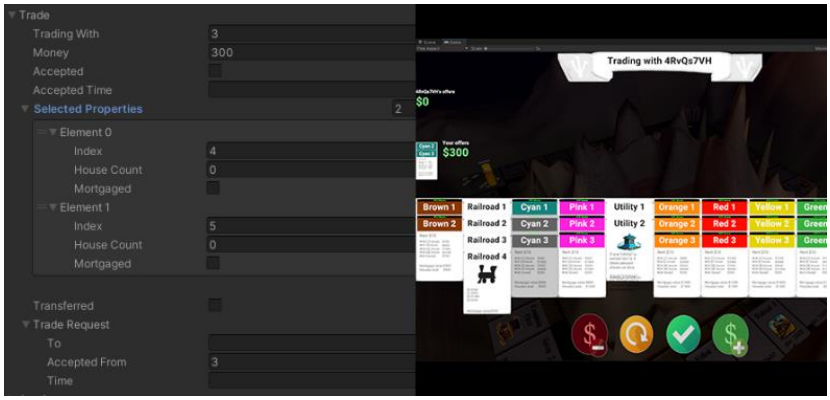
Jokaisella huoneessa olevalla pelaajalla on pääsy huoneluokan arvoihin. Pelaaja-luokassa on esimerkiksi roomIndex-niminen integer-kenttä, joka määritetään pelaajalle tämän liittyessään tai luodessaan uusi huone. Tämä on ratkaisuna palvelimelle paljon kevyempi, koska palvelimen ei tarvitse alkaa käydä läpi jokaista aktiivista huonetta ja vertailla pelaajien uniikkia-id arvoa muiden pelaajien kanssa vaan pelaaja itsessään kantaa tietoa omasta huonenumerosta. Palvelimella voi olla samanaikaisesti, jopa useita kymmeniä pelejä käynnissä. Pelaajat eivät voi kommunikoida huoneitten väliltä vaan jokainen pelissä oleva pelaaja saa vain tietoa siitä huoneesta ja huoneessa olevista pelaajista, jossa pelaaja on.

7.3 Player-luokka

Pelaajaluokka sisältää kaiken oleellisen pelissä olevien pelaajien tietojen säilyttämisen kannalta. Esimerkiksi alkuruudussa pelaaja pystyy itse määrittämään pelaajanimensä ja kuva-arvonsa. Pelaajaluokassa olevat arvot lähetetään palvelimelle samassa huoneessa olevien pelaajien luettavaksi, Tämä saa aikaa sen, että muut pelaajat pystyvät esimerkiksi näkemään pelaajan valitseman käyttäjänimen ja profiilikuvan, sekä muut toiminnot mitä pelaaja päättää tehdä.

Pelaajan pelin data on tallennettu luokkatiedostoon nimeltä CurrentGameData. Tämä alaluokka pitää sisällään kaiken datan, jota pelin sisällä käsitellään, kuten omistetut tontit, rahamäärä, pelaajan nykyinen ruutu, nopanarvot ja paljon muuta. Luokkaan kuuluu monia alaluokkia, kuten noppa, vuoro, vaihto, huutokauppa ja vankila. Nämä alaluokat taas sisältävät tietoa niistä pelivaiheista, jossa pelaaja on. Näitä pelivaiheita voi olla useampia kuten, vaihtokauppaa tekemässä (Kuva 9.), heittämässä noppaa, vankilassa. GameManager-skripti pitää tästä huolen, jotta pelaajalle näytetään oikeat valikot missäkin pelivaiheessa oman pelaajadatan ja huoneen tilan mukaisesti.

Kuva 9 Kahden pelaajan välinen vaihtokauppanäkymä. Vasemmalla näkyy pelaajan, trade-luokasta kuva, jossa pelaaja on asettanut kaksi tonttia ja rahaa vaihtonäkymään.



7.4 GameManager-skripti

GameManager-skriptin tehtävä on huolehtia pelin kulusta. Skriptissä on metodi nimeltä ConnectLoop, joka laukaistaan päälle pelin Start-metodissa. ConnectLoop-metodissa on silmukka, joka suoriutuu, joka requestTime-arvon välein. Tämän silmukan sisällä on 3 pientä silmukkaa, jolla on kukin oma tarkoitus.

7.4.1 Aula-silmukka

Aulasilmukan (Ohjelmakoodi 3.Ohjelmakoodi 2) tehtävä on suoriutua, kun huoneen state-arvo on asetettu arvoksi ’lobby’. Silmukan tehtävä on päivittää ja luoda liittyvien pelaajien 3D-pelinappulat laudalle sekä käyttöliittymäelementit näytölle, mikäli niitä ei ole vielä olemassa.

Ohjelmakoodi 3 Aulasilmukka

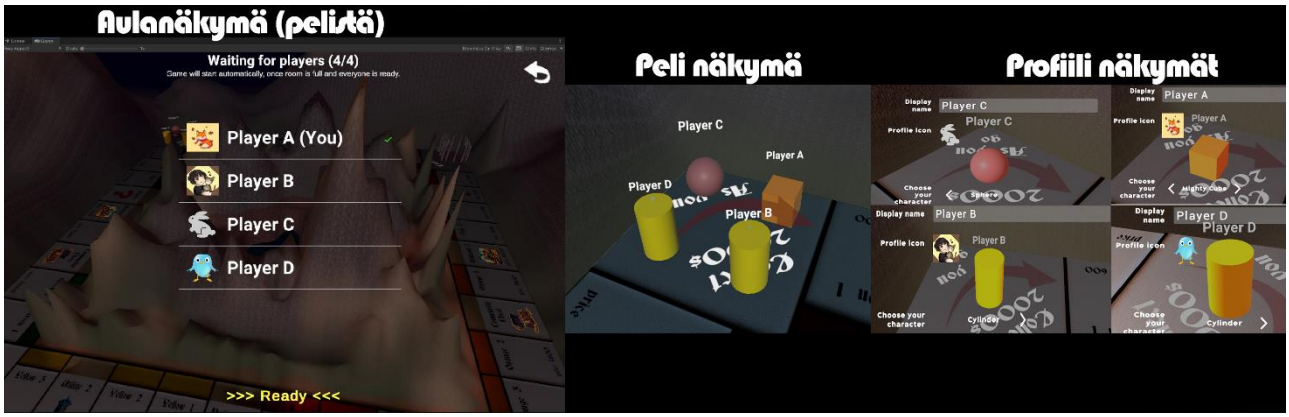
```
while (JSON.Instance.room.state == Room.State.lobby.ToString())
{
    WebsocketManager.Instance.Send(ConnectActions.waitInRoom);
    UI.mainMenu.waitingForPlayersScreen.UpdatePlayerUI();

    if (Player.WaitingInRoom.EveryoneReady(JSON.Instance.room))
    {
      WebsocketManager.Instance.Send(ConnectActions.gameStarting);
      while(!WebsocketManager.Instance.Wait(ConnectActions.gameStarting))
         yield return null;

          UI.mainMenu.waitingForPlayersScreen.UpdatePlayerUI();
       }
    InstantiatePlayerPieces();
    yield return new WaitForSecondsRealtime(WebsocketManager.Instance.requestRate);
```

Kuvassa (Kuva 10.) näkyy kuinka aulasilmukka päivittää pelaajan ’A’ huonenäkymää ja lisää pelaajat B, C ja D aulaan, jos näitä ei ole jo olemassa pelaajien valitsemiensa pelihahmojen perusteella.

Kuva 10 Vasemmalla aulanäkymä, keskellä pelinäkymä ja oikealla jokaisen pelaajan profiilinäkymä



Palvelinpuolelle on tehty myös metodi, joka tarkistaa milloin viimeksi pelaaja on ottanut yhteyttä palvelimelle. Mikäli pelaaja ei ole ottanut yhteyttä palvelimeen 15-sekunnin sisään nykyisestä kellonajasta niin palvelin poistaa pelaajan huoneesta, jolloin pelaajapuolella poistetun pelaajan huonenäkymä muuttuukin pelinaloitus näkymäksi. Tällä menetelmällä saadaan eliminoitua ne pelaajat pois huoneesta, jotka sulkevat selaimen suoraan ilman, että "ws onClose" -tapahtumakuuntelija suoriutuu.

7.4.2 InGame-silmukka

Ingame-silmukan tehtävä on suoriutua, kun huoneen state-arvo on asetettu arvoksi ’inGame’. Silmukan tehtävä on näyttää pelaajalle oikeat käyttöliittymäelementit tämän pelaajaluokan sisältävästä CurrentGameData-luokan arvoista. Silmukka pitää myös huolen laudalla tapahtuvista muutoksista kuten, pelaajien kiinnityksistä, sekä talojen ostamisista ja myymisistä. Näihin muutoksiin voi esimerkiksi kuulua, että pelaaja ’A’ lsähettää kauppapyynnön pelaajalle ’B’. Jolloin inGame-silmukan sisällä oleva metodi luo uuden ilmoituksen pelaajan ’B’ ruudulle, jolle pyyntö lähetettiin. Tämä käytännössä toimii siten, että pelaaja kuuntelee muiden pelaajien TradeRequest-alaluokan to-arvoa, jos tämä arvo on sama kuin pelaajan uniikki ID-arvo niin tällöin pelaaja saa ilmoituksen. (Ohjelmakoodi 4.)

Ohjelmakoodi 4 Luo ilmoituksen pelaajan näytölle, joka saa kauppapyynnön

```
if (playerWhoTradedMeuniqueID != "")
{
   Player _player = JSON.Instance.GetPlayerWithUniqueID(playerWhoTradedMeuniqueID);
   NotificationManager.Instance.Notification("Player " +
JSON.Instance.GetPlayerWithUniqueID(playerWhoTradedMeuniqueID).username + " wishes to trade
with you!", _player.profileImage, NotificationManager.NOTIFICATION_TIME,
playerWhoTradedMeuniqueID, NotificationMessageType.tradeRequest);
 }
```

inGame-silmukan sisällä laukaistaan kaikki pelissä olevat mahdolliset tapahtumat, josta pelaaja tarvitsee näytölleen päivityksen. Näitä voi olla esimerkiksi pelaajien raha-arvojen päivittäminen, vaihtokauppanäkymän muiden pelaajien arvojen päivittäminen tuoreimman JSON-viestin mukaiseen arvoon.

