JÄTEVEDENPUHDISTUKSEN KOULUTUSPELIN KEHITTÄMINEN

LAB-AMMATTIKORKEAKOULU
Insinööri (AMK)
Tieto- ja viestintätekniikka
Kevät 2020
Jukka-Pekka Nikunen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Nikunen, Jukka-Pekka	Opinnäytetyö, AMK	Kevät 2020
	Sivumäärä 27	

Työn nimi

Jätevedenpuhdistuksen koulutuspelin kehittäminen

Tutkinto

Insinööri (AMK)

Tiivistelmä

Työn tavoite oli kehittää koulutuspeli jätevedenpuhdistamoiden työntekijöiden tietämyksen kartoittamiseksi Unity-pelinkehitysympäristössä. Peli luotiin osana Euroopan unionin rahoittamaa IWAMA-hanketta.

Pelin jätevedenkäsittely mallinnettiin oikeiden jätevedenpuhdistamojen käsittelyvaiheiden pohjalta. Jäteveden puhtaus perustuu sen epäpuhtauksien pitoisuuksiin, joihin tulee käsittelyvaiheiden myötä reduktioita. Veden tulee olla lopuksi puhtauslaadultaan Euroopan unionin vaatimukset täyttävää.

Peli luotiin Unity-pelimoottorilla käyttämällä C#- ja HLSL-ohjelmointikieliä. Pelin luominen vaati tuntemusta sekä Unityn editorista että sen skriptausjärjestelmästä.

Pelissä käyttäjän pelaajahahmo on jätevedenpuhdistamon työntekijä, jonka tehtävänä on puhdistaa sisääntuleva jätevesi. Puhdistus tapahtuu minipelien kautta, jossa käyttäjän suoritus pisteytetään. Käyttäjälle esitetään myös jätevedenkäsittelyyn liittyviä väitteitä, joihin hänen tulee vastata oikein.

Työn tuloksena oli WebGL-alustalle luotu 2D-peli, joka on pelattavissa IWAMA:n virallisella kotisivulla.

Asiasanat

Unity, 2D, pelinkehitys, opetusohjelmisto

Abstract

Author(s)	Type of publication	Published
Nikunen, Jukka-Pekka	Bachelor's thesis	Spring 2020
	Number of pages	
	27	

Title of publication

Developing a water treatment training game

Name of Degree

Bachelor of Engineering

Abstract

The goal of the work was to develop an educational game for personnel working at water treatment facilities. The game was created as part of the IWAMA project funded by the European Union.

The game was modeled after the water treatment process of real water treatment facilities. The purity of the water is affected by the amount of impurities present, which get reduced during the treatment process. The purity of the water must satisfy the criteria set by the European Union at the end of the treatment process.

The game was developed in the Unity game development environment using C# and HLSL programming languages. The development required familiarity with both the Unity editor and the scripting system.

In the game, the user takes the role of an employee at a water treatment facility whose job is to purify the influent waste water. The treatment is done in the form of minigames where the user's performance is judged by a scoring system. The user is also presented with statements of the water treatment process to which he or she must answer correctly.

The result of the project was a 2D game for the WebGL platform. It is playable on the official website of IWAMA.

Keywords

Unity, 2D, game development, educational software

SISÄLLYS

1 JOHDANTO

Työn tavoitteena on suunnitella ja kehittää koulutuspeli jätevedenpuhdistamoiden työntekijöiden ammattitaidon kartoittamiseksi. Peli luodaan Unity Technologies -yhtiön kehittämällä Unity-pelimoottorilla. Pelissä käyttäjän tulee ohjata pelaajahahmoa, joka on jätevedenpuhdistamon työntekijä. Käyttäjän tulee puhdistaa sisääntuleva jätevesi niin, että se täyttää veden puhtausarvojen vaatimukset. Peli tulee koostumaan minipeleistä, jotka testaavat käyttäjän tietämystä jätevedenpuhdistuksesta. Minipelien lisäksi peli tulee koostumaan minipelien välisistä siirtymähuoneista ja valikoista. Lopullisesta pelistä tulee Unityn kaupallisella lisenssillä luotu WebGL-sovellus.

Työ tehdään osana Interactive Water Management eli IWAMA-hanketta. IWAMA-hanke on osana Euroopan unionin rahoittamaa ja Euroopan komission hyväksymää Interreg Baltic Sea Region Programme 2014 - 2020 -ohjelmaa. Ohjelmaan osallistuu EU-jäsenmaista Latvia, Liettua, Puola, Ruotsi, Saksan pohjoiset osat, Suomi, Tanska ja Viro. Lisäksi ohjelmaan osallistuu yhteistyömaina Norja, Valko-Venäjä ja Venäjän luoteiset osat. (Interreg Baltic Sea Region 2014.) IWAMA-hankkeen tarkoitus on edistää vedenpuhdistusprosessin kehitystä Baltian maissa ja sen lähialueella. IWAMA-hanke käynnistyi maaliskuussa 2016 ja saatiin päätökseen huhtikuussa 2019. Hankkeen budjetti on 4,6 miljoonaa euroa, josta 3,7 miljoonaa euroa on Euroopan aluekehitysrahaston rahoittamaa. (Interactive Water Management 2019). Työ alkoi Lahden ammattikorkeakoulun (nykyinen LAB-ammattikorkeakoulu) opiskelijaprojektina, jonka jälkeen se suoritettiin loppuun projektityönä.

Työn päätavoite on luoda jätevedenpuhdistamon työntekijöille mielenkiintoinen ja helppokäyttöinen koulutus- ja kartoituspeli, joka on helposti ja nopeasti pelattavissa lyhyilläkin tauoilla. Sivutavoitteena on kerätä tietoa työntekijöiden jätevedenpuhdistuksen tuntemustasosta, jonka avulla suurimmat ongelmakohdat voidaan löytää.

2 JÄTEVEDENPUHDISTUS

Vedenkäsittelyprosessi voidaan jakaa kymmeneen vaiheeseen, joista viisi liittyy suoranaisesti jäteveden puhdistukseen: sisääntulevan jäteveden pumppaus ja välppäys, hiekanerotus, esiselkeytys, ilmastus ja jälkiselkeytys. Toiset viisi vaihetta liittyvät puhdistamolietteen käsittelyyn: jätevedestä erotetun lietteen sakeutus, mädätys, metaanin kerääminen ja hyödyntäminen biokaasuna polttamalla, kuivaus sekä varastointi ja levitys. (Helsingin seudun ympäristöpalvelut HSY 2019.) Pelissä keskitytään veden puhdistamiseen liittyviin vaiheisiin, mutta käyttäjältä vaaditaan kokonaisvaltaista tuntemusta prosessin kokonaiskuvasta lietteenkäsittely mukaan lukien.

Pelissä veden puhtaus määritellään sen fosfori-, typpi-, BHK7- (biokemiallinen hapenkulutus) ja ammoniumtyppipitoisuuksien ja niiden reduktion perusteella. Pitoisuuksien hyväksyttävät rajat on mallinnettu Euroopan unionin direktiivien pohjalta. Hyväksyttävän veden puhtauden kriteerit vaihtelevat riippuen siitä, kuinka suurta asuinaluetta puhdistamo palvelee. Puhdistamon kapasiteetti ilmoitetaan asukasvastinelukuna. Yksikkö ilmaisee yhden hengen perhetalouden vuorokautista jätevesikuormaa. Pelissä käytettiin viittausarvona sadantuhannen asukasvastineluvun puhdistamoa. Näin ollen pelissä veden puhtausvaatimuksena on 70 prosentin reduktio BHK7:ssa, 80 prosentin reduktio kokonaisfosforissa ja 70 prosentin reduktio kokonaistypessä. Lisäksi pitoisuuksien tulee olla korkeintaan 25 mg/l BHK7:lle, 3 mg/l kokonaisfosforille ja 10 mg/l kokonaistypelle. (Official Journey of the European Communities 1991.)

Todellisuudessa vedenpuhdistusprosessiin kuluu aikaa parista viikosta kuukauteen riippuen sisääntulevan jäteveden likapitoisuuksista ja lämpötilasta. Nykyaikaisissa jätevedenpuhdistamoissa puhdistusprosessi on automatisoitu, jolloin puhdistamon työntekijöille jää tehtäväksi veden tilan monitoroiminen ja laitteiston toimivuuden ylläpito. Pelinkehityksessä käytettiin luovaa vapautta prosessien minipelien toteutuksessa. Pelin tarkoitus on olla opettavainen ja käyttäjää kiinnostava, ei todellisuutta tarkasti simuloiva.

Pelin kehityksen yhteydessä vierailtiin Lahti Aqua Oy:n Lahden Kariniemen puhdistamon tiloissa tutustumassa vedenpuhdistusprosessiin ja yksittäisten prosessointitilojen rakenteeseen. Paikan päältä kerättyä tietoa hyödynnettiin minipelien toiminnallisuuden sekä niiden graafisen ilmeen suunnittelussa ja toteutuksessa.

3 UNITY

3.1 Yleistä Unitystä

Unity on Unity Technologies -yhtiön kehittämä pelimoottori ja pelinkehittämisympäristö. Siihen sisältyy visuaalinen editori, ohjelmointikirjasto, valmiit komponentit, asset-kauppa, tuottavuustyökalut, pelaajastatistiikan keruu sekä pilvipalvelu versiohallinnalle. Unity on saatavilla ilmais-, kaupallis- ja koulutusversioina.

Unityssä skriptitiedostot kirjoitetaan C#-ohjelmointikielellä ja ne voivat muiden C#-ohjelmien tapaan käyttää .NET Framework -kirjaston ominaisuuksia. Shader-ohjelmat kirjoitetaan High-Level Shading Language (HLSL) -kielellä, joka muistuttaa syntaksiltaan C-ohjelmointikieltä. (Unity Technologies 2019a.) HLSL-kieli luotiin osana DirectX:n Direct3D-grafiikkaohjelmointirajapintaa. (Microsoft 2018.)

Unity tukee kohdealustoinaan työpöytäalustoja (Windows, MacOS, Linux), mobiilialustoja (Android, iPhone) sekä web-alustaa (WebGL). Unity oli alun perin tarkoitettu pääosin 3D-pelien kehittämistä varten, mutta versio 4.3 lisäsi virallisen tuen 2D-pelin kehitykselle. Unity on kuitenkin pohjimmiltaan 3D-pelimoottori, joten myös 2D-elementit on toteutettu 3D-teknologialla. Esimerkiksi 2D-spritet ovat yleensä yksipuolisia, kahden kolmion muodostamia quad-polygoneja, joilla on tekstuuri ja joiden pintanormaali osoittaa kohti kameraa, joka näkee pelimaailman ortografisesta perspektiivistä. Vaikka Unity antaakin uuden projektin luonnin yhteydessä pelikehittäjän valita 2D-pelin ja 3D-pelin väliltä, ne ovat kuitenkin käytännössä osa samaa pelimoottoria. Tämä tarkoittaa myös, että 3D-objekteja on mahdollista käyttää 2D-pelissä ja toisin päin.

3.2 Skenet, peliobjektit ja komponentit

Unityssä peli on jaettu erillisiin skeneihin, jotka rakentuvat peliobjekteista. Näistä skeneistä yksi on asetettu aktiiviseksi skeneksi. Skenen peliobjektit muodostavat pelin sen hetkisen tilanteen. Aktiivisen skenen voi ajonaikana vaihtaa toiseen skeneen. Skenen vaihtuessa aktiivisen skenen peliobjektit poistetaan pelistä ja tilalle tuodaan peliobjektit uudesta skenestä, joka sitten merkataan aktiiviseksi. Skenet luodaan aina niiden skene-tiedostojen pohjalta, joihin ei ajon aikana tule muutoksia. Jos siis jo aikaisemmin aktiivisena olleeseen skeneen palataan uudelleen, sen peliobjektit ovat oletusarvoissaan. Peliobjektit eivät tavallisesti säily skenen vaihtuessa, mutta tämän voi tarvittaessa välttää.

