Tampereen ammattikorkeakoulu

Unity-pelimoottorilla toteutetun 3D-ympäristön yhdistäminen verkkosovellukseen

Eetu Hentunen

OPINNÄYTETYÖ
Toukokuu 2025

Tietojenkäsittelyjen tutkinto-ohjelma
Games Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Games Production

HENTUNEN, EETU:
Unity-pelimoottorilla toteutetun 3D-ympäristön yhdistäminen verkkosovellukseen

Opinnäytetyö 31 sivua, joista liitteitä 0 sivua
Toukokuu 2025

Opinnäytetyössä tarkasteltiin kehittämisprojektia, jonka tavoitteena oli tuottaa interaktiivinen 3D-ympäristö osaksi selainpohjaista pelillistämistämisalustaa. Projektin ensimmäisessä vaiheessa todistettiin konseptin kelpoisuus siirtämällä alustan pelidataa Unityn ja asiakasyrityksen palvelimen välillä hyödyntämällä palvelimen ohjelmointirajapintaa ja Unityn sisäänrakennettuja verkko-ominaisuuksia. Toisessa vaiheessa projektin tavoitteeksi otettiin toteuttaa pienin toimiva tuote, jonka myötä saatiin lisää näyttöä konseptin skaalautuvuudesta sekä kaupallisesta potentiaalista.

Työssä kuvailtiin projektin etenemistä ja toteutusta näiden kahden kehitysvaiheen löydösten pohjalta. Lopullisessa toteutuksessa pelillistämisalustan selainkäyttöliittymä piirrettiin Unity-sovelluksen päälle ja kumpikin sovellus suoritettiin rinnakkain samassa selainikkunassa. Tällä tavoin Unityn ja verkkosovelluksen integraatio skaalautui hyvin ja se on yleistettävissä moniin erilaisiin Unityllä tuotettuihin pelityyppeihin. Työssä esiteltiin integraation taustalla olevia tekniikoita, sekä sen tuottamia haasteita ja niihin löydettyjä ratkaisuja.

Toteutuksen mahdollistaa Unity WebGL -alusta, jonka ansiosta pelejä voidaan kohdentaa verkkoselaimelle. Tämän lisäksi Unity tarjoaa työkaluja, joiden avulla voidaan viestiä selaimen JavaScript-moottorin ja C#-peliskriptien välillä. WebAssembly-standardin myötä selaimessa suoritettavat työpöytäsovellukset ovat yleistyneet, mutta vaikuttaa siltä, että Unityä käytetään edelleen verkkoalustalla lähinnä videopelien tuottamiseen. Tämän opinnäytetyön esittelemät tekniikat tarjoavat kuitenkin yhden tavan tuoda Unityn pelilliset ominaisuudet osaksi laajasti saavutettavaa hyötysovellusta.

Asiasanat: Unity, WebGL, WebAssembly, JavaScript, verkkosovellus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Games Production

HENTUNEN, EETU:
Integrating a 3D Environment Developed in the Unity Game Engine with a Web Application

Bachelor's thesis 31 pages, appendices 0 pages
May 2025

The thesis describes a development project that had the goal of producing an interactive 3D environment as a part of a browser-based gamification platform. The project started with a proof-of-concept phase in which game data from the platform was exchanged between Unity game engine and the platform backend using the backend API and Unity's built-in networking features. In the second phase, the goal was to create a minimum viable product that would lead to more insight into the concept's scalability and commercial potential.

The thesis follows the progress and implementation of the project through the findings of these development phases. In the final implementation the browser user interface of the gamification platform is rendered and overlayed on top of the Unity app and both applications are run together on the same browser tab. This allows the integration to be scalable and generalizable for many different game types made with Unity. The thesis explains the techniques used for this implementation and explores emergent challenges and resolutions.

Key words: Unity, WebGL, WebAssembly, JavaScript, web application

SISÄLLYS

ERITYISSANASTO

assetti	pelituotannossa käytetty resurssi
commit	versionhallintaan tehty muutos
GameObject	Unityn yleisluokka, johon kaikki pelisisältö kuuluu
hostaaminen	verkkosivun tallentaminen palvelimelle
MVP	minimum viable product, pienin toimiva tuote
POC	proof of concept, konseptitodistus
scene	Unity-assetti, johon tuotetaan pelisisältöä

1 JOHDANTO

Tässä opinnäytetyössä kuvataan kehittämisprojektia, jonka tarkoituksena oli toteuttaa kolmiulotteinen käyttöliittymä, tai käyttäjäkokemus, selainpohjaiselle pelillistämisalustalle. Alustalla on mahdollista luoda pelejä erilaisten koulutus- ja opetusmateriaalien pohjalta. Asiakasyritys, jonka tuote pelillistämisalusta on, oli toteuttanut ennen projektia pienimuotoisen testiprojektin, jossa alustan pelidataa oli haettu Unity-pelimoottoriin alustan palvelinohjelmointirajapintaa (engl. backend API) hyödyntäen. Asiakasyritys asetti tämän kehittämisprojektin tavoitteeksi toteuttaa konseptitodistuksen (engl. proof of concept, POC), jossa käyttäjä pystyisi tutkimaan Unityssa luotua 3D-maailmaa ja olemalla vuorovaikutuksessa tämän maailman kanssa, avaamaan sekä suorittamaan pelillistämisalustan harjoitteita. Konseptin tarkoituksena oli tuoda pelillistämisalustan peleihin syvempää interaktiivisuutta, sekä mahdollistaa vaikuttavampi tarinallisuus – molemmilla ominaisuuksilla ollen pedagogisia hyötyjä.

Projektin proof of concept -vaihe todettiin menestykseksi, minkä johdosta kehitystyötä jatkettiin. Kehittämisprojektin seuraava vaihe oli luoda pienin toimiva tuote (engl. minimum viable product, MVP). Tämän vaiheen aikana tulisi ratkaista POC:n aikana tunnistetut ongelmat, todentaa projektin skaalautuvuus sekä testata sen kaupallista potentiaalia. MVP:n tavoitteeksi asetettiin luoda kolme uutta 3D-maailmaa, joilla toisistaan poikkeavat teemat sekä muutamia yksilöllisiä toiminnallisuuksia. Tämän lisäksi pelimoottorin ja selainpohjaisen pelillistämisalustan integraatiota päätettiin muuttaa siten, että alustan käyttöliittymä ja toiminnallisuus piirrettäisiin Unity-sovelluksen päälle samassa selainikkunassa.

Edellä mainittu toteutus ja siihen liittyvien haasteiden ratkaiseminen on tämän opinnäytetyön pääasiallinen tarkastelukohde. Aloitan kuvailemalla teknologioita, joiden avulla Unity-sovellukset on mahdollista julkaista verkossa, ja joita hyödyntämällä eri sovellukset voivat viestiä keskenään. Tämän jälkeen avaan projektin alussa POC-vaiheessa tehtyjä havaintoja, joiden pohjalta päädyttiin lopulliseen toteutukseen. Seuraavassa osiossa kuvailen kuinka pelillistämisalustan ja Unityn lopullinen integraatio toteutettiin ja avaan kehitystyön aikana ilmenneitä ongelmia

sekä niihin löydettyjä ratkaisuja. Lopuksi käyn läpi projektin MVP-vaiheen päättymisen jälkeen todettuja jatkokehitystarpeita sekä yleisiä haasteita ja mahdollisuuksia, jotka liittyvät Unityn käyttämiseen osana selainsovellusta. Opinnäytteen tarkoituksena ei ole kuvata 3D-ympäristöjen toteuttamista tai tuotetun sovellusintegraation käyttäjäkokemusta tai muotoilua, vaan näkökulma on rajattu integraation tekniseen taustaan.

Vaikka Unity tarjoaa kattavat ja hyvin dokumentoidut ominaisuudet pelimoottorin ja selaimen väliseen viestimiseen, projektin ja tämän opinnäytteen taustoituksen aikana oli haastavaa löytää vastaavia toteutuksia, joissa Unity olisi tiukasti integroitu toiseen sovellukseen. Siinä missä tuottavuuskäyttöön tarkoitetut työpöytäsovellukset ovat yleistyneet verkossa WebAssemblyn myötä, Unity vaikuttaa olevan vielä niukasti käytetty muuhun, kuin selaimessa julkaistuihin videopeleihin. Tämän projektin toteutus kuitenkin avartaa ikkunaa monenlaisiin pelillisiin integraatioihin hyötysovellusten kanssa.