7.5 PlayerPieceController-skripti

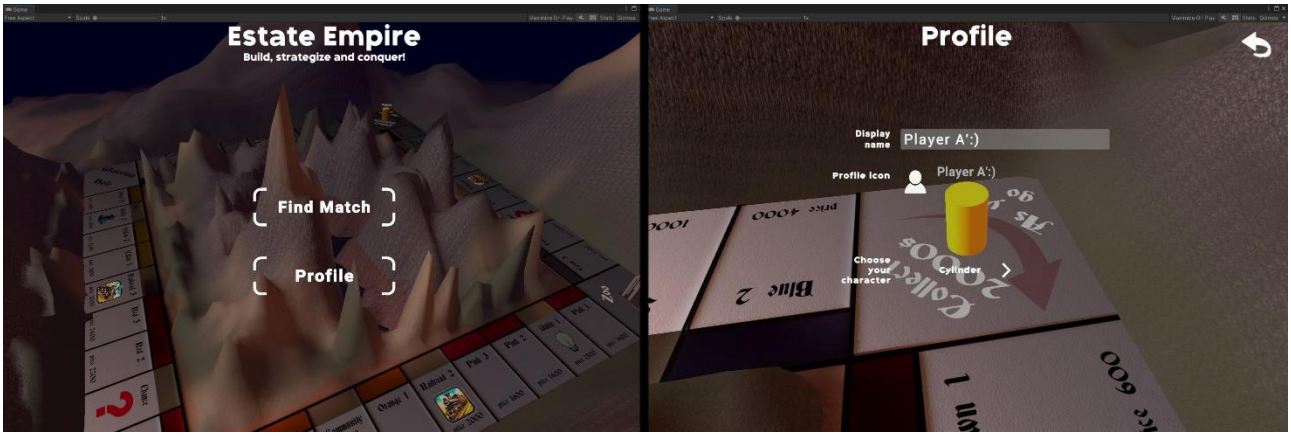
PlayerPieceControllerin tehtävä on liikuttaa pelaajan nappulaa oikeaan ruutuun pelaajan CurrentGameData-luokassa olevan square-arvon mukaan. Jokaisella laudalla olevalla ruudulla on oma indeksiarvo ruudut taulukossa (Kuva 11.). Nämä indeksit määrittävät sitten, pelaajan sijainnin laudalla. Liikkuminen tapahtuu siten, että pelaaja heittää noppaa. Palvelin arpoo pelaajalle arvon 2-12 väliltä. Pelaajan nykyiseen ruutuarvoon lisätään nopansumma, jolloin pelaajan nykyinen ruutuarvo ei käy toteen PlayerPieceController-luokassa olevan ’nykyinenRuutu’-arvon kanssa. Tällöin MovePiece-metodi aktivoituu ja liikuttaa pelaajaa eteenpäin.

Kuva 11 Pelilaudan ruutuarvot ruutulistassa

Jotta liikkuminen tapahtuisi sulavasti yksi ruutu kerrallaan niin pelaajan ’nykyinenRuutu’-arvoa nostetaan yksi kerrallaan ylöspäin. Tämän jälkeen skripti liikuttaa pelaajaa kohti ruutua `nykyistä ruutua + 1` ja odottaa niin kauan, kuin pelaajan ja uuden ruudun välinen etäisyys on pienempi kuin `0.05f`. Kun näin tapahtuu niin skripti nostaa taas ’nykyinenRuutu’-arvoa yhdellä niin kauan kuin nykyisen ruudun ja ’targetSquare’-arvot kohtaavat. Jos ’nykyinenRuutu’-arvo menee ruutulistan pituuden yli niin silloin arvo asetetaan takaisin 0.

7.6 CameraRotate-skripti

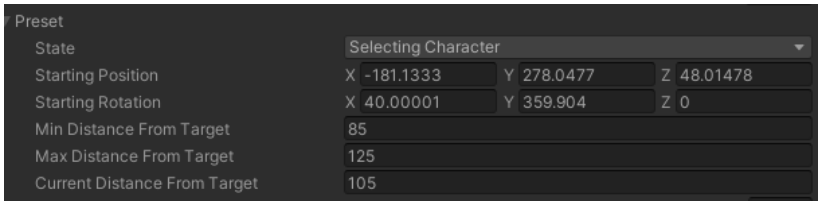
Kameraskriptin tehtävä on mahdollistaa kameran sulava liikuttaminen pelaajalle hiirtä käyttäen. Kameran liikeradan suunnitelussa piti ottaa huomioon ympäristö. Kamera on suuniteltu kiertämään kehää käyttäjän valitsemalla pituudella target-objektista. Tämä target-objekti voi vaihtua, joko pelilaudaksi tai pelaajan valitsemaksi hahmoksi, riippuen siitä, että kuinka lähelle pelaaja on zoomannut kameransa. Kamera on rakennettu niin, että muut skriptit voivat kontrolloida sitä tarvittaessa. Esimerkiksi pelaajan avattaessa profiilinäkymä kamera asettaa target-objektiksi pelaajan valitsemansa nappulan kuvan (Kuva 12.) mukaisesti, kohdentaen kameran pelaajan valitsemaansa pelinappulaan.



Kuva 12 Vasemmalla päävalikko. Oikealla profiilivalikko

Kameran sulavan liikuttamisen toimintaan saamiseen kameran esiasetusluokaan lisättiin kullekkin kamera-asetukselle omat perusarvot, jotka määrittävät sen, että kuinka pitkälle pelaaja pystyy rullaamaan kameraa. Presetit auttavat nopeasti tekemään uusia kamera-asetuksia peliin. Tällä hetkellä pelissä on 4 eri kamera-asetusta, jotka ovat nimeltään hahmon valinta, normaali tila, vankilan sisällä ja pelaaja liikkumassa. (Kuva 13.)

Kuva 13 Kamera "Selecting Character" vakioasetus arvoista kuva.



Kameran liikkuminen toimii seuraavalla laskutoimituksella (Ohjelmakoodi 5.). Ensin katsotaan, että ylittääkö nykyinen kameran etäisyys päävalikossa asetetun aktiivisen vakioasetuksen määrittämän `(maxDistanceFromTarget * cameraSwitchPercentage)`-arvon. Sen jälkeen tarkistetaan, että pelinappula ei liiku. Jos nämä kriteerit käyvät toteen niin target-objektiksi valitaan pelilauta, muussa tapauksessa pelaajan valitsema pelinappula.

Ohjelmakoodi 5 Mahdollistaa kameran liikuttamisen

```
if (distanceFromTarget > (preset.maxDistanceFromTarget * cameraSwitchPercentage) &&
!GameManager.Instance.playerPieceController.moving && preset.state == CameraState.normal)
this.transform.position = Vector3.Lerp(this.transform.position, target.position -
transform.forward * distanceFromTarget, currentCameraLerpSpeed * Time.deltaTime);
else
this.transform.position = Vector3.Lerp(this.transform.position,
GameManager.Instance.playerPiece.transform.position - transform.forward * distanceFromTarget,
currentCameraLerpSpeed * Time.deltaTime);
```

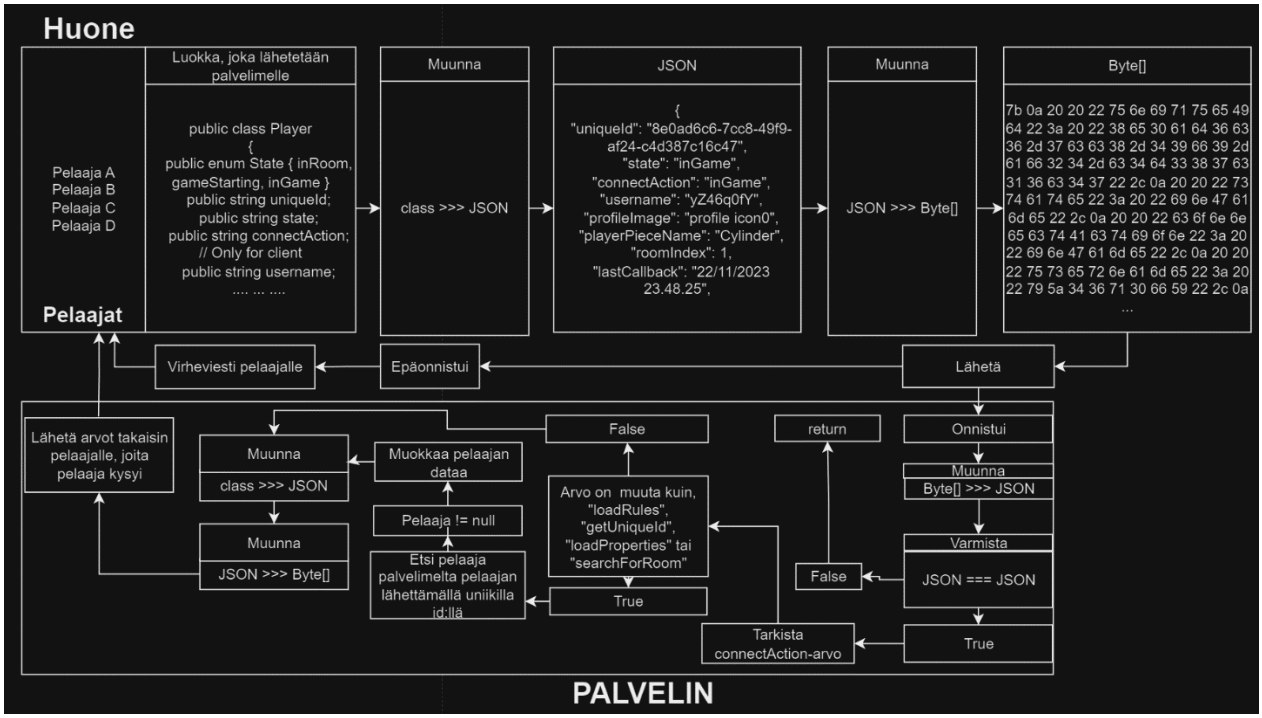
Kameran arvoja voi helposti säädellä Unityn omasta hierarkiasta, kun logiikka on kirjoitettu. Tämä mahdollisti myös kameran eri etäisyyksien testaamisen kehittämisvaiheessa ja optimaalisen etäisyyden löytämiseen. Jokaisella arvolla on oma merkitys kameran liikeradassa. Alla on kirjattu skriptin päämuuttujat, jotka ovat vastuussa kameran liikkumisesta.

distanceFromTarget:	Kameran ja ‘target’-objektin välinen etäisyys.
smoothTime:	Kameran sulavaan kääntämiseen liittyvä arvo.
cameraSpeed:	Arvo, joka määrittää kuinka nopeasti kamera kääntyy.
cameraAngle:	Kulma, jossa kamera pysyy target-objektiin nähden.
currentCameraLerpSpeed:	Arvo joka vaihtuu kameraa kääntäessä. Määrittää kameran kiihtyvyyden ja kääntönopeuden.
maxCameraLerpSpeed:	currentCameraLerpSpeed:in maksimi arvo.
minCameraLerpSpeed:	currentCameraLerpSpeed:in minimi arvo.
cameraLerpSpeedIncrement:	Arvo, joka määrittää kuinka nopeasti currentCameraLerpSpeed-muuttuja nousee. Eli kameran kääntövauhti kiihtyy sitä kääntäessä.
cameraSwitchPercentage:	Float-arvo väliltä (0.00 – 1.00). Määrittää kameran kohdennus objektin.

8 Pelaajien välinen viestintä

Pelaajien välinen viestintä tapahtuu palvelimen välityksellä aktiivisella websocket-yhteydellä, jossa pelaaja lähettää palvelimelle pelaajaobjektin byte-taulukko muodossa. Tämä prosessi tapahtuu siten, että pelaajanäkymässä pelaajaobjekti ensiksi muutetaan JSON-muotoon käyttäen Newtonsoft-kirjastoa. Tämän jälkeen JSON-muodossa oleva tieto muutetaan Byte-taulukko muotoon ennen tiedon lähetystä varten. Palvelinpuolella palvelin aloittaa varmistamalla, että data on tullut oikeassa muodossa muuntamalla pelaajalta saadun datan JSON-muotoon. Mikäli tämä prosessi onnistuu niin palvelin jatkaa tarkistamaan, että mikä pelaajan lähettämän ”connectAction”-muuttujan arvo on. Jos arvo vastaa switch-lausekkeissa olevia arvoja niin palvelin suorittaa näille ohjelmoidut toimenpiteet palauttaen pelaajalle dataa nykyisestä huoneesta tai muokaten sitä ennen palauttamista. (Kuva 14.)

Kuva 14 Pelaajan ja palvelimen välisestä viestinnästä kaavio alustetulla Websocket-yhteydellä..