Peliobjektin voi liittää joko suoraan skenen ylimmälle tasolle eli kantatasolle tai johonkin muuhun peliobjektiin, jolloin ne muodostavat emo-lapsi -hierarkian. Skene voi sisältää useita kantatason peliobjektia. Pelin ajon aikana skeneen voi ohjelmallisesti luoda uusia peliobjekteja tai siirtää niitä peliobjektihierarkioiden välillä. Tarvittaessa skenessä olevan peliobjektin voi joko poistaa skenestä tai sen voi merkata ei-aktiiviseksi, jolloin sama tapahtuu myös peliobjektin mahdollisille lapsipeliobjekteille.

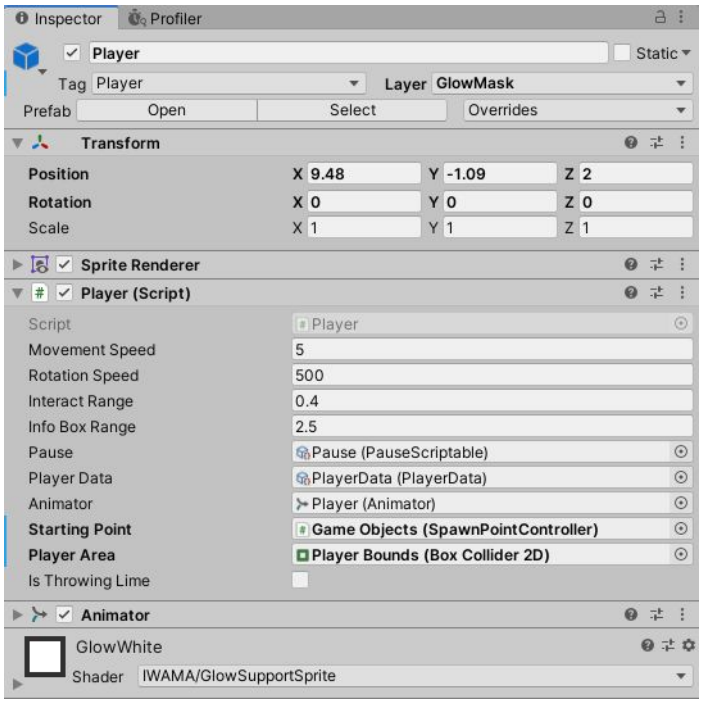
Jokaiseen peliobjektiin on mahdollista liittää nimetty tägi. Tägi kertoo peliobjektin roolista, kuten oletustägit MainCamera tai Player. Tägi on datatyypiltään string. Peliobjektin voi myös asettaa nimetylle kerrokselle. Kerrosta käytetään muun muassa valikoivaan renderöintiin ja fysiikkatestaukseen. Esimerkiksi hiiren kursorin poimimissäde voidaan kohdentaa vain kerrokselle, jolla on interaktiivisia objekteja. Näin muihin objekteihin ei tarvitse tehdä säteen leikkauksen fysiikkatestausta, mikä säästää suoritustehoa. Kerros on pohjimmiltaan 32-bittinen bittikenttä, joka mahdollistaa niihin kohdistuvien bitwise-operaatioiden suorittamisen. Ohjelmakoodissa kerroksen voi hakea joko sen bittikenttää edustavan kokonaisluvun arvon perustella tai sen nimen perusteella. Yksi peliobjekti on rajoitettu vain yhteen tägiin ja kerrokseen. Koko pelissä olevien tägien kokonaismäärää ei ole rajoitettu, kun taas kerroksissa niitä voi käytetystä datatyypistä johtuen olla enintään 32 kappaletta.

Peliobjektiin voi liittää komponentteja. Unity sisältää valmiiksi luotuja komponentteja yleisimmille peliohjelmointitoimille, kuten kameran, 3D-polygoniverkon tai 2D-spriten renderöinnin, äänentoiston, animaation ja fysiikkamallinnuksen. Komponentteja voi luoda, poistaa, merkata ei-aktiiviseksi ja niiden omistusta siirtää ajon aikana samalla tavalla kuin peliobjektejakin. Jotkin komponentit ovat uniikkeja, jolloin peliobjekti ei voi sisältää useampaa instanssia samasta tai samaan kategoriaan kuuluvasta komponentista. On kuitenkin mahdollista luoda peliobjektin alle uusi lapsipeliobjekti, joka emopeliobjektin tavoin sisältää halutun komponentin. Tällainen looginen hierarkiakokonaisuus on yleistä 3D-mallinnusohjelmista tuotavista malleissa, joissa esimerkiksi olkapäänivelen liikuttaminen vaikuttaa samalla mallin koko käden luiden ja nivelien liikkumiseen. Komponentit voivat olla myös pelikehittäjän luomia skriptitiedostoja. Nämä skriptikomponentit voivat tarkastella ja muuttaa peliobjektien sekä komponenttien tiloja. Tällä tavalla peliobjekteihin voi liittää toiminnallisuutta, joista pelilogiikka muodostuu.

Jokaisella peliobjektilla on oletuksena vähintään Transform-komponentti, joka määrittää peliobjektin sijainnin, suuntauksen ja skaalan pelimaailman 3D-avaruudessa. Kantatason peliobjekteilla nämä ovat absoluuttisia arvoja, kun taas lapsipeliobjekteille ne ovat suhteellisia arvoja niiden emopeliobjekteihin suhteutettuna. Näin ollen esimerkiksi emopeliobjek-

tin siirtäminen pelimaailmassa siirtää myös sen jokaista lapsipeliobjektia. Jos peliobjektin paikka pelimaailmassa ei muutu ajon aikana, voidaan se määrittää staattiseksi. Tällöin Unity käyttää sen oletusarvoja valmiiksi laskettuina arvoina, parantaen suorituskykyä.

Unityn peliobjektit ja komponentit muodostavat entiteetti-komponentti -järjestelmämalliin. Mallissa entiteetit ovat objekteja, jotka itsessään sisältävät vain minimaalisen toimivuuden, jonka on yhteistä kaikille objekteille, kuten komponenttien hallinnan ja uniikin identiteetin. Toimivuutta lisätään liittämällä komponentteja. Malliin sisältyy composition over inheritance -periaate, jossa vältetään olio-ohjelmoinnin tavanomaista perintää ja sen sijaan suositaan toiminnan liittämistä erikoistuneiden luokkien instansseilla. Malli on paljon käytetty peliohjelmoinnissa sen skaalautuvuuden ja joustavuuden ansiosta.



Kuva 1. Inspektori

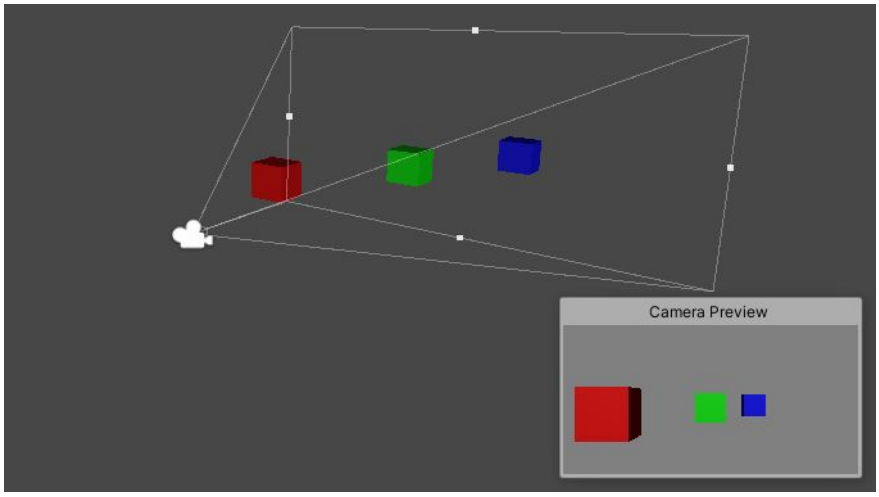
Peliobjektin ja sen komponenttien tilaa voi tarkastella sekä editoinnin että pelin ajon aikana Unityn inspektorilla (kuva 1). Se listaa kaikki peliobjektiin liitetyt komponentit sekä niiden ulospäin näkyvät kentät. Oletusarvoisesti jokainen julkinen kenttä näkyy inspektorissa, mutta on myös mahdollista muuttaa yksityinen tai suojattu kenttä näkyväksi serialisoimalla se SerializeField-attribuutilla. Näin kehittäjä pystyy asettamaan oletusarvon inspektorin kautta ilman, että ohjelmakoodin sisäinen kapselointi rikkoutuu. Kaikki datatyypit eivät kuitenkaan ole serialisoitavissa, kuten C#:n Dictionary datatyyppi tai get- ja set-propertyt.

Komponenttien viittaukset muihin peliobjekteihin ja komponentteihin tapahtuu myös yksinkertaisimmin inspektorin kautta. Ohjelmallisesti tämä vastaa inversion of control -järjestelmämallin dependency injection -toteutusta, jossa viittaukset ulkopuolisiin instansseihin viedään instanssiin, yleensä sen luomisen yhteydessä. Instanssi ei siis luo uusia ulkopuolisia instansseja sisäisesti itse. Näin ohjelmakoodiin saadaan vähemmän ohjelman osien välisiä kiinteitä kytköksiä, jolloin siitä tulee helpommin uudelleenkäytettävää ja testattavaa. Ohjelmakoodissa ei ole kuitenkaan mahdollista käyttää vastaavanlaista. Peliobjektit pitää etsiä joko hierarkiasta relatiivisesti parent-kentän tai koko skenestä Find-metodeilla joko sen nimen tai tägin perusteella. Kun viittaus peliobjektiin on muodostettu, voi sen komponentin hakea GetComponent-metodilla. Komponenttien keskeinen kommunikointi on myös mahdollista toteuttaa Unityn event-järjestelmällä tai käyttämällä kolmannen osapuolen laajennuksia.

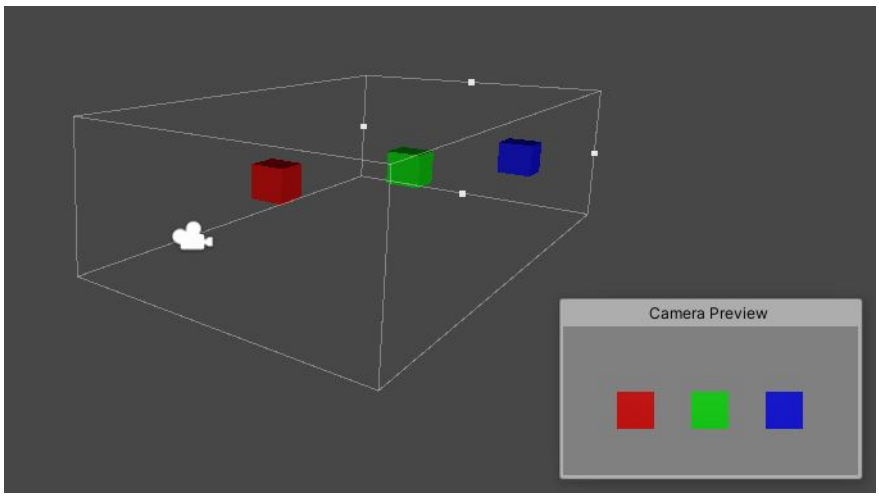
Peliobjekteista on mahdollista luoda prefab (prefabricated) -versio. Prefab toimii valmiina mallipohjana, josta voi instanssisoida peliobjekteja sekä editorissa että ajon aikana. Editorissa prefabista luodut peliobjektit on linkitetty prefabiin. Näin ollen prefabiin tehtävät muutokset suoritetaan myös siitä luotuihin peliobjekteihin.