2 KEHITTÄMISPROJEKTISSA KÄYTETYT TEKNOLOGIAT

2.1 UnityWebRequest ja REST-ohjelmointirajapinnat

Unity Manualin (2025) mukaan UnityWebRequest on Unityyn sisäänrakennettu modulaarinen systeemi, jonka avulla on mahdollista muodostaa HTTP-kutsuja sekä käsitellä HTTP-vastauksia. Sen tarkoitus on tarjota peleille mahdollisuus olla vuorovaikutuksessa palvelimien kanssa. UnityWebRequest tarjoaa kehittyneitä ominaisuuksia, joilla voi kehittäjät pystyvät hallitsemaan HTTP-otsakkeita ja -verbejä. (UnityWebRequest 2025) HTTP-verbeillä tarkoitetaan HTTP-protokollan määrittämiä pyyntömetodeja, jotka täsmentävät, mikä toiminto pyydetylle resurssille tahdotaan suorittaa. Yleisimpiä metodeja ovat GET, POST, PUT, PATCH ja DELETE. GET-metodi on pääasiallinen keino tiedon hakemiseen palvelimelta. Se noutaa halutun resurssin URL:n perusteella ja eikä muuta kyseisen resurssin tilaa. Näin ollen GET:iä pidetään turvallisena metodina. POST-metodi taas sisältää dataa pyynnön keho-osassa (engl. request body) ja tämä data on luultavimmin tarkoitettu tallennettavaksi palvelimelle. (Explain Request Verbs 2021) Tämän kehittämisprojektin piirissä vain GET- ja POST-metodit tulivat tarpeellisiksi.

HTTP-otsakkeet ovat pyyntöihin ja vastauksiin sisältyviä avain-arvo-pareja, jotka antavat tietoa asiakkaan ja palvelimen välisestä kommunikaatiosta. Otsakkeet voivat sisältää esimerkiksi tiedot käytetystä koodauksesta, sisällön tyypistä sekä autentikoinnista. (HTTP headers 2024) UnityWebRequest-toiminnallisuus mahdollistaa mukautettujen otsakkeiden lisäämisen pyyntöihin. Unityn systeemin toiminta on jaettu kolmeen osaan: pyyntöjen lähettämiseen, vastausten vastaanottamiseen ja HTTP-liikenteen hallitsemiseen. Jokaisesta kolmesta osa-alueesta vastaa oma olionsa. DownloadHandler vastaa saapuvan tiedon käsittelystä, kun taas UploadHandler vastaa palvelimelle lähetettävästä tiedosta. Näitä kahta oliota sekä HTTP-liikennettä hallinnoi UnityWebRequest-olio, johon määritellään pyynnön URL ja mukautetut otsakkeet. Se säilöö myös tiedon virheistä ja uudelleenohjauksesta. (UnityWebRequest 2025)

Käyttääkseen resursseja palvelimelta tai toisesta sovelluksesta kehittäjien on hyödynnettävä niiden ohjelmointirajapintoja (engl. Application Programming Interface, API). Korhosen (2018) mukaan API:llä voidaan tarkoittaa mitä tahansa rajapintaa – ohjelman sisäistä tai ulkoista. Ulkoiset rajapinnat julkaistaan käyttämällä web-palveluita kuten SOAP ja REST. (Korhonen 2018) Gupta (2025) kuvaa REST-ohjelmointirajapintoja (engl. REST API tai RESTful API) yleisimmäksi tavaksi toteuttaa ohjelmointirajapinta verkossa. Nimi on lyhenne sanoista Representational State Transfer. REST ei ole protokolla tai standardi, vaan ohjelmistoarkkitehtuurinen tyyli, joka sisältää ohjenuoria ja rajoitteita. Toteutustavat voivat erota, mutta näitä ohjenuoria noudattamalla ohjelmointirajapintaa voidaan kutsua REST-nimellä. (Gupta 2025) REST API:t käyttävät HTTP-protokollan metodeja määrittämään resursseille tehtäviä toimintoja ja ne palauttavat vastauksena dataa tyypillisimmin JSON-muodossa (engl. JavaScript Object Notation). (REST API Introduction 2025)

REST-arkkitehtuuri tarjoamia etuja ovat skaalautuvuus, joustavuus ja itsenäisyys. REST on tallenna vuorovaikutuksen tilaa, mikä tarkoittaa sitä, että jokainen vuorovaikutustapahtuma käsitellään itsenäisesti ilman, että aiemmat tapahtumat vaikuttavat siihen. Tilattomuus yhdistettynä asiakkaan ja palvelimen eriyttämiseen pitävät järjestelmän skaalautuvana ja joustavana. Ei ole myöskään väliä millä ohjelmointikielillä asiakas- ja palvelinpuoli ovat toteutettu, sillä niiden välinen vuorovaikutus RESTillä pysyy täysin itsenäisenä. (Amazon Web Services n.d.)

2.2 Unity WebGL ja WebAssembly

WebGL (Web Graphics Library) on JavaScript-ohjelmointirajapinta, joka mahdollistaa 3D- ja 2D-grafiikan piirtämisen HTML-canvas-elementille. Tämä on mahdollista ilman ulkoisia selaimen lisäosia. WebGL mukailee tarkasti OpenGL ES 2.0 standardia, minkä vuoksi se pystyy hyödyntämään käyttäjän laitteen grafiikkaprosessoria (engl. graphics processing unit, GPU). (WebGL: 2D and 3D graphics for the web 2024) Tapala (2025) kertoo, että WebGL mahdollistaa myös ohjelmoinnin GLSL-kielellä, jonka avulla kehittäjät voivat kirjoittaa shader-ohjelmia, eli ohjelmia, jotka suoritetaan grafiikkaprosessorilla (Tapala 2025). Toisin kuin monesti ajatellaan, WebGLFundamentals-verkkosivu (n.d.) huomauttaa,

että WebGL ei ole 3D API. Sivusto argumentoi WebGL:n olevan ennemminkin rasterointimoottori, sillä se ei tarjoa kehittäjälle työkaluja varsinaisten 3D-objektien, valaistuksen ynnä muun grafiikan luomiseen, kuten esimerkiksi three.js-kirjasto. Se on matalamman tason rajapinta, jolle on tarjottava kompleksia 3D-matematiikkaa sisältävät ohjeet. (WebGLFundamentals n.d.)

Unity-kehittäjänä ei ole kuitenkaan tarpeellista tuntea shader-ohjelmien kirjoittamista käyttääkseen Unityn WebGL-alustaa. Unity Manualin (2021) mukaan alusta käyttää HTML5/JavaScriptiä, WebAssemblyä, WebGL:ää sekä muita webstandardeja, jotta Unityllä kehitettyjä pelejä voitaisiin julkaista verkossa selaimella käytettäväksi (Getting started with WebGL development 2021). Unity suorittaa sovelluksen rakentamisen näille teknologioille ilman kehittäjän erityisosaamista. Tästä huolimatta erityisesti HTML- ja JavaScript-tuntemus ovat eduksi, sillä näiden teknologioiden avulla määritellään, kuinka rakennettu peli esitetään selaimessa ja kuinka se voi olla vuorovaikutuksessa selaimen kanssa.

WebAssembly on Assembly-ohjelmointikieltä muistuttava kieli, jota voidaan suorittaa moderneissa verkkoselaimissa. WebAssemblyn kompakti binäärimuoto mahdollistaa lähes natiivin suorituskyvyn selaimessa. Kehittäjien ei tarvitse osata kirjoittaa koodia suoraan WebAssemblylle, sillä muita ohjelmointikieliä, kuten C/C++:aa ja Rustia, on mahdollista kääntää WebAssembly-konekieleksi. Webstandardeja ylläpitävä W3C (The World Wide Web Consortium) kehittää WebAssemblyä standardiksi, millä on ollut merkittäviä vaikutuksia verkkokehitykselle. Sen myötä on ollut mahdollista tuoda verkkoalustalle kokonaisia sovelluksia, joita on aiemmin voinut käyttää vain natiivisti laitteelle asennettuina. (MDN 2025)

Unity WebGL -alustalle rakentaminen tapahtuu hyvin yksinkertaisesti valitsemalla kyseinen alusta editorin Build Settings -valikosta. Unityn suorituksen aikainen runtime-koodi on C/C#-koodia, joka käännetään WebAssembyyn emscripten-kääntäjällä. .NET-pelikoodi, eli pelikehittäjän tuottamat C#-skriptit, käännetään ensin vastaaviksi C++-lähdetiedostoiksi käyttäen Unityn kehittämää IL2CPP-teknologiaa (Intermediate Language To C++) ja niin edelleen emscriptem kääntää nämä tiedostot WebAssemblyksi. (Getting started with WebGL development 2021)

3 KEHITTÄMISPROJEKTIN PROOF OF CONCEPT -VAIHE

Kehittämisprojektin tarkoituksena oli tuottaa asiakasyritykselle Unity-pelimoottorissa 3D-pelimaailma, joka toimisi interaktiivisena käyttöliittymänä yrityksen verkkosovellukselle. Asiakasyrityksen tuote on pelillistämisalusta, jonka avulla pystyy luomaan pelejä erilaisista koulutusmateriaaleista. Alustalla luoduissa peleissä käyttäjä avaa harjoitteita kaksiulotteisella pelilaudalla ja kerää pisteitä niitä suorittamalla. Automaattisen pisteytyksen lisäksi pelin laatinut ohjaaja voi arvioida ja antaa palautetta pelaajille - tarvittaessa reaaliaikaisesti. Pelien harjoitteet koostuvat erilaisista tekstipohjaisista tehtävistä, kuten monivalintatehtävä, ’yhdistä parit’ ja ’täydennä puuttuva sana’. Tekstin lisäksi tehtävänannot sekä vastaukset voivat sisältää myös kuvia, videota ja ääntä.