8.1 Yhdistäminen ja tiedon lähettäminen

Yhteyden luominen tapahtuu luomalla uusi WebSocketServer-muuttuja ja alustamalla se palvelimen julkisella IP-osoitteella ja portilla. Jotta muut pelaajat pystyisivät yhdistämään peliin niin palvelinkoneella täytyy olla palomuuriasetukset kunnossa, jotta palomuuri hyväksyy tulevat ja lähtevät TCP-yhteydet pelin palvelimeen asetetusta portista. Pelaajan ja palvelimen välinen yhteys

luodaan lausekkeella (Ohjelmakoodi 6.). Pelissä pelin sisäisen datan tiedonsiirtona käytetään ei salattua tiedonsiirtomenetelmää eli `ws://`, koska pelaajaobjekti lähettää vain sellaista tietoa itsestänsä muille pelaajille mikä ei ole arkaluonteista ja on muutenkin jo näkyvissä muille pelaajille käyttöliittymäpuolella. Eri asia tässä olisi se, jos pelaajan nykyinen sijainti laudalla olisi salainen, että muitten pelaajien ei pitäisi esimerkiksi tietää, että missä pelaaja menee. Niin tällöin olisi tärkeää, että arvot salattaisiin, jotta huijausmenetelmien kehittäminen peliin olisi vaikeampaa.

Ohjelmakoodi 6 Palvelin- sekä pelaajapuolen yhteyden alustaminen.

```
// Palvelinpuolella. Palvelin laitteen Ipv4-osoite.
var server = new WebSocketServer("ws://192.168.100.11:7777");

// Pelaajapuolella.
ws = new WebSocket("ws://91.155.60.85:7777/game");
```

Palvelimen ja pelaajan välinen viestintä tapahtuu siten, että pelaaja lähettää viestin palvelimelle ja palvelin vastaa tähän pyyntöön niin onMessage-niminen tapahtumakuuntelija (Ohjelmakoodi 7.) suoriutuu ja palauttaa palvelimelta saadun arvon.pelaajalle tavutaulukkona. Tavut sitten muunnetaan ihmiselle luettavaan muotoon Encoding-kirjaston avulla.

Ohjelmakoodi 7 Tapahtuma kuuntelijoiden luominen.

```
ws.OnOpen += () =>
{
   Debug.Log("Connection open!");
   connectionReady = true;
};

ws.OnError += (e) =>
{
    Debug.Log("Error! " + e);
    connectionReady = false;
    ReConnect();
 };

 ws.OnClose += (e) =>
 {
    connectionReady = false;
    ReConnect();
    Debug.Log("Connection closed!");
 };

 ws.OnMessage += (bytes) =>
 {
    string responseText =
        System.Text.Encoding.ASCII.GetString(bytes);
        HandleResponse(responseText);
```

Jokaisella näistä 4 tapahtumakuuntelijasta on oma toiminnallisuus..

```
ws.OnOpen:      Suoriutuu, kun pelaaja avaa yhteyden palvelimelle.
ws.OnError:     Jos yhteyttä ei voida luoda, näyttää virhe viestin.
ws.OnClose:     Yhteyden katkaistaessa suoriutuu.
ws.OnMessage:   Suoriutuu, kun pelaaja saa viestin palvelimelta.
```

Ws.OnMessage-tapahtumakuuntelijan sisällä on funktio nimeltä HandleResponse, johon syötetään string-muodossa parametrina saatu vastaus palvelimelta. HandleResponse funktio käsittelee datan ja tallettaa sen pelaajan JSON-luokkaan, josta pelaajanäkymä lukee arvoja ja tekee valintoja näiden perusteella.

Tiedon tallettaminen noutovaiheessa helpottaa tiedonkäsittelyä. Muuten pelaajapuolella tiedon joutuisi lukemaan suoraan pyyntölistan JSON-viesteistä, mikä tekee pelinkehittämisestä hitaampaa ja sekavampaa. Esimerkkinä tästä on InGame-silmukan sisällä oleva else if-lauseke (Ohjelmakoodi 8.) . Käsittelemättömän tiedon käsittelyssä tekee vaikeaa sen, että tieto tulee palvelimelta huoneluokka-muodossa, joka tarkoittaa sitä, että ennen kuin voimme alkaa vertailla arvoja niin palvelimelta saadusta huonearvon pelaajalistasta pitää ensimmäisenä löytää pelaajalle kuuluva pelaajaobjekti.

Ohjelmakoodi 8 Havainnollistaa käsitellyn tiedon käytön verrattuna käsittelemättömään tietoon.

```
// Käsitelty tieto
else if (player.currentGameData.trade.tradingWith != "")

// Sama tieto käsittelemättömänä.
else if
(WebsocketManager.Instance.GetMyPlayer(WebsocketManager.Instance.GetJsonLatest())[“currentGame
Data][“trade”][“tradingWith”].ToString() != “”)
```

8.2 Pelaajien tunnistaminen

Yhdelle palvelimelle voi yhdistää monia satoja pelaajia samanaikaisesti, käyttäen samoja käyttäjänimiä. Tämän takia on tärkeää, että palvelin antaa pelaajalle liittymisvaiheessa oman uniikin merkkitunnisteen, jonka avulla pelaajan voi helposti löytää monien muiden palvelimella olevien pelaajien joukosta. Kaikki pelissä olevat muuttuvat arvokentät kuten muille pelaajille luodut käyttöliittymäelementit erotellaan toisistaan uniikilla ID:llä. Tämän toteuttamiseen käytin Microsoftin tarjoamaa GUID-kirjastoa (Ohjelmakoodi 9.), joka on lyhenne sanoista ’Globally Unique Identifier’. Kirjaston tehtävä on generoida kokonaan uniikki merkkijono, jota pelissä käytetään pelaajien tunnistamiseen toisistaan. Sillä pystyy generoimaan 128-bittisiä merkkijonoja ja mahdollisuudet siihen, että se palauttaisi 2 samaa merkkijonoa ovat äärinmäisen pienet (2 ^128). (Microsoft GUID)

Ohjelmakoodi 9 Funktio palauttaa uniikin-ID:n pelaajalle.

```
public static string GetUniqueID()
{
   Guid uniqueId = Guid.NewGuid();
   return uniqueId.ToString();
}
```

Palvelinpuolella pelaajan uniikkia ID:tä käytetään seuraavasti. Jokaisen pelaajan palvelimelle lähettämän viestin jälkeen palvelin suorittaa funktion nimeltä FindPlayer (Ohjelmakoodi 10.). Funktion tehtävä on löytää pelaaja palvelimelta ja palauttaa se. Tämä on tärkeä vaihe siihen, jotta aiempaa pelaajadataa voidaan muokata ja korvata aiempi pelaajan data muokatulla datalla. FindPlayer-funktio toimii seuraavasti. Funktio ottaa parametrina string-muuttujan ’uniqueId’. UniqueId on pelaajan lähettämä ja palvelimen aiemmin luoma uniikki identifioitu merkkijono tälle pelaajalle. Pelaajalle luodaan tämä merkkijono vain huoneeseen liittymisen yhteydessä, joten pelaaja-objekti kantaa sitä tästä eteenpäin mukana niin kauan kunnes pelaaja katkaisee yhteyden palvelimeen.

Ohjelmakoodi 10 FindPlayer-funktio, joka hoitaa oikean pelaajan löytämisen huoneista.

```
static Player FindPlayer(string uniqueId)
{
   if (json["roomIndex"] != null)
   {
       int roomIndex = int.Parse(json["roomIndex"].ToString());
       if (roomIndex != -1)
       {
          for (int j = 0; j < rooms[roomIndex].players.Count; j++)
          {
             if (rooms[roomIndex].players[j].uniqueId == uniqueId)
                         return rooms[roomIndex].players[j];
           }
       }
   }
   for (int i = 0; i < rooms.Count; i++)
      for (int j = 0; j < rooms[i].players.Count; j++)
      {
         if (rooms[i].players[j].uniqueId == uniqueId)
            return rooms[i].players[j];
       }
   return null;
}
```

9 Pelin päätoimintojen sekä näkymien luominen

Peli koostuu useasta eri näkymästä joita näytetään tai piilotetaan pelaajalle riippuen palvelimelta saaduista arvoista. Tätä prosessia kontrolloi pelin pääsilmukka joka pyörii GameManager-skriptissä. Pelin palvelin tiedon lähettämis- ja vastaanottamis- systeemi on rakennettu siten, että kaikki arvot, jotka halutaan näyttää muille pelaajille pitää tallentaa pelaajapuolella JSON-objektissa olevaan pelaajamuuttujaan tiedon lähettämistä varten. Palvelin käsittelee nämä tiedot ja lisää ne palvelinpuolella pelaajalistassa olevaan pelaajaobjektiin uniikin ID:n avulla. Pelissä on useita eri näkymiä ja toimintoja, jonka avulla käyttäjä pystyy muokkaamaan omaa dataansa, kuten nopan heittäminen, vuoron lopettaminen, tonttien kiinnitys aukaiseminen sekä talojen rakentaminen ja pelaajien vaihtokauppanäkymä, jossa pelaajat voivat vaihtaa keskenään tontteja ja rahaa.

9.1 Tonttien kiinnitys ja avaamis- sekä talojen myynti ja osto -näkymä

Pelin toiminnallisuuksiin kuuluu, että käyttäjällä on mahdollisuus kiinnittää ja avata tontteja, sekä ostaa tai myydä taloja omistamillensa tonteillensa. Näihin kuuluu myös säädöksiä, kuten pelaajan pitää omistaa kaikki samaa väriryhmää edustavat tontit tai, että jokin näistä tonteista ei saa olla kiinni, jotta taloja pystyttäisiin rakentamaan.

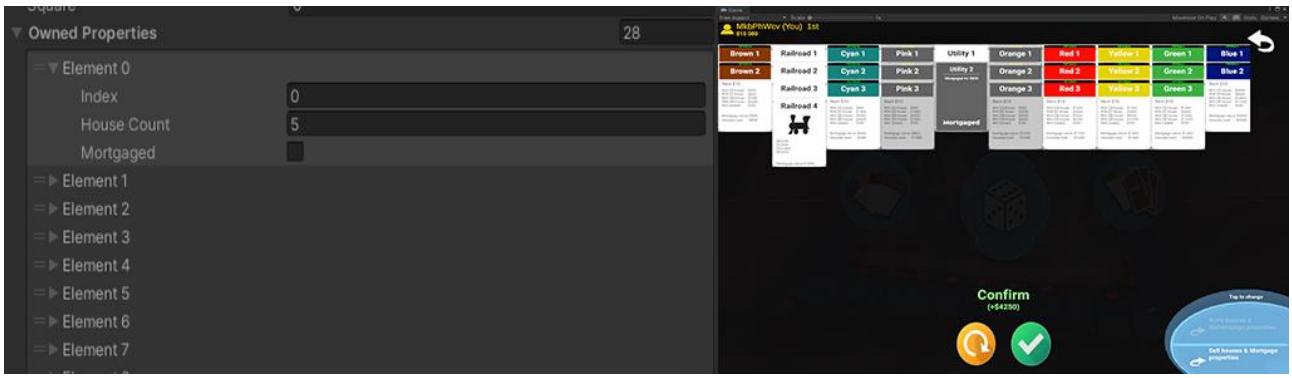
Tämä käytännössä toimii siten, että pelissä jokaisella tontilla on oma indeksi arvo, sillä tonttien indeksi arvot määräävät sen, että mitkä tontit pelaajalle näytetään. Tonttien visuaalisen näyttämisen pelaajalle hoitaa LoadProperties-luokka, johon on tehty funktiot tonttien UI-elementtien rakentamiseen annettujen parametrien perusteella. Jokaisella pelissä olevalla tontilla on oma indeksi-arvo valmiiksi määritellyssä tonttitaulukossa ja jokaisella tonttilla on nimi, väri, hinta, vuokra, talojen vuokrataulukko, kiinnitysarvo sekä talojen rakennushinta. Näkymän päätoiminnallisuuksiin kuuluu, että pelaaja voi tarvittaessaan kiinnittää sekä avata tontteja, rakentaa tai myydä taloja. Näkymä on suunniteltu siten, että sitä voi pelkästään hallita hiirellä tai kosketusnäytöllä vaihtamalla oikeassa alakulmassa olevan nappulan tilaa, joka mahdollistaa talojen ja tonttien kiinnittämisen- / avaamisen sekä talojen myymisen ja rakentamisen.