3.3 Kamera ja projektio

Eräs Unity-skenen keskeisimmistä komponenteista on kamera. Se piirtää skenen visuaaliset elementit ja näyttää ne pelaajan ruudulla. Unityn skenessä voi olla useita kameroita, mutta vain yksi ruudulle piirtävä pääkamera, mikä määritellään MainCamera-tägin kautta.



Kuva 2. Perspektiiviprojektio



Kuva 3. Ortografinen projektio

Kameralla on kaksi mahdollista projektiotyyppiä: perspektiiviprojektio (kuva 2) ja ortografinen projektio (kuva 3.) Perspektiiviprojektiossa kameran näköalue on sen linssiä huippupisteenä pitävä pyramidi, kun taas ortografisessa projektiossa se on suorakulmainen särmiö. Kameralla on kaksi leikkaustasoa, lähitaso ja kaukotaso, jotka leikkaavat tämän näköalueen. Tämän näköalueen kokonaan tai osittain sisällä olevat visuaaliset kohteet piirretään kokonaisuudessaan. Perspektiiviprojektio vastaa toiminnaltaan oikean kameran linssiä. Kohteen etäisyys kameran lähitasosta vaikuttaa sen visuaaliseen kokoon ja sen sädekulma näköalueen huipusta vaikuttaa sen kuvakulmaan. Ortografisessa projektiossa lähitaso ja kaukotaso ovat samankokonaisia, jolloin niiden väliset säteet ovat yhdensuuntaisia. Tästä johtuen kohteen etäisyys ei vaikuta sen visuaaliseen kokoon ja se näyttää aina litteältä. Projektioiden luonteista johtuen 3D-peleissä käytetään pääosin perspektiiviprojektiota ja 2D-peleissä ortografista projektiota.

3.4 MonoBehavior ja ScriptableObject

Unityn luokista tärkeimmät ovat MonoBehavior ja ScriptableObject, jotka pohjimmiltaan periytyvät Unityn Object-luokasta. Object määrittelee pohjan editorin serialisoinnin kanssa yhteensopiville luokille. Tästä periytyy Component-luokka, joka tarjoaa liitettävyyden peliobjektiin sekä ohjelmallisen pääsyn peliobjektiin ja sen komponentteihin. Luokka mahdollistaa viestien lähettämisen sen jokaiseen MonoBehavior-skriptikomponenttiin, joiden pohjalta niiden nimettyjä metodeja voi kutsua. Tästä luokasta periytyy Behavior-luokka, joka mahdollistaa skriptikomponenttien aktivoinnin ja aktivoinnin poistamisen. Lopulta tästä luokasta periytyy MonoBehavior-luokka, joka toimii alustana jokaiselle skriptikomponentille. ScriptableObject puolestaan periytyy suoraa Object-luokasta.

MonoBehavior määrittelee metodeja, joista komponentin ohjelmallinen elinkaari muodostuu. Yleisimpiä näistä ovat Awake-, Start-, Update- ja OnDestroy-metodit. Tämä poikkeaa tavanomaisesta C#-ohjelmakoodista, jossa luokkainstanssin elinkaari alkaa constructorista ja päättyy destructoriin. Unity ei takaa skriptikomponenteille kutsujärjestystä, tosin erikoisissa tapauksissa on mahdollista asettaa tietyllä skriptille viivepoikkeama niin, että sen metodia kutsutaan aikaisemmin tai myöhemmin kuin komponenttien, joilla ei ole sellaista ole.

Awake-metodi on toiminnallisuudeltaan constructorin tapainen. Sitä kutsutaan kerran skriptikomponentin luomisen yhteydessä. Awake-metodia käytetään skriptikomponentin sisäisen tilan initalisoimiseen niin, että muiden komponenttien on turvallista viitata siihen ja kutsua sen metodeja. Awake-metodin jälkeen kutsutaan Start-metodia, jos skriptikomponentti ja sen peliobjekti ovat aktivoituneena. Muussa tapauksessa sitä kutsutaan kerran sen aktivoituessa. Start-metodissa voi turvallisesti viitata ulkopuolisiin komponentteihin, jotka ovat initalisoituneet Awake-metodikutsun aikana. Update-metodia kutsutaan aina, kun ruutu päivittyy ja komponentti on aktiivinen. Valtaosa pelilogiikasta suoritetaan tässä metodissa. Koska päivitystiheys riippuu ruudun päivitystiheydestä, on hyvä tapa käyttää Time.deltaTime tai vastaavaa staattista muuttujaa liikkeen skaalaamiseen. Se kertoo, kuinka kauan on kulunut edellisestä ruudunpäivityksestä. Näin liikkeen nopeus voidaan liittää ajankuluun, jolloin liike on sama kaikilla laitteilla ruudun päivitystiheydestä riippumatta. Update-metodille on olemassa kaksi vaihtoehtoa, FixedUpdate- ja LateUpdate-metodit. FixedUpdate-metodia kutsutaan tietyin väliajoin ruudun päivitystiheydestä riippumatta. Pelikehityksessä on tyypillistä asettaa fysiikkamallinnuksen päivitykselle kiinteä aika-askel, jolloin se ei vaikuta pelin muuhun logiikkaan ja grafiikan renderöintiin. Metodin päivitystiheyden voi asettaa Unity-projektin asetuksissa. LateUpdate-metodi on kuin Update-metodi, mutta sitä kutsutaan myöhemmin. Tätä käytetään usein kameran liikkumiseen, varsinkin jos kamera on keskitetty johonkin kohteeseen. Näin voidaan varmistaa, että muut liikkuvat objektit ovat viimeisimmän päivityksen mukaisesti oikeissa paikoissaan. OnDestroy-metodi vastaa destructoria. Sitä kutsutaan, kun peliobjekti poistetaan skenehierarkiasta joko ohjelmallisesti tai automaattisesti skenen vaihtumisen yhteydessä. Tässä vaiheessa on hyvä poistaa viittaukset muista skriptitiedostoista, esimerkiksi jos käytössä on publisher-subscriber -malli. Myös ulkoiset resurssit ovat hyvä vapauttaa tässä vaiheessa. (Unity Technologies 2019b.)

ScriptableObject poikkeaa MonoBehaviorista siten, ettei sitä liitetä komponentiksi peliobjektiin, vaan se toimii skenehierarkiasta erillisenä skriptitiedostona, jonka pohjalta luodaan asset-tiedostoja. Nämä asset-tiedostot käyttävät ScriptableObject-tiedossa määriteltyjä

muuttujia ja metodeja. Koska asset-tiedostot eivät ole liitettynä tai riippuvaisia peliobjekteista tai skeneistä, ne ovat kätevä tapa säilyttää ja jakaa dataa skriptitiedostojen kesken skenestä riippumatta. ScriptableObject asset-tiedosto toimii hyvin datapankkina, jonka pohjalta voi luoda serialisoitavan ja deserialisoitavan tallennus- ja latausjärjestelmän.

3.5 Alustatuki ja WebGL

Unity tarjoaa mahdollisuuden luoda peli monelle eri kohdealustalle. Tässä pelissä käytetään alustana WebGL:ää, jolloin peliä voi pelata modernissa web-selaimessa. WebGL-tuki vaatii erillisen moduulin, jonka voi ladata ilmaiseksi. WebGL:lle peliä luodessa Unity käyttää skriptausbackendinaan IL2CPP (Intermediate Language To C++) -työkalua. Pelikehittäjän luomista C#-skriptitiedostot käännetään .NET-kääntäjän välityksellä IL-ohjelmakoodiksi. IL2CPP ottaa tämän IL-ohjelmakoodin sekä valmiiksi luodut IL-assemblyt ja kääntää ne C++ -ohjelmakoodiksi. Tämän jälkeen se käännetään natiiviksi binääritiedostoksi käyttöjärjestelmän C++ -kääntäjällä. Käännetty C++ -ohjelmakoodi käännetään tämän jälkeen WebAssemblyksi käyttämällä emscripten-työkalua. Kääntämisen lopputuloksena on valmis websivunäkymä, joka sisältää pelin suorittamiseen tarvittavat HTML-, CSS-, JavaScript-, UnityWeb- ja kuvatiedostot. HLSL-kielellä kirjoitettujen shader-ohjelmien tavukoodi käännetään GLSL ES-kieleksi, joka on OpenGL:n shader-ohjelmointikieli. (Unity Technologies 2019c.)

WebGL-peliä luodessa tulee etukäteen ottaa huomioon alustan poikkeavuudet yhteensopivuudessa. Koska WebGL-versio perustuu JavaScript-kieleen, se ei tue monisäikeisyyttä. Pelin kehittämishetkellä WebGL-versio perustuu OpenGL ES 2.0 ja 3.0 API-rajapintoihin, joten siitä puuttuu tuki geometry ja compute shaderille, jotka lisättiin OpenGL ES spesifikaatioon versioissa 3.1 ja 3.2 vastaavasti. (Khronos Group 2016, Khronos Group 2019.) Näitä teknologioita käytettiin peliä kehittäessä, mutta ne poistettiin lopullisesta käyttäjäversiosta.

Kuten muissa C++ -kääntäjissä, Unity käyttää kääntäessään objektitiedostoja, jotka linkitetään lopulta ajotiedostoksi. Objektitiedostot ovat välivaihe lähdekoodin kääntyessä lopulliseksi ajotiedostoksi. Jokainen lähdekooditiedosto saa sitä vastaavan objektitiedoston, jotka sitten linkitetään yhteen muodostaen lopullisen ajettavan tiedoston. Jos johonkin lähdekooditiedostoon ei jatkossa tehdä muutoksia, sitä ei tarvitse kääntää uudelleen objektitiedostoksi, mikä nopeuttaa kääntämisprosessia.

3.6 Tekstuurit, tekstuuriatlaat ja spritet

Unityssä valtaosa 2D-pelin visuaalisista elementeistä on sprite-tyyppisiä. Unityyn on mahdollista tuoda rasteriformaattisia kuvia, kuten PNG- ja JPG-tiedostoja, missä tapauksessa Unity luo niiden pohjalta sisäisesti tekstuuritiedostoja jättäen kuitenkin alkuperäiset tiedostot projektikansioon. Tekstuuritiedostoille on asetettava niiden tyyppi, joka kuvastaa niiden käyttötarkoitusta. Sprite-tyyppiä käytetään pelimaailman 2D-elementeissä sekä graafisessa käyttöliittymässä. Normaalista tekstuurista poiketen sprite-tyyppi antaa ylimääräisiä asetuksia, kuten mahdollisuuden käyttää tekstuuriatlasta ja Unityn sisäänrakennettua sprite-editoria tekstuuriatlaan luomiseen. Tekstuuriatlas on yksi tekstuuri, joka koostuu monesta pienestä kuvasta. Yksittäiseen kuvaan viitataan antamalla sen koordinaatit ja koko tekstuuriatlaasta, jolloin se leikataan yksittäiseksi kuvaksi. Tekstuuriatlasta käyttämällä grafiikkaprosessorin tarvitsee ladata vain yksi tekstuuritiedosto sen jokaisen osan renderöimiseen, mikä parantaa suoritustehoa. (Unity Technologies 2019d.)

Unity tarjoaa monta mahdollista tekstuuriformaattia, joista yleisimpiä ovat häviävää kompressointia käyttävät DXT-formaatit. DXT-formaatti kompressoi tekstuurin jakamalla sen 4x4 pikselin osiin. Osan sisällä olevat värit kartoitetaan RGB-avaruuteen, jonka läpi piirretään jana. Janan ääripäiden väriarvoja käytetään kahtena päävärinä, joiden väliltä interpoloidaan vielä kaksi muuta väriarvoa niin, että kaikki neljä väriarvoa ovat yhtä kaukana toisistaan janalla. Jokaiselle osan kuudelletoista pikselille annetaan 2-bittinen indeksi, joka viittaa lähimpään näistä neljästä väriarvosta. Väriarvot tallennetaan RGB565-muodossa, jossa vihreä kanava saa kuuden bitin tarkkuuden ja punainen sekä sininen viiden bitin tarkkuuden. Alkuperäisen kuvan RGBA8888-formaatista muuttaessa alimmat bitit pudotetaan pois. DXT5-formaatti antaa jokaiselle pikselille väri-indeksin lisäksi myös neljän bitin läpinäkyvyys arvon. (Intel Corporation 2012, van Waveren & Castaño 2007.)

