Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Henri Taussi

Maalaustyökaluliitännäisen toteutus pelimoottoriin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

21.11.2018

Tekijä Otsikko Sivumäärä Aika	Henri Taussi Maalaustyökaluliitännäisen toteutus pelimoottoriin 55 sivua 21.11.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Antti Laiho Pääohjelmoija Teemu Viisanen

Insinöörityön tavoitteena oli tehdä 3D-mallien maalaustyökalu liitännäinen Unity-pelimoottorille ja tutkia kilpailevia 3D-mallien maalaustyökaluja ja niiden ominaisuuksia. Projektina toteutettiin 3DM-Spary-niminen maalaustyökalu, jonka uniikki kilpailuvaltti oli sen tallennusominaisuus Unityn pelitilassa. 3DM-Spary-työkalun tallennus perustuu tietokoneen ”temporary”-kansioon tallennettuun väliaikaiseen tekstitiedoston käyttöön.

Kesken 3DM-Spary-työkalun kehityksen eräs kilpaileva 3D-mallien maalaustyökalu sai päivityksessään pelitilassa tapahtuvan tallennusominaisuuden, joka oli vastaava kuin 3DM-Spary-työkalussa toteutettu.

3DM-Spary-työkalun toiminta perustuu Unityn ensimmäisen persoonan pelihahmolle, jolle on mallinnettu maalia ampuva maaliruisku. Maaliruiskulle toteutettiin neljä erilaista sivellintä, jotka tuottavat erilaiset kuviot. Maaliruiskujen tuottaman kuvion väriä, läpinäkyvyyttä ja kokoa pystyy muuttamaan käyttöliittymästä.

Työn lopuksi näiden maaliruiskujen tehokkuutta mitattiin ja tuloksista tehtiin analyysi, jonka kaksi tehottomalla laskentaprosessilla toteutettua sivellintä päätettiin tulevaisuudessa muuttaa samaan muotoon kuin kaksi paremmin testissä pärjännyttä sivellintä. 3DM-Spary-työkalun kehitys jatkuu edelleen.

Avainsanat	pelimoottori, liitännäinen, ohjelmistotuotanto, C#

Author Title Number of Pages Date	Henri Taussi Paint tool plug-in execution to a game engine 55 pages 21.11.2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software engineer
Instructors	Senior Lecturer Antti Laiho Lead Programmer Teemu Viisanen

Goal of this thesis was to make a 3D-model paint tool for Unity game engine and investigate other 3D-model painting tools and their features. In this project painting tool named 3DM-Spary was created and it has a unique feature which enables saving from Unity on runtime. 3DM-Spary painting tool's saving feature is based on saving data to a computer's temporary folder, where it creates temporary text file.

In the middle of development one of the rival 3D-painting tool got a update where it introduced a similar saving feature as in 3DM-Spary painting tool.

3DM-Spary painting tool is based on Unity's first-person character controller, which was given a painting tool. The painting tool has four different brushes which all have their unique pattern. The color, transparency and size of the pattern in each brush can be changed from the inspector.

At the end of the development all four brushes and their performance were tested, and the results were analyzed. Two of the brushes didn't manage in the test, because of the ineffective execution. These brushes are going to be redone in the same way that the two better ones. 3DM-Spray tool's development is going to continue.

Keywords	game engine, add-on, software engineering, C#

Sisällys

Lyhenteet

Unity	Unity game engine. Unity-pelimoottori, jonka avulla voidaan tehdä videopelejä.
GPU	Graphics Processing Unit eli grafikkaprosessori, joka suorittaa 3D-mallien ja kuvien renderointia.
RGB	RGB color model. RGB-värimallin avulla luodaan värejä punaisen, vihreän ja sinisen valon avulla.
PNG	Portable Network Graphics, eli häviötön bittikarttagrafiikan tallennusformaatti.
MB	MB, eli megatavu on tiedon tallennuksen mittayksikkö, jossa mega tarkoittaa miljoonaa ja tavu tilan perusyksikköä.
CPU	Central Processing Unit eli prosessori, joka suorittaa tietokoneen konekieliset käskyt.

1 Johdanto

Insinöörin tilaajana toimi videopeliyritys Barracuda Disaster, jonka toimitusjohtajana ja 3D-mallintajana toimii Henri Taussi ja johtavana ohjelmoijana Teemu Viisanen. Insinöörityön tavoitteena oli toteuttaa 3D-mallien maalaustyökalu Unity-pelimoottoriin, joka säästäisi aikaa pelimaailman grafiikan monipuolistamisessa ja yksityiskohtien luomisessa. Työkalu on nimeltään 3DM-Spray ja sen toteutuksen onnistuessa se laitetaan kaupalliseen levitykseen Unityn Asset Store -kauppapaikkaan. 3DM-Spray-työkalu päätettiin toteuttaa sen ominaisuuksien mahdollistamien työskentelytapojen takia. Tässä työssä ei työn tilaajan määrittelemien rajoitusten takia käsitellä 3D-grafiikkaohjelmia yleisesti, vaan keskitytään Unity-pelimoottoriin tehtyihin toteutuksiin.

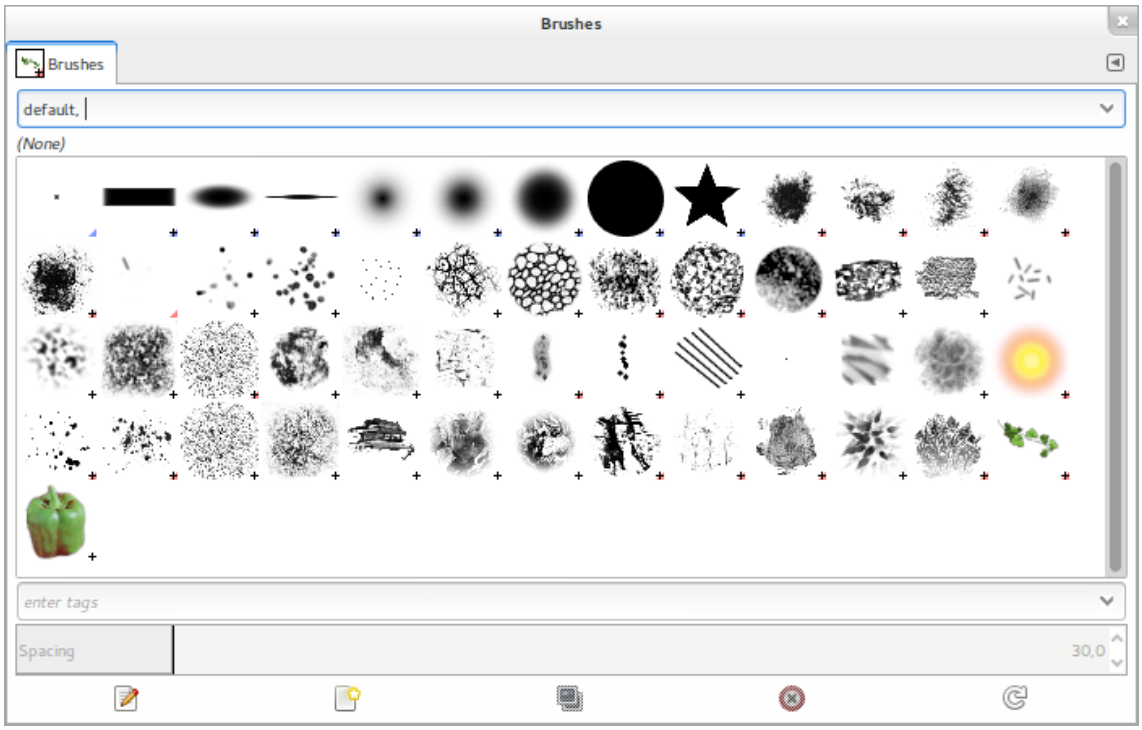
Ilman maalaustyökaluliitännäistä Unityn tehdyn pelimaailman tekstuurien monipuolistaminen vaatii manuaalisesti piirrettäviä muutoksia kuvankäsittelyohjelmalla tai 3D-mallinnusohjelmalla, jolla alkuperäinen tekstuuri on luotu. Tämä työtapa on hidas, varsinkin jos tavoitteena on lisätä pelimaailmaan paljon sattumanvaraisia kuvioita, kuten likaa, öljyä, lunta tai roskia kuvastavia läikkiä. 3DM-Spray-työkalun on tarkoitus helpottaa ja nopeuttaa pelimaailman tekstuurien yksilöllistämistä lisäämällä niihin haluttuja kuvioita. Toteutuksen tulisi olla myös miellyttävä käyttäjäkokemus, jonka takia liikkumiseen käytetään monelle pelinkehittäjälle tuttua perinteistä ensimmäisen persoonan pelihahmon liikkumista.

3DM-Spray-työkalun toimintaperiaate on liikkua ensimmäisen persoonan pelihahmolla pelitilassa Unityn luokassa (eng. scene), joka sisältää kaikki pelin silloisen tason komponentit ja niiden ominaisuudet. Ensimmäisen persoonan pelihahmolla on maaliruisku, jolla käyttäjä voi ampua haluamaansa väriä tai tekstuuria ja kuviota luokan 3D-malleihin. 3DM-Spray-työkalun tekemä kuvio siirretään 3D-mallien tekstuureihin siten, että ennen maalaamista jokaisen maalattavan peliobjektin materiaali ja tekstuuri kopioidaan. Vaatimuksena on, että maalattavan kuvion on oltava saumatonta eri peliobjektien välillä. Pelitilasta poistuttaessa ohjelman tulisi kysyä käyttäjältä, tallennetaanko tehdyt muutokset. 3DM-Spray-työkalun maaliruiskuun on tarkoitus toteuttaa käyttöliittymä, josta käyttäjä voi valita haluamansa kuvion, värin ja levittäytymisalueen.

2 Maalaustyökaluja ja niiden historiaa

2.1 Maalaustyökalujen toiminnallisuus ja tekstuuriin piirtäminen

Maalaustyökalussa on oltava jonkinlainen ohjattava piirtojälkeä tuottava sivellin tai ruisku. Tietokoneen hiiren kursoria seuraava sivellin on tyypillisin maalaustyökaluissa käytetty värin tuottaja, joka joissain työkaluissa saattaa olla graafisesti koristeltu tähtäimellä (eng. crosshair) tai jopa mallinnetulla väriä ampuvalla objektilla. Maalaamisen toimintaperiaate on tekstuurin pikselien RGB-arvojen muuttaminen halutuista koordinaateista. RGB-mallin avulla luodaan miljoonia eri värejä, jotka koostuvat punaisesta (R), vihreästä (V) ja sinisestä (B) väristä. [1.]



Kuva 1. Gimp2-kuvankäsittelyohjelman standardit siveltimet.

Maalaustyökalun värinlevitys tapahtuu kuvan 1 kaltaisten siveltimien, eli eri kuviomallien, tai Unityn fysiikkamoottoriin perustuvan partikkelien ampumisen avulla. Maalaus tapahtuu valitun siveltimen koon määrittämällä alueella siveltimen kuvion mukaisesti.

Toinen vaihtoehto, jolla määritetään kuvion syntyminen on käyttää hyväksi Unityn fysiikoita ja partikkeleita, jolloin siveltimen jäljestä saadaan sattumanvaraisempi. Tällaisessa toteutuksessa maaliruisku tai sivellin luo partikkeleita, joille se antaa voiman liikkua eteenpäin ja mahdollisesti sivuttaissuunnnassa. Muuttamalla partikkeleiden alkusijaintia, levittäytymisaluetta ja niihin kohdistuvia voimia voidaan saada erittäin satunnaisia ja uniikkeja kuvioita.

3D-mallin tekstuuriin piirtäminen Unity-pelimoottorissa

Tekstuuriin piirtäminen tapahtuu tekstuurin pikseleiden värjäämisellä valitun toimintatavan mukaan, kuten siveltimen kuvion mukaisesti. Maalaustyökalut tallentavat jokaisen siveltimen pikselin osuman tekstuuriin korvaten alkuperäisen pikselin RGB-arvon maalaustyökalusta valitulla värillä.

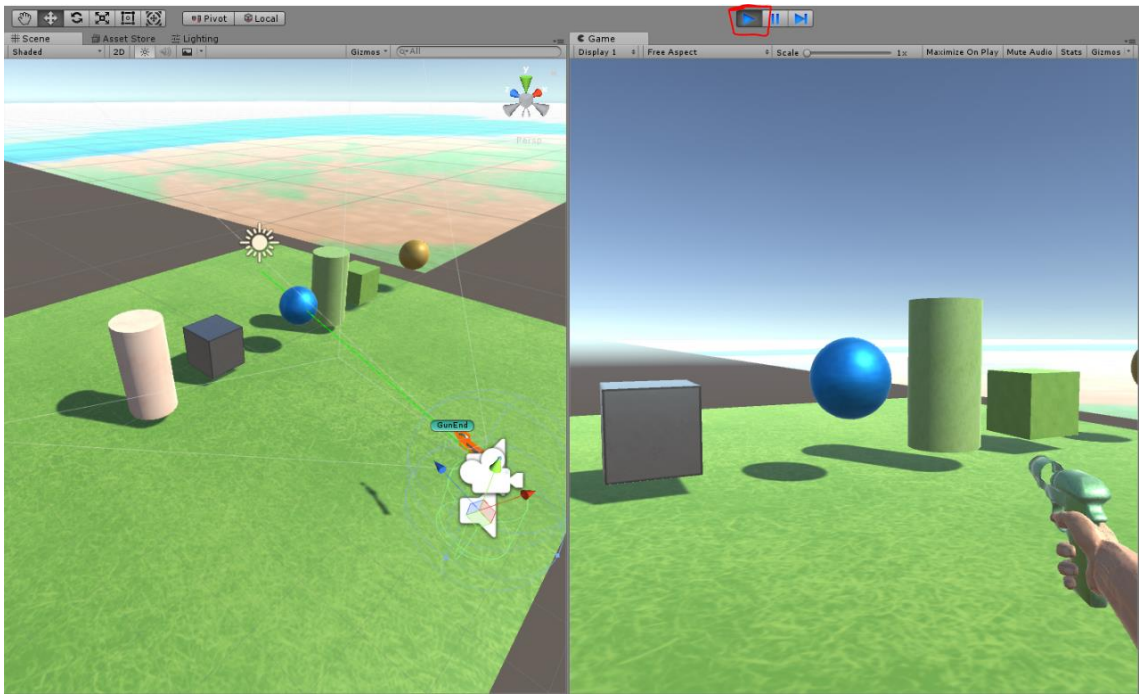
Unityssa tapahtuvan tekstuurien muokkaamisessa on otettava huomioon tekstuurien standardiasetukset, jotka estävät tekstuurin muokkaamisen. Tekstuurin voi aktivoida muokattavaksi tekstuurin omasta käyttöliittymästä tai koodin avulla. Aktivoinnin lisäksi tekstuurin formaatti on muutettava muotoon RGBA 16 bittiä, RGBA 24 bittiä, RGBA 32 bittiä, ARGB 16 bittiä tai RGBA Half. [2.]

Tekstuureihin piirrettäessä tulee kuitenkin muistaa, että useampi materiaali saattaa käyttää samaa tekstuuria tai sama materiaali saattaa olla useammassa objektissa. Useassa maalaustyökalussa tämä aiheuttaa ongelman, jossa yhteen objektiin maalattaessa useampaan objektiin ilmestyy tahattomasti maalattu kuvio. Tämän ehkäisyksi maalaustyökalun on ennen maalaamista kopioitava maalattavan objektin materiaali ja sen jälkeen kopioitava sen tekstuuri ja asetettava objektille kopioitu materiaali ja tekstuuri. Tällöin maalauksen aikana muokataan vain uusia liitännäisen luomia tekstuureita, eikä pelimaailmaan synny tahattomia kuvioita.