Kehittämisprojektin taustalla oli asiakasyrityksen halu lisätä pelien interaktiivisuutta sekä avata uusia mahdollisuuksia tarinankerronnallisuudelle. Ensimmäisenä tavoitteena oli toteuttaa kolmiulotteinen ympäristö, jota käyttäjä pystyy tutkimaan ensimmäisen persoonan perspektiivistä. Ympäristöä tutkiessaan käyttäjän tulisi pystyä avaamaan ja suorittamaan pelillistämisalustalla luotuja harjoitteita samalla tavalla kuin kaksiulotteisella pelilaudallakin. Käyttäjän antamien vastausten tuli myös tallentua alustalle ohjaajan arviointia ja palautetta varten.

Projektin ensimmäisenä tavoitteena oli tuottaa proof of concept (POC, suom. konseptitodistus), jonka tarkoituksena oli todentaa, että Unityyn on mahdollista tuoda pelidataa asiakasyrityksen palvelimelta ja niin ikään lähettää käyttäjän vastaukset takaisin palvelimelle. Tässä ensimmäisessä vaiheessa tutkittiin myös, kuinka harjoitteet tulisi esittää käyttäjälle ja millaisilla interaktiivisilla tavoilla ne olisivat avattavissa. Vaikka projektin alussa kehitystyö tapahtui natiivilla Windows-alustalle, oli tiedossa, että ympäristön tulisi olla käytettävissä ensisijaisesti verkkoselaimella, joten kaiken toiminnallisuuden tulisi olla tuettua myös WebGL-alustalla.

Unity tarjoaa olemassa olevat työkalut http-pyyntöjen lähettämiseen ja vastausten käsittelyyn, joten palvelimeen kanssa viestiminen ei oletetusti tuottanut haas-

teita. Datan muuntaminen palvelimen käyttämistä heikosti tyypitetystä tietotietorakenteista C#:n vahvasti tyypitettyyn sen sijaan aiheutti paljon poikkeuksellisia tilanteita, joissa yksittäiset avain-arvo-parit vaativat erityistä logiikkaa datan eheänä säilymiseksi. Tätäkin merkittävämpi haaste muodostui käyttöliittymän toteuttamisesta. Projektin tarkoituksena oli tuoda pelillistämisalustan verkkosovelluksen toiminnallisuus interaktiivisempaan 3D-ympäristöön, mutta tämä tarkoitti sitä, että kaikki sovelluksen vuosien mittaan kehitetyt ominaisuudet ja käyttöliittymät tulisi rakentaa vastaavanlaisina Unityn sisällä. Tämä olisi äärimmäisen resurssi-intensiivinen prosessi, joka vaatisi jatkuvaa ylläpitämistä, sillä verkkosovellukseen lisätään jatkuvasti uusia ominaisuuksia ja sen käyttäjäkokemusta kehitetään. Tämä lähestymistapa edellyttäisi, että kaikki verkkosovelluksen käyttäjäpuoleen kohdistunut kehitystyö tulisi tehdä aina toistamiseen Unityssä täysin eri työkaluilla, mikä lopulta sai projektitiimin kokeilemaan toista toteutustapaa.

3.1 Pelillistämisalustan palvelinrajapinnan hyödyntäminen

Pelillistämisalustalla luotujen pelien data haetaan Unityyn hyödyntämällä asiakasyrityksen palvelinohjelmointirajapintaa (engl. backend API). Unity tarjoaa sisäänrakennettuna ominaisuutena UnityWebRequest-moduulin, jonka avulla on mahdollista lähettää http-pyyntöjä ja vastaanottaa palvelimen vastauksia. Ohjelmointirajapinoille tyypilliseen tapaan (GeeksforGeeks 2025) asiakasyrityksen API palauttaa pelaajadatan JSON-merkkijonona. Jotta JSON-data saadaan käyttöön Unityn komponenteille, se on muunnettava C#-tietorakenteiksi. Tämän projektin tapauksessa palvelimelta saatava data sisältää toisinaan erittäin suuren määrän kenttiä, jotka voivat olla sisäkkäisiä ja pitää arvoinaan pitkiä merkkijonoja. Tässä tapauksessa oli hyödyllistä käyttää muuntajaohjelmaa, joka generoi syötetystä JSONista automaattisesti C#-luokat vastaavilla jäsenmuuttujilla. Käyttämäni muuntajaohjelma löytyy verkosta osoitteesta www.json2csharp.com.

Kun palvelinpuolta vastaava tietorakenne on toteutettu Unityssä, vastaanotettu JSON-data tulee jäsentää C#-olioiksi ohjelman suorituksen aikana. Unityyn on sisäänrakennettu tähän tarkoitukseen JsonUtility-luokka, joka on kuitenkin jokseenkin rajallinen ja tämän projektin erityispiirteitten takia itsessään riittämätön.

Unityn dokumentaatio (2025) kertoo, että JsonUtilityn FromJson-metodi hyödyntää Unityn serialisoijaa, joka tukee rajattua määrää tyyppejä. Hakemistot ovat esimerkki tietotyypistä, jota ei tueta (JSON Serialization 2025) ja joita asiakasyrityksen palvelinpuolen data sisältää. Data sisältää myös olioita toisten olioiden sisällä (engl. nesting). Keskustelu Unityn keskustelupalstalla (2023) paljastaa, että JsonUtility on rajattu kymmeneen tasoon sisäkkäisiä olioita (Disabling JsonUtility Serialization Depth Limit of 10 2023), mutta tämän projektin tapauksessa moniosaisten tehtävien vastauskentät saattoivat ylittää tuon syvyyden. Lisäksi tehtävätyypin mukaan samassa avainkentässä voidaan tarjota erityyppisiä mukautettuja tietotyyppejä merkkijonoina, eikä ohjelman toiminnallisuuden kannalta olisi mielekästä säilöä tätä dataa sellaisenaan.

Edellä luetelluiden syiden takia, Unityn oman JSON-serialisoijan rinnalla käytettiin ulkoista Newtonsoft Json -pakettia. Newtonsoftin serialisoija mahdollistaa JSON-datan deserialisoimisen JObject-tyyppiseksi JSON-olioksi. JObject-luokka perii Newtonsoft.Json.Linq-nimiavaruudesta, joka tarjoaa C#:n LINQ-kirjaston toiminnalisuuden JSON-datan käsittelyyn (Newtonsoft n.d.). Microsoftin dokumentaatio (2023) kertoo, että LINQillä viitataan kokoelmaan teknologioita, jotka integroivat kyselyjen teon suoraan C#:iin. Perinteisiin kyselykieliin verrattuna tämä suora integraatio tarjoaa vahvan tyypityksen sekä IntelliSense-tuen. (Microsoft 2023) JObject-oliosta on helppoa kysellä täsmällisesti haluttuja avainkenttiä syvältä datan monista kerroksista. Avainkentät ovat niin ikään deserialisoitu JToken-olioiksi, jotka voidaan muuttaa takaisin JSON-merkkijonoksi ja jäsentää vastaaviin C#-olioihin käyttämällä joko Unityn JsonUtilityä tai Newtonsoftin Json-liitännäistä. Tällä tavoin voidaan esimerkiksi täsmentää datasta tehtävän tyyppi ennen jäsentämistä ja valita oikea tietotyyppi, johon jäsentäminen suoritetaan.

3.2 Selainsovelluksen käyttöliittymän toisintaminen Unityssä

Pelillistämisalustan selaimella käytettävässä pelinäkymässä käyttäjä avaa harjoitteita klikkaamalla karttapinniä muistuttavia ikoneja. Harjoitteet avautuvat pelilaudan päälle erillisessä modaalissa, joka sisältää harjoitteet tehtävät. 3D-maailmassa pelaaja tutkii ympäristöä ensimmäisestä persoonasta ja klikkaa samalla

tavoin esineitä, joiden yläpuolella leijuu karttapinni-ikoni. POC-vaiheessa tavoitteena oli toteuttaa kaikille kuudelle eri tehtävätyypille selainsovellusta vastaava käyttöliittymä Unityn käyttöliittymätyökaluilla. Median tukeminen tehtävänannoissa sekä käyttäjän vastauksissa rajattiin tässä kohtaa tavoitteiden ulkopuolelle.

Pelillistämisalustan selainsovellus on toteutettu käyttämällä React -JavaScript-kirjastoa. React on suunniteltu dynaamisten ja interaktiivisten käyttöliittymien toteuttamiseen, minkä lisäksi se yksinkertaistaa yksisivuisten verkkosivujen luomista ja ylläpitämistä (React Introduction 2025). Toisin kuin React ja muut perustavammat web-teknologiat kuten CSS ja HTML, Unity UI -käyttöliittymätyökalu ei ole lähtökohtaisesti suunniteltu skaalautuvien ja dynaamisten dokumenttien suunnitteluun. Videopelien valikot ja HUD-näkymät (heads-up-display) ovat verrattain staattisia eikä niillä ole samanlaista tarvetta mukautua käyttäjän laitteeseen sovelluksen suorituksen aikana. Web-teknologiat piirtävät sivun hyödyntäen vektorigrafiikkaa, kun taas Unityn käyttöliittymätyökalu käyttää bittikarttoja. Bittikartat yhdessä animaatiotyökalun kanssa antavat suuremman taiteellisen vapauden, joka soveltuu fantastisten ja tyyliteltyjen käyttöliittymien suunnitteluun, mutta lähestymistapa on työläämpi, se vaatii paljon editorinäkymän sekä ulkoisten työkalujen käyttämistä eikä ole yhtä vahvasti hallittavissa koodista käsin. Unityyn kehitetään uutta UI Toolkit -systeemiä, joka pohjautuu samaan dokumenttilähtöiseen lähestymistapaan kuin web-teknologiat ja on siten luultavimmin tutuntuinen web-suunnittelijoille. Systeemiä kuitenkin kehitetään edelleen ja se ei sisällä toistaiseksi kaikkia vanhan Unity UI -systeemin ominaisuuksia (Comparison of UI systems in Unity 2025). Tästä syystä, sekä koska vanha systeemi on itselleni tuttu, toteutus tehtiin vanhalla Unity UI -systeemillä.