9.1.1 Käyttöliittymäpuolen toiminnallisuus

Pelissä jokaisella tontilla on oma indeksi arvo, sillä tonttien indeksi arvot määräävät sen, että mitkä tontit pelaajalle näytetään. Tonttien visuaalisen näyttämisen pelaajalle hoitaa LoadProperties-luokka, johon on tehty funktiot tonttien UI-elementtien rakentamiseen annettujen parametrien perusteella. Jokaisella pelissä olevalla tontilla on oma indeksi-arvo valmiiksi määritellyssä tonttitaulukossa ja jokaisella tonttilla on nimi, väri, hinta, vuokra, talojen vuokrataulukko, kiinnitysarvo sekä talojen rakennushinta. Näkymän päätoiminnallisuuksiin kuuluu, että pelaaja voi tarvittaessaan kiinnittää sekä avata tontteja, rakentaa tai myydä taloja. Näkymä on suunniteltu siten, että sitä voi hallita pelkästään hiirellä tai kosketusnäytöllä vaihtamalla oikeassa alakulmassa olevan nappulan tilaa, joka mahdollistaa talojen ja tonttien kiinnittämisen- / avaamisen sekä talojen myymisen ja rakentamisen.

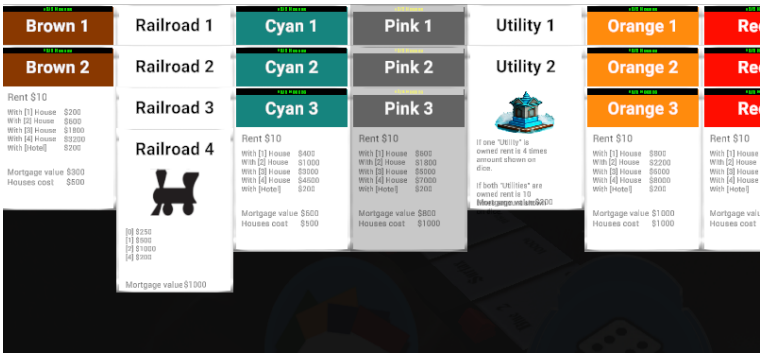
Monopolissa on sääntönä, että ei voi rakentaa tai myydä taloja kahden erolla, sekä ei voi ostaa taloja, jos yksi tonteista on kiinni, jotka piti ottaa myös huomioon hyväksymis- ja laskentafunktiota suunnitellessa. Näkymän laskurin arvo päivittyy automaattisesti ilman erillisiä viestejä palvelimelle, jotta käyttäjä ei pystyisi yli kuormittamaan palvelinta klikkailemalla nopeasti tontteja auki ja kiinni. Tämä lähestymistapa tarjoaa myös responsiivisemman kokemuksen käyttäjille, koska käyttäjät näkevät heti muutokset ilman palvelinpuolen viivettä. Tämä tietenkin vaatii sen, että tonttien arvot ovat ladattu valmiiksi jo pelaajapuolelle palvelimelta, mikä tapahtui pelin käynnistys vaiheessa sekä sen, että laskenta funktio on luotu molemmille pelaaja- ja palvelin puolelle arvojen validointia varten. (Kuva 15.)

Kuva 15 Vasemmalla pelaajan OwnedProperties-luokka, jonka avulla käyttöliittymä tietää, että mitä tontteja pelaajalle pitää näyttää. Oikealla pelaajan kiinnitys-/ talojen rakennus-/myynti -käyttöliittymänäkymästä kuva

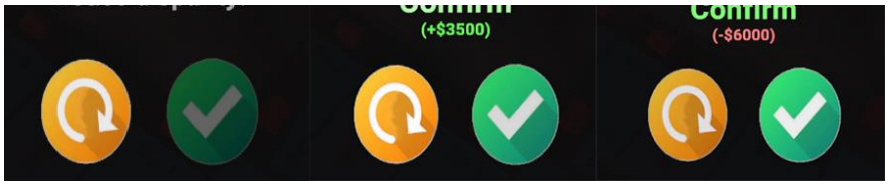


Jokaisella instantisoidulla tontilla on tapahtumakuuntelija (Kuva 16.), joka mahdollistaa tontin ja pelaajan välisen vuorovaikutuksen. Pelaaja pystyy valita haluamansa tontit tonttilistaan, joka lähetetään palvelimelle lopullista hyväksymistä varten. Tätä ennen kuitekin käyttöliittymäpuoli pitää huolen siitä, että pelaaja on tehnyt muutokset sääntöjen mukaisesti ja antaa pelaajan vasta sitten painaa hyväksymisnäppäintä. (Kuva 17.)

Kuva 16 Pelaaja on muokannut kolmea tonttiä kiinnitys näkymässä, mutta ei ole vielä hyväksynyt muutoksia.



Kuva 17 ToggleProperty-funktion asettamista arvoista kuva, käyttäen apunaan laskentafunktioita.



Näkymässä keskellä alhaalla on arvo kunika paljon rahaa pelaaja tulee menettämään tai saamaan rahaa valitsemistansa muutoksista. Tämän osuuden hoitaa laskurifunktio, joka on luotu molemmille puolille (pelaaja ja palvelin) arvojen validointia varten. Laskentafunktion (Ohjelmakoodi 11.) tehtävä on palauttaa rahallinen arvo integer-muodossa pelaajalle tonttien valitsemisen yhteydessä. Kun pelaaja avaa tonttien kiinnitys- ja avausnäkymän, niin LoadProperties-luokka rakentaai tontit pelaajan näkymään asettaen jokaiselle tontille nimen kyseisellä kaavalla: `(propertyColor:green; propertyOrder:1; propertyIndex:22; mortgaged:0; propertyName:Green 1;) Green 1.` Jotta arvot saadaan päivittymään responsiivisesti niin alkuperäiset eli muokkaamattomat arvot pitää tallentaa väliaikaisesti muistiin ja tämän avulla voidaan laskea ero nykyisestä tilasta verrattuna uuteen tilaan. Lausekkeessa ei kuitenkaan ole talojen määräarvoja. Tämä johtuu siitä, että ne ovat tallennettu tontin lapsiobjektin, jonka arvo näyttää tältä: `+5/5 Houses.` Arvolla on kaksi käyttötarkoitusta pelissä. Ensimmäinen on visuaalinen puoli, eli näyttää pelaajalle nykyisen ja alkuperäisen talojen määrän. Toinen on, että

käyttää laskentafunktiossa näitä kahta lukua - nykyistä ja alkuperäistä - laskeakseen pelaajalle kuuluvan lopullisen rahasumman valintojen jälkeen.

Ohjelmakoodi 11 Laskentafunktio

```
public int SelectedMortgageListValue(RectTransform[] properties)
{
      JSON.Instance.GetPlayer().currentGameData.mortgagePropertyScreen.mortgageMoney = 0;
      float total = 0;

      foreach (RectTransform property in properties)
      {
         int propertyIndex = Str.AttributeToInt(property.name, "propertyIndex");
         bool propertyMortgaged = Utility.PickRectTransformFromRectTransform(property,
      "Mortgaged").localScale == new Vector3(1, 1, 1);

         TextMeshProUGUI textMortgagedHouseCount =
      Utility.PickTextMeshProUGUIFromTransformChildren(property, "Text - Mortgaged House Count");
         int maxHouseCount = -1;
         int currentHouseCount = -1;
         if (textMortgagedHouseCount != null)
         {
            maxHouseCount = Str.SplitToIntBetween(textMortgagedHouseCount.text, '/', ' ');
            currentHouseCount = (Str.SplitToInt(textMortgagedHouseCount.text, '/', 0));
          }

          foreach (CurrentGameData.OwnedProperties op in
          JSON.Instance.GetPlayer().currentGameData.ownedProperties)
          {
             if (propertyIndex == op.index)
             {
                if (currentHouseCount != -1)
                {
                   // Player building houses
                   if (currentHouseCount > op.houseCount)
                           total -= (currentHouseCount - op.houseCount) *
                           Property.properties[op.index].housePrice;

                    // Player selling houses
                    if (currentHouseCount < op.houseCount)
                           total += (op.houseCount - currentHouseCount) *
                           (Property.properties[op.index].housePrice / 2);
                     }

                     // Player mortgaged property
                     if (propertyMortgaged && !op.mortgaged)
                         total += Property.properties[propertyIndex].mortgageValue;

                     // Player unmortgaged property
                     if (!propertyMortgaged && op.mortgaged)
                           total -= Property.properties[propertyIndex].mortgageValue +
                           (Property.properties[propertyIndex].mortgageValue * 0.1f);
                      }
                  }
           }

           return Mathf.RoundToInt(total);
      }
```

9.1.2 Palvelinpuolen toiminnallisuus

Toimivan käytöliittymän jälkeen piti luoda ratkaisu saada pelaajan valitsemansa tiedot palvelimelle, jossa palvelin pystyisi tekemään lopullisen päätöksen pelaajan haluamista muutoksista ja palauttaa uudet arvot. Tälle prosessille kehitin oman alaluokan näkymän nimen mukaisesti pelaajan CurrentGameData-luokkaan nimeltä ”MortgagePropertyScreen. MortgagePropertyScreen-alaluokan tehtävä on pitää kirjaa pelaajan valitsemistansa tonteista listamuodossa. Kun pelaaja viimein hyväksyy muutokset niin tieto lähetetään palvelimelle, jossa palvelin tekee lopulliset

muutokset pelaajan omistamiinsa tontteihin muokaten näitten kiinnitys sekä taloarvoja ja vähentääkseen tai lisätääkseen rahaa riippuen pelaajan lähettämistä arvoista.

Luokan rakentajan sisällä oleva logiikka alkaa siten, että luodaan silmukka, joka käy läpi jokaisen pelaajan valitseman tontin. Kummastakin listoista ”ownedProperties” sekä ”selectedProperties” otetaan tontin indeksiarvot varmistusta varten, että pelaaja omistaa valitsemansa tontit. Mikäli tämä käy toteen niin varastoidaan tontin vanha sekä uusi arvo ”ownedProperties” ja mortgageList”-listoista ja tutkitaan näiden kahden eroavaisuudet talojen määrissä ja kiinnitysarvoissa. Mikäli eroavaisuuksia löytyy niin ”total”-muuttujaan lisätään tontin talonhinta tai kiinnitysarvo, jotka haetaan palvelimen properties-taulukosta tontin indeksin perusteella. Lopulta total-muuttujan arvo lisätään tai vähennetään pelaajan raha-arvoon riippuen siitä, että onko se positiivinen vai negatiivinen. Tämä lähestymistapa saa myös aikaa sen, että pelaaja ei voi käyttää näitä muistiosotteita huijatakseen pelissä, koska tonttien arvot ladataan, joka kerta palvelimelta kiinnityksen yhteydessä.Lopulta luokan arvot muunnetaan JSON-objektiksi käyttäen Newtonsoft-kirjastoa ja palautetaan pelaajalle bittitaulukko muodossa.