Maalaustyökalujen tekemien muutosten tallennustapoja

Tallentaminen voidaan toteuttaa joko automaattisesti tai manuaalisesti. Automaattinen tallennustapa tallentaa muutokset välittömästi niiden tullessa näkyviin ja yleensä tällaisen tallennusratkaisun yhteydessä on myös peruuta-toiminto (eng. Undo), jolla tehdyn

muutoksen voi perua. Manuaalinen tallennus toimii yleensä oman tallennusnappulan kautta. Editointitilassa kummalla tahansa tavalla tallentaminen on mahdollista, kun pelitilassa tallentaminen onnistuu vain manuaalisesti. Tämä johtuu Unity-pelimoottorin ominaisuudesta tehdä varmuuskopio projektista, joka kerta, kun pelitila käynnistetään, ja palauttaa tämä varmuuskopio, kun pelitilasta poistutaan. Unityn pelitilaan siirrytään ja poistutaan kuvasta 2 löytyvän käynnistä-nappulan (eng. play) avulla. Varmuuskopion takia kaikki pelitilassa tehdyt muutokset katoavat, joten tallentaminen pelitilassa on tehtävä Untiyn ulkopuolelle tietokoneen muistiin.



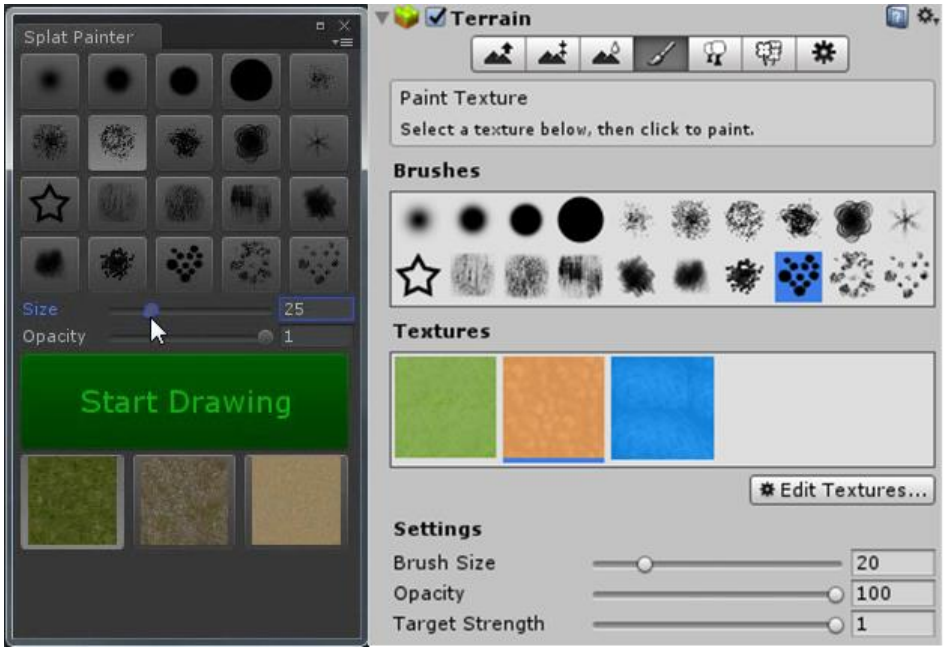
Kuva 2. Vasemmalla Unity-editointitilan näkymä ja oikealla pelitilan näkymä 3DM-Spray-työkalun testiprojektissa. Käynnistä-nappula on korostettu punaisella laatikolla.

Käytännön kannalta automaattitallennus voi olla tehokas ja nopea tapa työskennellä, mutta virheiden sattuessa se saattaa aiheuttaa paljon korjattavaa. Joissakin maalaustyökaluissa toki on peruuta-nappula (eng. undo), jolla virheellisen muutoksen saa peruttua. Manuaalisen tallennuksen etuna on mahdollisuus kokeiluun ja valtaan päättää haluaako todella tallentaa muutokset. Manuaalisen tallennuksen suurin heikkous on käyttäjävirheen mahdollisuus, eli tallennuksen epäonnistuminen, mikäli käyttäjä unohtaa sen tehdä.

Maalaustyökalujen käyttöliittymät ja niiden asetukset

Käyttöliittymiä on yhtä monta erilaista kuin on liitännäisiäkin, mutta niissä on paljon yhteisiä piirteitä. Maalaustyökalujen tärkeimpiin omaisuuksiin kuuluu maalattavan kuvion muokattavuus, joista yleisimpiä on värin ja tekstuurin vaihtaminen, eri siveltimien vaihtaminen, kohdealueen tai siveltimen koon muuttaminen ja läpinäkyvyys (eng. transparency). Värin vaihtaminen tapahtuu käyttöliittymän väripaletista, joka usein liitännäisissä on Unityn materiaalien väripaletin kaltainen. Käyttöliittymään on useasti lisätty myös toiminnallisuuteen liittyviä nappuloita, jotka liittyvät tallentamiseen, tekstuurien asetuksiin ja varjostimien käyttöön.

Tekstuurin valinta maalattavaksi kuvioksi on tyypillisesti toteutettu Unityn tekstuurin valintavalikon kautta tai työkalun käyttöliittymään kustomoidun pikavalikon kautta, joka toimii samalla tavalla kuin Unityn Terrainin käyttöliittymästä löytyvä teksturiin valintavalikko. Kuvassa 3 on Teenothequen Splat Painter -liitännäisen ja Unityn Terrainin käyttöliittymät vierekkäin ja niiden käytössä olevat tekstuurit on asetettu niiden valintavalikkoihin, joissa ne näkyvät kuvakkeina ja niiden välillä voi vaihdella aktiivisena olevaa tekstuuria klikkaamalla halutun tekstuurin kuvaketta. [3.]



Kuva 3. Vasemmalla Teenothequen Splat Painter-liitännäisen käyttöliittymä ja oikealla Unityn Terrain käyttöliittymä.

Eri siveltimien valinta mukailee tyypillisesti jonkin yleisessä käytössä olevan kuvankäsittelyohjelman valikkoa ja sisältää yleisimmät sivellinkuviot: erikokoiset ympyrät reunojen sumennuksella, neliön, ruiskukuviot ja roiskekuviot. Unityn Terrain käyttöliittymä on tässäkin kohdassa antanut monelle liitännäiselle paljon vaikutteita.

Kohdealueen tai siveltimen koon muuttamiseen, värin tai kuvion läpinäkyvyyden muuttamiseen ja maksimiläpinäkyvyyden (eng. target strength) muuttamiseen on lähes poikkeuksetta käytetty vierityspalkkeja (eng. scroll bar) ja teksti-ikkunaa numeeriselle arvolle. Käyttöliittymät näiden asetusten muuttamiseen ovat lähes standardinomaisesti ulkonäöltään samanlaisia kuin Unityn Terrain-käyttöliittymässä, johon muutettavat asetukset on listattu päällekkäin ja jokaisen vieressä on niihin kuuluva vierityspalkki ja teksti-ikkuna.

2.2 Katsaus kilpaileviin maalaustyökaluihin

Katsaus kilpailevaan Play Time Painter -liitännäiseen

3DM-Spray-työkalu toimii samalla periaatteella kuin Iurii Selinnyi:n kehittämä Unity liitännäinen Play Time Painter (julkaistu 8. heinäkuuta 2017) [4], jossa pelitilassa voi maalata objektien tekstuureihin ja tallentaa tehdyt muutokset. Se on myynnissä Unityn Asset Strore -verkkokaupassa hintaan 13,40 dollaria, joka on 10.10.2018 11,65 euroa [5]. Play Time Painter ominaisuuksien kehityksessä on painotettu eri siveltimien ominaisuuksiin ja niiden tehokkaaseen toteutukseen.

Maalaustyökalun lisäksi Play Time Painter:ssa on lukuisia muita ominaisuuksia, kuten 3D-mallien muokkausominaisuus ja tuki mobiilialustoille optimoitujen materiaalien varjostimille (eng. shader). Varjostin on Unityn ohjelma, joka pyörii grafiikkaprosessorin (eng. GPU) avulla ja renderöi objektit. Useasti niistä puhuttaessa tarkoitetaan kuitenkin niiden ominaisuuksia, jolla materiaaliin saadaan luotua varjoja ja se saadaan näyttämään graafisesti realistisemmalta [6].

Kuva 4. Play Time Painterin lokakuun päivityksessä parani myös työkalun käyttöliittymä.

Play Time Painter -liitännäiseen tuli uusin päivitys lokakuussa 2018, jossa se esitteli uuden ominaisuuden pelitilassa tehtyjen muutosten tallentamisen ja kuvan 4 mukaisen käyttöliittymäpäivityksen. Tallentaminen toimii tallennus- ja latausominaisuuksien avulla, joten Iurii Selinnyin on ratkaisut tallentamiseen liittyvät ongelmat vastaavalla toteutuksella kuin 3DM-Spray-työkalu. Play Time Painterin viimeisimmän päivityksen myötä 3DM-Spray-työkalun tallentamisominaisuus ei enää ole ainutlaatuinen, ja se jakaa parhaimman myyntivalttinsa huomattavasti kehittyneemmän kilpailijansa kanssa.

```
67          public string SaveInPlayer()
68          {
69              if (texture2D != null)
70              {
71                  if (destination == TexTarget.RenderTexture)
72                      RenderTexture_To_Texture2D();
73
74                  var png = texture2D.EncodeToPNG();
75
76                  string path = Path.Combine(Application.persistentDataPath, savedImagesFolder);
77
78                  Directory.CreateDirectory(path);
79
80                  string fullPath = Path.Combine(path, "{0}.png".F(SaveName));
81
82                  System.IO.File.WriteAllBytes(fullPath, png);
```

Kuva 5. Kuvakaappaus Play Time Painter -liitännäisen lähdekoodista, jossa on tallennus pelitilassa-funktio [7].

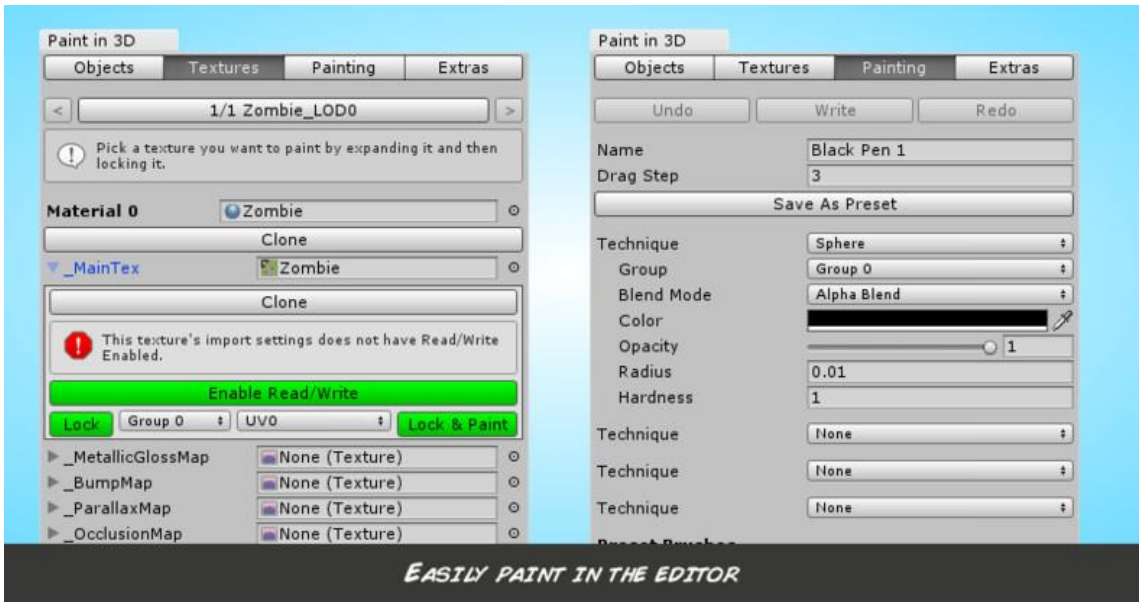
Iurii Selinnyi:ltä tallennus ominaisuudesta kysyttäessä hän vastasi koodin olevan avointa lähdekoodia ja antoi linkin koodiin Github-palveluun [7]. Kuvassa 5 näkyy Play Time Painter -liitännäisen lähdekoodista poimittu tallennus pelitilassa-funktio, joka tallentaa kuvan uuteen kansioon Unity:n osoittamaan kansioon tietokoneelle [8]. Tämä tapahtuu ”Application.persistentDataPath”-tiedostopolun avulla, joka vie julkiseen kirjastoon ja on tarkoitettu tiedon tallentamiseen pelitilojen välillä. Kuvassa 6 näkyvä lataus-funktio vastaavasti lataa kuvan tiedostopolun avulla, joten tallennusperiaate on sama kuin 3DM-Spray työkalussa.

```
100         public void LoadInPlayer(string path)
101         {
102             if (File.Exists(path))
103             {
104                 var fileData = File.ReadAllBytes(path);
105                 if (!texture2D)
106                     texture2D = new Texture2D(2, 2);
107 
108                 if (texture2D.LoadImage(fileData))
109                     Init(texture2D);
```

Kuva 6. Iurii Selinnyi:n tallennus- ja latausfunktiot toimivat myös tallentamalla Unity:n ulkopuoliseen kansioon.

Katsaus kilpailevaan Paint in 3D-liitännäiseen

Carlos Wilkens:n maalaustyökalu Paint in 3D (julkaistu 3. marraskuuta 2015) on kilpailevista liitännäisistä yksinkertaisin ja ominaisuuksiltaan suppein [9]. Se on myynnissä Unityn Asset Store -verkkokaupassa hintaan 26,80 euroa, joka on 02.11.2018 30,62 dollaria [10]. Sen tärkeimmät ominaisuudet ovat maalausominaisuus erilaisilla kuvioilla 3D-malleihin ja niiden muokkaaminen. Ominaisuuksista löytyy myös pelitilassa maalaaminen, mutta tallennusominaisuutta pelitilassa ei ole listattu.



Kuva 7. Paint in 3D-käyttöliittymästä löytyy tekstuurin muokkaukseen liittyvien asetuksien hallinnan lisäksi tallennusmahdollisuus ja maalaustyökalun asetukset [9].

Kuvan 7 mukaisesti Paint in 3D -liitännäisen käyttöliittymä on yksinkertainen, eikä sisällä kilpailijoidensa tavoin graafisia elementtejä. Käyttöliittymä on koko liitännäisen tavoin tehokas siinä, mihin se on suunniteltu, eikä sisällä ylimääräisiä ominaisuuksia. Painikkeissa ja väripaleteissa on käytetty Unityn standardien mukaista ulkonäköä ja asettelua, joka nopeuttaa liitännäisen opettelua ja käyttöönottoa. Käyttöliittymästä löytyy myös varoitusikkuna ja painike, mikäli tekstuuriasetuksissa sen muokkaamista ei ole aktivoitu. Automatisoidun asetuksien muokkaamisen tai manuaalisesti tekstuurin oman käyttöliittymän kautta asetuksen muuttamisen sijaan käyttöliittymän aktivoi muokkausnappula (eng. Enable Read/Write) on yksinkertainen ratkaisu vaikeaan ongelmaan.

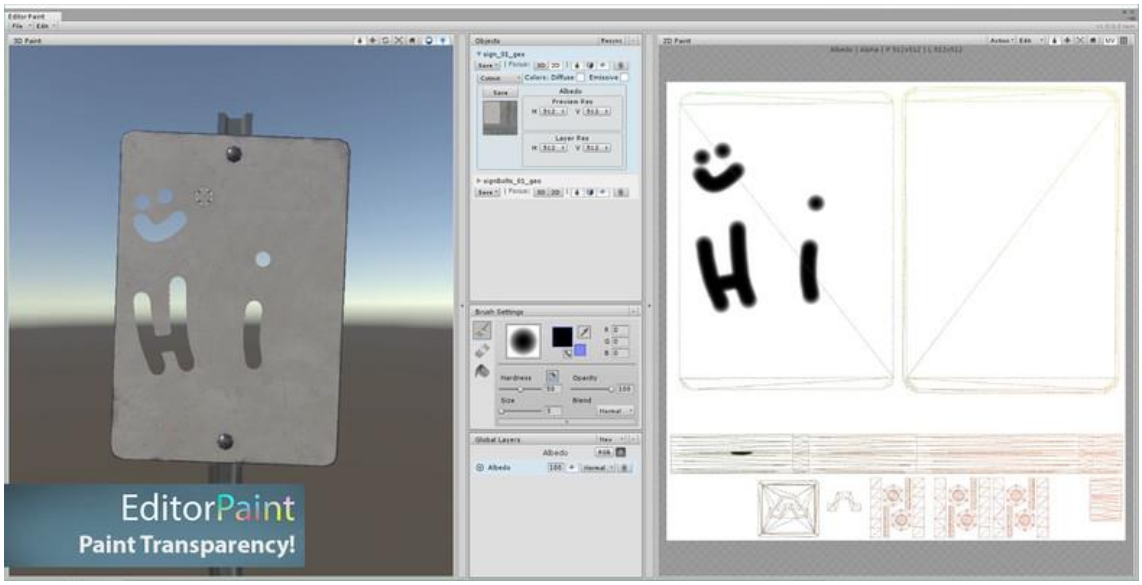
Päivitykset ja tuki

Paint in 3D -liitännäiseen on tullut julkaisunsa kahdeksan päivitystä, joista viimeisin tuli 2. marraskuuta 2018. Päivityksissä on esitelty uusia ominaisuuksia ja korjattu ohjelmointivirheitä. Liitännäinen toimii Unityn versioilla 4.6.1, 5.0.0, 5.5.0, ja 2017.1.0.