Usealle alustalle peliä luodessa on myös mahdollista luoda alustakohtaiset tekstuuriasetukset. Jos jokin tekstuurin sivuista ei ole arvoltaan kahden potenssi, sitä ei välttämättä voi kompressoida tekstuuriformaatista ja tekstuuriasetuksista riippuen. Tekstuuriformaatista riippuen Unity tarjoaa myös mahdollisuuden käyttää Crunch Compression -kompressointimenetelmää, joka kompressoi tekstuuriformaatin häviävästi toistamiseen. Formaatti on erittäin suorituskykyinen ja vie huomattavasti vähemmän tilaa, mutta voi tuoda mukanaan huomattavia kompressioartifakteja. (Suvorov 2019.)

4 PELIN TOTEUTUS

4.1 Graafinen käyttöliittymä

Graafinen käyttöliittymä on kokoelma 2D-elementtejä, jotka sijoitetaan näytettävän pelimaailman kuvan päälle. Sen tarkoitus on näyttää käyttäjälle tietoa sekä toimia valikkona, josta käyttäjä voi muuttaa pelin asetuksia. Unityssä jokaisella käyttöliittymäelementillä tulee olla emopeliobjekti, jolla on Canvas-komponentti. Canvas-komponentti piirtää sen sisällä olevat graafiset elementit sille valitulla renderöintitilalla. Tiloja on kolme: Screen Space - Overlay, Screen Space - Camera ja World Space. Screen spacessa käyttöliittymä renderöidään pelimaailman kuvan päälle. Overlay renderöi elementit litteästi, kun taas Camera renderöi ne kameran nykyisillä perspektiiviasetuksilla. World Space siirtää käyttöliittymäelementit osaksi pelimaailmaa, jonka avulla pelimaailman sisälle voidaan upottaa esimerkiksi tekstiä.

Pelin graafinen käyttöliittymä luotiin peliobjektina, jonka kontrolleriskripti käyttää singleton-mallia. Singleton on ohjelmointimalli, jossa luokalla voi olla vain yksi aktiivinen instanssi. Yleensä luokassa on julkinen ja staattinen viittaus tähän instanssiin, jotta siihen on globaali pääsy muista luokista. Instanssin luomisen yhteydessä instanssi tarkistaa, onko samasta luokasta luotu jo aikaisempi instanssi. Jos instanssi on jo olemassa, uusi instanssi poistetaan käytöstä. Unityssä tämä onnistuu GameObject.Destroy-metodin avulla, jolloin koko peliobjekti poistetaan skenestä. Singleton-malli muistuttaa toiminnaltaan staattista luokkaa, mutta toisin kuin staattinen luokka, singleton-luokka voi periä muusta luokasta. Tämä on erityisen tärkeää Unityssä, jossa MonoBehaviorista periminen mahdollistaa päivitettävän pelilogiikan luomisen. Skriptikomponentti käyttää myös GameObject.DontDestroyOnLoad-metodia, jota käyttämällä peliobjekti ei tuhoudu skeneä vaihdettaessa.

Graafinen käyttöliittymä näyttää käyttäjälle sen hetkisen veden puhtauden sekä hänen keräämänsä pisteet. Tätä varten graafisella käyttöliittymällä on viittaukset pelaajadataan ja vesidataan, joihin se rekisteröityy. Datan päivittyessä graafinen käyttöliittymä saa ilmoituksen hakea data uudelleen, jolloin päivitetty data palautuu ja sen voi esittää käyttäjälle. Joissakin minipeleissä vain tietyt veden elementit voivat muuttaa arvoaan juuri siinä puhdistusvaiheessa. Tällöin vaiheen kannalta tarpeettomat arvot haalistetaan, jolloin käyttäjän huomio voidaan kiinnittää vain oleellisiin arvoihin. Ruudun vasemmalla laidalla on kolme ikonia, joiden kautta käyttäjä pääsee pelin asetuksiin ja pelilogiin, joka pitää kirjaa käyttäjän etenemisestä. Klikkaamalla jotakin ikonia pelin aikana peli pysäytetään ja ikonia vas-

taava sivu avataan. Kun sivu on auki ja jotakin toista ikonia klikataan, korvataan nykyinen sivu valitulla sivulla. Jos nykyisen sivun ikonia klikataan uudelleen, käyttöliittymä sulkeutuu ja peli jatkuu. Käyttöliittymän sivua avatessa sen sisältö ei ole heti näkyvillä, vaan sivulla oleva sisältö tulee esiin animoituna ja yksitellen. Tätä varten käynnistetään korutiini, jonka toimii kuten perinteinen funktiokutsu, mutta sen suoritus tapahtuu usean päivityksen aikana. Korutiini kykenee palautumaan kesken funktion ja jatkamaan myöhemmin toimintaansa siitä, mihin se jäi edellisellä kerralla. Käynnistetty korutiini tuo esille sivun peliobjektin alla olevat lapsipeliobjektit. Nämä edustavat sivun sisältöä ja niillä on komponentteina kuva-, painike- tai tekstielementti, joka näytetään sivulla. Animaatioiden toteutuksessa käytettiin kolmannen osapuolen luomaa DOTween-lisäosaa, joka tekee monivaiheisten animaatiosekvenssien luomisesta yksinkertaista.

Graafinen käyttöliittymä sisältää myös avustusikkunan, joka kertoo käyttäjälle tietoa sen hetkisestä vaiheesta. Jokaisella skenellä on omat avustustekstinsä, jotka määritellään ohjelmakoodin puolella. Jos skenessä on useampi avustusteksti, ikkunan vasemmalle ja oikealle laidalle luodaan nuolinäppäimet, joiden kautta käyttäjä voi vaihtaa näytettävää avustustekstiä. Ikkuna voidaan avata skenen latauksen yhdessä automaattisesti ja tarvittaessa käyttäjälle voidaan näyttää avustustekstien joukosta jokin tietty avustustekstisivu. Peli pysäytetään avustusikkunan ollessa auki ja peli jatkuu sulkemalla avustusikkunan sen oikeasta ylälaidasta.

Graafista käyttöliittymää luodessa tulee huomioida, että jokainen eri fonttia käyttävä tekstikomponentti luo ylimääräisen piirtokutsun skenen renderöintivaiheessa. Suorituskyvyn takaamiseksi tulee käyttää samaa fonttia mahdollisimman monessa paikkaa.

4.2 Tekstuurien optimointi

Koska kyseessä on selainpohjainen peli, sen laitteistovaatimukset yritetään pitää mahdollisimman pienenä. Tämän vuoksi tekstuurien enimmäiskooksi valittiin 2048x2048 pikseliä. Valtaosa markkinoilla olevista näytönohjaimista tukee tätä tekstuuriresoluutiota, mukaan lukien mobiililaitteet, kuten matkapuhelimet ja tabletit. Matala tekstuuriresoluutio pitää myös tarvittavan videomuistin määrän alhaisena. Työssä toimineen graafikon luomat kuvat olivat kokonaisia kuvia tiloista, jotka olivat liian suuria mahtuakseen tekstuurikoon rajoituksiin. Nämä kuvat leikattiin moneksi enintään 2048x2048 kokoisiksi paloiksi ImageMagick-työkalulla ja asetettiin skenessä vierekkäin niin, ettei niiden välinen sauma näy käyttäjälle. Crunch-kompressointia käytettäessä palojen välille tuli huomattavia värieroja, minkä takia kompressointia ei otettu käyttöön. Sprite-animaatioiden yksittäiset kuvat olivat tal-

lennettu samaan kuvatiedostoon niin, että niitä olisi voinut käyttää tekstuuriatlaana. Animaation toiston sulavuuden takaamiseksi jokaisen yksittäisen kuvan tulee kuitenkin olla samankokoinen. Tämänkin tekemiseksi käytettiin ImageMagick-työkalua. Joissain tapauksissa animaatiokuvia oli niin monta, etteivät ne olisi mahtuneet samaan teksturiatlaaseen. Näissä tapauksessa osa kuvista poistettiin.

4.3 Toimisto

Varsinainen peli alkaa jätevedenpuhdistamon toimistosta. Käyttäjän ensimmäinen tehtävä on kerätä huoneessa olevat turvavarusteet. Käyttäjä tutustuu tässä vaiheessa pelaajahahmon liikuttamiseen ja peliympäristönsä interaktiivisuuteen. Huoneessa on info-palloja, jotka tulevat näkyviin pelaajanhahmon lähestyessä niitä. Nämä pallot avustavat pelaajaa etenemään tehtävässään. Kun käyttäjä poimii turvavarusteen, pelaajahahmon ulkoasu päivittyy näyttämään sen yllänsä. Kolme turvavarustetta on toimistossa käyttäjän näkyvillä, mutta viimeinen on piilossa yhdessä kaapeista.

Kuva 4. Toimisto-skene

Toimisto on ensimmäinen varsinainen peliskene. Käyttäjällä etenee jätevedenpuhdistamossa pelihahmon kautta. Pelaajahahmolla on skriptikomponentti, jonka avulla hän liikkuu ja vuorovaikuttaa ympäristöönsä. Pelaajahahmo liikkuu käyttäjän kursorilla klikkaamaan kohtaan. Jos kohdassa on objekti, johon hän voi vuorovaikuttaa, se merkitään pelaajahahmon aktiiviseksi kohteeksi. Kun pelaajahahmo on saavuttanut tietyn etäisyyden kohteesta, suoritetaan objektin ja pelaajan vuorovaikutusfunktio. Vuorovaikutettavat objektit ja niiden rooli määritellään Unityn tägijärjestelmän avulla. Joissakin tapauksissa käyttäjältä otetaan

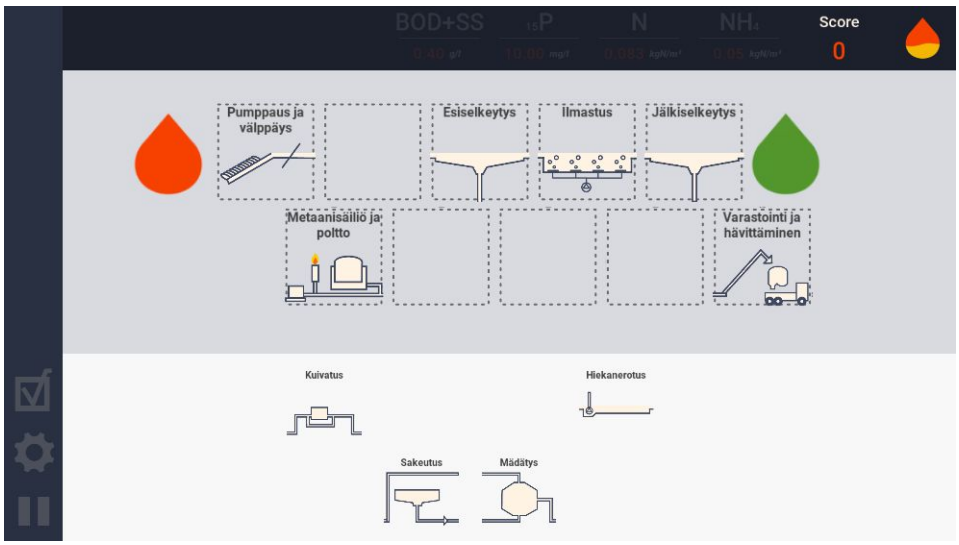
pois mahdollisuus liikuttaa pelaajahahmoa, esimerkiksi sillä aikaa kun hän säätää venttiilejä ilmastusosiossa.