POC-tuotokseen rakennettiin onnistuneesti oma lomake tai käyttöliittymä jokaiselle kuudelle tehtävätyypille. Lopputulos oli hyvä, mutta käyttöliittymän skaalaaminen tehtäväsisällön mukaan, sekä palautelaatikon ja annettujen pisteiden sulava animoiminen tuotti merkittävän paljon vaikeasti muokattavaa ja ylläpidettävää koodia. Kun POC-tavoitteisiin oli päästä, koodipohja koettiin jo niin kompleksiseksi, että tulevien ominaisuuksien lisääminen olisi pahentanut tilannetta enti-

sestään. Asiakasyrityksen toiveena oli erityisesti kuvien ja videoiden sisällyttäminen, mikä olisi jälleen vaatinut oman logiikan erilaisten kuvasuhteiden skaalaamiseen sopivalla tavalla.

Toinen ilmeinen haaste web-kehityksen ja pelimoottorin vuoropuhelussa muodostui HTML-muotoilutunnisteista. Palvelimelta saatavan datan tekstikentät ovat HTML-koodattuja. Unityn TextMesh Pro -paketti tukee rikasta tekstiä, mutta tunnisteet sekä niiden attribuutit eivät vastaa HTML-tunnisteita. Tunnisteiden muuntaminen olisi vaatinut oman erityisen logiikkansa, joka sekin olisi sisältänyt useita erityistapauksia esimerkiksi linkkien käsittelyyn liittyen. Vaikka kyse ei ollut merkittävästä ongelmasta, se loi jälleen kitkaa kahden erilaisen kehitysympäristön välille ja osoitti kuinka Unity ei ole erityisen yhteensopiva työkalu web-kehityksen kanssa.

Myös tekstin piirtäminen näytölle pelimoottorissa poikkeaa selaimesta. Selaimet käyttävät kirjasintiedostoja, jotka esittävät merkit vektorimuodossa ja ne ovat näin ollen skaalattavissa mihin tahansa kokoon. TextMesh Pro sen sijaan generoi kirjaisintiedoston pohjalta atlakseksi kutsutun bittikartan, joka sisältää tarvittavat merkit. Projektissa käytettiin atlasta, jonka suuruus on 512 kertaa 512 pikseliä, mikä riittää esittämään laajennetut latinalaiset aakkoset hyvällä tarkkuudella. Suurentamalla atlaksen resoluutiota on mahdollista esittää merkit paremmalla tarkkuudella tai sisällyttää suurempi määrä merkkejä, mutta samalla suurempi tiedostokoko lisää muistinkäyttöä. Erityisesti lihavoidusta tekstistä kuitenkin huomaa, ettei lopputulos ole yhtä siisti kuin vektorimuodosta piirretty teksti. Laajennetut latinalaiset aakkoset ovat todella kattavia, mutta esimerkiksi arabiankielisille aakkosille tulisi generoida oma atlas.

3.3 Proof of concept -vaiheen johtopäätökset

Projektin ensimmäisessä vaiheessa välitettiin onnistuneesti tietoa pelimoottorin ja palvelimen välillä. Unityllä luotiin näyttävä 3D-ympäristö, jossa käyttäjä pystyi liikkumaan vapaasti ja avaamaan pelillistämisalustan harjoitteita. Harjoitteiden suorittaminen sai aikaan muutoksia ympäristössä, mikä lisäsi interaktiivisuuden ja toimijuuden tuntua. Projektitiimi koki, että konseptissa oli potentiaalia ja että

videopelimäiset ympäristöt mahdollistaisivat merkityksellisemmän tarinankerronnan alustalla luoduissa peleissä, mikä osaltaan syventäisi oppimista. Tämä kyseinen ympäristö sopisi kuitenkin vain yhdenlaiseen narratiiviin ja teemaan, joten konseptin pitäisi olla yleistettävissä erilaisiin 3D-maailmoihin. Todettiin myös, että selainsovelluksen käyttöliittymän ja sen toiminnallisuuden toistaminen Unityn sisäisesti vaatisi valtavasti resursseja välittömästi sekä jatkuvana ylläpitämisenä eikä lopputulos olisi koskaan yhtä hyvä, kuin Reactilla rakennettu selainversio. Jatkokehitystä varten olisi etsittävä toinen ratkaisu.

4 KEHITTÄMISPROJEKTIN MINIMUM VIABLE PRODUCT -VAIHE

4.1 Selainsovelluksen käyttöliittymä Unity-sovelluksen rinnalla

Proof of concept -vaiheessa pelillistämisalustan yhdistäminen Unityyn ja 3D-ympäristöön oli todennettu toimivaksi konseptiksi ja sen hyödyt oli tunnistettu. Projektin seuraavaksi tavoitteeksi asetettiin tuottaa pienin toimiva tuote (engl. minimum viable product, MVP), jota voitaisiin tarjota yrityksen asiakkaiden käyttöön ja siten todentaa projektin kaupallinen potentiaali. Ennen kuin varsinainen jatkokehitys kohti pienintä toimivaa tuotetta voitiin aloittaa, oli ratkaistava kaksi edellisessä vaiheessa avoimeksi jäänyttä ongelmaa: projektin skaalautuvuus, sekä tehokkaampi käyttöliittymän integrointi.

Tässä vaiheessa projektia oli todettu, että Unity-sovelluksen tulisi oltava käytettävissä verkkoselaimessa aivan kuten pelillistämisalustan omat ohjaaja- ja pelisovelluksetkin. Käyttäjän tietokoneelle asennettavaa Unity-sovellusta ei pidetty käypänä vaihtoehtona, joten kehittäminen pystyttiin rajaamaan kokonaan selaimille tarkoitetulle WebGL-alustalle. Tämä rajaus nosti esiin mahdollisuuden suorittaa pelillistämisalustan selainsovellusta ja Unity-sovellusta yhtä aikaa samassa ikkunassa. Alustan olemassa olevassa käyttöliittymä valikkoineen ja pistetauluineen on piirretty erillisinä modaaleina kaksiulotteisen pelilaudan päälle. Nämä modaalit olisi yhtä lailla mahdollista piirtää myös 3D-ympäristön päälle, sillä WebGL piirtää Unity-instanssin tavalliselle HTML-canvas-elementille. Täytyi ainoastaan löytää keino kommunikoida Unity-instanssin ja pelillistämisalustan käyttöliittymän kesken.

WebGL-alustalle rakennettu Unity-sovellus generoi index.html-tiedoston, joka määrittää rakenteen verkkosivulle, jolla sovellus esitetään. Tätä index-tiedostoa on mahdollista muokata kuten mitä tahansa verkkosivua ja muokatun tiedoston pystyy asettamaan Unityn projektiasetuksissa mallipohjaksi, jota käytetään aina sovellusta rakentaessa. HTML-dokumentin body-osiossa pellistämisalustan käyttöliittymä lisätään ulkoisina JavaScript-skripteinä. Samalla canvas-elementti, jo-

hon Unity-sovellus piirretään, asetetaan resoluutioltaan koko dokumentin kokoiseksi, niin että se täyttää selainikkunan. Näin on onnistuttu näyttämään käyttäjälle kaksi erillistä sovellusta päällekkäin aseteltuna verkkosivulla.

4.2 Viestintä Unityn ja selaimen JavaScript-moottorin välillä

Jotta pelillistämisalustan käyttöliittymä ja Unity-instanssi pystyvät välittämään tietoa toisillee, Unityn on kyettävä kommunikoimaan selaimen JavaScript-moottorin kanssa. Unity tarjoaa valmiin toiminnallisuuden C#-metodien kutsumiseen JavaScriptistä. Kun Unity-instanssi on luotu in index-tiedostossa, tämä instanssi voidaan asettaa muuttujaan. Unityn dokumentaatio (2025) kertoo, että tämä olio sisältää SendMessage-funktion, jonka parametreiksi annetaan kohteena olevan GameObjectin nimi, kutsuttavan metodin nimi ja vaihtoehtoisesti yksi parametri kutsuttavalle metodille. Parametrejä voi olla vain yksi ja sen on oltava joko merkkijono tai numero. (Call Unity C# script functions from JavaScript 2025) Tässä projektissa sallittujen parametrien niukka määrä ei ollut rasite, sillä boolean-arvotkin on mahdollista esittää kokonaislukuina 0 ja 1.