Katsaus Editor Paint -liitännäiseen

Prized Goatin liitännäinen Editor Paint (julkaistu 26. tammikuuta 2018) on maalaamisen laatuun ja monipuolisuuteen keskittyvä työkalu [11]. Maalausominaisuuden lisäksi Editor Paint:n lisäominaisuudet ovat vähäiset, eikä siinä ole pelitilan ominaisuuksia ollenkaan. Se on myös kilpailijoistaan kallein hinnalla 34,84 dollaria, joka on 10.10.2018 30,30 euroa. [12.]

Editor Paint on kilpailijoitansa monipuolisempi maalaamisen ominaisuuksiltaan, jotka mahdollistavat jopa varjostimien (eng. shader) käytön maalauksessa ja läpinäkyvyydellä maalaamisen, jonka avulla kuvan 8 mukaisesti valmiiseen 3D-malliin voi maalata reikiä.



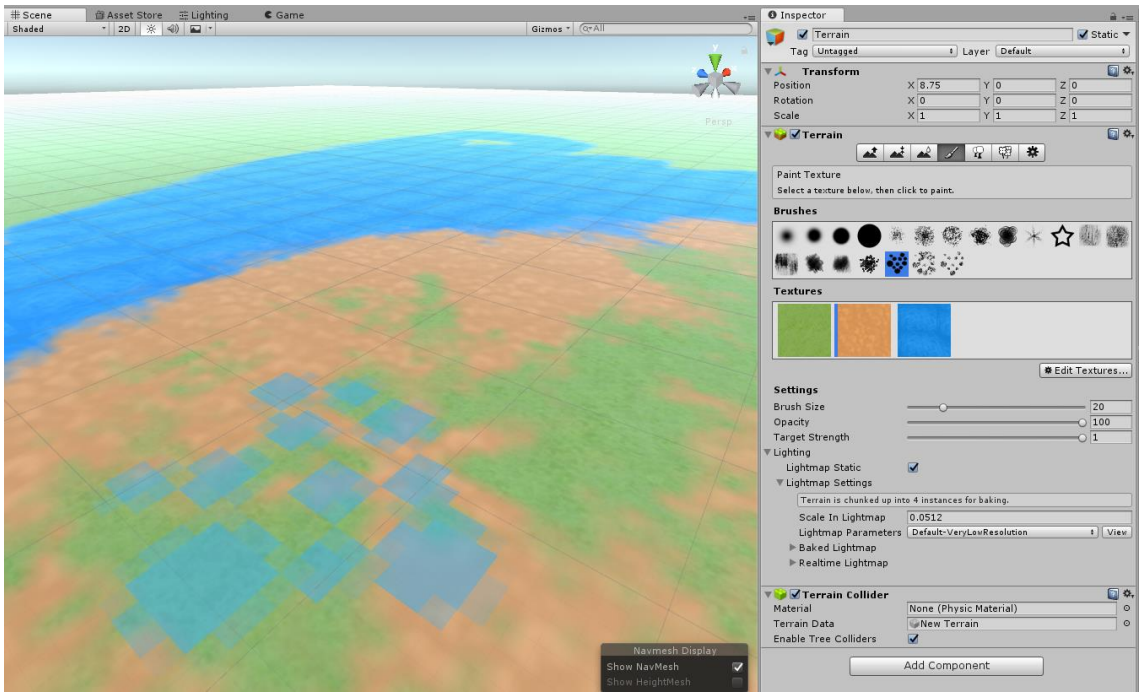
Kuva 8. Tavallisen maalaamisen lisäksi Editor Paintilla voi hyödyntää varjostimien ominaisuuksia.

Kuten kilpailijansakin Editor Paint-työkalu on saanut korjaavia päivityksiä julkaisunsa jälkeen. Viimeisin versio oli versio 0.9, jossa työkalu sai uusia ominaisuuksia. Yksi uusista ominaisuuksista on partikkeliefektien maalaaminen, joka toimii vain 2D-tilassa.

2.3 Maalaustyökalujen historia

Unityn Terrain muokkaustyökalu

Kaikkien Unityn maalaustyökalujen inspiraationa on alun perin ollut Unityn oma Terrain-peliobjekti, jolla on lukuisien muiden ominaisuuksien lisäksi yksilöllinen maalaustyökalu [13], josta on kuvakaappaus kuvassa 9. Tämä maalaustyökalu on ensimmäinen Unityssa siveltimillä toiminut maalaustyökalu, joka on liitetty Terrain-peliobjektiin eikä toimi sen ulkopuolella.



Kuva 9. Vasemmalla Unity:n Terrain editointi ikkunassa ja oikealla Terrain:n maalaustyökalun käyttöliittymä, johon on lisätty kolme eri tekstuuria.

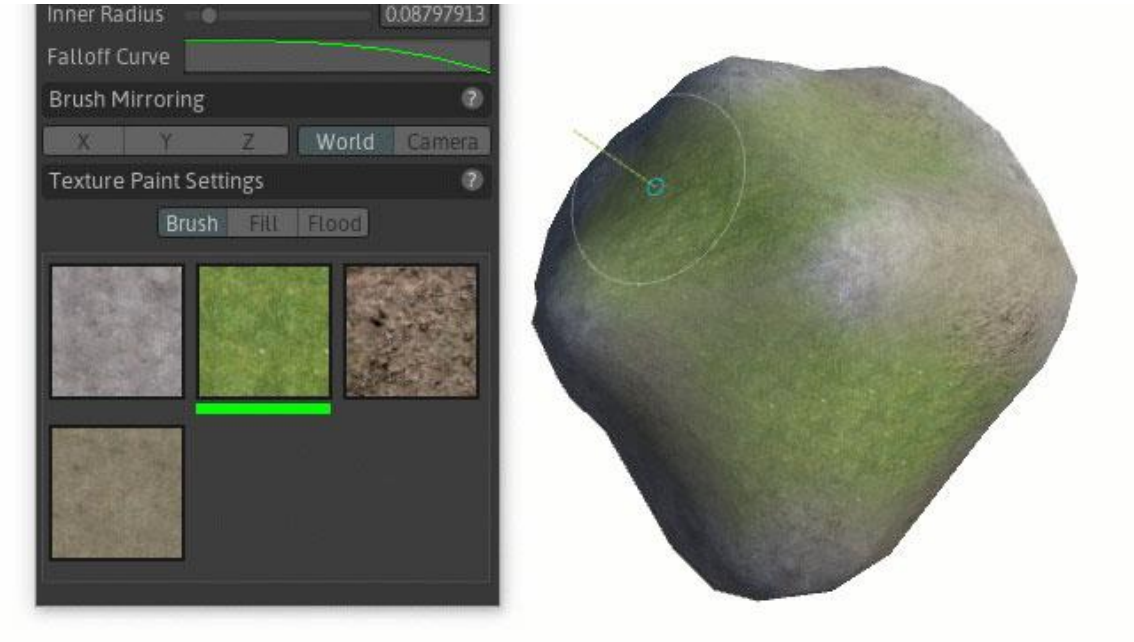
Maalaustyökaluliitännäisistä poiketen Terrain-maalaustyökalu ei maalaa tekstuuriin vaan tallentaa maalattavaa tekstuuria bittikarttaan, johon tallennetaan myös Terrainin korkeuserot. Unity:n Terrain-maalaustyökalussa on tyypillinen siveltimen valinta, koko ja läpinäkyvyys asetukset, sekä tekstuurin valintaominaisuus. Terrain voidaan kuvitella olevan kolmiulotteinen kangas, johon voi maalata ja jonka maalatun pinnan päälle voidaan maalata useamman kerran.

Ensimmäiset 3D-mallien maalaustyökalut Asset Storesta

10. marraskuuta 2010 julkaistiin verkkokauppapaikka Unity Asset Store, josta lähes kaikki Unityn liitännäiset ovat ostettavissa tai ladattavissa [14]. Erilaisia maalaustyökaluliitännäisiä on ladattu Unity Asset Store -verkkokauppaan useita kymmeniä, joiden käyttötarkoitukset ja ominaisuudet ovat hyvin erilaisia. Yksinkertaisimmat maalaustyökalut toimivat vain 2D-peliprojekteissa ja niillä voi levittää väriä vain editointitilassa. Osa maalaustyökaluista on spesifioituja tiettyihin ominaisuuksiin, kuten pelitilassa roiskittavat verimaalausominaisuudet, öljyväreihin erikoistunut maalaustyökalu tai lukuisat Unityn Terrain-maalaustyökalun lisäosat [15].

Polybrush, tulevaisuudessa markkinoita dominoiva maalaustyökalu

Polybrush liitännäinen on Unity Technologiesin omistama maalaustyökalu, joka oli alun perin Gabriel Williamsin ja Karl Henkelin tekemä liitännäinen [16]. Unity Technologies osti Probuilderin ja Polybrushin vuonna 2018 ja palkkasi kyseiset kehittäjät [17]. Tämä mahdollisti Polybrushin ylivoimaisen kehityksen maalaustyökalujen joukossa. Se on mitä todennäköisimmin tulevaisuudessa markkinoita dominoiva maalaustyökalu Unityssa toimivien maalaustyökalujen joukossa, koska se on ilmainen ja sillä on Unityn oma tekninen tuki takana. Sen edistyksellisyyden voi huomata jo käyttöliittymän asetuksista, kuten kuvassa 10 näkyvässä pudotus-vektorimuuttujasta (eng. Falloff curve), jolla värin läpinäkyvyyttä voidaan muuttaa siveltimen reunoilta.



Kuva 10. Polybrushlla voi maalata tekstuureja 3D-malleihin.

Polybrush on monipuolisin maalaustyökalu, mitä markkinoilta löytyy, koska sillä on mahdollista veistää ja muokata muotoja, maalata tekstuureilla ja väreillä, maalata vertekseillä, eli tekstuurittoman materiaalin maalaaminen usealla eri värillä [18], ja objektien lisäämisen siveltimellä. Monipuolisesta maalaustyökalusta huolimatta Polybrush ei toimi pelimuodossa ja sen toiminta muistuttaa hyvin paljon Unity Terrain -työkalun toimintaa, koska silläkin voi muokata objektien muotoja, sekä lisätä objektin pintaan pienempiä objekteja.

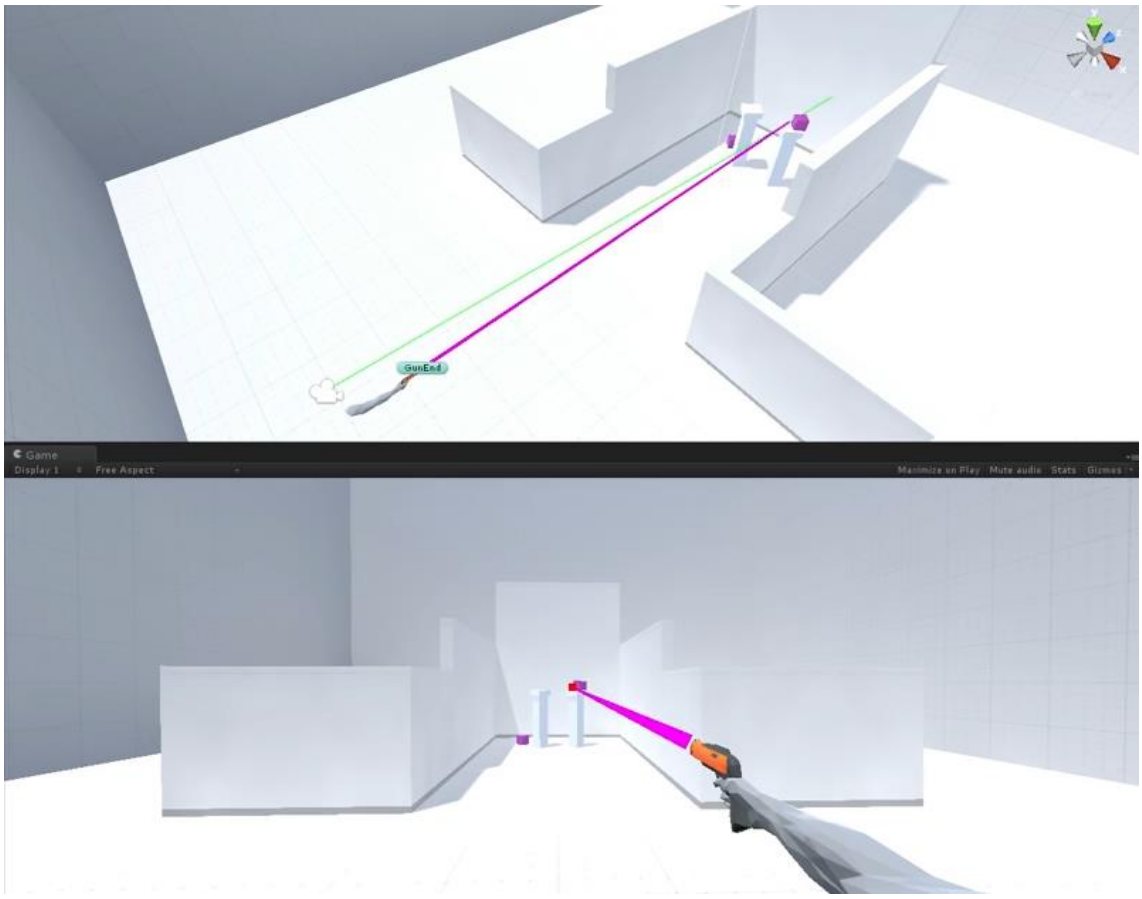
3 3DM-Spray-työkalun vaatimusmäärittely ja ominaisuudet

3.1 Yleiset ohjelmistovaatimukset 3DM-Spray-työkalulle

3DM-Spray-työkalu toimii Unity-pelimoottorilla Windows-käyttöjärjestelmässä. Ensimmäinen versio toimii Unityn versiolla 2017.3.1 ja saa olla maksimissaan 26 MB. Maalaustyökalu tarvitsee toimiakseen maalattavan 3D-mallin, jossa on verkkotörmäytin (eng. mesh collider) ja UV-kartta. Sillä on oltava myös materiaali, jolla on ”albedo” -teksuuri.

3DM-Spray-työkalun on tarkoitus olla ensimmäisen persoonan pelattava hahmo, joka koostuu yhdestä paketista (eng. prefab). Tarkoitus on minimoida Asset-kansioiden määrä ja pyrkiä minimoimaan liitännäisen mukana tulevat pelikomponentit, jotta projektin hallinta ei vaikeudu liiallisen komponenttimäärän takia. Työkalun koko voi olla suhteellisen iso ja tiedostoja sillä voi olla niin monta, kuin on tarpeellista. Työkalu tulisi tukea myös kaikilla uudemmilla Unity-versioilla ja tulevaisuudessa päivittää toimimaan tulevilla versioilla. Ohjelman tulee pitää lokia konsoli-ikkunassa ja kirjata siihen sen suorittamat toiminnot.