Pelaajahahmoon on liitetty ScriptableObject, joka seuraa käyttäjän etenemistä pelissä. Käyttäjän suorittamat vaiheet, väitekysymysten vastaukset ja keräämät pisteet tallennetaan tähän objektiin. Muut peliobjektit voivat seurata tämän kautta käyttäjän etenemistä. Esimerkiksi toimistossa oleva ovi ei avaudu ennen kuin käyttäjä on pelannut prosessipalapelin. Pelaajalla on myös spawn-kontrolleri, joka määrittelee, mikä on pelaajahahmon paikka skenen latautuessa. Toimistossa pelaajahahmon saama paikka voi olla päävalikosta pelin käynnistymisen yhteydessä saatu oletuspaikka, käytävästä toimistoon palattaessa saatu oven edustan paikka tai tallennetiedostoon kirjattu aikaisemman pelitilan pelaajahahmon paikka.

Käyttäjän tulee kerätä toimistossa olevat turvavarusteet ja jatkaa sitten prosessipalapelin kokoamistehtävään (kuva 4.) Toimiston kontrollerilogiikka perustuu pitkälti tapahtumiin. Käyttäjän poimii turvavarusteen vuorovaikuttamalla sen kanssa, joka aiheuttaa tapahtumakutsun. Turvavarusteen peliobjekti poistetaan skenestä, pelaajan ulkoasu päivitetään ja skene päivittää puuttuvien turvavarusten määrän monitorille.

4.4 Prosessipalapeli

Prosessipalapelissä käyttäjän tulee osoittaa tuntemuksensa jätevedenkäsittelyn vaiheista rakentamalla niiden pohjalta kuva kokonaisprosessista käyttämällä palapelipaloja. Tämä minipeli on kaksivaiheinen. Ensimmäisessä vaiheessa käyttäjän tulee asettaa käsittelyvaiheiden nimillä varustetut palapelipalat oikeille paikoilleen. Toisessa vaiheessa käyttäjän tulee yhdistää vaiheet ja vaihekuvaukset. Käyttäjän suoritus pisteytetään molempien suoritusten jälkeen. Tämän jälkeen käyttäjän pelihahmo siirtyy takaisin toimistoon, josta hän voi siirtyä ensimmäiseen käytävään.



Kuva 5. Prosessipalapeli-skene

Skenessä on kymmenen aukkoa ja kymmenen palaa. Ensimmäisessä vaiheessa paloilla on prosessia osoittava kuva ja nimi. Skenen punainen vesipisaran kuva esittää sisääntulevaa likaista jätevettä ja vihreä vesipisara uloslähtevää puhdistettua jätevettä. (kuva 5.)

Skenen latauduttua palat asetetaan riviin satunnaisessa järjestyksessä. Pelin ensimmäisessä testausvaiheessa palojen siirto tapahtui klikkaamalla, jolloin ensimmäinen klikkaus kiinnitti palan kursoriin ja toinen klikkaus päästi siitä irti. Käyttäjätestauksessa kuitenkin palojen raahaaminen vaikutti luontevammalta. Kun palasta päästetään irti, palan etäisyys jokaiseen aukkoon mitataan ja pala asetetaan lähimpään vapaaseen aukkoon, jos se on tarpeeksi lähellä. Jos pala on liian kaukana, palautuu pala automaattisesti sen oletuspaikalle. Tämä estää myös käyttäjää pudottamasta paloja pelialueen ulkopuolelle, jossa niitä ei pystyisi poimimaan uudestaan.

Palapelipalat ja palapeliaukot ovat peliobjekteja, joille on annettu niiden käsittelyvaihetta kuvaava nimi. Näin jokaiselle palalle löytyy sitä vastaava aukko nimen perusteella. Minipelin controller-skripti seuraa, mikä pala on missäkin aukossa. Kun käyttäjä on asettanut jokaisen palan aukkoon, hänelle näytetään painike, jota klikkaamalla palojen paikat tarkistetaan. Palat väritetään vihreällä tai punaisella riippuen siitä, ovatko ne oikeissa aukoissa. Väärässä aukossa olleet palat siirretään automaattisesti oikeille paikoilleen, jonka perusteella käyttäjä näkee oikeat vastaukset ja jonka pohjalta palapelin toinen vaihe voidaan käynnistää. Toisessa vaiheessa käyttäjälle näytetään kuvaus hänen poimiessaan palan.

Käyttäjän poimiessa palan sen kokoa suurennetaan. Jos pala asetetaan aukkoon, sen Z-koordinaattia siirretään kauemmas kamerasta. Näin muut palat saavat piirtoprioriteetin au-

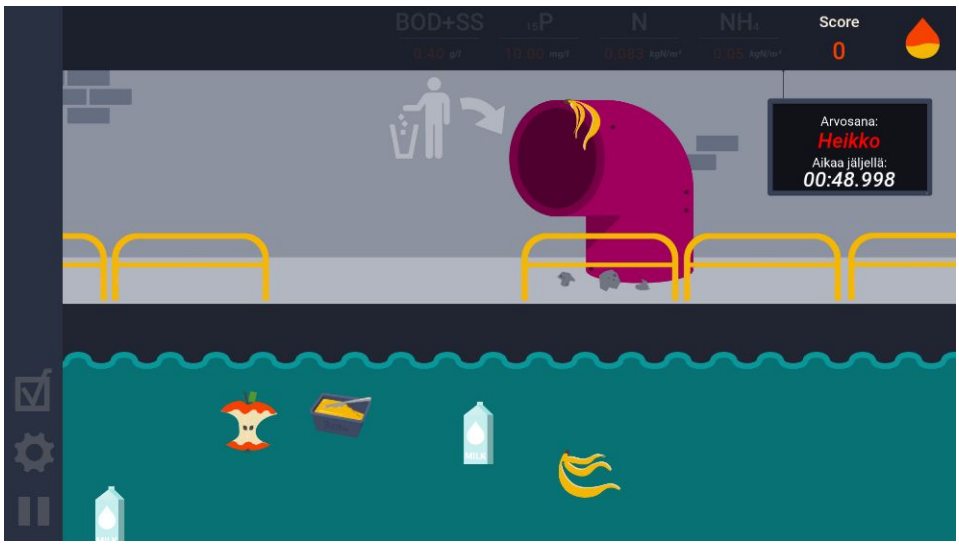
koissa olevien palojen yli, jolloin ne eivät tule peittämään käyttäjän poimimaa aktiivisena olevaa palaa.

4.5 Käytävät

Pelissä on kaksi käytäväskeneä, jotka toimivat minipeliskenejen välisinä siirtymisväylinä. Käytäville on asetettu ovia, joiden kautta käyttäjä voi siirtyä minipeliskeneihin. Käytävien controller-skriptit seuraavat käyttäjän etenemistä pelissä, avaten ja sulkien ovia niin, ettei käyttäjä voi jatkaa eteenpäin ennen kuin vaaditut minipelit on suoritettu. Käytäville on myös asetettu väitekysymyspalloja, jotka testaavat käyttäjän tuntemuksen jätevedenpuhdistuksen vaiheista. Klikkaamalla palloa avautuu ikkuna, joka esittää väitteen puhdistusprosessista. Ikkunan alalaidassa on kaksi painiketta, joilla käyttäjä voi vastata myönteisesti tai kieltävästi väitteeseen. Kysymykset on ryhmitetty käytäville niin, että alkupään kysymykset käsittelevät jätevedenpuhdistusprosessin alkupään vaiheita. Näin sekä väitteet että käyttäjän etenevät samaa tahtia.

4.6 Välppäys

Välppäysminipelissä käyttäjän tulee poimia sisääntulevasta jätevesisyötteestä suuret vieraat objektit pois. Skenen controller-skripti valitsee vieraan objektin tyypin sekä sen Y-koordinaatin satunnaisesti. Jätevesi ja sen mukanaan kuljettamat objektit virtaavat kuvan vasemmalta laidalta oikealle laidalle. Käyttäjän tulee poistaa objektit ennen kuin ne ovat virranneet pois kuvasta. Tässä minipelissä käytetään ajastinta, joka alkaa minuutista ja laskee alaspäin. Veden virtausnopeus kasvaa ajan myötä. Kun aika on loppunut, minipeli päättyy ja käyttäjän suoritus pisteytetään. Käyttäjä saa pisteitä jokaisesta poimimastaan objektista ja menettää pisteitä objektien virratessa kuvasta pois. Tämän jälkeen käyttäjä siirtyy takaisin ensimmäisellä käytävälle, josta hän siirtyy ilmastukseen.



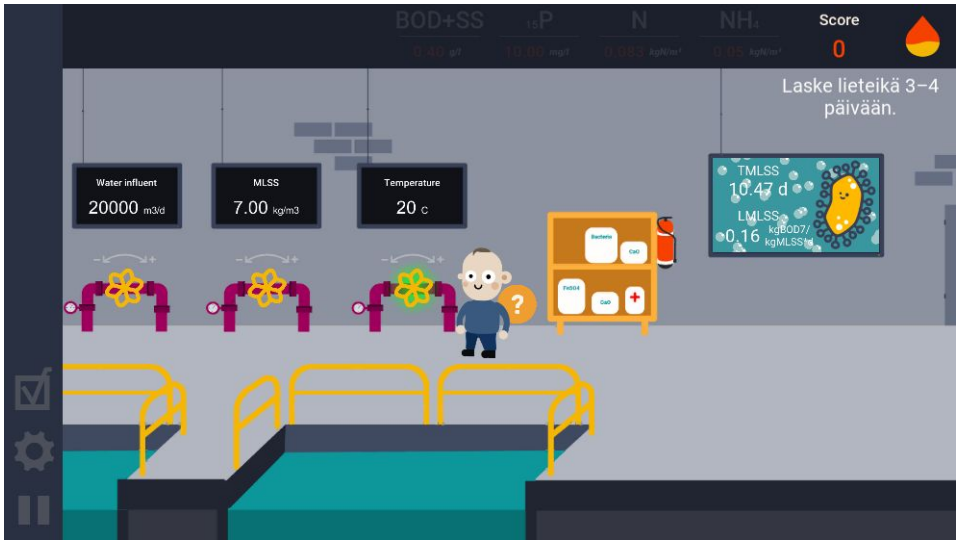
Kuva 6. Välppäys-skene

Käyttäjä poistaa vieraat objektit vedestä klikkaamalla niitä. Pelissä vieraiden objektien kuvaamisessa käytettiin tyhjiä maitokartonkeja, voipaketteja, kiviä, omenoita ja banaaninkuoria (kuva 6.) Jokaisen objektin etäisyys hiiren kursoriin mitataan ja niistä lähin valitaan. Jos tämän valitun objektin etäisyys kursoriin on tarpeeksi pieni, sen ympärille luodaan hehkuefekti, joka osoittaa, että se on poimittavissa. Klikkaamalla objektia se heitetään ruudulla olevaan jäteputkeen, josta käyttäjälle annetaan pisteitä. Annetuista ja menetetyistä pisteistä luodaan ylöspäin leijuva tekstikomponentti, joka osoittaa annettujen tai menetettyjen pisteiden määrän.