C#-koodista JavaScriptille viestimiseen ei ole tarjolla yhtä valmista ratkaisua, mutta Unityn manuaalista (2025) löytyy kattava ohjeistus tämän toiminnallisuuden pystyttämiseen: Unityn projektikansioon tulee luoda JavaScript-liitännäinen, joka on tiedostotyypiä .jslib. Tähän tiedostoon nimetään ja määritetään JavaScript-funktioita dokumentaation määrittämällä syntaksilla. (Set up your JavaScript plug-in 2025) Kun funktiot ovat määritelty omaan kirjastoonsa, eli tarkemmin JavaScript-liitännäiseen, niitä voidaan linkittää C#-skripteihin DllImportAttribute-attribuuttiluokan avulla. Unityn dokumentaation tarjoamissa esimerkeissä attribuutin parametriksi syötetään merkkijono ”__Internal”. Dokumentaatiosta ei käy ilmi merkityksettömästi nimetyn parametrin tarkoitusta, mutta StackOverflow-keskustelu (2026) paljastaa, että tällä tavoin linkitys määritetään staattiseksi, oletusarvoisen dynaamisen linkityksen sijaan. Alustat kuten iOS ja Xbox sallivat ainoastaan staattisen linkityksen. (StackOverflow 2016) Linkitettyjä funktioita voi tämän jälkeen kutsua tavallisesti C#-koodin sisällä ja ne suoritetaan selaimen JavaScript-moottorissa. Tässä kohtaa sekä JavaScript-, että C#-skriptit pystyvät kutsumaan toistensa funktioita tai metodeja.

Pellistämisalistan ja Unityn vuoropuhelu toteutettiin tapahtumapohjaisesti. JavaScript-tapahtumien hyödyntäminen mahdollisti selkeän ja helposti ylläpidettävän rakenteen ja helpon tavan usealle kehittäjälle työskennellä yhdessä. Pelillistämisalustan Reactilla kehitetty käyttöliittymä ladataan html-tiedostossa ulkoisena JavaScriptinä. Asiakasyrityksen React-kehittäjä ohjelmoi käyttöliittymän lähettämään JavaScript-tapahtumia olennaisten tilamuutosten yhteydessä: esimerkiksi pelaajan kirjauduttua peliin ja harjoitusmodaalin sulkeutuessa. Unity-kehittäjänä lisäsin html-tiedostoon näitä tapahtumia kuuntelevat funktiot, jotka niin ikään lähettävät viestin Unityyn käyttäen Unity-instanssin SendMessage-funktiota. Jotta html-tiedosto pysyisi mahdollisimman selkeänä ja helposti ymmärrettävänä myös selainpuolen kehittäjille, kaikki SendMessage-funktiot kohdistuvat samaan GameManager-peliobjektiin. GameManager sisältää EventListener-komponentin, johon on määritelty kaikkia selainpuolen JavaScript-tapahtumia vastaavat C#-tapahtumat sekä -metodit.

Kertauksena, prosessi etenee seuraavasti: ulkoinen selainkäyttöliittymä lähettää tapahtumia, joita kuunnellaan verkkosivun html-dokumentissa. Tapahtumasta lähetetään tieto Unitylle SendMessage-funktiolla, jonka parametreina annetaan kohde-GameObject GameManager, sekä kutsuttavan metodin nimi, joka vastaa kyseistä tapahtumaa. Nämä metodit on määritetty EventListener-komponentissa ja ne laukaisevat JavaScript-tapajtumaa vastaavan C#-tapahtuman. Prosessin lopputuloksena JavaScript tapahtumat ovat ikään kuin muunnettu C#-tapahtumiksi. Unityn kommunikaatio selaimen suuntaan tapahtuu niin ikään lähettämällä JavaScript-tapahtumia JavaScript-liitännäisessä määritetyillä funktioilla, joita voidaan kutsua suoraan C#-koodista.

Tämä lähestymistapa sisältää monta välivaihetta, jotka olisivat karsittavissa siten, että SendMessages-funktiolla kohdistettaisiin suoraan tapahtumia kuluttavia komponentteja ja niiden metodeja, jotka suorittavat varsinaista pelilogiikkaa. Tästä huolimatta, tapahtumapohjaisen ohjelmointityylin jatkaminen natiiveihin C#-tapahtumiin asti erottaa kaksi eri ohjelmointikerrosta toisistaan hyvin selkeällä tavalla. Tällöin React-kehittäjä ja Unity-kehittäjä pystyvät kommunikoimaan erilaisista tapahtumista, joita heidän täytyy lähettää tai kuluttaa ja toinen osapuoli

pystyy toteuttamaan niitä natiivissa ympäristössään. Html-dokumentti ja JavaScript-liitännäinen ovat tällöin vain linkkejä, joiden ylläpito vaatii ainoastaan kuuntelijoiden lisäämisen samannimisiin funktioihin.

Uuden toteutustavan tehokkuudesta kertoo se, että käytettyjen tapahtumien määrä jäi hyvin suppeaksi, vain noin kymmeneen. Selainkäyttöliittymä ja 3D-ympäristö toimivat hyvin itsenäisinä kokonaisuuksina, jotka tarvitsevat vain muutamia yhteyspisteitä toistensa toimintaan. Selainkäyttöliittymä hallitsee käytännössä kaikkea viestintää palvelimen kanssa, mikä poisti valtavan määrän kitkaa palvelimen ja C#:n erilaisten tietorakenteiden välillä. POC-vaiheessa toteutetun Backend-komponentin rooli supistui lopulta pelkästään pelidatan hakemiseen pelaajan kirjautumistiedoilla. Tämäkin data olisi mahdollista välittää suoraan käyttöliittymältä, jolloin Unity ei vaatisi minkään laista verkkotoiminnallisuutta. Uudessa toteutuksessa koko kirjautumisprosessi käyttää pelillistämisalustan omaa kirjautumissivua ja mikäli kyseiseen peliin on lisätty 3D-maailman tunniste ohjaaja-asetuksissa, käyttäjä uudelleenohjataan verkkosivulle, jossa kyseinen 3D-maailma on hostattu ja kirjautumistiedot välitetään Unitylle tapahtuman avulla maailman alustamista varten. Varsinaisen pelin aikana yhteydenpito liittyy lähinnä harjoitteiden avaamiseen ja suorittamiseen. Kun käyttäjä klikkaa 3D-maailmassa harjoitetta, lähetään tapahtuma, jonka parametrina on harjoitteet tunniste. Tämä tapahtuma avaa käyttäjälle tehtävämodaalin. Tehtävät suoritettuaan selainkäyttöliittymä lähettää tapahtuman, jonka parametrina kulkee päivitetty tehtävädata. Tämän tapahtuman perusteella Unity-sovellus tietää palata takaisin pelimoodiin ja saa ajantasaisen kopion palvelimen tehtävädatasta.

4.3 Pelidatan hyödyntäminen Unityssä

MVP-vaiheen tavoitteeksi asetettiin luoda kolme uutta maailmaan ensimmäisen rinnalle. Jokainen maailma liittyy erilaiseen teemaan ja niihin erilaisia maailmakohtaisia ominaisuuksia, jota ovat riippuvaisia pelillistämisalustan datasta. Suurinta osaa pelidatasta ei hyödynnetä Unityssä, mutta se tallennetaan kokonaisena mahdollisia jatkokehityskohteita varten. Unity tarvitsee sellaista pelidataa, joka vaikuttaa käyttäjän edistymiseen 3D-ympäristössä. Lisäksi osa datasta voidaan ilmaista osana ympäristöä. Olennaisimpia tietoja ovat harjoitteiden määrä

ja järjestys, niistä saatavat maksimipisteet sekä tehtävätyypit. Selainkäyttöliittymä ja palvelin ylläpitävät tietoa pelin ja harjoitteiden tilasta, mutta koska juuri harjoitteet vaikuttavat 3D-ympäristöön, niille toteutettiin oma Unityyn oma Exercise-komponentti. Tämä komponentti säilöö kopiota harjoitteet tilasta ja paljastaa tärkeimmät tiedot muiden komponenttien käytettäväksi.

Maailmoissa käyttäjän etenemistä rajataan, kunnes tämä on suorittanut vaaditut harjoitteet. Kun harjoite on suoritettu, sillä on aina jonkinlainen vaikutus ympäristöön. Usein nämä ovat animaatioita tai partikkelisysteemien käynnistämistä ja joissakin tapauksissa animaatiot avaavat käyttäjälle reitin eteenpäin. Vaikutus saattaa myös määräytyä käyttäjän keräämien pisteiden pohjalta. Kaikki nämä toiminnallisuudet kuuntelevat tapahtumaa jonka Exercise-komponentti lähettää, saatuaan itse harjoitteen suorittamisesta kertovan tapahtuman selainkäyttöliittymältä. Tapahtuman rekisteröimisen jälkeen eri komponentit voivat tarkastella kuinka moni Exercise-komponenteista on ”suoritettu”-tilassa ja kuinka paljon niille on rekisteröity pisteitä.