Ensimmäisen persoonan kontrolleri



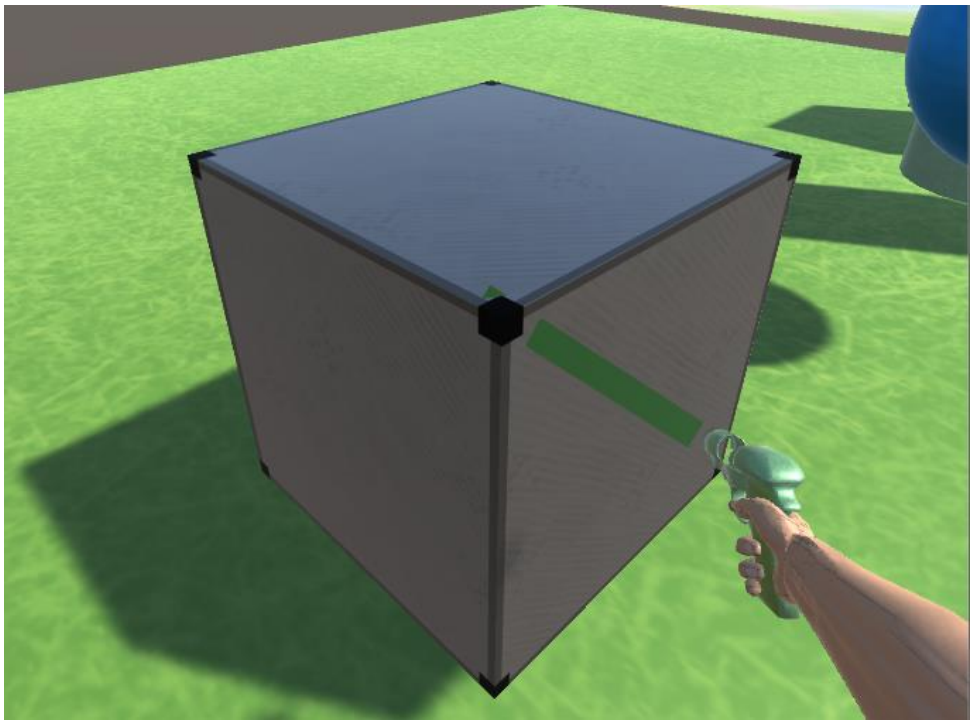
Kuva 11. Unity-kurssi:” Let’s Try: Shooting with Raycast” -säteen ampuminen toiminnassa.

3DM-Spray-työkalun pohjana toimii Unityn standardi ensimmäisen persoonan kontrolleri ja sen lisäksi siihen on luotava maaliruisku ja säteen ampuminen (eng. raycast) kuvan 11 mukaisesti. Maaliruiskuun tehdään 3DM-Spray-työkalun käyttöliittymä ja sen

toiminnan mahdollistava koodi. Se saa hierarkiassa lapsikseen (eng. children) 3D-mallit maaliruiskuista ja pelaajan kädestä. Säde, joka ammutaan, toteutetaan soveltamalla Unity kurssin (eng. tutorial): ”Let’s Try: Shooting with Raycast” -artikkelissa käytettyjä metodeita [19]. Säteen ampuminen perustuu suoran viivan laskemiseen aseen kärjestä näytön keskikohtaan halutun matkan päähän. Säde ilmestyy vain, jos ampuminen edessä olevaan objektiin on mahdollista.

Hyväksyttävät rajoitukset ja puutteet

Teemu Viisasen aikaisemman kokemuksen ansiosta oli tiedossa, että objektien törmäyttimet (eng. collider) saattavat aiheuttaa ongelmia piirto-ominaisuuden kanssa. Tavallisen verkkotörmäyttimen (eng. mesh collider) tulisi toimia normaalisti, mutta Unityn optimoidut törmäyttimet voi jättää huomiotta, mikäli ratkaisua ei löydy. Unityn optimoituja törmäyttimiä ovat laatikko- (eng. box collider), pallo- (eng. sphere collider) ja kapselitörmöyttimet (eng. capsule collider) sekä verkkotörmäyttimen optimoitu muoto konveksi verkkotörmäytin (eng. convex mesh collider).



Kuva 12. Kuvakaappaus Unityn pelitilassa, jossa 3DM-spray-työkalulla on ammuttu laatikkotörmäyttimellä varustettua objektia.

Törmäyttimien käytöstä tutkittaessa selvisi, että laatikkotörmäyttimellä varustettuun objektiin maalaaminen on mahdollista. Valintasäde ammuttiin keskelle objektin sivua useaan eri kohtaan, mutta pikselit värjäytyivät kuvan 12 mukaisesti vain objektin kulmista, joissa tekstuurin reunatkin sijaitsevat. Tästä voidaan päätellä, että valintasäde osuu aina kohtaan 0,0 tekstuurin x,y-koordinaatistossa. Tämä johtuu siitä, että natiivitörmäyttimillä ei ole koordinaatistoa.

Tekstuurin vaatimukset

Mikäli objektin materiaalin tekstuurin asetukset eivät ole oikein, ohjelma vaihtaa ne automaattisesti. Tekstuurin asetuksista on aktivoitava tekstuurin muokkausmahdollisuus ja tekstuurin formaatti. 3DM-Spray-työkalun on muutettava kaikki käsittelemänsä tekstuurit käyttäjän käyttöliittymästä valitsemaan muotoon. Formaattivaihtoehdot ovat RGBA 16-bittinen, RGBA 24-bittinen ja RGBA 32-bittinen. Tekstuurin formaatin muuttaminen tapahtuu pelitilassa ja sen muuttamisen edellytyksenä on myös tekstuurin asetuksien muuttaminen ”Override for Linux, PC and Mac Standalone” -tilaan, joka tarkoittaa koontiversion (eng. build) tekemistä Linux-, PC- ja Mac-tietokoneiden käyttöjärjestelmiä tukevaksi [20]. Tekstuurin on myös oltava materiaalissa ilman toistoa (eng. tiling), koska maalattu kuvio toistuu muuten aivan kuten alkuperäinen tekstuurikin.

Käyttöliittymän vaatimukset

3DM-Spray-työkalun käyttöliittymän tulisi koostua kolmesta osasta: maalaustyökalun asetuksista, teknisistä asetuksista ja tallentamiseen liittyvistä nappuloista. Maalaustyökalun asetukset ja tallennusnappulat olisi hyvä olla omassa yhteisessä ”Painting”-käyttöliittymässään.

Teknisien asetuksien käyttöliittymä on yhdessä 3DM-Spray-työkalun ”Painter” -komponentin käyttöliittymässä. Käyttöliittymästä tulisi löytyä tekstuurin oletusformaatin valinta, aseen äänen valinta ja työn aikana ilmenneet muut asetuksia vaativat asiat.

Maalaustyökalun asetukset

Maalaustyökalun asteukset sijoitetaan 3DM-Spray-käyttöliittymä ikkunan yläosaan. Maalaustyökalun asetukset koostuvat väripaletista, sivellinpaletista ja yleisistä asetuksista. Maalaustyökalun asetuksien valikon on oltava vähintään vastaavan tasoinen valikko kuin Unityn Terrainista löytyvässä maalaustyökalun käyttöliittymässä on sillä poikkeuksella, että valoasetukset eivät ole tarpeelliset. Yleisissä asetuksissa tulisi olla ainakin siveltimen koko, läpinäkyvyys ja kantomatka.

Tallennus- ja latausnappulat

Tallennus- ja latausnappulat sijoitetaan 3DM-Spray-käyttöliittymä ikkunan alaosaan. Tallennus- ja latausnappulat pitäisi olla suurempia kuin muut ja niiden tulisi vaihtaa väriä riippuen, ovatko ne käytettävissä vai ei. Niiden tulisi olla vihreitä aktiivisina ja harmaita epäaktiivisina. Niiden alapuolelle tulee myös teksti-ikkunat edellisistä muokatuista materiaaleista, sekä lista kaikista tallentamattomista maalatuista objekteista.

3.2 3DM-Spray-työkalun ominaisuudet

Liikkuminen

Hyödynnetään ensisijaisesti Unityn ensimmäisen persoonan kontrollerissa olevaa kävelyä ja juoksemista simuloivaa liikkumista. Mikäli resurssit riittävät toteutetaan myös vapaasti liikkuva lentämistä simuloiva tila.

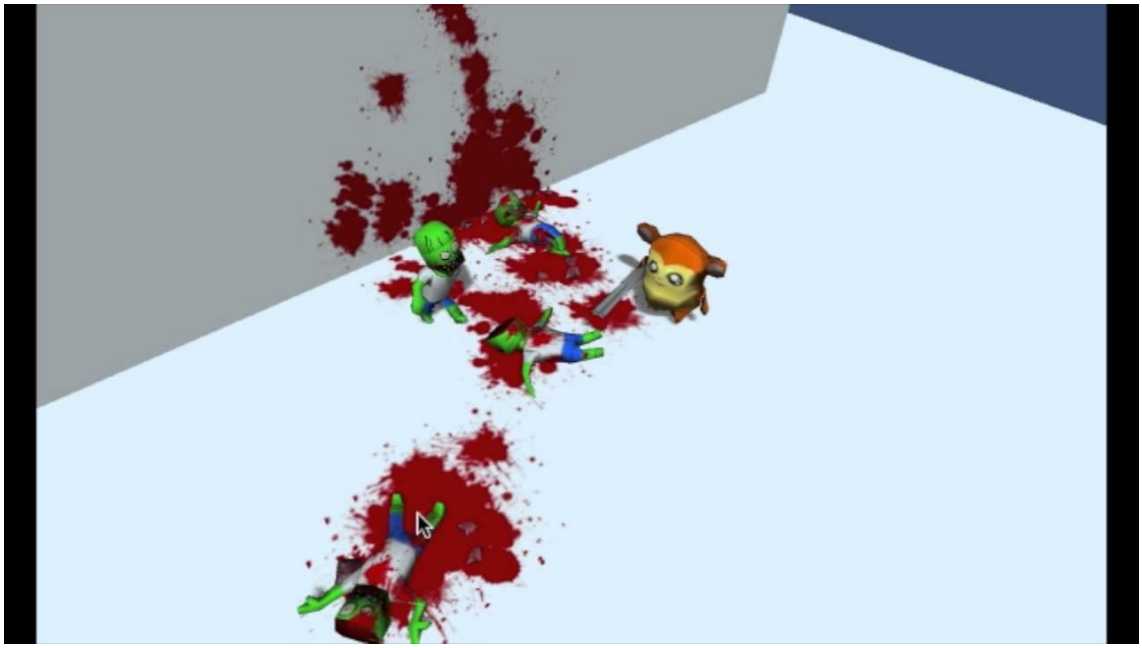
Maalaus siveltimillä

Työkaluun on toteutettava maalaus erilaisilla siveltimillä pelitilassa. Siveltimillä maalattaessa useampaan objektiin samanaikaisesti lopputuloksen on oltava saumatonta eri objektien välissä. Koska Unityssa on mahdollista käyttää samaa materiaalia usealle eri objektille ja vastaavasti samaa tekstuuria usealle eri materiaalille, on ohjelman maalattava objektin alkuperäisestä materiaalista ja tekstuurista tehdyille kopioille. Muutoin samoja materiaaleja käyttävät objektit tai samoja tekstuureja käyttävät materiaalit maalautuvat

myös tahattomasti. Uudet materiaalit ja tekstuurit tallennetaan niille luotuihin uusiin kansioihin Resources-kansion sisään.

Maalaus Unityn fysiikkamoottorilla

Mikäli aika riittää, toteutetaan Unityn fysiikkamoottoriin pohjautuva maalaustila, jossa aseen kärkeen luodaan partikkeleita, jotka ammutaan käyttöliittymässä määritettävällä voimalla. Näihin partikkeleihin voidaan kohdistaa satunnaisia ylimääräisiä voimia kuten tuulta ja maan vetovoimaa, jolloin niiden luomista kuvioista saadaan sattumanvaraisempia. Partikkeleiden osuessa objekteihin ne maalaavat osumakohtaansa siveltimen määräämän kuvion.



Kuva 13. Kuvakaappaus Robert Attardin youtube -videosta: Unity 3d blood test [18].

Toimintaperiaate olisi hyvin samantyyppinen kuin Robert Attardin youtube-videossaan esittelemässä veren roiskumista simuloivassa tekstuurin maalauksessa [21]. Kuva 13 on kuvakaappaus videolta, jossa näkyy, kuinka vihollista ammuttaessa, seinään tai lattiaan muodostuu roiskejälkiä.

Tekstuuriin tehtyjen muutosten tallennus ja lataus

Tallennus tapahtuu tallennusnappulasta pelitilassa, jolloin ohjelma luo uuden tekstitiedoston tietokoneen temporary-kansioon. Käyttäjän palatessa editointitilaan latausnappula aktivoituu ja sen avulla ohjelma lataa tehdyt muutokset tekstitiedostosta ja poistaa tekstitiedoston.

Tämä tallennusratkaisun ansiosta 3DM-Spray-työkalu oli ainutlaatuinen, ja tämä oli paras kilpailuvaltti muihin maalaustyökaluliitännäisiin nähden. Kilpaileva Play Time Painter -liitännäinen sai kuitenkin päivityksen kesken 3DM-Spray-työkalun kehitystä, jossa siihen esiteltiin vastaava ominaisuus, joka mahdollistaa pelitilassa tallentamisen.

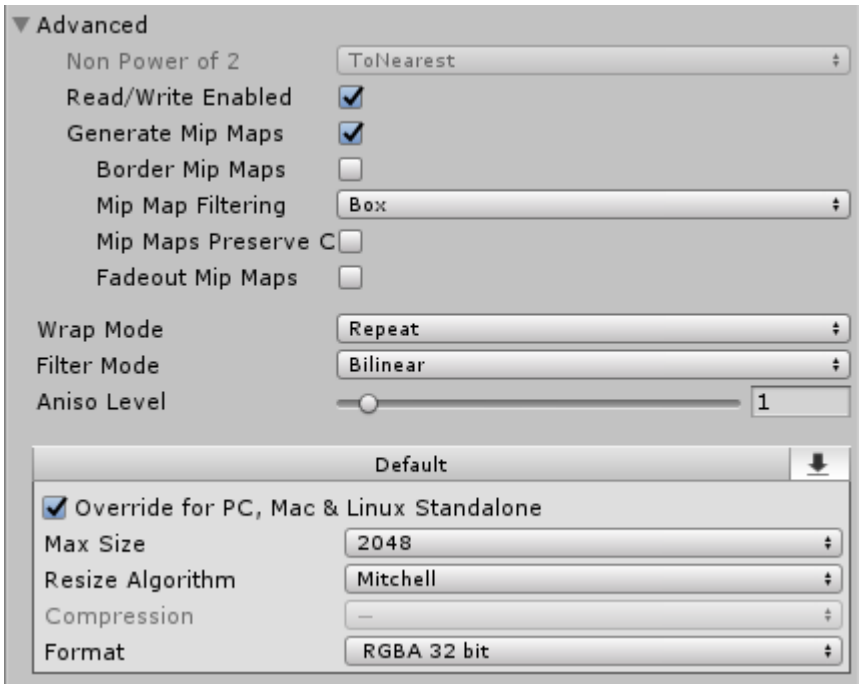
4 3DM-Spray-maalaustyökalun toteutus

4.1 Työkalun kehityksessä esiintyneet ongelmat ja niiden ratkaisut

Tallennusratkaisu ja ensimmäinen piirto-ominaisuus

Ennakkoselvityksessä kävi ilmi, että 3DM-Spray-työkalun haastavin työvaihe oli tallennuksen tekeminen pelitilassa, joten siitä aloittaminen oli järkevä vaihtoehto. Tallennuksen testaamista varten oli kuitenkin pakko tehdä yksinkertainen runko maalaustyökalulle, joten ensimmäiseen versioon tehtiin piirto-ominaisuus, objektin valintasäde ja ensimmäinen 3D-malli maaliruiskuksi. Pelitilaan mennessä Unity luo varmuuskopion projektista ja palauttaa sen pelitilasta poistuttaessa, joten tietokantaan tehtyjä muutoksia lukuun ottamatta mitkään tehdyt muutokset eivät tallennu. Tallennusominaisuus tehtiin aluksi John Pile Jr:n esimerkin mukaisesti vain objektien sijanneille, jolla varmistettiin tallennusratkaisun toteutuksen mahdollistaminen. [22.]