9.2 Vaihtokauppa näkymä

Vaihtokauppa näkymään liitttyy kaksi eri näkymää, se mistä pelaaja voi lähettää toiselle pelaajalle vaihtokauppa kutsun, ja se, missä pelaajat voivat käydä kauppaa keskenään. Kutsun lähettäminen vaatii myös tavan toiselta käyttäjältä hyväksyä vaihtokauppa. Tämä tapahtuu siten, että GameManager-skriptin InGame-silmukassa kuunnellaan, että vastaako jonkin huoneessa olevan pelaajan tradeRequest-luokan ”to” arvo omaa uniikkia id:tä. Jos näin tapahtuu niin pelaajan ruudulle, jolle lähetetään vaihtokauppa pyyntö, luodaan ilmoitus (Kuva 18.), jossa on tapahtumakuuntelija mikä mahdollistaa pelaajan, jolle ilmoitus tulee, hyväksymään sen klikkaamalla ilmoitusta. Tämä lähettää pyynnön palvelimelle, jossa palvelin asettaa molempien pelaajien trade-luokan tradingWith-muuttujaksi toisen pelaajan uniikin id:n arvon. Tämä saa aikaa sen, että pelaajanäkymässä on luotu InGame-silmukkaan if-lause, joka tarkistaa, että pitäisikö näiden pelaajien olla vaihtokauppanäkymässä. Jos tämä käy toteen niin molemmilta pelaajiilta aukeaa automaattisesti vaihtokauppanäkymä. (Kuva 19.)

Kuva 18 Pelaaja saa kutsun

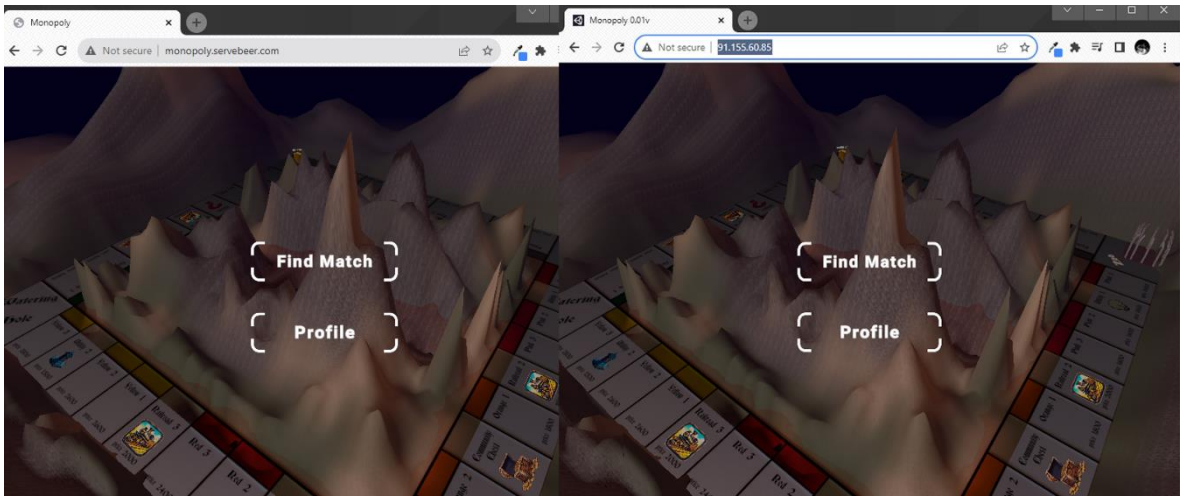
Kuva 19 Kahden pelaajan välinen vaihtokauppanäkymä

10 HTTP-palvelin (XAMPP)

Oleellinen osa WebGL-ohjelmointirajapinnan sovelluksissa ja peleissä on se, että ne pyörivät web-palvelimella. Tämä on erillinen palvelinohjelmisto ja protokolla siitä missä pelaajat viestittelevät keskenään pelissä. IP-osolitteet ovat hankalia muistaa ja keskiverto käyttäjälle myös pelottavan näköisiä. Siksi kehitettiin Domain Name System (DNS). Tämä järjestelmä tekee internetin käytöstä paljon helpompaa ja käyttäjäystävällisempää, sillä sen ansiosta käyttäjät voivat kirjoittaa tuttuja osoitteita verkkoselaimen osoiteriville sen sijaan, että heidän pitäisi muistaa monimutkaisia numerojonoja. Tämä on ikään kuin puhelinluettelo internetissä, joka kääntää helposti muistettavat nimet IP-osoitteiksi. (Cloudflare2023a)

NO-IP-palvelu mahdollistaa ilmaisen verkkotunnuksen, jonka avulla pystyn ohjaamaan verkkoliikenteen haluamaani IP-osoitteeseen. Verkkoliikenteenohjaukselle on kaksi vaihtoehtoa maskin kanssa tai ilman. Verkkoliikenteenohjaus maskin kanssa toteutuu siten, että käyttäjän valitsemalle domainille luodaan IFrame-ikkuna, jonka sisältö on käyttäjän valitsema kohdesivusto. Ilman maskia tarkoittaa sitä, kun käyttäjä avaa linkin niin verkkotunnus vaihtuu palvelimen ip-osoitteeksi. (Kuva 20.)

Kuva 20 Vasemmalla NO-IP:kanssa (maskillisena). Oikealla ilman NO-IP:tä



11 Testaaminen

Pelin kehitys toteutettiin WebGL-ohjelmointirajapinnalle, mikä tarkoittaa, että se toimii suoraan nettiselaimessa. Tämä edellyttää kuitenkin toimivaa web-palvelinta. Ratkaisuksi valittiin XAMPP, avoimen lähdekoodin verkkopalvelinpaketti, jonka avulla luotiin helppo paikallinen testiympäristö. Konfigurointivaiheessa XAMPP asetettiin kuuntelemaan isäntälaitteen paikallista IPv4-osoitetta. Tämä mahdollisti sen, että peliin pääsi helposti käsiksi syöttämällä osoitteen http://localhost/ isäntälaitteelta.

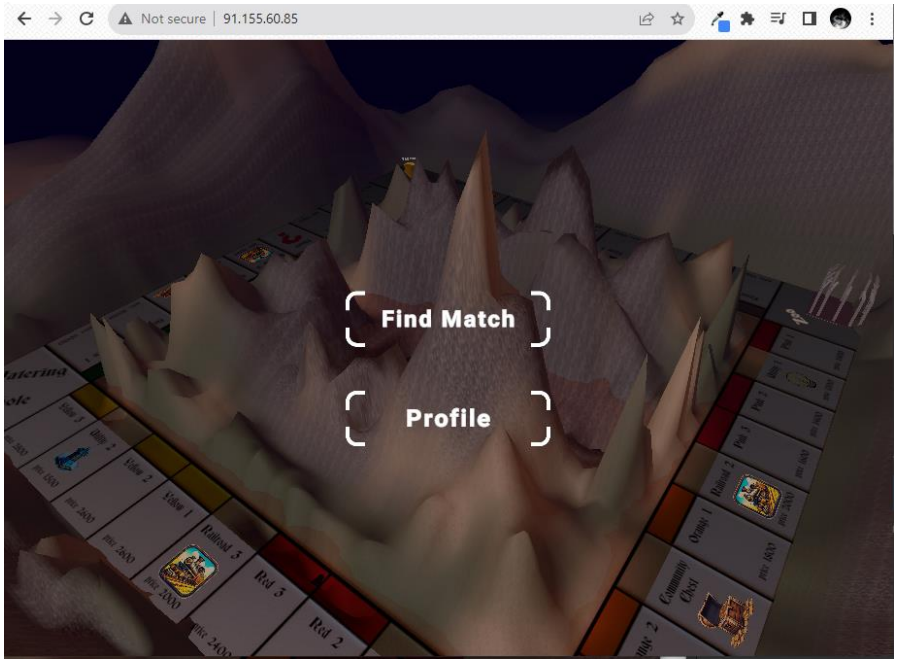
Peli on tarkoitettu nettipeliksi ja idea on siinä se, että pystyt pelaamaan muiden pelaajien kanssa kotiverkon ulkopuolelta. Ratkaisuna tähän ongelmaan on muutamia, kuten portin ohjaus tai VPS-palvelun osto, mutta koska pelin budjetti on 0€ niin portin ohjaus itse asettamaan XAMPP-ympäristöön oli osava ratkaisu. Tämä tapahtui siten, että modeemin asetuksista luotiin uusi portinohjaus sääntö porteille 80 ja 7777 palvelimen isäntälaitteelle. Portti 80 on yleinen http-liikenteen portti, kun taas portti 7777 toimii pelissä datan välityksen porttina.

Jotta peliä pääsee kokeilemaan niin XAMPP-webserveri ja pelin websocket-sharp-palvelin pitää asettaa päälle. XAMPP-käyttöliittymässä se tapahtuu avaamalla ohjauspaneeli sovelluksen ja painamalla start-näppäintä. Kehitysympäristönä palvelimen teossa käytin Microsoftin tarjoamaa Visual Studio Code-sovellusta. Palvelin on luotu konsolisovellukseksi ja sen päälle laittaminen tapahtuu komennolla `dotnet run`

On tärkeää myös ottaa huomioon palvelinlaitteen palomuuri. Pelin palvelimeen yhdistäessä ei ilmennyt ongelmia, mutta näin voi tapahtua, joten on tärkeää, että palvelimella konfiguroidaan palomuuri hyväksymään verkkoliikenne porteista 80 ja 7777 TCP-protokollalle.

Kuvassa (Kuva 21.) näkyy, että yhteys XAMPP-palvelimeen, palvelimen laitteen julkisella IP-osoitteella toimii. Peli lataa alustusvaiheessa tonttien tiedot ja sääntöarvot palvelimelta. Tämä tarkoittaa sitä, että portinohjaukset porteille 80 ja 7777 toimivat.

Kuva 21 WebGL-näkymästä kuva XAMPP-palvelimelta.



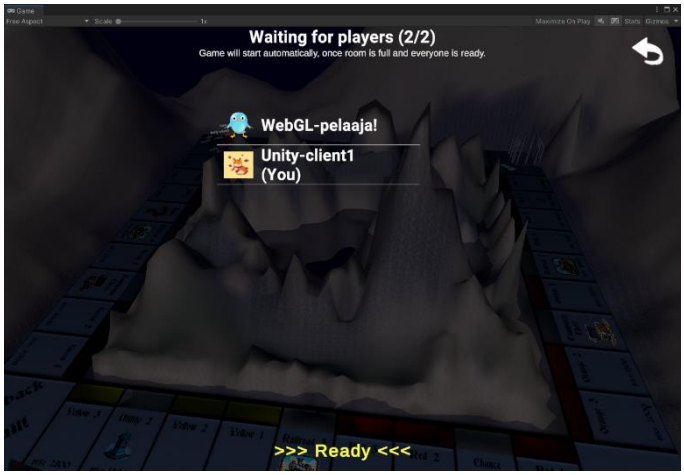
Seuraavana vuorossa on testata, että näkeekö pelaajat toisensa. Tämän testauksen suoritin Unityllä ja selaimella. Peliin piti kehittää ratkaisu, miten pelaajat löytäisivät toisensa. Tähän oli monia eri vaihtoehtoja, kuten kolmannen valikon luominen missä pelaajat voisivat luoda huoneita ja liittyä muiden huoneisiin. Peli on kuitenkin tarkoitettu nopeasti pelattavaksi vähäisillä käyttäjäsyötteillä niin ”Find Match”-niminen painike tekee kaiken tämän automaattisesti. Mikäli palvelimella ei ole aktiivisia huoneita niin painike luo huoneen ja liittää pelaajan sinne Find Match-painikkeen algorithmi priorisoi huoneeseen liittymisessä huoneet, jossa on eniten pelaajia, mutta missä kuitenkin on tilaa.