Pelidataa itsessään voidaan myös sisällyttää 3D-maailmaan. Kun pelaaja lähestyy tehtäväpistettä, hänelle avautuu leijuva kehote, johon kirjataan harjoitteen tehtävätyyppi ja maksimipistemäärä. Erääseen maailmaan sisällytettiin videoruutuja, joilla on mahdollista näyttää pelaajan lähettämiä videovastauksia. Tämä oli ensimmäinen hyvin konkreettinen tapa muokata ympäristöä pelillistämisalustan pelidatan pohjalta, mutta kehitystiimi tunnisti välittömästi monia tulevaisuudessa tavoiteltavia ominaisuuksia. Käyttäjä voisi esimerkiksi asettaa ohjaajasovelluksessa oman organisaationsa väripaletin ja logon, jotka sitten näkyisivät osana 3D-maailmaa.

4.4 Osoittimen hallinta ensimmäisen persoonan selainsovelluksessa

Ensimmäisen persoonan 3D-sovelluksessa käyttäjä liikuttaa kameraa hiiren avulla. Jotta kamera voisi kääntyä ympäri täydet 360 astetta, hiiren osoitin tyypillisesti lukitaan keskelle näyttöä ja piilotetaan näkyvistä. Tällöin osoitin ei törmää ruudun laitaan ja estä enempää kääntymistä. Videopeleissä osoitin on helppo

vapauttaa ja lukita valikkoihin siirtyessä ja niistä poistuttaessa. Verkkoselain sovelluksen käyttäminen verkkoselaimessa aiheuttaa kuitenkin muutamia erikoistilanteita, etenkin kun osoittimen on oltava sekä Unity- että selainsovelluksen käytössä.

Kuten useimpiin videopeleihin, tähänkin Unity-sovellukseen haluttiin toteuttaa taukovalikko, josta pelaaja voi muuttaa asetuksiaan. Koska pelillistämisalustan oma käyttöliittymä on jo piirrettynä Unity-sovelluksen päälle, Unityn valikko pystyttiin jättämään hyvin niukaksi. Sen sisältöön kuului näppäinten selitykset ja yksinkertainen käyttöohje, asetukset hiiren herkkyydelle ja kamera-akselien kääntämiselle sekä painikkeet jatkamiselle, äänen mykistykselle ja uloskirjautumiselle. Esc-näppäin on varattu selaimen turvaominaisuuksissa osoitinlukosta ja koko näytön tilasta poistumiseen, mikä sopii tämän projektin käyttötarkoitukseen, sillä yleisen käytännön mukaan taukovalikko avataan Esc-näppäimellä ja selain poistaa automaattisesti osoittimen lukituksen. Myös valikosta poistuminen takaisin peliin olisi tyypillistä sitoa Esc-näppäimeen, mutta tässä tauksessa se ei ole mahdollista, sillä näppäimen painallus kumoaisi osoittimen lukitsemisen. Tämän lisäksi, jotta Unity pystyy hyödyntämään käyttäjän syötettä näppäimistöltä, on sen oltava sivun aktiivinen elementti. Valikossa sijaitseva ”jatka”-painike toimii siis myös keinona saada käyttäjä klikkaamaan canvas-elementtiä, jossa Unity-sovellus esitetään. Näin elementti muuttuu aktiiviseksi ja Unity pystyy pyytämään osoittimen ja näppäimistösyötteen omaan käyttöönsä.

Käyttäjän syöte halutaan kuitenkin siirtää Unityltä selaimelle, myös muissa tapauksissa – lähinnä silloin, kun käyttäjä klikkaa tehtäväpistettä ja avaa harjoitemodaalin. Onneksi selaimet tarjoavat ohjelmointirajapinnan osoittimenlukitsemiseen (Pointer Lock API), jonka avulla syötteiden siirtoa sovellukselta toiselle voidaan hallita ohjelmoinnillisesti. Rajapinta laajentaa html-elementtejä lisäämällä niihin metodeja ja ominaisuuksia, joilla voidaan hallita osoittimen lukitsemista (MDN 2024). Kun osoitin täytyy lukita ensimmäisen persoonan ohjausta varten, kutsutaan canvas-elementin requestPointerLock-metodia ja vastaavasti siitä poistutaan document-olion exitPointerLock-metodilla. Mikäli käyttäjä on poistunut osoitinlukosta käyttämällä Esc-oletusnäppäintä tai osoitin lukitaan ensimmäistä kertaa, rajapinta vaatii lukituspyynnön lisäksi tapahtuman, joka on syntynyt käyttäjän syötteestä (MDN 2024). Tämä tilanne syntyy joka kerta, kun käyttäjä avaa

taukovalikon Esc-näppäimellä ja sen seurauksena osoittimen lukitseminen epäonnistui käyttäjän palatessa pelitilaan.

Ratkaisu ongelmaan löytyi Pointer Lock -rajapinta sisältämistä onpointerlockchange ja onpointerlockerror -tapahtumista. Enimmäinen tapahtuu aina, kun osoittimen lukituksen tila muuttuu onnistuneesti. Sen sijaan, että Unity hallitsisi omaa tilaansa ja osoitin pyrittäisiin lukitsemaan asiaankuuluvasti selaimessa, toteutus käännettiin niin päin, että onpointerlockchange-tapahtuma ohjaa Unityn tilaa. Selaimen turvaominaisuudet ja kolmesta eri lähteestä tulevat komennot (Unity, React-käyttöliittymä ja selaimen oma koodi) tekevät jokseenkin epävarmaa ja alkuperäisessä toteutuksessa käyttäjä ajautui jatkuvasti tilanteeseen, jossa Unity siirtyi takaisin pelitilaan, mutta osoitinta ei ollut lukittu. Kun Unityn tila sidottiin tapahtumaan, käyttäjä ei joudu koskaan tilanteeseen, jossa selain ja Unity ovat keskenään eri tilassa – edes silloin, kun osoittimen lukitus on epäonnistunut.

Selaimen turvaominaisuuden kiertämiseen taas löytyi vähemmän elegantti ratkaisu. Kun käyttäjä painaa taukovalikon ”jatka”-painiketta ja pyytää siten osoittimen lukitusta, selain ei hyväksy pyyntöä ennen kuin käyttäjä on tarjonnut jonkin eleen selaimelle, esimerkiksi painanut jotakin näppäintä. Selain lähettää tässä tapauksessa onpointerlockerror-tapahtumat. Sidoin tämän tapahtuman asynkroniseen funktioon, joka odottaa 500 millisenkuntia ja pyytää lukitusta uudelleen. Tämä rekursio jatkuu, kunnes käyttäjä jatkaa pelaamista ja antaa syötteellään selaimelle eleen, jonka myötä lukitus voidaan hyväksyä. Toteutus ei ole järin elegantti, mutta se ratkaisi ongelman. Yleisesti on todettava, että verkkoselaimet eivät ole paras alusta ensimmäisen persoonan sovelluksille. Normaalissa selainpelissä osoitin lukitaan kerran, minkä jälkeen näppäimistön ja hiiren hallinta säilyy oletettavasti koko pelaamisen ajan pelillä itsellään. Tämän projektin tapauksessa, kun syöte on siirrettävä toistuvasti Unityn ja selaimen välillä, muutoin triviaalit ongelmat ilmenevät käyttäjälle jatkuvasti ja vaativat erityisiä ratkaisuja myös selaimen puolella.

4.5 Toteutuksen skaalautuminen

Projektin POC-vaiheessa toteutettiin yksi 3D-ympäristö, johon pelillistämisalusta integroitiin menestyksekkäästi. Tämä ympäristö ja sen sisältämät tehtäväanimaatiot suunniteltiin spesifin teeman ja narratiivin pohjalta. Tästä syystä ympäristöä pystyisi käyttämään tarkoitetulla tavalla vain kyseiseen teemaan liittyvissä peleissä. Jotta 3D-integraatiolla olisi kaupallista potentiaalia, käyttäjille tulisi pystyä tarjoamaan useita erilaisiin teemoihin ja peleihin liittyviä ympäristöjä ja tarinoita. Tämä tarkoitti sitä, että Unity-integraation tulisi olla yleistettävissä siten, että sen pystyisi implementoimaan nopeasti ja luotettavasti muihin Unity-WebGL-projekteihin. MVP-vaiheessa tavoitteeksi asetettiin tuottaa ensimmäisen 3D-maailman lisäksi kolme uutta maailmaa, jotka olisivat teemoiltaan ja osin toiminnallisuudeltaan erilaisia.

Unity (2021) suosittelee scenejen, työkalujen ja muiden assettien jakamiseen Unity-pakettien (engl. Unity package) käyttämistä. Vaihtoehtoisesti assetteja pystyy siirtämään projektien välillä käyttämällä Asset package -formaattia. (Creating your own Asset packages 2021) Unity-paketin täytyy noudattaa tarkasti määritettyä rakennetta, jotta se on yhteensopiva Unityn Package Managerin kanssa. Package Manageria käyttäessä etuna on pakettiversioiden helppo päivittäminen, minkä takia se on suositeltu vaihtoehto, kun halutaan jakaa toiminnallisuutta useiden projektien kesken. MVP-vaiheessa päädyttiin kuitenkin käyttämään Asset package -pakettiformaattia, sillä se oli merkittävästi nopeampi tapa testata toiminnallisuuden siirtämistä projektista toiseen. Kuten zip-tiedosto, Asset package säilyttää purettuna tiedostorakenteensa sekä tiedostojen metadatan (Asset packages 2025). Paketin vieminen projektista vaatii ainoastaan toivottujen assettien valitsemisen Unityn editorissa ja tämän jälkeen **Assets > Export Package** -vaihtoehdon valitsemisen editorin valikosta. Assettien tuominen tapahtuu valitsemalla **Import Package** samasta valikosta ja valitsemalla pakettitiedosto tietokoneen levyltä.