Ensimmäiset ongelmat tekstuuriin piirtämisessä



Kuva 14. Formaatin muuttamisen mahdollistamiseksi tekstuurin asetuksissa on oltava valittuina ruudut ”Read/Write Enabled” ja oltava ”Override for PC, Mac and Linux Standalone”.

Ensimmäinen ongelma ilmeni piirto ominaisuudessa, vaikka tekstuuri oli manuaalisesti vaihdettu muokkauksen sallivaan tilaan. Ohjelma antoi virheilmoituksen tekstuurin formaatista, jonka selvittämisen jälkeen todettiin, että tekstuurin on oltava kuvan 14 mukaisesti ”Override for PC, Mac and Linux Standalone”-tilassa ja formaatin on oltava RGBA 32 bittiä. Tässä vaiheessa ne vaihdettiin manuaalisesti, jotta tallentamisen testaaminen mahdollistettiin.

Tekstuurin kopioinnista isoin ongelma

Tekstuurin kopioinnissa esiintyi useita ongelmia, joista vaativimmat olivat tekstuurin asetuksista muokattavuuden aktivointi ja formaatin muuttaminen. Tekstuurin asetuksista muokattavuuden aktivointiin löytyi useita eri tapoja, joista vain yksi onnistuu pelitilassa. Ennen oikean ratkaisun löytymistä kokeiltiin Programmer nimimerkin ratkaisua, jossa annetaan tekstuurin luontivaiheessa sille uusi formaatti. [23.] Tämä ratkaisu ei kuitenkaan

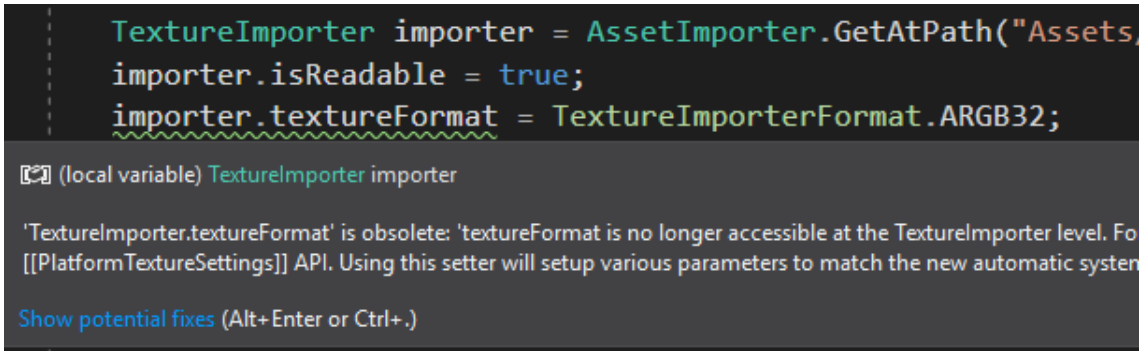
toimi, koska ”Override for PC, Mac and Linux Standalone” -tila on edelleen epäaktiivinen. Programmer-nimimerkin koodissa oleva ratkaisu:

```
//Create new empty Texture
Texture2D newTex = new Texture2D(2, 2, newFormat, false);
//Copy old texture pixels into new one
newTex.SetPixels(oldTexture.GetPixels());
//Apply
newTex.Apply()
```

Ensimmäinen toimiva ratkaisu löytyi Unity-foorumeilta, jonne MirrorIrorriM-nimimerkki oli jakanut ratkaisunsa, jossa käytettiin apuna textureimporter-luokkaa [24]. Ratkaisussa luodaan ensin importer-niminen textureimporter-luokka, jolle annetaan tekstuurin tiedot. TextureImporter-luokan avulla tekstuurin asetuksia voi muokata, joista MirrorIrorriM-nimimerkin esimerkissä muutettiin tekstuurin tyyppiä, formaattia, muokattavuutta, filtteritilaa ja kokoa.

```
TextureImporter importer = assetImporter as TextureImporter;
importer.textureType  = TextureImporterType.Advanced;
importer.textureFormat = TextureImporterFormat.AutomaticTruecolor;
importer.isReadable = true;
importer.filterMode = FilterMode.Point;
importer.npotScale = TextureImporterNPOTScale.None;
```

Tämä ratkaisu toimi vanhemmissa Unityn versioissa, mutta Unity on muuttanut 2017 vuoden versioon TextureImporter-luokan toimintaa, kuten kuvasta 15 näkyy, eikä sen sisältä löydy enää textureFormat-toimintoa. Luokasta löytyy kuitenkin vielä isReadable-toiminto, jonka avulla kopioidun tekstuurinmuokattavuus asetus saadaan aktivoitua.



Kuva 15. importer.textureFormat-toiminto on Unityn 2017 versiossa vanhentunut, mutta isReadable toimii vielä.

Seuraavana oli tarkoitus kiertää ongelma ja käyttää Unityn ehdottamaa PlatformTexture-Settings-luokkaa, mutta siinä aiheutui samankaltainen virheilmoitus. [25.] Tämän ratkaisun etuna olisi ollut myös alustan vaihtaminen, eli ”Override for PC, Mac and Linux Standalone” -tilan aktivointi.

```
    string platform = "Standalone";
    int maxTextureSize = 1024;
    TextureImporterFormat format = TextureImporterFormat.RGBA32;
    bool allowsAlphaSplit = false;
    int compressionQuality = 0;
    importer.SetPlatformTextureSettings(platform, maxTextureSize, format,
compressionQuality, allowsAlphaSplit);
```

Esimerkkikoodi 1. Ote laaditusta koodista.

Unityn dokumentaation mukaan uusi toimintatapa on käyttää TextureImporterPlatform-Settings-luokkaa [26]. Unityn koodausesimerkkejä ja tutoriaaleja sisältävältä SarperSoher.com-nettisivulta löytyi dokumentaatio kustomoidusta välittäjästä ”Conditional Asset Importer”, jonka koodin seassa oli esimerkki TextureImporterPlatformSettings-luokan käytöstä [27]. Tämä ratkaisu toimi ja sen myötä tekstuurien kopiointi saatiin toimimaan.

```
        // Standalone
        // Construct the class that contains our importer settings, we'll re-use this class
per platform by changing any fields we need changed
        // Let's start with standalone build target, "name" field determines the target
platform
        var tips = new TextureImporterPlatformSettings() {
                                                    allowsAlphaSplitting = true,
                                                    compressionQuality   = 100,
                                                    crunchedCompression  = true,
                                                    format               =
importer.DoesSourceTextureHaveAlpha() ? TextureImporterFormat.DXT5Crunched :
TextureImporterFormat.DXT1Crunched,
                                                    maxTextureSize       = 4096,
                                                    name                 =
"Standalone",
                                                    overridden           = true,
                                                    textureCompression   =
TextureImporterCompression.CompressedHQ
                                                };
        importer.SetPlatformTextureSettings(tips);
```

Kuva 16. Kuvakaappaus SarperSoher-nettisivulta löytyneen pitkän esimerkki koodin seasta löytyi toimiva esimerkki [27].

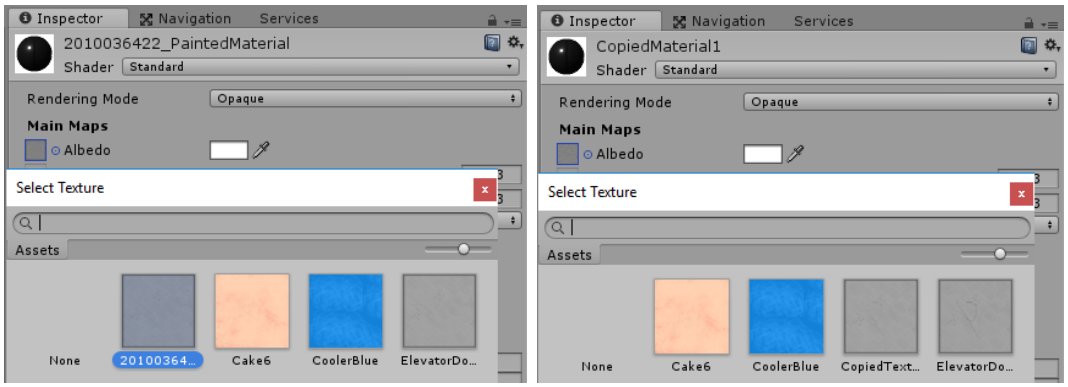
Kuvassa 16 näkyy kuvakaappaus SarperSoher-nettisivuilta, jossa muokataan tekstuurin asetuksista sen alpha-arvojen muokkausta, laatua, formaattia, kokoa ja ”overriden”-asetusta.

Tekstuurin tallentaminen materiaalille ja siinä esiintyneet ongelmat

```
rend.sharedMaterial.mainTexture = copiedTex;
```

Esimerkkikoodi 2. Koodirivi, jossa tekstuuri kopioidaan objektin materiaalin tekstuuriksi.

Tekstuurin tallentaminen materiaalille on yksinkertainen toiminto, jossa valitaan objektin tekstuuri ja korvataan se kopioidulla tekstuurilla. Tämä toiminto toimii editointitilassa moitteettomasti, mutta pelitilassa se jää väliaikaiseen muistiin. Pelitilassa toiminto ei toimi, koska materiaalin käyttöikkunasta löytyvästä Albedo-muuttujasta löytyy materiaalin tekstuuri, jonka pitäisi olla tekstuuri-ikkunassa korostettuna projektin muiden tekstuurien joukossa. Kuten kuvasta 17 voi huomata korostusta ei kuitenkaan näy, koska objektin materiaalille on annettu vain copiedTex-tekstuurin data itse tekstuurin sijaan.



Kuva 17. Materiaalin käyttöliittymästä löytyy Albedo-muuttuja, joka avaa tekstuuri-ikkunan.

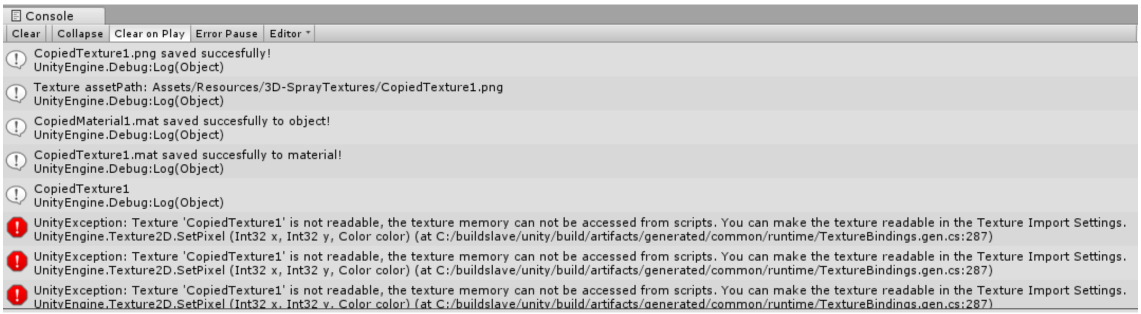
Ratkaisuna käytettiin samaa toimintatapaa kuin materiaalin kopioinnissa. Tekstuuri haetaan Resources-kansiosta ja ladataan sieltä valitun objektin materiaalille.

```
rend.sharedMaterial.mainTexture = (Texture2D)AssetDatabase.LoadAssetAtPath(as-
setPathTex, typeof(Texture2D));
```

Esimerkkikoodi 3. Koodirivi, jossa tekstuuri asetetaan objektin materiaaliin tietokoneen tietokannasta.

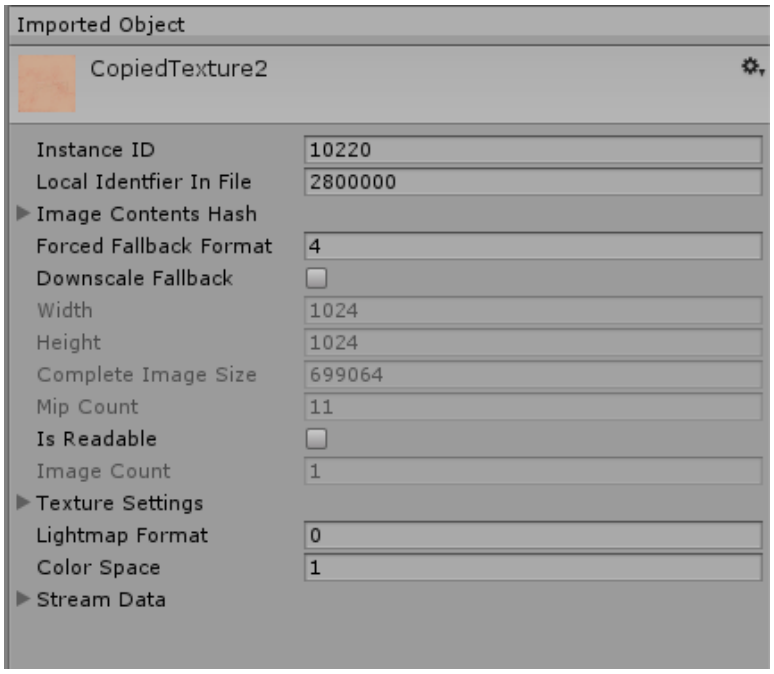
Tämä ratkaisu johti kuvan 18 virheilmoitukseen aina, kun objektiin yritettiin piirtää. Virheilmoitus paljasti puutteen tekstuurin asetuksista tai piirto-ominaisuudesta. Tämän virheilmoituksen pitäisi olla mahdoton, sillä kyseinen kopioitu tekstuuri on ainakin

käyttöliittymän mukaan jatkuvasti muokattavassa muodossa. Mielenkiintoiseksi virheilmoituksen teki se, että se ei esiintynyt, mikäli pelitilasta poistuttiin ja palattiin piirtämään samaa objektia, mikä viittaisi virheilmoituksen liittyvän materiaalin ja tekstuurin luontivaiheeseen.



Kuva 18. Kuvakaappaus Unityn konsolista, jossa näkyy virheilmoitus.

Virheilmoitusta tutkittaessa kävi ilmi, että tekstuureilla on muokkauksen aktivoimisen lisäksi myös luettavuuden (eng. Is Readable) aktivointi, joka löytyy kuvan 19 mukaiselta tekstuurin käyttöliittymän vianmääritysvälilehdeltä. Luettavuusmuuttuja aktivoituu, mikäli pelitilasta poistutaan ja palataan piirtämään kyseiseen objektiin, joten kyseiseen ohjelmointivirheeseen löytyy ratkaisu jo valmiista koodista ja kyse on todennäköisesti koodin järjestyksestä.