Find Match -painikkeen huoneen valitsemisalgorithmi toimii siten, että palvelinpuolella suoritetaan case-lausekkeen sisällä oleva silmukka, joka tarkistaa jokaisen palvelimella olevan huoneen ja listaa sen väliaikaisesti huonelistoihin pelaajamäärän mukaan. Seuraaksi tarkistetaan korkeimmasta pienimpään pelaajamäärien omaavien huonelistojen pituudet ja arvotaan 0 ja listan koon väliltä numero. Numero toimii huoneen indeksinä, jonne pelaaja lopulta lisätään. Palvelimen liittäessä pelaajan huoneeseen, palvelin antaa pelaajalle uniikin ID:n. Uniikin merkkijonon avulla

pelaaja pystyy kommunikoimaan palvelimen kanssa, tunnistamaan itsensä ja tämän avulla muokkaamaan palvelimelle tallennettuja arvoja.

Kuvassa (Kuva 22.) näkyy nyt lopputulos, kun molemmat pelaajat ovat painaneet Find Match-näppäintä. Ylempi eli ’WebGL-pelaaja!’ yhdisti ensimmäisenä, joka tarkoittaa sitä, että hänen lähettämä viesti palvelimelle teki tarkistuksen, että onko vapaita huoneita olemassa. Mikäli huoneita ei ollut hän loi uuden huoneen palvelimelle. Alempi pelaaja ’Unity-client1’ taas yhdisti peliin myöhemmin, joten palvelin löysi valmiiksi luodun huoneen ja lisäsi tämän pelaajan huoneeseen. Jokaisessa huoneessa on oma maksimi pelaajamäärä. Mikäli huoneen nykyinen pelaajamäärän arvo vastaa huoneen maksimi pelaajamäärää niin palvelimen Find-Match-painikkeen algorithmi skippaa nämä huoneet ja tekee uuden huoneen. Tämä on pakollinen tarkistus sen varalle ettei huoneeseen voi tulla määräänsä enempää pelaajia.

Kuva 22 Lopputulos toimivasta huoneen luomis- / löytämis funktiosta



Pelaajille piti keksiä tapa näyttää saatu huone data palvelimelta. Tätä hallinnoi UI-luokassa sijaitseva alaluokka nimeltä ’WaitingForPlayersScreen’. Luokan metodejen päätehtäviin kuuluu päivitää pelaajan näkymää saadun tiedon perusteella palvelimelta. Mikäli huonedata ei täsmää olemassa olevien käyttöliittymäelementtien kanssa ’WaitingForPlayersScreen’-näkymässä niin tällöin metodi poistaa tai luo uuden käyttöliittymäelementin pelaajan saadun tiedon perusteella. Jokainen käyttöliittymäelementin-parent peliobjekti nimetään pelaajan uniikin ID:n mukaan. Uniikkia ID:tä käytetään palvelindatan ja käyttöliittymässä olevien elementtien identifioimiseen. Käyttöliittymän jokaiseen pelaaja-elementin lapsiobjektiin asetetaan myös käyttäjänimi ja profiilikuva palvelimelta saatujen tietojen perusteella.

12 Pelin käyttöliittymän toteutus ja näyttäminen

Pelin tekemiseen liittyy monia eri osa-alueita ja yksi näistä on käyttöliittymä. Siisti ja selkeä käyttöliittymä on tärkeä osa pelin käyttäjäkokemusta, koska se mahdollistaa pelaajien vuorovaikutuksen pelin maailman kanssa. Hyvin suunniteltu käyttöliittymä on yksi syy sille mikä lisää houkuttelevuutta peliisi.

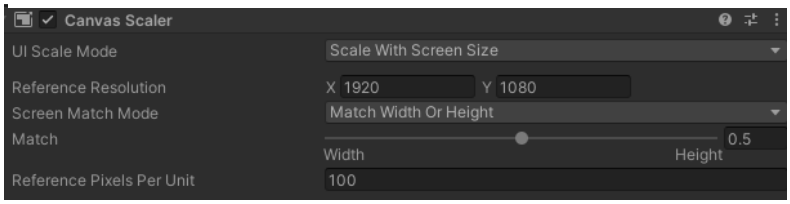
Ensin luodaan mekaniikka ja siihen sopiva käyttöliittymäelementti. Tämä on järjestys, jossa käyttöliittymä luodaan peliin. Käyttöliittymää suunnitellessa pitää ottaa huomioon, että kaikki ihmiset ovat erilaisia, jotkut ovat kuuroja tai sokeita, toiset taas nuoria tai vanhoja. Tämän takia on tärkeää yrittää tehdä sellainen käyttöliittymä, joka soveltuu mahdollisimman monelle ihmiselle, ottaen kuitenkin pelin teeman ja mekaniikat huomioon.On tapoja luoda jokaiselle ryhmälle omat käyttöliittymätoiveet asetusvalikon avulla, josta pelaajat pystyvät esimerkiksi skaalaamaan käyttöliittymää omien tarpeidensa ja halujen mukaisesti.

12.1 Canvas

Canvas on Unityssä pääkomponentti, joka sisältää kaiken mikä liittyy käyttöliittymään. Kaikki pelin 2D-tekstit, kuvat, painikkeet ja tekstinsyöttökentät ovat lapsiobjekteja canvas-peliobjektille. Tämä mahdollistaa sen, että canvas-peliobjektiin voi lisätä hyödyllisiä komponentteja kuten Canvas Scaler-komponentin.

Canvas Scaler-komponentti (Kuva 23.) mahdollistaa peliin responsiivisen käyttöliittymän Tämä toimintaperiaate perustuu pääasiassa siihen, että peli asettaa itselleen referenssiresoluution, joka tässä tapauksessa on 1920x1080 pikseliä. Kun pelaajan käyttämä resoluutio ylittää tämän referenssiresoluution, peli sovittaa pelaajan resoluution automaattisesti referenssiresoluutioksi. Toisaalta, jos pelaajan käyttämä resoluutio on pienempi kuin referenssiresoluutio, Canvas-Scaler-komponentti skaalaa käyttöliittymän ylöspäin, jotta se mahtuu näyttöön selkeästi.

Kuva 23 Canvas Scaler-komponentti



Match-liukusäätimen tehtävänä on päättää, käytetäänkö skaalauksessa leveyttä vai korkeutta referenssinä vai niiden yhdistelmää toisistaan. Nämä asetukset varmistavat, että pelin käyttöliittymäelementit näkyvät samalla tavalla eri resoluutioita käyttäville pelaajille. (Kuva 24.)

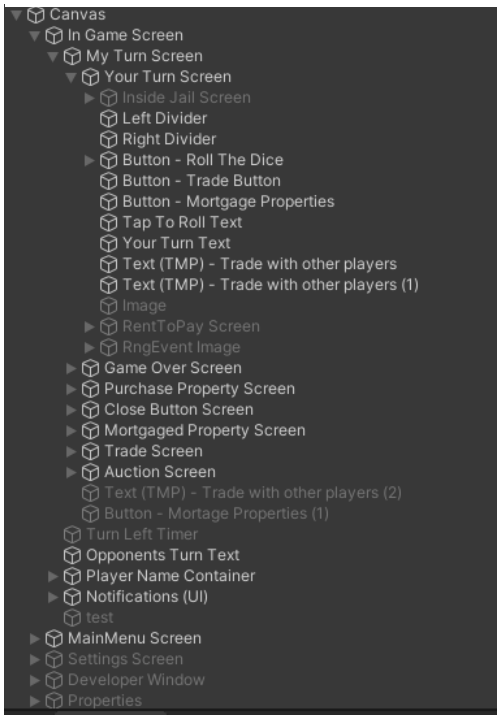
Kuva 24 Pelin aloitusruutu usealla eri resoluutiolla ja kuvasuhteella.

Mitä pienempi resoluutio on, sitä suttuisemmiksi pelin näytöllä olevat elementit muuttuvat. Statcounter-sivuston tekemien tutkimusten mukaan elokuusta 2022 - elokuuhun 2023 maailmanlaajuisesti yleisin tietokonekäyttäjien resoluutio oli 1920x1080 eli 1080p, ja sitä käytti 23,39% käyttäjistä. Toiseksi yleisin oli 1366x768 (13.99%), kolmanneksi 1536x864 (10.19%), neljänneksi 1440x900 (6.55%), viidenneksi 1280x720 (5.53%) ja viimeisenä 1280x1024. (Statcounter2023a)

12.2 Valikoiden näyttäminen eri pelin vaiheissa

Pelin yksi toimintamekanismeista perustuu käyttöliittymäelementtien (Kuva 25.) kytkemiseen pelaajan ruudulta päälle ja pois. Kytkemisjärjestys tapahtuu pelaajan saaman JSON-tiedon perusteella palvelimelta. Eli jos on pelaajan ’A’ vuoro niin pelaajan peli kytkee päälle Your Turn Screen-peliobjektin eli vuororuudun tälle pelaajalle, samalla kuin muiden pelaajien käyttöliittymästä kytketään tämä ruutu pois. Tämä kaikki tapahtuu ConnectLoop-silmukassa, joka pyörii JSON.cs-skriptissä olevan requestRate-arvon mukaan. Mitä pienempi tämä luku on niin sitä nopeammin pelaaja saa tiedon muilta pelaajilta näiden arvojen muutoksista ja täten ruudut päivittyvät nopeammin pelaajalle.

Kuva 25 Unityn hierakia:sta canvas-peliobjektin sisältävistä käyttöliittymä-elementeistä



Jotta käyttöliittymäelementtien hallinta oli selkeämpää niin projektiin UI.cs-skripti tiedoston, joka hoitaa kaiken käyttöliittymäelementtien hallinnan. Skriptin rakenne menee samalla tavalla kuin Unityn hierakia näyttää kyseiset käyttöliittymäelementit. Jokaisen pääluokan loppuun on kirjoitettu nimike ”Screen”, joka kuvastaa jokaista käyttöliittymäelementtiä, mikä sisältää useamman eri käytettävän objektin. Joissakin ruuduissa on useampi alaruutu ja useampi alaobjekti. Jokaiselle pelissä käytettävälle muokattavalle käyttöliittymäelementille on kirjoitettu oma funktio sille näyttämiseen ja piilottamiseen sille nimettyyn luokkaan. Tämä helpottaa elementtien aktivoimista koodista käsin.

Esimerkkinä (Ohjelmakoodi 12.), jossa ConnectLoop-silmukassa sijaitsevasta koodinpätkästä, joka avaa pelaajan vuororuudun auki. Tässä ensimmäisenä katsotaan, että onko pelaajalla maksettavia velkoja, jos näin on, niin suoritetaan ensimmäisen if-lausekkeen sisällä olevat komennot, jos näin ei ole niin silloin varmistetaan, että maksuruutu menee kiinni ja avataan pelaajan vuororuutu. Samalla vaihdetaan vuororuudussa olevan nappulan kuva-arvoksi nopan kuva. Kaikki tämä tapahtuu ison silmukan sisällä toistuvasti, joka suoriutuu vain silloin kun pelaajan Room-luokan state arvo on asetettu ”inGame”.