Asset package -paketin käyttäminen oli nopeaa ja mutkatonta, koska sillä oli mahdollista siirtää ja testata assetteja ilman minkään laista alkuvalmistelua. Kuitenkin Unity-projektien ajan tasalla ylläpitäminen muuttui todella haastavaksi, kun niiden kokonaismäärä oli kasvanut neljään. Toisin kuin Unity-paketit, joihin voi

julkaista päivityksiä ladattavaksi Package Managerin kautta, Asset package on vietävä joka kerta ajantasaisesta projektista ja tuotava manuaalisesti kaikkiin muihin projekteihin. Integraation alkuvaiheessa projektin kansiorakenne oli jatkuvassa muutoksessa parasta käytäntöä etsiessä ja tällöin kätevä zip-tiedostoa muistuttava ratkaisu oli perusteltu. Kuitenkin assettien ja kansioiden rakenteen vakiinnuttua olisi ollut viisasta määritellä Unity-paketti. Se jäi projektin jatkokehityksen kärkiprioriteeteiksi. Kehitystyön aikana puntaroitiin myös vaihtoehtoa luoda kaikki maailmat erillisinä sceneinä saman Unity-projektin sisällä, jolloin ylläpidettäväksi jäisi vain yksi projekti. Tämä lisäisi kuitenkin projektin kokoa merkittävästi, mikä olisi voinut aiheuttaa ongelmia GitHubin kanssa, sillä palvelu rajoittaa yksittäisten commitien tiedostokokoa.

Pellistämisalustan ja Unityn integraatio jaettiin kahteen osaan. Ensimmäinen oli ydintoiminnallisuus, joka sisälsi viestinnän Unityn ja selaimen kesken, sekä pelillistämisalustan datan ja harjoitteiden käsittelyn. Toinen osa liittyi ensimmäisen persoonan käyttäjäkokemukseen, jonka perustalle kaikki neljä erilaista ympäristöä rakennettiin. Ensin mainittu ydintoiminnallisuus on mahdollista tuoda mihin tahansa Unity-WebGL-sovellukseen. Jokin Unity-sovelluksen toiminto täytyy vain ohjelmoida lähettämään tapahtuma, joka antaa selainkäyttöliittymälle käskyn avata tehtävämoduuli tietyn harjoitteen tunnistekoodilla. Konseptitodistuksena integroin pelillistämisalustan erääseen opiskeluprojektina tehtyyn tasohyppelypeliin. Peliä pystyi pelaamaan selaimessa pelillistämisalustan käyttöliittymän kanssa ja aina pelaajan kerättyä 5 elämäpistettä, selainkäyttöliittymä avasi uuden harjoitteen. Toteutus ei ollut täysin saumaton, mutta tämän POC:n toteuttaminen vaati ainoastaan alle kaksi tuntia työaikaa. Oli siis todettavissa, että integraatio on eriytetty riittävästi varsinaisesta pelimekaniikasta, jotta se on helposti yleistettävissä erilaisin pelityyppeihin ja käyttäjäkokemuksiin. Myös tälle projektille spesifi ensimmäisen persoonan käyttäjäkokemus yleistettiin onnistuneesti, niin että kameran liikuttelu, maailman sisäinen käyttöliittymä ja 3D-tilassa sijaitsevat tehtäväpisteet pystyttiin siirtämään vaivatta eri Unity-projekteihin. Käytännössä tämä mahdollistaisi sen, että pelillistämisalustaa käyttävät yritykset ja organisaatiot voisivat tarjota omia 3D-ympäristöjään tai Unity-scenejään ja ensimmäisen persoonan pelikokemus pelillistämisalustalla voitaisiin toteuttaa näissä ympäristöissä verrattain pienillä resursseilla.

5 JOHTOPÄÄTÖKSET JA JATKOKEHITYS

Kehittämisprojektissa onnistuttiin kehittämään asiakasyrityksen pelillistämisalustalle kolmiulotteiseen ympäristöön sijoittuva käyttäjäkokemus. Toteutuksessa käytettiin uudenlaista lähestymistapaa, jossa pelillistämisalustan käyttöliittymä piirretään modaaleina natiivisti selaimessa Unityn 3D-maailman päälle. Tämä mahdollisti asiakasyrityksen ydintuotteen sekä 3D-ominaisuuden rinnakkaisen ja toisistaan erillisen kehittämisen. Tällä tavalla toteutus on myös helposti yleistettävissä erityyppisiin WebGL-sovelluksiin tai vaikkapa videosoittimeen. Aiemmassa osiossa erittelin eri kehitysvaiheissa ilmenneitä ongelmia sekä niihin löydettyjä ratkaisuja. Avaan seuraavaksi haasteita, jotka ilmenivät itse kehitystyön osalta.

Projektin lopullisessa toteutustavassa Unity sekä verkkoselain linkittyivät toisiinsa hyvin vahvasti. Käyttäjäkokemuksen ydintoiminnallisuus jakautui tasaisesti ja toisistaan riippuvaisesti Unityn ja pelillistämisalustan selainkäyttöliittymän kesken. Tämä aiheutti merkittävän haasteen ja rasitteen kehitystyölle. Integraatiota kehittäessä ainut mahdollisuus testata kahden alustan yhteensopivuutta oli suorittaa sovellusta sen lopullisessa käyttöympäristössä, verkkoselaimessa. Tämä tarkoitti sitä, että Unityn editorissa tai C#-koodissa tehdyt muutokset eivät olleet testattavissa editorin sisällä, vaan jokaisen muutoksen jälkeen sovellus oli rakennettava uudestaan sekä pyyhittävä selaimen välimuistista ja ladattava sivu uudelleen. Sovelluksen rakentaminen kesti tämän projektin tapauksessa noin kolmesta kuuteen minuuttia, riippuen tehtyjen muutosten laajuudesta. Tämä teki bugien korjaamisesta huomattavan turhauttavaa, sillä monesti löysin itseni tilanteesta, jossa lisään Unityssä yhden tai muutaman tulostuksen konsoliin, odotan 3 minuuttia sovelluksen rakentumista, totean että vika on jossakin muualla ja toistan edeltävät askeleet. Tällaisella työskentelytavalla kolmen eri tulostusrivin kokeileminen saattaa viedä yli vartin aikaa. Sovelluksen rakentumiseen kuluvaa aikaa ei voi varsinaisesti käyttää hyödyksi ja katkonainen työskentely haittaa keskittymistä ja asiaan paneutumista.

Unity Editorista on mahdollista lähettää HTTP-kutsuja siinä missä selaimestakin, joten pelillistämisalustan käyttöliittymä ei ole teknisesti välttämätön alustan pelien

suorittamiseen Unityssä. Näin itse asiassa tehtiinkin projektin POC-vaiheessa. Unityyn olisi mahdollista jatkokehittää jonkinlainen pelkistetty käyttöliittymä tai näkymätön toiminnallisuus, jolla pelidataa voitaisiin käsitellä, lähettää ja vastaanottaa suoraan editorissa, mikä helpottaisi ei-selainspesifien ominaisuuksien kehittämistä ja bugien ratkomista. Vaikka tällainen systeemi olisi lopulta nopeuttanut kehitystyötä, olisi sen kehittäminen itsessään vienyt MVP-vaiheessa liikaa resursseja ja huomiota varsinaiselta toiminnallisuudelta.

Myös selainten väliset erot aiheuttivat haasteita, joihin Unity-kehittäjä ei ole välttämättä tottunut. Projektin aikana ilmeni ainoastaan Firefox-selaimella esiintyvä bugi, jossa Unity-instanssin suorittaminen pysähtyi, kun käyttäjä oli lisännyt mediaa pelillistämisalustan selainkäyttöliittymässä. Suorittaminen jatkui vasta, kun käyttäjä oli vaihtanut aktiivista ikkunaa tai välilehteä ja palannut takaisin. Valitettavasti tämä ongelma ei ratkennut projektin aikana. Firefoxin toiminnassa on muihin selaimiin verrattuna jotakin eroa, mistä ei löydy dokumentaatiota eikä vastaavia kokemuksia muilta kehittäjiltä. Toinen selainkohtainen ongelma ilmeni Safarissa: selainten turvaominaisuuksiin kuuluu ponnahdusilmoitus, joka ilmoittaa käyttäjälle, mikäli verkkosivu on lukinnut hiirenosoittimen. Suurimassa osassa selaimista tämä viesti ilmestyy modaalina sivun päälle. Safari kuitenkin avaa ilmoituksen omana palkkinaan sivun ylälaidan ja osoiterivin väliin. Tämä saa koko pelinäkymän siirtymään kyseisen palkin verran alaspäin. Koska käyttäjä vaihtelee jatkuvasti ensimmäisen persoonan tutkiskelun ja selainkäyttöliittymän välillä, tämä ponnahdusviesti liikuttaa pelinäkymää toistuvasti aiheuttaen epämukavan käyttökokemuksen. Valitettavasti selainten turvaominaisuudet ovat kehittäjän vaikutusmahdollisuuksien ulkopuolella, joten Safarin implementaatio teki pelaamisesta epäsuositeltavaa kyseisellä selaimella.