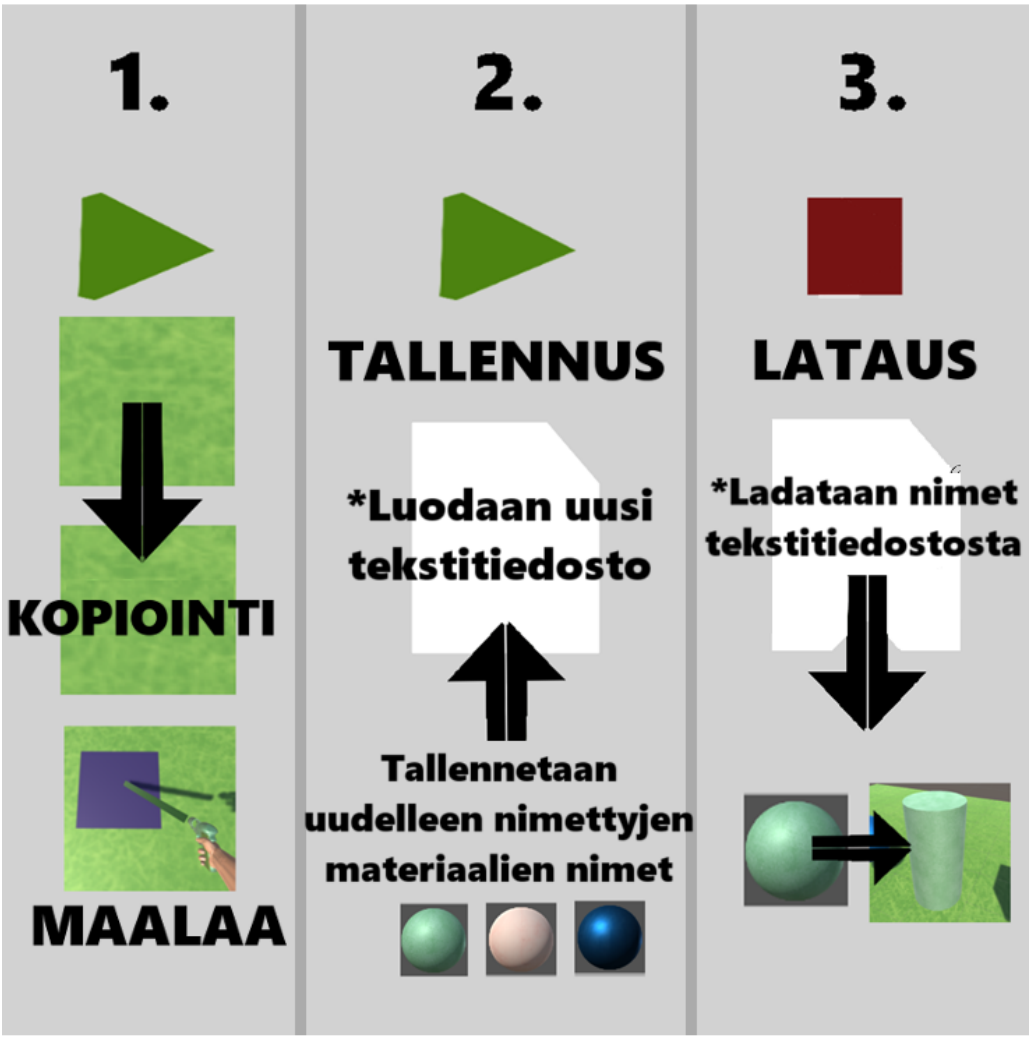


Kuva 19. Kuvakaappaus tekstuurin käyttöliittymän vianmääritysvälilehdestä, josta löytyy toinen muokattavuuteen vaikuttava muuttuja.

Ratkaisu ongelmaan löytyi Unityn keskustelufoorumeilta, jossa nimimerkki Carlos Ulloa vastasi vastaavaan ongelmaan [28]. Hän tarjosi ratkaisua, jossa tietokannalle annetaan tekstuurin välittäjän asetusten muuttamisen jälkeen pakkopäivitys-käsky (eng. ForceUpdate), mikä pakottaa tietokannan päivityksen. Tämän ratkaisun myötä tekstuurien kopiointi ja materiaalien kopioiminen toimii vaatimusten mukaan.

Tallennuksen muuttaminen oikealle toteutukselle

Objektien sijantien tallennuksen muuttaminen objektien materiaalien tallennukseksi ei sujunut täysin ongelmitta, koska oikean materiaalin asettaminen oikeaan objektiin ei onnistunut. Ensimmäinen suunnitelma oli tallentaa tekstitiedoston objektien sijaintitietojen sijaan objektien nimet ja nimetä objektien materiaalit samannimisiksi. Tällöin latausnappulaa painettaessa ohjelma kävisi listan objektit läpi ja asettaisi niille samannimiset materiaalit. Ratkaisu toimisi, mikäli Unity-objektit säilyttäisivät uudet nimensä pelitilasta poistuttaessa tai mikäli jokainen objekti olisi valmiiksi nimetty eri tavalla. Varsinkin suurissa projekteissa samannimisiä objekteja esiintyy, joten tällaista olettamusta ei voida liitännäiseen jättää.



Kuva 20. Tallennusmekanismin kolme askelta, joista kaksi ensimmäistä tapahtuu pelitilassa.

Kuvan 20 suunnitelman mukaan 3DM-Spary-työkalun toiminnan voi jakaa kolmeen osaan. Ensimmäinen osa tapahtuu pelitilassa ja siinä materiaalit, sekä tekstuurit kopioidaan ja asetetaan objekteille, jonka jälkeen tapahtuu objekteihin maalaaminen. Kun halutut muutokset halutaan tallentaa, siirrytään kohtaan kaksi, joka toimii myös pelitilassa. Osassa kaksi käyttäjä painaa tallenna-nappulaa, jolloin ohjelma nimeää muokatut materiaalit uudestaan siten, että niiden uusiin nimiin tulee niiden objektin uniikki sarjanumero. Uudelleen nimettyjen materiaalien nimet tallennetaan tekstitiedostoon Unityn ulkopuolelle. Tallennuksen jälkeen käyttäjän on siirryttävä takaisin editointitilaan, jolloin tehdyt muutokset katoavat. Osan kolme toteutus tapahtuu käyttäjän painaessa lataus nappulaa. Lataus toiminto hakee tekstitiedostosta materiaalien nimet ja pilkkoo niistä objektin sarjanumerot, joiden avulla se etsii oikeat objektit. Tämän jälkeen ohjelma tallentaa

kohdassa yksi kopioidut uudelleen nimetyt materiaalit oikeille objekteilleen, jolloin maalatut muutokset tulevat jälleen näkyviin.

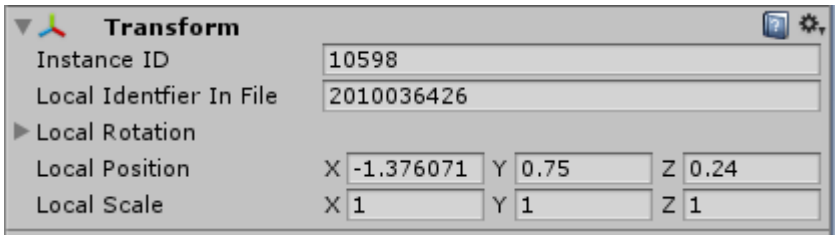
Objektien sijantien tallennuksen muuttaminen objektien materiaalien tallennukseksi ei sujunut täysin ongelmitta, koska ongelmia tuotti oikean materiaalin asettaminen oikeaan objektiin. Ensimmäinen suunnitelma oli tallentaa tekstitiedoston objektien sijainti tietojen sijaan objektien nimet ja nimetä objektien materiaalit saman nimisiksi. Tällöin latausnappulaa painettaessa ohjelma kävisi listan objektit läpi ja asettaisi niille samannimiset materiaalit. Tällainen ratkaisu toimisi, mikäli Unity-objektit säilyttäisivät uudet nimensä pelitilasta poistuttaessa tai mikäli jokainen objekti olisi valmiiksi nimetty eri tavalla. Varsinkin suurissa projekteissa samannimisiä objekteja esiintyy, joten tällaista olettamusta ei voida liitännäiseen jättää.

Seuraavana tehtävänä on etsiä jokaisesta objektista uniikki arvo ja käyttää sitä vastaavasti siten, että nimetään materiaali sen mukaan. Unity antaa jokaiselle objektille InstanceID-arvon, joka on jokaiselle objektille uniikki [29]. Tämä ratkaisu toimi siihen asti, kunnes kävi ilmi, että jokaisen pelitilan käynnistyskerran aikana Unity generoi uudet InstanceID-arvot jokaiselle objektille, joten se ei ole sama tallenna- ja lataus-nappuloita painaessa.

```
int objectId = paintedObjects[j].GetInstanceID();
String objectIdString = objectId.ToString();

//Rename copiedmaterials
Renderer paintedRend = paintedObjects[j].GetComponent<Renderer>();
String[] oldName = paintedRend.sharedMaterial.name.Split(" "[0]);
String tempName = oldName[0] +".mat";
String assetPath = "Assets/Resources/3D-SprayMaterials/"+tempName;
String newName = objectIdString + "_PaintedMaterial";

AssetDatabase.RenameAsset(assetPath, newName);
AssetDatabase.SaveAssets();
AssetDatabase.Refresh();

//GuID
String ObjectAssetPath = AssetDatabase.GetAssetPath(objectId);
Debug.Log(AssetDatabase.GetAssetPath(objectId));
string t = AssetDatabase.AssetPathToGUID(ObjectAssetPath);
Debug.Log(paintedObjects[j] + "GUID" + t);
```

Esimerkkikoodi 4. Ote laaditusta koodista, jossa haetaan InstanceID-arvo ja GUID-tiedostopolku objektille.

InstanceID-arvon lisäksi jokaisella objektilla on yksilöllinen GUID-tiedostopolku [30]. Tämäkään ei toiminut ratkaisuna, koska GUID-tiedostopolku on olemassa vain niillä objekteilla, jotka löytyvät tietokannasta. GUID-tiedostopolun avulla tehty ratkaisu toimisi, mikäli projektin kaikki objektit olisi tallennettu erillisiin paketteihin (eng. prefab), jolloin tietokannasta ensimmäisen ratkaisun mukaisesti nimien muuttaminen olisi mahdollista.

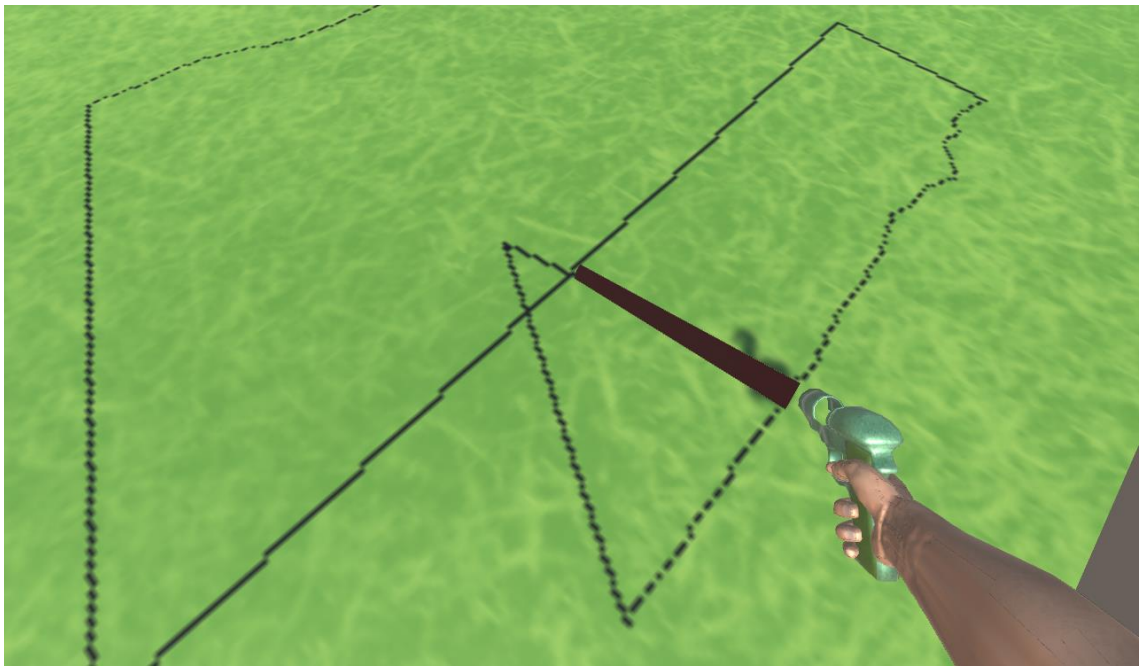


Kuva 21. Unityn peliobjektin debug-tilasta löytyy ”Local Identfier In file” -sarjanumero.

Unityn objekteista löytyy yksi ainutlaatuinen muuttumaton muuttuja nimeltä ”Local Identfier In File”, joka on vaikean tavoitettavuuden lisäksi, kuten kuvasta 21 näkee, väärin kirjoitettu. TheLackey3326-nimimerkin kirjoittaman toteutuksen avulla objektille saadaan yksilöllinen numerosarja, jonka avulla tallennus voidaan toteuttaa [31]. TheLackey3326-nimimerkin koodissa luodaan uusi sarjallistettu (eng. serialize) luokka, jolle annetaan parametriksi objekti, jonka yksilöllinen numerosarja halutaan.

Piirto-ominaisuus ja siveltimien toteutus

Ensimmäisen version piirto-ominaisuudeksi tehtiin vain yksinkertainen pikselinvärjäämisoperaatio valittuun tekstuuriin, kun hiiren vasenta näppäintä painetaan, joka sai aikaan kuvan 22 mukaista piirtojälkeä. Toteutus tehtiin Barracuda Disasterin toisen projektin koodin ja Brackeys-nimimerkin Youtube-tutoriaalin ”Shooting with Raycasts – Unity Tutorial” avulla [32]. Tämä testiversio toimi kahdessa funktiossa, jossa ensimmäisessä määritettiin alku- ja loppupisteet. Toisessa funktiossa ohjelma piirtää suoran viivan näiden välille, jolla saadaan piirrettyä yhtäjaksoista viivaa ja maalattua valintasäteen matkalta myös ne pikselit, joita ohjelma ei ehdi piirtää. Tämä ratkaisu säilytetään yhtenä 3DM-Spray-työkalun ominaisuutena.



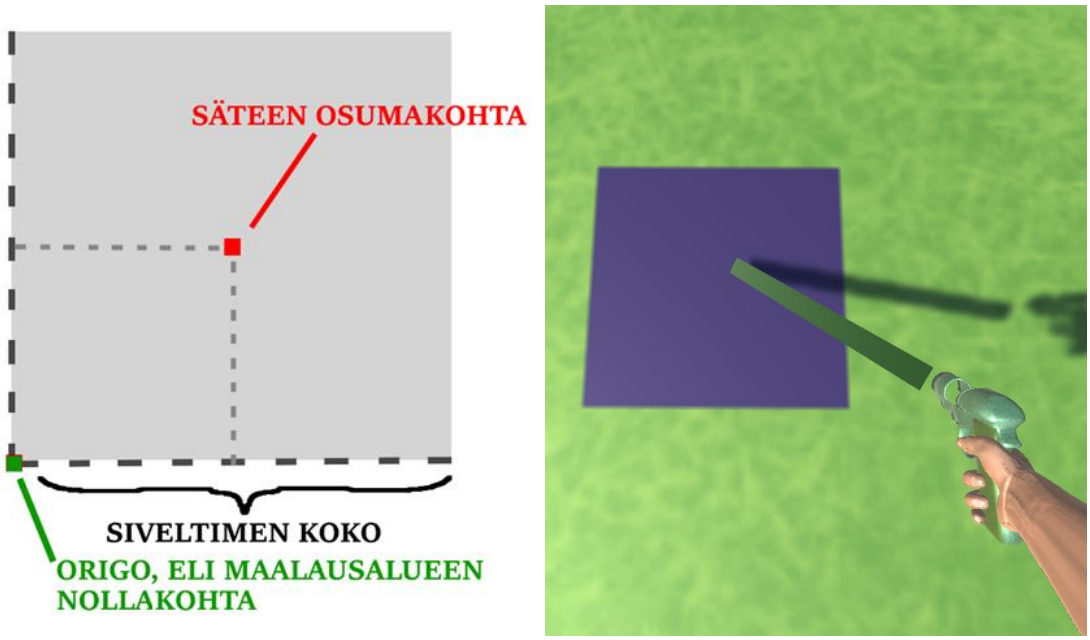
Kuva 22. Piirtämisen ensimmäinen versio piirtää kahden eri pisteen välille viivan.

Maalausominaisuuden toteutus aloitettiin värin ja siveltimen koon määrityksellä. Väripaletin liittäminen oli yllättävän yksinkertainen operaatio, koska Unity-dokumentaatio tarjosi yhden koodirivin ratkaisun [33]. Unityn kirjastoista löytyy väripaletti, jonka saa käyttöliittymään määrittämällä yleisen muuttujan ”Color”.

```
public Color chooseColor = Color.black;
```

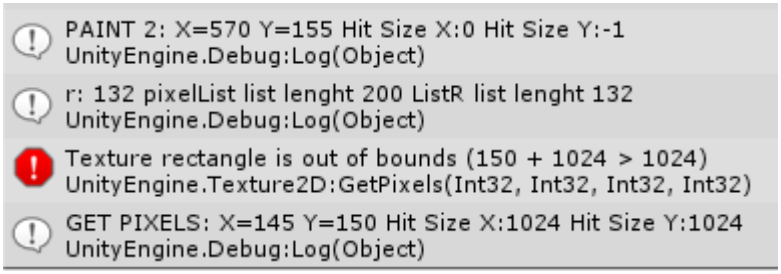
Esimerkkikoodi 5. Koodirivi, jolla luodaan käyttöliittymään värinvalintamuuttuja.