Ohjelmakoodi 12 Pelin pääsilmukka, joka suoriutuu, mikäli pelaaja on pelinäkymässä.

```
while(JSON.Instance.room.state == Room.State.InGame.ToString()){

// Check if player has to pay something, before enabling diceButton.
if (player.currentGameData.turn.isMyTurn && !playerPieceController.moving &&
RollTheDiceButtonTimerReady() && !player.currentGameData.auction.active &&
!UI.inGameScreen.tradeScreen.IsOpen())
{
        if(player.currentGameData.turn.rent.amount > 0)
        {
           UI.inGameScreen.yourTurnScreen.Open();
UI.inGameScreen.yourTurnScreen.rollTheDice.leftToPay.Open();
        }
        else
        {
        UI.inGameScreen.yourTurnScreen.rollTheDice.leftToPay.Close();

        // Players first diceRoll
        if (!player.currentGameData.turn.propertyPurchased &&
        player.currentGameData.turn.payAmount <= 0 &&
        !player.currentGameData.dice.rolledDice ||
        !player.currentGameData.turn.propertyPurchased &&
        player.currentGameData.turn.payAmount <= 0 &&         player.currentGameData.dice.pair
        ){
        UI.inGameScreen.yourTurnScreen.Open();
        UI.inGameScreen.yourTurnScreen.rollTheDice.Butto      nDiceImage(true);
        }
        else if …
        … … …
        }
… … …
}
… … …
… … …
… … …
if(WebsocketManager.Instance.IsReadyToSend())
   WebsocketManager.Instance.Send(ConnectActions.inGame);
…
}
   … … …
   … … …
   yield return new WaitForSeconds(WebSocketManager.Instance.requestRate);
}
```

13 Kehittämistyön tavoite ja tarkoitus

Tämän opinnäytetyön tarkoituksena oi luoda selaimella toimiva vuropohjainen nettipeli, käyttäen Unity 3D-pelimoottoria ja websocketteja luodakseen yhteyden palvelimeen ja tätä kautta toisiinsa. Opinnäytetyössä tarkoituksena oli toteuttaa tämä kokonaisuus alusta loppuun asti, jossa pelaajat voisivat avata pelin selaimella, vaihtaa hahmoa ja nimeä päävalikon profiilivalikossa. Hahmon valinnan jälkeen etsiä huone ja aloittaa peli. Lopulta palata päävalikkoon, muiden pelaajien mentyä konkurssiin. Tavoitteena itselleni oli saada käytännönkokemus pelaajien ja palvelimen välisestä viestinnästä vuoropohjaista lautapeliä luodessa. Viestintä tapahtui pelissä JSON-muodossa, joten arvojen validointi oli selkeää. Tavoitteena ei ollut luoda cross platform-elementtiä peliin, mutta se tuli WebSocketteja käyttämällä. Pelin testaaminen tapahtui pääosin Unityn omalla play-editorilla ja nettiselaimessa pyörivällä rakennetulla WebGL-projektilla. Palvelinpuolella tavoitteena oli luoda ratkaisu siihen, että yksi palvelin voisi sisältää useita eri huoneita, ja liittää pelaajan vapaaseen huoneeseen, jossa on eniten pelaajia, mutta mikä ei kuitenkaan ole täynnä yksinkertaisella ’Find-Match’-painikkeella. Tämä onnistui myös, lisäämällä ’searchForRoom’-funktion sisälle palvelinpuolella, että liittymisvaiheessa pelaaja katsoo huoneella olevan jokaisen listan. Lisää int-tyyppiseen listaan kaikkien niiden huoneiden indeksinumerot, jossa on eniten pelaajia ja jos näitä on useampia niin arpoo näistä indekseistä, että mihin huoneeseen pelaaja liitetään. Tämän jälkeen palvelin palauttaa pelaajalle arvotun huone-indeksi arvon, joka sinetöi tämän päätöksen, että pelaaja kuuluu nyt huoneeseen ’x’. Lautapeli sisälsi useita eri toiminnallisuuksia, jotka piti luoda, kuten tonttien ja tonteilla olevien talojen ostamisen, kiinnittämisen, huutokauppaamisen ja vaihtamisen, sekä pelaajan nopanheittämisen,, liikuttamisen, vankilaan menemisen ja voittamisen / häviämisen. Mitä tulee esimerkiksi pelaajan tontin ostamiseen niin tämä liittyi siihen, että palvelimella on lista jokaisesta pelissä olevasta tontista ja näiden rahallisista arvoista, kuten kiinnityskuluista, vuokrista yms. Palvelinpuolella taas oli funktio, joka muutti pelaajan nykyisen ruutuarvon tontti-indeksiksi, jonka avulla palvelin tiesi, että minkä tontin palauttaa pelaajalle nopanheiton jälkeen, mutta ennen palauttamista palvelimen piti tarkistaa, että onko tontti jo joillakin pelaajista. Tämä tapahtui siten, että silmukoidaan läpi kaikki huoneessa olevat pelaajat. Jos pelaajalla on tontti ja uniikki-id ei matchaa oman pelaajan uniikkia-id:tä niin silloin pelaajan rent-alaluokan arvoihin syötettiin summa paljonko pelaaja on velkaa ja sen pelaajan uniikki-id, että kenelle se on velkaa. Tätä arvoa InGame-silmukka pitää silmällä ennen vuoron lopettamista, eli pakottaa pelaajan maksamaan.

Velan määrittelyssä piti ottaa myös huomioon tontin talomäärä tai se, että jos pelaaja omistaa kaikki kyseisen värin tontit. Toiminnon laski funktio, jolle piti tehdä omat tarkistukset kolmelle eri tonttityypille ’rr’, ’utility’ ja ’color’. Ensimmäisestä eli ’rr katsottiin vain, että omistaako pelaaja samantyyppisiä, jos näin oli niin vuokra = määrä * tontin vuokra. ’Utility’ taas määrittää, että kerrotaanko nopanluku 4 tai 10:llä. Color oli taas näistä monimutkaisin. Siinä piti ensimmäisenä

tarkistaa, että omistaako pelaaja taloja. Tämä oli helppo rasti, koska tonttiluokassa on arvo tälle muuttujalle. Jos pelaaja ei omistanut taloja niin sitten katsottiin, että omistaako pelaaja 3 samaa väriä, jos näin oli niin `vuokra = vuokra * 2`. Jos tämäkään ei käynyt toteen niin funktio palautti arvoksi vuokran. On myös olemassa vaihtoehto missä pelaajalla ei ole varaa maksaa vuokraa, eli pelaajan kokonaisvarallisuus ei riitä niin tällöin pelaaja automaattisesti leimataan hävinneeksi ja lisätään huoneessa olevan hävinneet listaan.

Tonttien talojen ostamistoimintoa tehdessä, piti ottaa huomioon se, että pelaaja on valtuutettu tähän, eli hän omistaa kolme samaa väriä. Kiinnittämisessä piti kanssa tehdä muutama tarkistus, että jos pelaaja omistaa taloja niin talot myydään ensimmäisenä ennen kuin pelaaja pystyisi kiinnittämään tontin.

Huutokauppa osiossa, kuten muissakin piti luoda uusi näkymä ja tarkistus pelin pääsilmukkaan, että jos jollakin pelaajalla on huutokauppa aktiivisena niin muillakin pelaajilla avautuu huutokauppa ruutu, missä he pystyvät sitten huutokaupata tontista. Huutokauppa kestää 15 sekunttia ja aika katsotaan DateTime-luokan UniversalTime().now-alustetun ajan mukaan verrattuna nykyiseen aikaan. Tämän arvon voi muuttaa palvelimen sääntö osioista.

Nopanheittäminen ja liikkuminen toimii käsikädessä. Nopanheitto pääosin toimii siten, että palvelin arpoo kaksi arvoa 1-6 väliltä per arvo. Lisää näiden lukujen summan pelaajan nykyiseen ruutumuuttujaan ja tämän jälkeen tekee tarkistukset, että mihin ruutuun pelaaja päätyi. Jos tämä ruutu sattui olemaan vankilaruutu niin silloin pelaajan vankila-arvoksi laitetaan ’true’. Tämä otetaan huomioon aina ennen nopanheittoa, että jos olet vankilassa niin sillon nopansumma ei vaikuta ruutuarvoon, vasta tuplien tai maksun jälkeen. Palvelin hoitaa logiikan ja säännöt, pelissä oleva ”liikkuminen” tapahtuu vain sen jälkeen kun palvelin antaa pelaajalle uuden ruutuarvon ja jos nykyinen ruutuarvo ei matchaa tätä uutta ruutuarvoa niin liikkuminen aloitetaan pelaajapuolella. Koska peli oli vuoropainotteinen niin yksi tärkeistä elementeistä oli luoda huoneelle tapa päättää, että kenen pelaajan vuoro on. Tämä funktio toimii siten, että silmukoidaan jokainen pelaaja huoneesta ja annetaan vuoro seuraavalle listasta. Eli pelaajalistan ensimmäinen aloittaa.

Viimeisenä osiossa oli voittaminen ja häviäminen. Tämä liittyy pääosin pelaajan kokonaisvarallisuuteen. Palvelinpuolelle piti luoda funktio, joka palauttaa pelaajan kokonaisvarallisuuden, jos tämä varallisuus menee alle pelaajan heittämän tontin vuokran, tuloveron tai lisäveron niin pelaaja automaattisesti leimataan konkurssiin ja hänelle annetaan sijoitusluku.

Pelissä piti lisäksi tehdä jokaisen käyttöliittymäelementtien näyttämiseen ja piilottamiseen metodeja UI-luokassa oleviin alaluokkiin, josta näitä hallinnoitiin koodipuolella.

14 Projektin suunnittelu ja toteutus

Projektin teko vaiheessa ensimmäisiä isoja esteitä oli etsiä toimiva nettikirjasto, joka tukee nettiselainten välistä kommunkaatiota. Tähän löytyi useampi vaihtoehto kuten Unityyn suunnitellut Mirror ja FishNet-nettikirjastot. Perinteinen C#:lla toimiva System.net-kirjastoa käyttämä tcp-socket ratkaisu ei toiminut, koska nettiselain ei tukenut tätä. Lopulta kuitenkin löysin toimivan verkkokirjasto ratkaisun Websocketit. Koodasin palvelinpuolelle todella simppelin chat-tyyppisen palvelinratkaisun, joka ottaa käyttäjältä syötteen ja näyttää sen palvelimella. Pystytettyään webbipalvelimen, konffattua palomuurin, asetettua portinohjaukset ja rakennettuaan projektin WebGL-moduulilla oli aika testata yhteyttä. Suoritin tämän testauksen käyttäen toista laitetta, joka on eri verkossa palvelin laitteen kanssa. Testi onnistui positiiviesti tieto meni perille, sen jälkeen olikin aika lähteä suunnittelemaan itse peliä.

Pelin suunnittelu alkoi ulkoasulla. Ennen isompien pelaajaluokkien suunnittelua niin seuraava osa oli tehdä itse pelilauta ja suunnitella pelinappuloiden liikkuminen laudalla. Tämä tapahtui siten, että laudan jokaisella ruudulla on oma indeksiarvo ruututaulukossa. Tähän seuraava askel oli luoda pelaajaluokkaan arvo, joka pitää kirjaa nykyisestä ruutuarvosta. Pelinappulan liikuttamista varten piti taas luoda oma skripti, joka piti kirjaa pelinappulan nykyisestä sijainnista ja pitää huolen tämän liikuttamisesta oikeaan ruutuun palvelimelta saadun tiedon perusteella.