4.7 Ilmastus

Ilmastuksessa käyttäjän tulee optimoida vedenpuhdistuksen tehokkuutta säätämällä käytettävän hapen määrää, sisään tulevaa jätevesisyötteen volyymia, kiintoainepitoisuutta (MLSS; Mixed Liquor Suspended Solids) ja veden lämpötilaa. Nämä muuttujat vaikuttavat jäteveden lieteikään (TMLSS) ja lietekuormaan (LMLSS). Käyttäjälle annetaan tehtäviä, kuten lieteiän laskeminen kolmesta neljään päivään. Käyttäjä säätää muuttujia liikuttamalla pelaajahahmon venttiilien lähelle, josta käsin hän voi säätää muuttujia. Kun käyttäjä aktivoi jonkin venttiilin ensimmäistä kertaa, peli antaa hänelle tietoa muuttujasta. Venttiilin avulla käyttäjä voi laskea tai suurentaa muuttujan kuormaa. Tehtävästä riippuen joitakin muuttujia ei voi säätää, vaan käyttäjän tulee tietää, miten annetuilla muuttujilla voidaan säätää lietekuormaa ja lieteikää. Peli näyttää sen hetkisen lieteiän ja lietekuorman monitorilla, joka on aina ruudulla näkyvillä. Käyttäjän asetettua muuttujat hänen tulee palata monitorille, jossa suoritus pisteytetään ja seuraava tehtävä ilmoitetaan. Kun käyttäjä on suorittanut esimerkkitehtävät, hänen tulee puhdistaa jätevesi. Tämä vaikuttaa veden BHK7- ja

ammoniumtypen reduktioon. Kun käyttäjä on suorittanut vedenpuhdistustehtävän, tulee hänen ennen ilmastushuoneesta poistuessa varmistaa, että ilmastuksen hapensyöttö on suljettuna. Käyttäjä poistuu ensimmäiselle käytävälle ja siirtyy sieltä toiselle käytävälle.



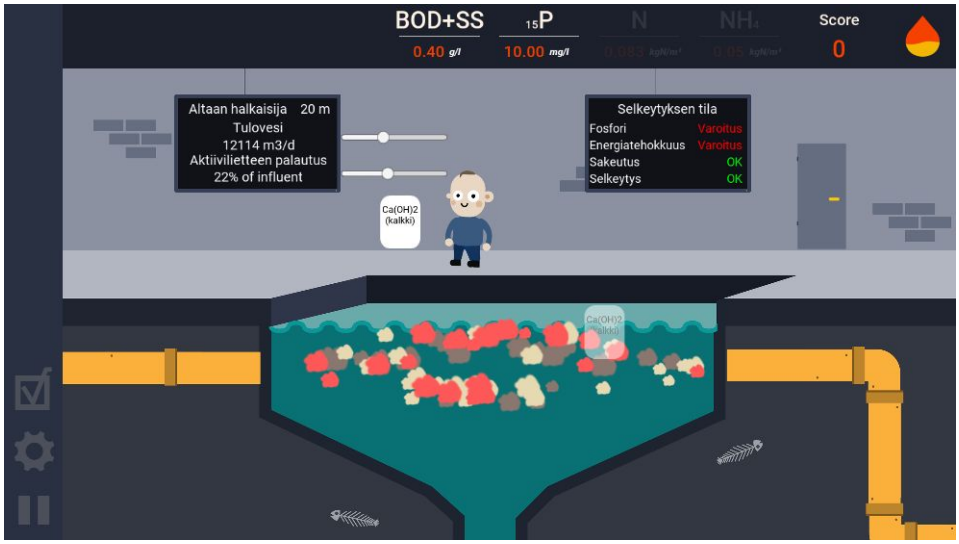
Kuva 7. Ilmastus-skene

Skenen controller-skripti seuraa sen hetkistä tehtävää, jonka perusteella muuttujien aloitusarvot sekä venttiilien lukkiutuminen määräytyy. Skenessä on monitori, joka näyttää lietekuorman ja lieteiän. Jos käyttäjä liikuttaa pelaajahahmoa ja sen kautta kameraa niin, ettei monitoria ole ruudullä näkyvissä, aktivoituu skenen toissijainen kamera. Tämä kamera renderöin vain monitorin lähialueen ja tallentaa tuloksen renderöintitekstuurina. Tämä tekstuuri asetetaan sitten graafiseen käyttöjärjestelmään kameran ruutuavaruuteen, jolloin se näkyy aina käyttäjälle. Monitorin taustalla esiintyy ylöspäin nousevia kuplia, joiden määrä kertoo hapen määrän (kuva 7.) Käyttäjä voi aktivoida venttiilin liikuttamalla pelaajahahmon sen lähelle ja klikkaamalla sitä. Tämän jälkeen kamera siirtyy lähemmäs monitoria ja pelaajahahmon nykyinen paikka lukitaan. Käyttäjä voi kääntää venttiiliä vasemmalle tai oikealle klikkaamalla ja pitävällä painiketta pohjassa.

4.8 Selkeytys

Todellinen selkeytysprosessi on kaksiosainen, jossa esiselkeytysvaihe tapahtuu ennen ilmastusta ja jälkiselkeytysvaihe ilmastuksen jälkeen. Selkeytyksessä jätevesi sisältää rasvaa, lietettä ja fosforia, jotka käyttäjän tulee poistaa. Käyttäjä voi säätää sisääntulevan jätevesisyötteen volyymia sekä sen palautuslietteen (RAS; Return Activated Sludge) osuutta. Nämä muuttujat vaikuttavat jäteveden selkeytykseen, sakeutukseen ja prosessin energiatehokkuuteen. Tehtävässä käytetään State Point Analysis -menetelmää. Sisääntulevan

jäteveden volyymi vaikuttaa siinä olevan rasvan pinnalle nousemisnopeuteen ja lietteen laskeutumisnopeuteen. Pohjalle laskeutunut liete poistuu automaattisesti, kun taas pinnalle noussut rasva pitää poistaa käyttäjän toimesta. Pinnalle muodostuu rasvakerros, jota käyttäjä voi klikata poistaakseen sen. Kun prosessin sen hetkinen puhdistuksellinen ylläpidettävyys on stabilisoitu hyväksyttävästi, huoneeseen ilmestyy kanisteri kalsiumhydroksidia. Käyttäjän tulee liikuttaa pelaajahahmo kanisterin viereen, josta käsin hän voi lisätä sitä jäteveteen. Kalsiumhydroksidi poistaa jäteveden sisältämää fosforia. Käyttäjän suoritus pisteytetään prosessitehokkuuden sekä veteen jääneen rasvan, lietteen ja fosforin määrään perustuen. Käyttäjä voi tämän jälkeen siirtyä takaisin toiselle käytävälle.



Kuva 8. Selkeytys-skene

Käyttäjä säätää muuttujia käyttämällä liukusäätimiä. Muuttujien arvot näkyvät vasemmassa monitorissa ja selkeytyksen tila oikeassa monitorissa (kuva 8.) Käyttäjä voi valita kanisterin klikkaamalla, jonka jälkeen hiiren osoittimeen ilmestyy läpikuultava versio kanisterista, jonka voi lisätä veteen klikkaamalla altaan aluetta. Skenen controller-skripti luo taustalla State Point Analysis -menetelmän pohjalta graafisen kaavion, joka muodostuu kahdesta viivasta ja käyrästä. Menetelmän tavoite on säätää muuttujien arvot niin, että kahden viivan leikkauspiste sijaitsee käyrän sisällä. Tällöin prosessi toimii tehokkaasti. Mitä korkeampi leikkauspisteen Y-koordinaatti on käyrän sisällä, sitä tehokkaammin se toimii. Tehokkuus vaikuttaa käyttäjän suorituksen pisteytykseen. Pelin kehityksen aikana kaaviolle luotiin visuaalinen esityskaavio käyttämällä compute ja geometry shader-ohjelmia. WebGL-versio ei kuitenkaan tue näitä, joten ne jätettiin pois lopullisesta pelistä.

4.9 Pisteytys ja etäserveri

Lopuksi käyttäjä siirtyy toiselta käytävältä tuloshuoneeseen. Huoneessa oleva monitori näyttää, onko käyttäjä suorittanut jätevedenpuhdistuksen hyväksytysti ja kuinka monta pistettä hän on kerännyt. Tämän jälkeen kerätyt pisteet lähetetään etäserverille, jossa ne tallennetaan tilastointia varten. Käyttäjä siirtyy tämän jälkeen takaisin päävalikkoon.

Etäserverille lähetetään käyttäjän nimimerkki, asuinalue, väitekysymysten vastaukset sekä minipelien pisteet. Tiedon lähetys toteutetaan luomalla instanssi WWWForm- ja UnityWebRequest-luokista. WWForm-olioon lisätään kenttä jokaista lähetettävää dataa kohden, jonka jälkeen UnityWebRequest-olio lähettää HTTP POST-pyynnön, joka sisältää WWWForm-olion kentät. Serverin osoite luetaan StreamingAssets-kansiossa olevasta tekstitiedostosta. Tämä mahdollistaa sen, että pelin siirtyessä IWAMA:n kotisivuille ylläpitäjä voi muokata tekstitiedostoa osoittamaan päivitettyä osoitetta. Näin osoitteen muutos tapahtuu ilman, että koko peliä tarvitsee kääntää uudelleen.

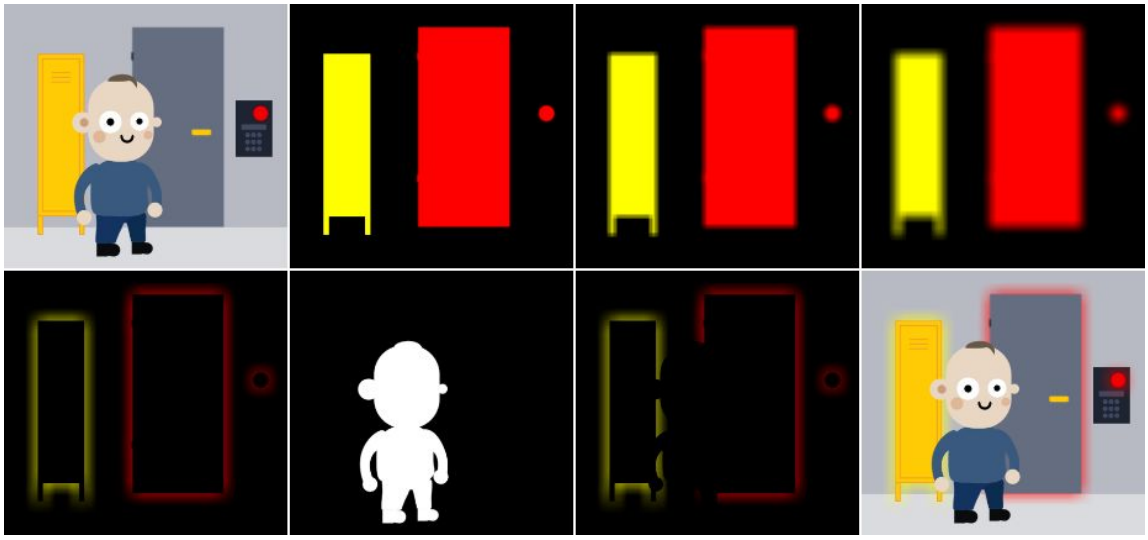
Etäserverillä on PHP-tiedosto ja MySQL-yhteensopiva tietokanta. Serveri lukee vastaantulevan HTTP POST -pyynnön ja kerää sen kentistä käyttäjän tiedot. Tiedot sanitoidaan SQL-injektion varalta ja syötetään sitten tietokantaan.

4.10 Kuvaefektit

Luomalla hehkuefektin spritejen ympärille voidaan käyttäjälle osoittaa, että hänen valitsemansa kohde on interaktiivinen. Hehkuefektin värillä voidaan myös kommunikoida tietoa kohteen tilasta käyttäjälle, esimerkiksi oven vihreä ja punaisella hehkuefektillä voidaan osoittaa, onko ovi sillä hetkellä aukaistavissa vai lukittuna.

MonoBehavior-luokalla on OnRenderImage-metodi, joka suoritetaan sen jälkeen, kun pääkamera on renderöinyt skenen ja on valmis näyttämään sen käyttäjän ruudulla. Tällä metodilla on kaksi RenderTexture-parametriä: lähde- ja kohdetekstuuri. Lähdetekstuuri on sisääntuleva, kameran valmiiksi renderöity tekstuuri, ja kohdetekstuuri on tekstuuri, joka lopulta näytetään ruudulla. Tekemällä muutoksia lähdetekstuuriin dataan ja näyttämällä se kohdeteksuurissa voidaan luoda kuvaefektejä, kuten kuvan sumentaminen tai värien muuttaminen.