Siveltimen koon muuttaminen oli monimutkaisempi prosessi, jossa tehtiin Paint-funktio. Paint-funktiossa ohjelma laskee ammuntasäteen osumapisteen ympäriltä käyttäjän määrittämän siveltimen koon (eng. BrushSize) verran pikseleitä ja värjää ne valitulla värillä. Kuvan 23 mukaisesti ohjelma vähentää osumapisteen koordinaateista siveltimen koon, joka on jaettu kahdella. Sen jälkeen lasketaan x- ja y-koordinaatit kahdessa sisäkkäisessä niin kauan kuin-silmukassa, joissa jokaisen uuden koordinaatin kohdalla pikseli värjätään.



Kuva 23. Siveltimen koon mukaan määräytyvä alue värjäytyy valitulla värillä.

Pikselien värjäämisessä käytetään ”aseta pikseli” -toimintoa (eng. setPixel), joka on vie useasti käytettynä paljon suoritustehoa. Unityssa on kuitenkin toiminto, jolla pystyy värjäämään useamman pikselin kerralla käyttämällä ”aseta pikselit” -toimintoa (eng. setPixels) [34]. ”Aseta pikselit” -toiminnossa värit, joilla halutaan maalata, tallennetaan värilistaan, joka annetaan ”Aseta pikselit” -toiminnolle parametrina. Unityn dokumentaatiosta löytyneen esimerkin avulla 1024x1024 pikseliä kokoisen tekstuurin kaikki pikselit saatiin värjättyä ilman viivettä, toisin kuin ”aseta pikseli” -toimintossa ilmeni viive jo 100x100 pikseliä kokoisen alueen maalaamisessa. ”Aseta pikselit” -toiminto ei kuitenkaan toimi halutulla tavalla, koska oikeiden koordinaattien asettaminen värilistaan ja sen lukeminen ei onnistu.



Kuva 24. Kuvakaappaus Unityn vikailmoitus lokista, josta esiintyi virheilmoitus: ”Texture rectangle is out of bounds”

”Aseta pikselit” -toiminnon avulla pikselien värjäämisen onnistuminen jäi kiinni ”hae pikselit” -toiminnosta (eng. getPixels), jossa värilistaan lisättiin valittu pikseli x,y-koordinaatistosta. Tämä rivi antoi kuitenkin kuvan 24 mukaisen virheilmoituksen: ”Texture rectangle is out of bounds”, jonka syyn selvittämiseen löytyi Unityn keskustelufoorumeilta apua [35]. Nimimerkki PaulUsul perusteli virheilmoituksen johtuvan tallennettavan listan koon olevan suurempi kuin siihen tallennettavien värien määrä, mikä johtaisi kyseiseen virheilmoitukseen.

```
cols = tex.GetPixels(x, y, width, height);
```

Esimerkkikoodi 6. Koodirivi, joka aiheutti kuvan 24 virheilmoituksen.

Useista listan koon korjausyrityksistä huolimatta virheilmoituksesta ei päästy eroon, joten ratkaisua piti muuttaa siten ettei ”hae pikselit” -toimintoa tarvinnut käyttää. Tämä onnistui käyttämällä pikselien tallennuksessa järjestettyä listaa (eng. array list) värilistan (eng. colorlist) sijaan.

4.2 3DM-Spray-työkalun toteutus

Update-funktio

Update-funktiolla tarkoitetaan C#-ohjelmointikielessä käytettyä funktiota, jota kutsutaan jokaisessa ruudunpäivityksessä [36]. Update-funktiota käytetään maalaustyökalussa tarkkailemaan objektin valintasäteen tilaa ja hiiren klikkauksia. Funktion tarkistaa ensimmäisenä, onko maalaustyökalun käyttöliittymässä määrittelyn kantomatkan päässä maalattavaksi kelpaavia objekteja, joiden kohdalla se renderöi objektin valintasäteen. Objektin ollessa kantoetäisyydellä piirtämisen aloittaminen on mahdollista hiiren klikkauksella, joka aktivoi materiaalin ja tekstuurin kopiointi- sekä piirtofunktion.

```
// Update is called once per frame
void Update()
{
      RaycastHit hit;
      laserLine.SetPosition(0, gunEnd.position);

      if (Physics.Raycast(fpsCam.ScreenPointToRay(Input.mousePosition).origin,
fpsCam.ScreenPointToRay(Input.mousePosition).direction, out hit, 4, Phys-
ics.DefaultRaycastLayers))
      {
            laserLine.enabled = true;
```

```
            laserLine.SetPosition(1, hit.point);

            if (Input.GetButton("Fire1"))
```

Esimerkkikoodi 7. Ote laaditusta koodista, jossa on Update-funktio.

Ensimmäisenä ampumisfunktio käynnistää maalausäänen ja tallentaa kohdeobjektin muuttujaan ”targetItem”, jonka avulla objektin renderöijä tallennetaan rend-muuttujaan. Seuraavaksi tarkistetaan ehto-lauseen avulla, onko objektin materiaalista tehty jo kopiota, eli onko 3DM-spray-työkalulla maalattu jo kyseistä objektia.

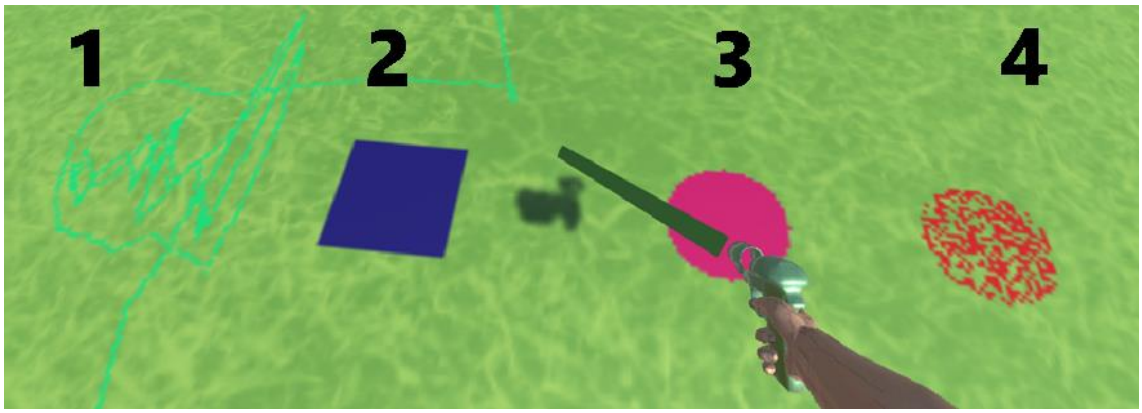
```
gunAudio.Play();
targetItem = hit.collider.gameObject;
// Copy material and texture
Renderer rend = targetItem.GetComponent<Renderer>();
if (!rend.sharedMaterial.name.Contains("CopiedMaterial"))
      {
      i++;
      StartCoroutine(CopyTexture(rend));
      AssetDatabase.Refresh();
      paintedObjects.Add(targetItem);
      j++;
      }
```

Esimerkkikoodi 8. Ote laaditusta koodista, jossa on Update-funktion ehto-lause.

Jos objektia on maalattu aikaisemmin, funktio siirtyy suoraan piirtämisoperaatioon. Muuten se suorittaa tekstuurin- ja materiaalinkopiointifunktiot. Lisäksi se lisää muuttujiin i ja j yhden numeron lisää. Muuttujaa i käytetään materiaalin kopioinnissa ja muuttuja j pitää lukua tallennukseen liittyvästä listasta ”paintedObjects”, johon objekti myös lisätään.

Siveltimien toteutus ja pikselien maalaus tekstuureihin

3DM-Spray-työkalun ensimmäiseen versioon toteutettiin neljä eri sivellintä, joiden jälki näkyy kuvassa 25. Sivellin yksi toimii piirtotyökalun avulla ja sivellin kaksi maalaa siveltimen koon muukaan määritellyn neliön. Sivellin kolme maalaa kokonaisen ympyrän ja sivellin neljä maalaa ympyrän sisältä sattumanvarasiesti pikseleitä. Siveltimen neljä sattumanvaraisesti maalattujen pikselien määrä riippuu käyttöliittymän Spread-muuttujasta.



Kuva 25. Kuvakaappaus tekstuurista, johon on maalattu kaikilla neljällä siveltimellä. Kuvioiden yläpuolelle on lisätty siveltimien numerot.

Pikselien maalaus tapahtuu Paint-funktiossa, jonka parametreiksi on annettu maalattava tekstuuri ja valintasäteen osumakohdan x- ja y-koordinaatit. Paint-funktio käyttää yleisiä muuttujia: ”Brush”, ”hitSize” ja ”chooseColor”, jotka voidaan määrittää käyttöliittymässä. Paint-funktiolla on myös koordinaattien laskemista varten seitsemän kokonaislukumuuttujaa, sekä yksi vektorilista ja yksi vektori muuttuja. Kokonaislukumuuttuja ”dxy” saa arvokseen käyttöliittymässä määritellyn siveltimen koon, jonka perusteella neliön ja ympyrän säde lasketaan. HitX- ja HitY-muuttujiin siirretään x- ja y-muuttujien arvot, joista miinustetaan ympyrän säteen verran.

```
void Paint(Texture2D tex, int x, int y) {
int dxy = (int)(hitSize);
int hitX = x;
int hitY = y;
int countX = dxy;
int countY = dxy;
int Origo = y - (dxy / 2);
int r = 0;
Vector2[] pixelList = new Vector2[dxy * dxy*2];
Vector2 pixel = new Vector2(hitX, hitY);
```

Esimerkkikoodi 9. Ote laaditusta koodista, jossa alustetaan siveltimen kuvioiden maalaamista.

Siveltimet 2, 3 ja 4 kutsuvat Paint-funktiota ja sen rakenne erottaa nämä kaksi sivellintä ehtolauseilla, joissa tarkastetaan kumpi sivellin on käytössä. Ehto-lauseiden sisällä on kaksi sisäkkäistä niin kauan kuin -silmukkaa, joiden ehdon täyttymisenä countX- tai countY-muuttujien on oltava suurempia tai yhtä suuria kuin nolla. CountX- tai countY-muuttujille on asetettu arvoiksi siveltimen koko ja niiden arvot laskevat yhdellä jokaisessa niin kauan kuin -silmukan kierroksessa. HitX- ja HitY-muuttujin lasketaan joka

kierroksella yksi kokonaisluku lisää siten, että jokainen pikseli siveltimen kokoiselta alueelta on laskettu.

```
if (brush == 2)
      {
      while (countX >= 0)
            {
            HitY = OrigoY;
            countY = dxy;
            while (countY > 0)
                  {
                  pixel.x = hitX;
                  pixel.y = hitY;
                  pixelList[r] = pixel;
                  r++;
                  countY--;
                  hitY++;
                  }
            pixel.x = hitX;
            pixel.y = hitY;
            pixelList[r] = pixel;
            r++;
            hitX++;
            countX--;
            }
      }
```

Esimerkkikoodi 10. Ote laaditusta koodista, jossa funktio laskee maalattavan neliön.

Jokainen laskettu pikseli lisätään vektorimuodossa pixelList-listaan ja jokaisen lisäyksen yhteydessä kokonaislukumuuttujaan r lisätään kokonaisluku, jotta listaan lisättyjen vektorien määrä on helposti saatavilla. Siveltimet 3 ja 4 eroavat yllä esitetystä siten, että niillä on ehto-lause ennen kuin pixelList-listaan lisätään vektori. Tässä jos-lauseessa on ehtona CountCircle-funktio, jonka on palautettava kokonaisluku 1, jotta vektorin lisääminen listaan tapahtuisi.

```
if (CountCircle(pixel, x, y, dxy) == 1)
      {
            pixelList[r] = pixel;
            r++;
      }
```

Esimerkkikoodi 11. Ote laaditusta koodista, jossa on ehto-lauseessa olevan CountCircle-funktion avulla lasketaan maalattava ympyrä.

Molempien ehto-lauseiden päätteeksi pixelList-listasta löytyvät kaikki maalattavat pikselit vektorimuodossa. Ne maalataan niin kauan kun-silmukassa tekstuuriin käyttämällä aseta pikseli-toimintoa (eng. setPixel) ja valvomalla listan kokoa muuttujalla r.

```
      int s = 0;
      while (s < r)
            {
            tex.SetPixel((int)pixelList[s].x, (int)pixelList[s].y, chooseColor);
            s++;
            }
      tex.Apply();
}
```

Esimerkkikoodi 12. Ote laaditusta koodista, jossa listaan asetetut pikselit asetetaan tekstuuriin valitulla värillä.

CountCircle-funktio laskee siveltimien kolme ja neljä pyöreän muodon sekä sattumanvaraisuuden. Funktio laskee ensin uudet arvot x2 ja y2, jotka ovat valitun vektorin pisteiden kohdat sillä ympyrällä, jonka origo on maalattavan kohdealueen nollakohdassa. Mikäli kyseessä on sivellin kolme, ohjelma laskee vain pisteen etäisyyden origosta ja palauttaa sen perusteella kokonaislukuarvon yksi tai kaksi. Ohjelma laskee annetun pisteen etäisyyden ympyrän keskipisteestä pythagoran lauseen avulla ja pisteen ollessa ympyrän säteen ulottuvissa palauttaa toteutuessaan kokonaisluvun yksi [37]. Siveltimen neljä kohdalla ohjelma vertaa ensin Spread-muuttujaa Unityn satunaisgeneroituun lukuun, joista molemmat ovat väliltä 0-100. Mikäli käyttöliittymässä annettun Spread-muuttujan arvo on isompi kuin satunaisluku, ohjelma palauttaa arvon nolla ja pikseliä ei maalata. Muutoin ohjelma siirtyy ehto-lauseeseen laskemaan pisteen sijaintia.

```
public int CountCircle(Vector2 pixel, int x, int y, int dxy)
{
      int x2 = (int)pixel.x-x;
      int y2 = (int)pixel.y-y;
      int rnd = UnityEngine.Random.Range(0, 10);
      if (brush == 4){
            if (Spread >= rnd)
            {
                  return 0;
            }
            if ((x2 * x2 + y2 * y2) <= (dxy / 2) * (dxy / 2))
            {
                  return 1;
            }
            else
            {
                  return 0;
            }}
      if ((x2 * x2 + y2 * y2) <= (dxy / 2) * (dxy / 2))
      {
            return 1;
      }
      else
      {
            return 0;
      }
```

Esimerkkikoodi 13. Ote laaditusta koodista, jossa näkyy CountCircle-funktion toiminta.

Sivellin 2 ja 3 kolme saivat uudet toimintaperiaatteet, kun aseta pikselit -toiminto (eng. setPixels) saatiin toimimaan. Sivellin 2 käyttää aseta pikselit -toimintoa, jonka pitäisi olla huomattavasti tehokkaampi kuin aseta pikseli -toiminto. Sivellin 2 toiminta perustuu aseta pikselit -toiminnon perusmekaniikalle, jossa sille annetaan x- ja y-koordinaatit tekstuurista sekä maalattavan alueen koko.

```
Color[] colors = new Color[dxy * dxy];
     for (int i = 0; i < dxy * dxy; i++)
     {
     colors[i] = chooseColor;
     }
tex.SetPixels(x, y, dxy, dxy, colors);
tex.Apply();
```

Esimerkkikoodi 14. Ote laaditusta koodista, jossa esimerkki aseta pikselit -toiminnolle.

Sivellin 3 kutsuu funktiota, jossa lasketaan ympyrä pisteet x- ja y-koordinaattien ympäriltä siten, että muuttuja r on puolet käyttäjän antamasta siveltimen koosta, sillä sitä käytetään ympyrän säteenä. Funktio tallentaa ympyrän pisteet Array-listaan, jonka jälkeen se värjää pikselit SetPixel32-komennolla. Sivellin 3 käyttää aseta pikselit32-toimintoa (eng. setPixels32), jonka pitäisi olla tehokkaampi kuin aseta pikselit -toiminto.

```
public void Circle(Texture2D tex, int cx, int cy, int r, Color col)
    {
        int x, y, px, nx, py, ny, d;
        Color32[] tempArray = tex.GetPixels32();

            for (x = 0; x <= r; x++)
            {
                d = (int)Mathf.Ceil(Mathf.Sqrt(r * r - x * x));
                for (y = 0; y <= d; y++)
                {
                    px = cx + x;
                    nx = cx - x;
                    py = cy + y;
                    ny = cy - y;
                    tempArray[py * tex.width + px] = col;
                    tempArray[py * tex.width + nx] = col;
                    tempArray[ny * tex.width + px] = col;
                    tempArray[ny * tex.width + nx] = col;
                }
            }
        tex.SetPixels32(tempArray);
        tex.Apply();
    }
```

Esimerkkikoodi 15. Ote laaditusta koodista, jossa lasketaan ympyrä Cicle-funktion avulla.