Myös WebGL-alustalle kehittäminen vaatii erityistä huomiota, sillä monet Unityn erityisesti graafisista komponenteista eivät ole tuettuja WebGL:llä. 3D-ympäristöjä luodessa käytettiin Unityn Asset Store -kaupasta hankittuja assettipaketteja. Monet näistä paketeista sisälsivät erilaisia materiaalishadereita, joiden avulla ympäristöistä pystyttiin luomaan erittäin näyttäviä ja todentuntuisia, vaikkakin tyyliteltyjä. Valitettavasti juuri läpikuultavia ja heijastavia pintoja ei ole mahdollista toteuttaa WebGL:llä yhtä realistisesti, kuin natiivilla Unity-alustalla. Myös Unityn vi-

suaalisiin efekteihin käytettävä Visual Effects Graph -ominaisuus hyödyntää laskentashadereita, jotka eivät ole WebGL:n tuettu ominaisuus ja täten efektien käyttö on rajattava perinteisiin partikkelisysteemeihin. Oli myös huomioitava, että pelillistämisalustaa käytetään pääsääntöisesti kouluissa ja työpaikoilla, jolloin pelaaminen tapahtuu usein matalatehoisilla kannettavilla tietokoneilla. Tästä syystä ympäristöjen valaistus, reaaliaikaiset efektit ja valaistus sekä geometrian määrä oli pidettävänä maltillisena. Eritoten 3D-objekteissa käytettävien tekstuurien tulee olla resoluutioltaan pieniä ja äänitiedostojen pakattuja, jottei niiden lataaminen verkkosivulle kestä kohtuuttoman pitkän aikaa heikommilla verkkoyhteyksillä.

Projektin aikana pelillistämisalustan integrointiin tarvittavat Unity-assetit ja C#-koodi järjestettiin siten, että ne ovat eriytettävissä ja vietävissä toisiin Unity WebGL -projekteihin. MVP-vaiheessa projektin skaalaamiseen ja neljän eri Unity-projektin ylläpitämiseen käytettiin Unityn Asset package -formaattia, joka mahdollisti nopean testaamisen. Asset package on helppokäyttöinen, mutta sen suunniteltu käyttötarkoitus on siirtää kokoelma assetteja yhden kerran projektista toiseen; ei ylläpitää päivittyvää toiminnallisuutta usean projektin välillä. Tähän tarkoitukseen sopivampi vaihtoehto on Unity Package Manager ja Unity Package -formaatti, joiden avulla ylläpito tapahtuu keskitetysti. Kehitysprojektin lopuksi Unity Package -paketin määrittäminen asetettiin korkean prioriteetin jatkokehitystavoitteeksi.

Lopullisessa toteutuksessa teknisen toiminnan kannalta Unityn rooliksi jäi ainoastaan ilmoittaa selainkäyttöliittymälle, kun käyttäjä on avannut harjoitteen. Tämä mahdollistaa hyvin monimuotoisten Unity-pelien ja -kokemusten käyttämisen pelillistämisalustan pohjana. Käyttäjä voisi esimerkiksi tutkia kaksiulotteista ylhäältä kuvattua ympäristöä vanhojen seikkailupelien tapaan tai avata harjoitteita pelaamalla minipelejä. Toisaalta olisi mahdollista toteuttaa videopelimäisempi kokemus, jossa pelaaja kehittää hahmoaan suorittamalla harjoitteita ja keräämällä pisteitä pelillistämisalustan pelissä. Kehittäjänä täytyy ainoastaan ohjelmoida tapa, jolla harjoitteet avataan, minkä jälkeen käyttäjän vastausdataa ja ansaitsemia pisteitä voidaan käyttää erilaisilla luovilla tavoilla. Integraation yleisluontoisuus ja selkeä vastuunjako eri sovellusten välillä avaa valtavan määrän erilaisia mahdollisuuksia hyödyntää pelillistämisalustan ja Unityä yhdessä. Tässä opinnäytteessä

kuvatut tekniikat toimivat myös yleisluontoisena tapana luoda yhdistettyä toiminnallisuutta Unity WebGL-pelien ja verkkosivujen välillä.

LÄHTEET

Amazon Web Services. n.d. What is RESTful API? Verkkosivu. Viitattu 28.4.2025. https://aws.amazon.com/what-is/restful-api/

GeeksforGeeks. 2021. Explain Request Verbs. Verkkosivu. Viitattu 28.4.2025. https://www.geeksforgeeks.org/explain-request-verbs/

GeeksforGeeks. 2024. HTTP headers. Verkkosivu. Viitattu 28.4.2025. https://www.geeksforgeeks.org/http-headers/

GeeksforGeeks. 2025. React Introduction. Verkkosivu. Viitattu 5.4.2025. https://www.geeksforgeeks.org/reactjs-introduction/

GeeksforGeeks. 2025. Rest API Introduction. Verkkosivu. Viitattu 3.4.2025. https://www.geeksforgeeks.org/rest-api-introduction/

Gupta, L. 2025. What Is REST? Verkkosivu. Viitattu 28.54.2025. https://rest-fulapi.net

Json2CSharp. 2025. Convert Json to C# Classes Online. Verkkosivu. Viitattu 3.4.2025. https://json2csharp.com

Korhonen, P. 2018. Pieni API-sanakirja. Blogikirjoitus. Viitattu 28.5.2025. https://www.cgi.com/fi/fi/blogi/pieni-api-sanakirja

MDN. 2024. Pointer Lock API. Mozillan dokumentaatio. Viitattu 21.4. https://developer.mozilla.org/en-US/docs/Web/API/Pointer_Lock_API

MDN. 2025. WebAssembly. Mozillan dokumentaatio. Viitattu 1.5.2025. https://developer.mozilla.org/en-US/docs/WebAssembly

MDN. 2024. WebGL: 2D and 3D graphics for the web. Verkkosivu. Viitattu 29.4.2025. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

Microsoft. 2023. Language Integrated Query (LINQ). C#-dokumentaatio. Viitattu 3.4.2025. https://learn.microsoft.com/en-us/dotnet/csharp/linq/

Newtonsoft. n.d. Newtonsoft.Json.Linq Namespace. Newtonsoftin dokumentaatio. Viitattu 3.4.2025. https://www.newtonsoft.com/json/help/html/N_Newtonsoft_Json_Linq.htm

StackOverflow. 2016. "[DllImport("__Internal")]" - what does the "__Internal" mean? Foorumikeskustelu. Viitattu 6.4.2025. https://stackoverflow.com/questions/38584361/dllimport-internal-what-does-the-internal-mean

Tapala, K. 2025. WebGL-shaderien kanssa alkuun. Blogikirjoitus. Viitattu 29.4.2025. https://www.karhuhelsinki.fi/blogi/webgl-shaderien-kanssa-alkuun/

Unity. 2025. Asset packages. Unity-dokumentaatio. Viitattu 25.4. https://docs.unity3d.com/2022.3/Documentation/Manual/AssetPackages.html

Unity. 2025. Comparison of UI systems in Unity. Unity-dokumentaatio. Viitattu 5.4.2025. https://docs.unity3d.com/6000.0/Documentation/Manual/UI-system-compare.html

Unity. 2021. Creating your own Asset packages. Unity-dokumentaatio. Viitattu 25.4. https://docs.unity3d.com/2022.3/Documentation/Manual/AssetPackages-Create.html

Unity. 2021. Getting started with WebGL development. Unity-dokumentaatio. Viitattu 30.4.2025. https://docs.unity3d.com/2020.1/Documentation/Manual/webgl-gettingstarted.html

Unity. 2025. JSON Serialization. Unity-dokumentaatio. Viitattu 3.4.2025. https://docs.unity3d.com/2021.3/Documentation/Manual/JSONSerialization.html

Unity. 2025. Set up your JavaScript plug-in. Unity-dokumentaatio. Viitattu 6.4.2025. https://docs.unity3d.com/6000.0/Documentation/Manual/web-interacting-browser-js.html

Unity. 2025. UnityWebRequest. Unity-dokumentaatio. Viitattu 28.4.2025. https://docs.unity3d.com/2022.3/Documentation/Manual/UnityWebRequest.html

Unity Discussions. 2023. Disabling JsonUtility Serialization Depth Limit of 10. Foorumikeskustelu. Viittattu 3.4.2025. https://discussions.unity.com/t/disabling-jsonutility-serialization-depth-limit-of-10/919647

WebGLFundamentals. n.d. WebGL - Rasterization vs 3D libraries. Verkkosivu. Viitattu 30.4.2025. https://webglfundamentals.org/webgl/lessons/webgl-2d-vs-3d-library.html