Hehkuefekti toteutettiin käyttämällä kameran RenderWithShader-metodia, joka renderöi skenen sen argumentiksi annetulla shader-ohjelmalla. Tätä metodia ei voi kutsua kamerassa, joka on jo samaan aikaan suorittamasta renderöintiä. Tämän vuoksi luotiin toissijainen hehkukamera, joka asetettiin pääkameran lapseksi ja joka jakaa asetukset pääkameran kanssa. Toissijainen kamera renderöi vain hehkukerrokselle asetetut kohteet. Kohteiden materiaaleihin on liitetty shader-ohjelma, jonka määrittelee niille värikentän, jota käytetään hehkuefektin värinä. Ennen efektin luomista jokaisen kameralle rekisteröidyn hehkukohteen sen hetkinen kerros kerätään talteen ja ne siirretään hehkukerrokselle, jossa hehkukamera voi renderöidä ne. Hehkukameran RenderWithShader-metodissa määritelty shader-ohjelma korvaa kohteiden tekstuurivärin näillä värikenttien värillä, jolloin kohteista tulee yksivärisiä.



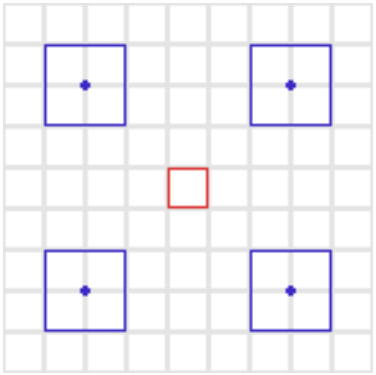
Kuva 9: Hehkuefektin vaiheet

Efektin alussa hehkukamera renderöi hehkukerrokselle asetetut kohteet niille asetetuilla väreillä ja tallentaa tämä väridata väliaikaiseen renderöintitekstuuriin Colors, joka luodaan RenderTexture.GetTemporary-metodilla (Kuva 9, 2. vasemmalta ylhäällä). Metodi ottaa argumenteikseen tekstuurin halutun leveyden ja korkeuden. Tekstuurin resoluution pienentyessä sen pikselitarkkuus laskee, mutta samalla siihen sovelletun fragment shaderin suoritusaika paranee. Tässä tapauksessa käytettiin lähdetekstuurin resoluutiota, sillä resoluutioerosta syntyviä pikselivirheitä ei haluta hehkuefektin ja spriten välille.

Seuraavaksi hehkukohteiden väridata sumennetaan. Tätä varten luodaan kaksi uutta väliaikaista renderöintitekstuuria, Blur1 ja Blur2. Colors-tekstuurin väridata piirretään Graphics.Blit-metodilla Blur1-tekstuuriin halutun shader-ohjelman pass-lohkoa käyttäen, joka tässä vaiheessa on Kawase-sumennus kernelkoolla 1 (Kuva 9, 3. vasemmalta ylhäällä).

Tämän jälkeen sumennus toistetaan kernelkoolla 2 Blur1-tekstuurista Blur2-tekstuuriin (Kuva 9, 4. vasemmalta ylhäällä). Lopuksi sumennetaan kernelkoolla 4 Blur2-tekstuurista Colors-tekstuuriin käyttämällä blendausmoodia OneMinusDstAlpha Zero. Tätä blendausmoodia käyttäessä Blur2-lähdetekstuurista vähennetään värikanavista Colors-kohdetekstuurin läpinäkyvyyskanavan verran, jolloin Colors-tekstuurissa oleva sumentamaton ja opaakki väridata tekee spriten kokoisen aukon tulokseen. Tästä syntyy spriten ympärille hehkuefekti (Kuva 9, 1. vasemmalta alhaalla).

Ei kuitenkaan haluta, että hehkuefekti peittäisi taakseen hehkukohteiden edessä olevia kohteita, esimerkiksi graafisen käyttöliittymän elementtejä ja pelaajahahmoa. Luodaan uusi renderöintitekstuuri Mask, johon hehkukamera renderöi UI- ja hehkumaskikerroksille asetetut kohteet (Kuva 9, 2. vasemmalta alhaalla). Colors-tekstuuri blendataan mask-tekstuuriin edellisellä blendausmoodilla, jolloin hehkuefekti on valmis (Kuva 9, 3. vasemmalta alhaalla). Lopuksi Mask-tekstuuri blendataan kameran lähdetekstuurin kanssa blendausmoodilla One OneMinusSrcAlpha (Kuva 9, 4. vasemmalta alhaalla). Kun efekti on valmis, väliaikaiset renderöintitekstuurit vapautetaan ja peliobjektit palautetaan niiden edellä tallennetuille kerroksille.



Kuva 10. Kawase-sumennus

Sumennuksen shader-ohjelmassa käytettiin Kawase-sumennusta. Muita suosittuja vaihtoehtoja sumennuksen toteuttamiseen ovat laatikkosumennus ja Gauss-sumennus. Ohjelman fragment shaderissa tekstuurin pikselin (kuva 10, punainen alue) väridata muodostetaan sen diagonaaliseista naapuripisteistä (kuva 10, siniset pisteet.) (Kawase 2003.) Haettaessa tekstuurin pisteen väridataa on hyvä sijoittaa piste pikselien raja-arvolle, jolloin palautuu usean pikselin bilineaarisen interpolaation arvo yhdellä hakukutsulla (kuva 10, siniset alueet.) Koska Kawase-sumennuksessa naapuripisteet poikkeavat käsiteltävästä pikselistä molemmilla koordinaattiakseleilla, voidaan pisteet sijoittaa neljän pikselin rajalle. Näin saadaan neljän pikselin värien keskiarvo yhdellä tekstuurihaulla. Sumennuksen lop-

putulokseen vaikuttaa suoritetuskertojen lisäksi sumennusalueen resoluutio ja matriisikertoimet. Tätä hehkuefektiä luodessa käytettiin 3-pass Kawase-sumennusta, joka tuotti suorituskyvyn ja kuvanlaadun kannalta hyväksyttävän tuloksen. Merkittävin vajavuus oli heikko sumennustulos kohdissa, joissa diagonaalista väridataa oli lähialueella vähän käytettävissä, kuten kaapin ohut jalusta ja terävät kulmat.

Sumennusefektiä käytettiin haalistusefektin ohella myös pelin ollessa tauolla, esimerkiksi kun käyttäjä avaa graafisen käyttöliittymän valikon tai kun hänelle näytetään viesti-ikkuna. Haalistetulla ja sumennetulla näkymällä osoitetaan käyttäjälle, että pelimaailma ei sillä hetkellä ole aktiivinen ja hänen fokuksensa alla. Tämän efektin intensiteetti on asteittainen. Pelin mennessä tauolle efekti voimistuu yhden sekunnin ajan ennen kuin lopullinen näkymä saavutetaan. Ohjelmakoodista käsin on mahdollista muuttaa shader-ohjelman ominaisuuksien arvoja sen materiaalin kautta. Tässä tapauksessa käytettiin materiaalin SetFloat-metodia, joka muuttaa shader-ohjelmassa käytetyn efektin intensiteetin kerrointa.

4.11 Serialisointi ja pelitilan tallennus

Serialisoinnilla tarkoitetaan ajonaikaisen ohjelmatilan eli olioiden sen hetkisten muuttujien tilan tallentamista muotoon, jossa se on siirrettävissä ja tallennettavissa ohjelman ulkopuolelle. Videopeleissä on yleistä, että käyttäjällä on mahdollisuus tallentaa sen hetkinen pelitila. Pelitila tallennetaan pysyvään muistiin, kuten kiintolevylle tai muistikortille. Käyttäjä voi tämän jälkeen lopettaa videopeliohjelman tai katkaista virran laitteistosta. Myöhemmin pelaaja voi ladata pelitilan ja jatkaa pelaamista siitä kohtaa, johon hän aikaisemmin jäi. Tällöin tallennettu data muutetaan takaisin ohjelmatilaksi eli toisin sanoen olioiden muuttujien tila palautetaan datan pohjalta. Tätä kutsutaan deserialisoinniksi.

Työn pelissä on myös käytössä tallennus-lataus -järjestelmä. Käyttäjä voi toimistossa ollessaan tallentaa sen hetkisen pelitilan graafisen käyttöliittymän kautta. Pelitila tallennetaan käyttäjän selaimen paikalliseen tallennustilaan. Käyttäjä voi myös ladata tästä tallennustilasta aikaisemmin luodun pelitilan milloin tahansa.

Tallenusjärjestelmää varten luotiin ScriptableObject, jonka kautta voi tallentaa, ladata ja poistaa pelitiloja. Objekti määrittää tallennuspaikan käytettävälle alustalle, tallennettavien tiedostojen nimet sekä tallennettujen tiedostojen kirjoituksen ja lukemisen. Pelitilat tallennetaan JSON-muodossa ja WebGL-versiossa ne tallennetaan selaimen IndexedDB-paikallistietokantaan. Tiedostoon tallennetaan käyttäjän nimimerkki ja sijainti, pelaajahahmon Transform-komponentin tila, jäteveden puhtausdata, pelaajan suorittamat tapahtumat, ak-

tiivisen skenen tila, käyttäjän keräämä pistemäärä sekä tallennuksen aikaleima ja kuvakaappaus.

Tätä ScriptableObject-objektia käyttää tallennuskontrolleri. Sen tehtävä on tallennuksen yhteydessä kerätä tallennettava data ja latauksen yhteydessä jakaa ladattu data. Kontrolleri huolehtii myös tallennuspaikkojen ulkoasun formatoinnista ja käyttölogiikasta.

Kätevä tapa tallentaa dataa on käyttää Dictionary-datatyyppiä, joka on avain-arvo -pareista koostuva sanakirja. Unity ei kuitenkaan tue Dictionary-tyypin serialisointia sellaisenaan. Tämän takia geneerisen Dictionary<TKey, TValue>-luokan pohjalta luodaan uusi abstrakti sanakirjaluokka, joka toteuttaa ISerializationCallbackReceiver-rajapinnan. Rajapinta määrittelee kaksi metodia, OnBeforeSerialize ja OnAfterDeserialize. Serialisoinnin ja deserialisoinnin yhteydessä näitä metodeja kutsutaan, minkä avulla pohjautuvan Dictionary-tyypin sisältö voidaan esittää vaihtoehtoisesta kahdella List-datatyypillä. List-tyyppi on serialisoitava ja järjestetty, joten Dictionary-tyypin avaimet voidaan tallentaa yhteen listoista ja sen arvot toiseen listoista. Nämä listat sitten tallennetaan. Ladattaessa pohjautuva Dictionary-tyyppi täytetään listojen niiden avaimien ja arvojen pohjalta. Täytyy kuitenkin huomioida, että myös käytettyjen avainten ja arvojen tulee olla serialisoitavissa.

Aikaleima tallennetaan numeraalisesti Unix-aikaleimana, joka kertoo, kuinka monta sekuntia on kulunut hetkestä 1. tammikuuta 1970 (UTC), joka on määritelty arvon nollakohdaksi. Aikaleima esitetään käyttäjälle tekstinä hänen paikalliskielellänsä formatoituna. Kuvakaappaus otetaan siitä hetkestä, kun pelaaja avaa graafisen käyttöliittymän ennen kuvaefektien käyttöönottoa ja valikon avautumisanimaation toistoa. Kuvakaappaus tulee tallennusjärjestelmälle Texture2D-tyyppinä, jonka resoluution pohjalta luodaan uusi, alaspäin skaalattu tekstuuri. Skaalatun tekstuurin pikselit käydään läpi ja niiden pohjalta haetaan vastaava pikseliväri alkuperäisestä kuvakaappauksesta käyttämällä Texture2D.GetPixelBilinear-metodia. Skaalatun tekstuurin väridata lisätään, kun kaikki pikselivärit on kerätty. Tämän jälkeen tekstuuri muutetaan ensin PNG-rasterikuvamuotoon ja sitten base64-stringiksi, jonka voi helposti serialisoida. Graafisessa käyttöliittymässä kuvakaappaukset näkyvät RawImage-tyyppinä, joka sisältää tekstuurin ja värisävyn. Base64-stringin kuvakaappausdata muutetaan tavutaulukoksi, jonka pohjalta luodaan uusi tekstuuri. Tämä tekstuuri liitetään RawImage-tyyppin, jolloin se on näkyvissä käyttäjälle.