Seuraava iso askel oli suunnitella pelaajaluokka, joka pitää sisällään kaiken pelissä olevan pelaajan tiedon, kuten uniikin ID:n, pelissä olevat arvot, kuten omistetut tontit, rahamäärän, käyttäjänimen, profiilikuvan, huoneindeksin sekä nopanheittoon, pelaajan vuroon, rankkaukseen, vaihtoon, huutokauppaan ja vankilaan liittyvät alaluokat. Pelaajaluokan ja pelissä olevien näkymien tekeminen tapahtui yhtäaikaisesti. Pelissä olevat näkymät hallinnoi tavalla tai toisella pelaajaluokassa olevia arvoja. Palvelimelle piti kehittää myös ratkaisu hyväksymään ja tallentamaan pelaajien lähettämät pelaajaluokat. Tähän ratkaisuna oli luoda palvelimelle huoneet. Palvelimen huoneet toimivat siten, että huoneluokassa on arvot huoneen asetuksille kuten maksimi pelaajamäärä, tila, huonenumero, käytetyt kortit, ranking-lista ja tärkein eli lista pelaajaluokista. Viestintä tapahtuu siten, että pelaaja lähettää oman pelaajaluokkansa palvelimelle, palvelin tallentaa tämän pelaajaluokassa olevan huonenumeron ja uniikin ID:n perusteella oikeaan huoneeseen ja oikealle pelaajalle. Palvelin antaa vastausviestinä pelaajalle koko huoneluokan arvon. Huoneluokka sisältää tässä vaiheessa kaiken tiedon muista ja omasta pelaajasta. Viestin tultua pelaajalle, tieto käsitellään ja eritellään. Pelaajapuolella käyttöliittymä ja pelilaudalla olevien tonttien tilat päivitetään saadun datan mukaisesti. Tätä pelaajaluokkaa lähetetään edestakaisin pelaajan ja palvelimen välillä.

15 Johtopäätökset ja pohdinta

Tämän opinnäytetyön tavoitteena oli luoda toimiva nettipeli verkkoselaimelle käyttäen Unityä ja NativeWebsocket-kirjastoa. Projektia tehdessä tuli tutuksi moni muukin verkkokirjasto Unityyn. Esimerkiksi Mirror Networking-kirjasto tarjosi työkalut ja valmiit koodinpätkät reaaliaikaisen verkkopelin luomiseen, mutta ideana tässä oli saada parempi käytännöllinen ymmärrys palvelimen ja pelaajan välisestä kommunikaatiosta. Websocket tarjosi kevyen JSON-viestittelyn pelaajien ja palvelimen välillä. Palvelinpuolella pitää ottaa kuitenkin paljon asioita huomioon kuten, että pelaaja ei syötä väärää dataa, joka voisi mahdollisesti johtaa palvelimen kaatumiseen. Pelaajien välinen välinen kommunikaatio toimi suunnitelmien mukaisesti ja kussakin vaiheessa näytti pelaajalle oikeat käyttöliittymäelementit ruudullaan.

Jatkoa ajatellen olisi tärkeää ottaa huomioon palvelimen skaalattavuus. Tällä hetkellä peli on rakennettu siten, että se yhdistää vain yhdelle palvelimelle, jossa voi maksimissaan olla tietty määrä huoneita. Tähän pitäisi tulevaisuudessa kehittää systeemi, joka mahdollistaisi usean samanaikaisen palvelimen käytön. Tämä toimisi siten, että olisi pääpalvelin joka on yhteydessä muihin pelin palvelimiin ja päättää, missä palvelimessa olisi vapaita huoneita tai tilaa uudelle huoneelle, johon pelaaja sijoitetaan. Myöskään vasteaika ei ole ongelma vuoropainotteisessa pelissä niin tällöin pelaajat eri puolilta maailmaa pystyisivät pelaa yhdessä samalla palvelimella. Tietenkin se hidastaa toimintoja, kuten nopanarvon saamista palvelimelta tai yleisesti ottaen muiden pelaajien arvoja ruudulla. Vasteajan korjaamiseen ja pelin mahdollistamiseen mahdollisimman sujuvaksi monelle pelaajalle ympäri maailmaa olisi se, että vuokraisi VPS-palvelun eri alueilta, mutta se on tulevaisuuden kysymys, jos projekti menestyy.

Pelin jatkon kannalta on tärkeää myös ajatella pelaajan datan tallettamista muistiin. Tämä kuitenkin vaatisi sen, että pelaajan pitäisi rekisteröidä tili. Tiliien tallettamiseen taas voisi käyttää Xampin tarjoamaa MariaDB-tietokantaa. Tämä lisäisi kuitenkin paljon kysymyksiä tietoturvaan, kun on kyse pelaajen datasta niin sitä pitää säilyttää ja käsitellä varoen. Pitäisi luoda automaattinen ohjelma, joka ottaa varmuuskopiota tietyn väliajoin pelaajien datasta ja palvelin koneellta pitäisi tehdä tietoturvamuutoksia kansioon, jossa dataa säilytetään. Myös itse kirjautumis- ja rekisteröitymiskentät pitäisi rakentaa tietoturvaa miettien, jotta käyttäjä ei pysty syöttämään SQL-komentoja ja tulostamaan kaikkia tallennettujen pelaajien arkaluontoisia tietoja näytölleen. Tämän takia on hyvä, että arkaluontoiset asiat kuten, salasanat salataan ennen tallentamista. Lopulliseen julkaisuun olisi myös tärkeää luoda käyttäjiä koukuttavia elementtejä, kuten pelivaluutta ja kauppa, lisää pelinappuloita, pelinappuloiden muokkausmahdollisuuksia ja erikoisefektejä.

16 Yhteenveto

Opinnäytetyön tavoitteena oli selvittää, millaisia rajoitteita saattaa ilmetä WebGL-ohjelmointirajapinnalle julkaistussa pelissä? Mitä erilaisia tapoja pelaajien väliseen viestittelyyn löytyy? Miten pelaajien välinen viestittely tapahtuu Websocket-protokollalla?

Ensimmäiseen tutkimuskysymykseen eli millaisia rajotteita saattaa ilmetä WebGL-ohjelmointirajapinnalle julkaistussa pelissä, vastaaminen onnistui hyvin ja suurin osa rajotteista olikin graafiikkaan liittyviä, mikä rajotti yksityiskohtien lisäämistä pelimaailmaan. Selvisi myös syy, miksi WebGL-ohjelmistorajapinnalla luodut sovellukset ovat raskaita ja mikä niistä tekee raskaan, sekä miten pystytään optimoimaan WebGL-alustalle suunniteltuja pelejä ja sovelluksia.

Toiseen tutkimuskysymykseen eli mitä erilaisia tapoja pelaajien väliseen viestittelyyn löytyy? Opinnäytetyössä käytiin läpi Unityssä toimiva verkkovaihtoehtoja, kuten Fishnet ja Mirror… sekä tutustuttiin erilaisiin verkkotopologioihin, jotka auttavat lukijaa ymmärtämään paremmin, miten tieto kulkee ja miksi toiset palvelinratkaisut ovat parempia tietyissä tilanteissa kuin toiset.

Viimeiseen tutkimuskysymykseen eli miten pelaajien välinen viestittely tapahtuu Websocket-protokollalla? Opinnäytetyössä selitettiin kappale kappaleelta tähän vastaus toiminnallisessa osuudessa, viestipakettien luomisesta alustamiseen Newtonsoft-kirjastoa käyttäen ja siitä lähettämiseen asti aktiivisella Websocket-yhteydellä. Tähän kysymykseen löydettiin myös teoriaosuudessa ratkaisuja erilaisille viestintämenetelmille, sekä miksi valita tietyt viestintävaihtoehdot muiden sijasta.

Lähteet

Adobe2023a. Understanding physically based rendering. Viitattu 18.12.2023
https://www.adobe.com/products/substance3d/discover/pbr.html#:~:text=Physically%20based%20rendering%20(PBR)%2C,light%20interacts%20with%20material%20properties.

Cloudflare2023a. What is DNS?. Viitattu 13.12.2023
https://www.cloudflare.com/learning/dns/what-is-dns/#:~:text=The%20Domain%20Name%20System%20(DNS,browsers%20can%20load%20Internet%20resources.

datatracker2023a. RFC6455. Viitattu 22.10.2023
https://datatracker.ietf.org/doc/rfc6455/

GG2023a. What is websocket and how it is different from HTTP. Viitattu 22.10.2023
https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/

Github 2023a. NativeWebSockets. Viitattu 24.09.2023
https://github.com/endel/NativeWebSocket

Hackernoon2023a. Game Network Topologies. Viitattu 09.12.2023
https://hackernoon.com/unity-realtime-multiplayer-part-6-game-network-topologies

Microsoft2023a. A tour of the C# language. Viitattu 18.12.2023
https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Mozilla2023a. The Websocket API. Viitattu 18.12.2023
https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Occasoftware 2023a. Michael Sacco. Understanding URP, HDRP and Built-In Render Pipeline. Viitattu 15.10.2023
https://www.occasoftware.com/blog/unity-understanding-urp-hdrp-built-in#:~:text=URP%20is%20ideal%20for%20developers,graphics%20for%20high%2Dend%20hardware.

owasp2023a. Cross Site Scripting (XSS)
https://owasp.org/www-community/attacks/xss/

Photon2023a. Photon Engine. Viitattu 08.12.2023
https://www.photonengine.com/pun/pricing

Procedural-Worlds 2023a. Viitattu 11.10.2023
https://www.procedural-worlds.com/products/indie/gaia/

Procedural-Worlds 2023b. Gaia Water Settings. Viitattu 01.11.2023
https://canopy.procedural-worlds.com/library/tools/gaia-pro-2021/written-articles/creating_runtime/1d-gaia-water-settings-r63/

Statcounter2023a. Screen resolution stats worldwide. Viitattu 18.11.2023
https://gs.statcounter.com/screen-resolution-stats

Unity 2022.3a. WebGL browser compatibility. Viitattu 05.10.2023
https://docs.unity3d.com/2023.3/Documentation/Manual/webgl-browsercompatibility.html

Unity 2022.3b. Textures. Viitattu 13.10.2023
https://docs.unity3d.com/Manual/Textures.html#:~:text=You%20should%20make%20your%20textures,appear%20in%20the%20Project%20View.&text=See%20in%20Glossary%20or%20Normalmap,of%20applications%20in%20the%20game.

Unity 2022.3c. Creating and editing Terrains. Viitattu 11.10.2023
https://docs.unity3d.com/Manual/terrain-UsingTerrains.html

Unity 2022.3d.Shader Performance. Viitattu 21.10.2023
https://docs.unity3d.com/Manual/SL-ShaderPerformance.html

UnityToS 6.11.2023a. Unity Editor Software Terms. Viitattu 30.11.2023
https://unity.com/legal/editor-terms-of-service/software

UnrealEngineToS 6.11.2023a. Unreal Engine End User Lisence Agreement. Viitattu 30.11.2023
https://www.unrealengine.com/en-US/eula/unreal

Unknown Cheats (23.01.2022). Cheat table. Viitattu 24.09.2023
https://www.unknowncheats.me/forum/grand-theft-auto-v/487033-wicked-menu-cheat-engine.html