Vedenpuhtaudelle luotiin ScriptableObject, joka sisältää vedenpuhtauteen vaikuttavia elementtejä, kuten BHK7 ja typpi. Näillä elementeillä on aloitusarvo ja nykyinen arvo. Laskemalla näiden suhde saadaan elementin reduktio, jota pelin lopussa verrataan puhtauden

hyväksyttäviin arvoihin. Tämä ScriptableObject toimii myös publisher-roolissa, ilmoittaen graafiselle käyttöliittymälle arvojen muutoksista.

4.12 Monikielisyystuki

Koska pelin kohdeyleisö koostuu usean maan kansalaisista, on pelissä oltava tuki monikielisyydelle. Tukea varteen luotiin Unityn sisäänrakennetun Text-komponentin pohjalta uusi luokka, joka käyttää sille editorin kautta asetettua tekstiä näytettävän tekstin sijaan tekstiavaimena. Komponentti säilyttää tämän tekstiavaimen käynnistyessään ja pyytää sillä sitä vastaavan tekstin nykyisellä kielellä. Kun varsinainen teksti palautuu, komponentti asettaa tekstikseen tämän tekstin.

Kaikki tekstit sijaitsevat Dictionary<string, Dictionary<string, string>>-tyyppissä sanakirjassa. Avaimena käytetään kielen nimeä, esimerkiksi "fi", ja arvona on avain-arvo -pari tekstiavaimelle ja vastaavalle tekstille, esimerkiksi "INFO_WELCOME_HEADER" ja "Tervetuloa!". Tietokanta alustetaan ensimmäisen tekstikomponentin Awake-metodikutsussa. Komponentti käynnistää korutiinin, joka hakee tekstikäännökset sisältävän tiedoston. Tiedosto on Microsoft Excel-tiedosto, joka sisältää jokaiselle kielelle oman sivun. Jokaisella sivulla on kahden sarakkeen tekstiavain-teksti -pareja. Excel-tiedoston lukemiseen käytettiin kolmannen osapuolen luomaa ExcelDataReader-kirjastoa. WebGL-versiossa tiedosto pyydetään serveriltä UnityWebRequest-luokan SendWebRequest-metodilla. Serveriltä tulevasta vastauksesta luodaan MemoryStream-muistivirta, jolle ExcelDataReader voi luoda lukijan. Lukijan nykyinen nimi kertoo, millä sivulla Excel-tiedostoa ollaan, toisin sanoen mitä kieltä käsitellään. Kieli lisätään sanakirjaan ja se saa arvoksi sanakirjan tekstiavain-teksti -pareja, jotka ovat Excel-sivulla. Lukija lakkaa lukemasta, kun se kohtaa tyhjän rivin. Tämän jälkeen sanakirja on valmis eikä sitä tarvitse luoda uudelleen ajon aikana.

Koska Unity ei pakkaa Excel-tiedostoa sen asset-pakkauksen sisälle, tulee tiedosto asettaa Unityn StreamingAssets-kansioon. Peliä käännettäessä Unity kopioi kansiossa olevat tiedostot ja sisällyttää sen valmiiseen ohjelmaan. Excel-tiedosto valittiin, koska monella tietokoneella työskentelevällä löytyy perustaito sen käyttämiseen. Vaihtoehtoina oli käyttää CSV- (Comma-Separated Values) tai JSON-tiedostomuotoa, mutta nämä olisivat voineet olla kääntäjille teknillisesti hankalia. Excel-tiedoston käyttö toi kuitenkin ongelmia mukanaan. Unity ja .NET Framework eivät sisällä natiivia kirjastoa Excel-tiedostojen lukemiseen, joten käyttöön otettiin kolmannen osapuolen kirjasto. Tämä kirjasto vaati toimiakseen System.Data-kirjastoa, joka ei pelin kehityshetkellä ollut osana Unity-projektin referenssilistaa, vaan tämä kirjasto piti kopioida Unityn asennuskansiosta. Joissain tapauksis-

sa Excel tulkitsi kääntäjien soluihin syötetyn tekstin vääränä datatyyppinä. Esimerkiksi väitekysymysten vastauksien tekstit "totta" ja "ei totta" tulkittiin boolean-datatyypin True- ja False-arvoina, jolloin niiden käännökset katosivat.

Luotu tekstiluokka sisältää listan kielistä ja nykyisen kielen. Luokassa käytetään publisher/subscriber-mallia, jossa komponentti-instanssit rekisteröivät itsensä kielen vaihdon varalta. Kielen vaihtuessa jokaista rekisteröityä komponentille ilmoitetaan tästä, jolloin ne lähettävän tekstiavaimensa uudelleen. Tämä tekstiavain palauttaa tekstin uudella kielellä. Järjestelmän avulla käyttäjä voi vaihtaa kieltä ajon aikana saumattomasti ja kaikki tekstikomponentit päivittyvät välittömästi.

Joissakin tapauksissa teksti vaatii jälkikäsittelynä formatointia, esimerkiksi "Vastatut kysymykset ({0} / {1})", jossa "{0}" on vastattujen kysymysten määrä ja "{1}" on kysymysten kokonaismäärä. Komponenttiin on mahdollista lisätä formatointifunktio, joka palauttaa formatoinnissa käytettävän string-taulukon. Funktiota kutsutaan tekstinhaun yhteydessä ja sitä voi kutsua myös manuaalisesti, esimerkiksi jos tekstin tila vaatii päivittämistä käyttäjän vastatessa uuteen kysymykseen. Funktion ja sen palauttaman arvon määrittelee ulkopuolinen kontrolleriskripti.

WebGL-versiota luodessa tulee käytettävät fontit pakata mukaan peliin. WebGL-versiolla ei ole mahdollisuutta hakea fontteja käyttäjän laitteistolle asennetuista TrueType-fonteista, jolloin käyttöjärjestelmäfontit eivät välttämättä renderöi muita kuin latinalaisia merkistöjä oikein.

5 YHTEENVETO

Työn tavoitteena oli kehittää koulutuspeli, joka on käyttäjän kannalta mielenkiintoinen ja opettava. Työn kehittämiseen kuului ohjelmistokehityksen lisäksi kuvanmuokkaus, minipelien suunnittelu, pisteytyksen toteutus ja jätevedenpuhdistukseen liittyvien matemaattisten mallien soveltaminen pelin ohjelmakoodissa. Työn toteutus vaati Unity-pelimoottorin osaamista sekä editorin että skriptien ohjelmoinnin suhteen. Työn toteutuksessa käytettiin C#-, HLSL- ja PHP-ohjelmointikieliä.

Ohjelmointiteknisesti työstä tuli toimiva, tosin WebGL-version yhteensopivuudessa tuli haasteita työn kehityksen aikana. WebGL-peliä kehittäessä tulee alusta alkaen ottaa huomioon fontti- ja DLL-tiedostojen oikeanlainen pakkaaminen pelin mukaan. Käännöksissä käytettiin Microsoft Excel -tiedostoa sen helppokäyttöisyyden vuoksi, tosin tämä aiheutti ongelmia solujen datatyyppien kanssa.

Grafiikan osalta pelin taustojen käyttö olisi sujunut paremmin, jos yksittäiset visuaaliset elementit olisivat olleet tallennettu yksittäisiksi tiedostoiksi. Kuvia luodessa tulee niiden mittasuhteet pitää kahden potenssissa, jotta niiden tekstuurit kompressoituvat tehokkaasti. Animaatiokehyksten luonnissa tulee mittasuhteet pitää vakiona ja grafiikka keskitettynä, jotta tekstuuriatlaan voi ottaa suoraan käyttöön.

Pelin kehityksen haastavin osa oli puhdistuksessa käytettävien biokemiallisten kaavojen toteuttaminen, minipelien suunnittelu sekä käyttäjän suorituksen pisteyttäminen. Nämä olisi pitänyt saada päätettyä ylimääräisen ohjelmointityön välttämiseksi.

Lopullisesta pelistä tehtiin WebGL-versio ja se siirrettiin IWAMA:n serverille. Pelistä tuli valmis ja se julkaistiin IWAMA:n virallisella kotisivulla, jossa se on pelattavissa.

LÄHTEET

Council of the European Communities. 1991. Council Directive of 21 May 1991 concerning urban waste water treatment (91/271/EEC) [viitattu 29.4.2020]. Saatavissa: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31991L0271

Helsingin seudun ympäristöpalvelut HSY. 2019. Miten jätevesi puhdistetaan [viitattu 1.5.2020]. Saatavissa: https://www.hsy.fi/fi/asiantuntijalle/vesihuolto/jatevedenpuhdistus/Sivut/default.aspx

Intel Corporation. 2012. Fast CPU DXT Compression [viitattu 23.3.2020]. Saatavissa: https://software.intel.com/en-us/articles/fast-cpu-dxt-compression

Interactive Water Management. 2019. About IWAMA [viitattu 31.3.2020]. Saatavissa: http://www.iwama.eu/about

Interreg Baltic Sea Region. 2014. About the Programme [viitattu 1.4.2020]. Saatavissa: https://www.interreg-baltic.eu/about-the-programme.html

Kawase, M. 2003. Frame Buffer Postprocessing Effects in DOUBLE-S.T.E.A.L (Wreckless) [viitattu 23.3.2020]. Saatavissa: http://www.daionet.gr.jp/~masa/archives/GDC2003_DSTEAL.ppt

Khronos Group. 2016. OpenGL ES Version 3.1 (November 3, 2016) [viitattu 29.4.2020]. Saatavissa: https://www.khronos.org/registry/OpenGL/specs/es/3.1/es_spec_3.1.pdf

Khronos Group. 2019. OpenGL ES Version 3.2 (October 22, 2019) [viitattu 29.4.2020]. Saatavissa: https://www.khronos.org/registry/OpenGL/specs/es/3.2/es_spec_3.2.pdf

Microsoft. 2018. Microsoft Dev Center, HLSL [viitattu 29.4.2020.] Saatavissa: https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl

Peterson, J. 2015. An introduction to IL2CPP internals [viitattu 23.3.2020]. Saatavissa: https://blogs.unity3d.com/2015/05/06/an-introduction-to-ilcpp-internals/

Suvorov, A. 2019b. Updated Crunch texture compression library [viitattu 31.3.2020]. Saatavissa: https://blogs.unity3d.com/2017/11/15/updated-crunch-texture-compression-library/

Unity Technologies. 2019a. Shading language used in Unity [viitattu 29.4.2020]. Saatavissa: https://docs.unity3d.com/Manual/SL-ShadingLanguage.html

Unity Technologies. 2019b. Order of Execution for Event Functions [viitattu 29.4.2020]. Saatavissa: https://docs.unity3d.com/Manual/ExecutionOrder.html

Unity Technologies. 2019c. How IL2CPP works [viitattu: 23.3.2020]. Saatavissa: https://docs.unity3d.com/Manual/IL2CPP-HowItWorks.html

Unity Technologies, 2019d. Sprite Atlas [viitattu 29.4.2020]. Saatavissa: https://docs.unity3d.com/Manual/class-SpriteAtlas.html

van Waveren, J.M.P. & Castaño, I. 2007. Real-Time YCoCg-DXT Compression [viitattu 23.3.2020]. Saatavissa: https://developer.download.nvidia.com/whitepapers/2007/Real-Time-YCoCg-DXT-Compression/Real-Time%20YCoCg-DXT%20Compression.pdf