Maalattavan tekstuurin kopiointi

3DM-Spray-työkalu tallentaa kopioitavat tekstuurit Recourses-kansion sisälle uuteen kansioon 3D-SprayTextures-kansioon. Ohjelma tarkistaa ennen kyseisten kansioiden luontia niiden olemassa olon päällekkäisyyksien ehkäisemiseksi. Tallennusratkaisun kannalta on välttämätöntä, että ohjelma tietää kopioitujen tekstuurien tiedostopolun ja nimen. ”Resources”-kansio mahdollistaa Lataus-operaation (eng. Load) käytön koodissa. Mahdollisten uusien kansioiden luomisen jälkeen luodaan uusi tekstuuri 3D-SprayTextures-kansioon nimellä ”copiedTexture”, jonka perässä on järjestysnumero.

```
IEnumerator CopyTexture(Renderer rend)
     {
     //Check that Assets/Materials/3D-SprayMaterials folders exists
     if (!Directory.Exists("Assets/Resources"))
          {
          AssetDatabase.CreateFolder("Assets", "Resources");
          AssetDatabase.Refresh();
          Debug.Log("Textures folder created");
          }
     if (!Directory.Exists("Assets/Resources/3D-SprayTextures"))
          {
          AssetDatabase.CreateFolder("Assets/Resources", "3D-SprayTextures");
          AssetDatabase.Refresh();
          Debug.Log("3D-SprayTextures folder created");
          }
```

Esimerkkikoodi 16. Ote laaditusta koodista, jossa luodaan uudet kansiot tekstuureille.

Tekstuurin kopiointi suoritetaan luomalla ensin uusi tekstuuri Unityn Texture2D-formaatissa ja antamalla sille perusarvoina kopioitavan tekstuurin arvot. Tämän jälkeen kopioidaan vanhan tekstuurin kaikki tiedot ja data CopyRawdata-metodilla [38], mikä käytännössä tarkoittaa raakadatan hakemista vanhasta tekstuurista ja sen lataamista kopioitavaan tekstuuriin.

```
     oldTex = rend.sharedMaterial.mainTexture as Texture2D;
     copiedTex = new Texture2D(oldTex.width, oldTex.height, TextureFor-
mat.RGBA32, false);
     copiedTex.LoadRawTextureData(oldTex.GetRawTextureData());
     copiedTex.Apply();
```

Esimerkkikoodi 17. Ote laaditusta koodista, jossa kopioidaan ja tallennetaan alkuperäisen tekstuurin raakadata uuteen tekstuuriin.

Seuraavaksi tekstuuri on ladattava tietokantaan uutena tekstuurina, joka on png-kuvamuodossa. Png on lyhenne englanninkielisistä sanoista Portable Network Graphics, mikä on kuvan tallennusformaatti. Tekstuurin tallennusmuodon muuttaminen png-

kuvaksi toteutetaan ”EncodeToPNG”-metodilla, joka muuttaa sen bittijonoksi [39]. Tämän jälkeen se ladataan haluttuun kansioon tiedostopolun avulla ja nimetään samalla ”.png”-loppuisena.

```
      // Encode texture into PNG
      byte[] bytes = copiedTex.EncodeToPNG();
      File.WriteAllBytes(Application.dataPath + "/Resources/3D-SprayTex-
tures/CopiedTexture" + i + ".png", bytes);
```

Esimerkkikoodi 18. Ote laaditusta koodista, jossa tekstuuri puretaan png-muotoon.

Bittijonon lataaminen uudeksi tekstuuriksi tietokantaan on raskas prosessi ja kestää sen takia joitakin sekunnin sadasosia. Tämä aiheutti virheilmoituksen, joka johtui koodin liikkumisesta eteenpäin, vaikka tekstuuri ei vielä ollut tallentunut. Ratkaisuna oli muuttaa koko CopyTexture-funktio coroutine-funktioksi, eli mahdollistaa ajankäyttäminen parametrina. Coroutine-funktion ominaisuuksia hyödynnettiin niin kauan kuin-silmukassa, joka tarkistaa, löytyykö Resources-kansiosta oikean nimistä tekstuuria. Mikäli sitä ei löydy ohjelma odottaa yhden näyttökuvan päivityksen ja antaa Unityn konsoliin ilmoituksen, että ohjelma odottaa tekstuurin tallentamista.

```
      while (Resources.Load("CopiedTexture" + i + ".png") != null)
            {
            yield return new WaitForFixedUpdate();
            Debug.Log("Waiting for CopiedTexture" + i + ".png to be saved...");
            }
      Debug.Log("CopiedTexture" + i + ".png saved succesfully!");
      yield return new WaitForFixedUpdate();
```

Esimerkkikoodi 19. Ote laaditusta koodista, jossa varmistetaan, että uusi tekstuuri on luotu ennen, kuin ohjelma siirtyy käsittelemään sitä.

Kun ohjelma löytää ladatun tekstuurin, se ilmoittaa Unityn konsolissa tallennuksen onnistuneen ja odottaa vielä virheiden ehkäisemiseksi yhden näyttökuvan päivityksen. Sen jälkeen ohjelma hakee juuri ladatun tekstuurin tietokannasta ja luo sille tekstuurinvälittäjän (eng. TextureImporter), jonka avulla tekstuurin asetuksista voidaan aktivoida tekstuurin muokkattavuus.

```
      TextureImporter importer = AssetImporter.GetAtPath("Assets/Resources/3D-
SprayTextures/CopiedTexture" + i + ".png") as TextureImporter;
      importer.isReadable = true;
      AssetDatabase.Refresh();
```

Esimerkkikoodi 20. Ote laaditusta koodista, jossa luodaan tekstuurinvälittäjä (eng. TextureImporter).

SarperSoher.com-nettisivuilta löytyi ”TextureImporterPlatformSettings”-funktio, jolla toteutettiin tekstuurin formaatin muuttaminen RGBA 32 bit -muotoon ja alusta saatiin muutettua ”Override for PC, Mac and Linux Standalone” -tilaan. Funktion syötetään halutut arvot ja sen jälkeen se annetaan parametrina tekstuurin välittäjälle. Ongelmia aiheuttaneen tekstuurin luettava-asetuksen aktivoinnin päivity suoritetaan tietokannan päivityksen pakotuksella, jonka jälkeen tekstuurin kopiointi on valmis ja käynnistetään materiaalin kopiointifunktio.

```
tips = new TextureImporterPlatformSettings()
      {
      allowsAlphaSplitting = false,
      compressionQuality = 0,
      format = importer.DoesSourceTextureHaveAlpha() ? TextureImporterFor-
mat.RGBA32 : TextureImporterFormat.RGBA32,
      maxTextureSize = 1024,
      name = "Standalone",
      overridden = true,
      textureCompression = TextureImporterCompression.CompressedHQ
      };
importer.SetPlatformTextureSettings(tips);
AssetDatabase.ImportAsset("Assets/Resources/3D-SprayTextures/CopiedTexture" +
i + ".png", ImportAssetOptions.ForceUpdate);
AssetDatabase.Refresh();
CopyMaterial(rend);
```

Esimerkkikoodi 21. Ote laaditusta koodista, jonka avulla tekstuurin formaatin muuttaminen on mahdollista.

Alkuperäisen materiaalin kopiointi ja sen asettaminen objektille

3DM-Spray-työkalua käyttävän projektin projektinhallinnan kannalta on tärkeää, että kopioitavat materiaalit tallennetaan asianmukaisiin kansioihin. Tallennusratkaisun kannalta on välttämätöntä, että ohjelma tietää kopioitujen materiaalien tiedostopolun ja nimen. Materiaalin kopiointi aloitetaan tarkastamalla, onko kansioita ”Resources” ja ”3D-SprayMaterials” olemassa ja luoda sellaiset, mikäli projektista ei löydy kyseisiä kansioita. ”Resources”-kansio mahdollistaa Lataus-operaation käytön koodissa. Ohjelman varmistuessa kansioiden olemassaolosta se luo uuden materiaalin ”3D-SprayMaterials”-kansioon nimellä ”copiedMaterial” ja järjestysnumerolla.

```
//Check that Assets/Resources/3D-SprayMaterials folders exists
      if (!Directory.Exists("Assets/Materials"))
      {
            AssetDatabase.CreateFolder("Assets", "Resources");
            AssetDatabase.Refresh();
            Debug.Log("Materials folder created");
      }
```

```
      if (!Directory.Exists("Assets/Resources/3D-SprayMaterials"))
      {
            AssetDatabase.CreateFolder("Assets/Resources", "3D-SprayMaterials");
            AssetDatabase.Refresh();
            Debug.Log("3D-SprayMaterials folder created");
      }
//Create new material
      mat = Material.Instantiate(rend.material);
      AssetDatabase.CreateAsset(mat, "Assets/Resources/3D-SprayMaterials/Cop-
iedMaterial"+i+".mat");
```

Esimerkkikoodi 22. Ote laaditusta koodista, jossa luodaan uudet kansiot materiaaleille.

Ohjelma ei siis suoraan kopioi vanhaa materiaalia, vaan luo uuden, johon se sen jälkeen kopioi vanhan materiaalin ominaisuudet (eng. properties). Ohjelma tarkistaa, ettei valittu objekti ole tyhjä ja sen jälkeen kopioi sen ominaisuudet juuri luotuun materiaaliin. Materiaalin luominen ilmentymän luomiskomennolla (eng. Instantiate) aiheuttaa ongelman, koska se luo materiaalille väliaikaisen kopion, joka on objektin materiaalina pelitilasta poistumiseen asti. Väliaikaisen kopion ilmestymisen ehkäisemiseksi ilmentymän luomiskomento on jätettävä pois ja kaikki ”.material”-viittaukset on muutettava ”.sharedMaterial”-muotoon, jolla tarkoitetaan jaettua materiaalia tietokannassa [40].

```
//Place new material to the object
      if (rend != null)
      {
      //Copy properties from old material
      rend.material = mat;
      rend.material.mainTexture = copiedTex;
      AssetDatabase.Refresh();
      }
```

Esimerkkikoodi 23. Ote laaditusta koodista, jossa uusi materiaali ja tekstuuri asetetaan objektille.

Materiaalin kopioinnissa kävi ilmi, että sen ominaisuuksien muuttaminen pelitilassa vaatii tietokannan päivittämisen ja tällä ratkaisulla sen tiedot kopioituvat objektille. Pelkkien tietojen kopioituminen materiaalille tarkoittaa, että sitä ei kuitenkaan ole osoitettu objektille ja sama pätee tekstuurin osoittamista materiaalille. Oikea menettelytapa on käyttää lataa tiedostopolku -toimintoa (eng. LoadAssetPath), jonka avulla objektille voidaan osoittaa oikea materiaali ja sille tekstuuri.

```
if (rend != null)
      {
      rend.sharedMaterial = (Material)AssetDatabase.LoadAssetAtPath(assetPath,
typeof(Material));
      AssetDatabase.Refresh();
      Debug.Log("CopiedMaterial" + i + ".mat saved succesfully to object!");
```

```
      rend.sharedMaterial.mainTexture = (Texture2D)AssetDatabase.LoadAs-
setAtPath(assetPathTex, typeof(Texture2D));
      AssetDatabase.Refresh();
      Debug.Log("CopiedTexture" + i + ".mat saved succesfully to material!");
```

Esimerkkikoodi 24. Ote laaditusta koodista, jossa tallennetaan materiaali tietokantaan.

Tallentaminen sarjanumeron ja tekstitiedoston avulla

Tallennusfunktio tallentaa jokaisen maalatun objektin erikseen niin kauan kuin -silmukan sisällä, jonka koko on määritelty yleisessä muuttujassa j. Funktio laskee muuttujan j arvoa jokaisella silmukallaan ja lopettaa, kun j:n arvo on yksi.

```
if (j != 0)
      {
      List<string> saveObjects = new List<string>();
      while (j > 0)
            {
            j--;
```

Esimerkkikoodi 25. Ote laaditusta koodista, jossa muuttujan j avulla lasketaan niin kauan kuin -silmukan kesto.

Funktio kutsuu findID-funktiota, joka palauttaa paintedObjects-listassa sijalla j-muuttujan arvon olevan objektin yksilöllisen sarjanumeron ja tallentaa sen muuttujaan ”localId”.

```
//Local Identfier In File
            int localId = findID(paintedObjects[j]);
```

Esimerkkikoodi 26. Koodirivi, jossa tallennetaa objektin yksilöllinen sarjanumero localId-muuttujaan.

Luodaan kolme muuttujaa selkeyttämään koodia. Ensimmäinen on rendereri ”paintedRend”, jolle annetaan paintedObjects-listan sijalla j-muuttujan arvon olevan objektin rendereri. Loput kaksi ovat kirjainjonoja (eng string), joihin tallennetaan kopioidun materiaalin tiedostopolku ja toiseen materiaalin uusi nimi.

```
//Rename copiedmaterials
            Renderer paintedRend = paintedObjects[j].GetComponent<Renderer>();
            String assetPath = "Assets/Resources/3D-SprayMaterials/" + paint-
edRend.sharedMaterial.name+".mat";
            String newName = localId + "_PaintedMaterial.mat";
```

Esimerkkikoodi 27. Ote laaditusta koodista, jossa uudet materiaalit nimetään uudelleen ja uusiin nimiin liitetään objektin yksilöllinen sarjanumero.

Uudelleennimetään objektin materiaali, jotta lataamisvaiheessa se voidaan yhdistää objektiin sen nimeen liitetyn sarjanumeron (localId) avulla. Tietokanta pitää tallentaa ja päivittää, jotta muutokset tulevat näkyviin projektissa.

```
AssetDatabase.RenameAsset(assetPath, newName);
AssetDatabase.SaveAssets();
AssetDatabase.Refresh();
```

Esimerkkikoodi 28. Ote laaditusta koodista, jossa uuden nimen tallennus ja tietokannan päivitys tapahtuu.

Luodaan kirjainjono ”objectName”, johon tallennetaan objektin nimi ja sen sarjanumero, jotta lataamisvaiheessa objektin löytäminen sarjanumeron avulla on mahdollista. Objektin nimen ja sarjanumeron väliin lisätään kirjainjono ”_3DMspray_”, jonka tarkoitus on erottaa ne toisistaan. Tämän jälkeen kirjain jono lisätään saveObjects-listaan.

```
String localIdString = localId.ToString();
String objectName = paintedRend.name + "_3DMspray_" + localIdString;
saveObjects.Add(objectName);
}
```

Esimerkkikoodi 29. Ote laaditusta koodista, jossa luodaan uusi kirjainjono johon lisätään objektin nimi ja sen yksilöllinen sarjanumero.

Kun niin kauan kuin-silmukka on käynyt kaikki saveObjects-listan objektit läpi, se tallennetaan saveObjects-lista tietokoneen temp-kansiossa sijaitsevaan tekstitiedostoon ”SacedObjectNamesFor3DsprayTool”.

```
    System.IO.File.WriteAllLines(System.IO.Path.GetTempPath() + "SavedObject-
NamesFor3DsprayTool.txt", saveObjects.ToArray());
    }
```

Esimerkkikoodi 30. Ote laaditusta koodista, jossa lista kirjainjonoista tallennetaan väliaikaiseen tekstitiedostoon.

Muutosten lataamisen toteutus editointitilassa

Latausfunktion ensimmäinen tehtävä on listata projektin kaikki objektit allObjects-listaan sarjanumeron vertailua varten. AllObjects-listan käyttö on suoritustehollisesti raskain operaatio maalaustyökalun toiminnassa. Funktio lataa tietokoneen temp-kansiosta tekstitiedoston ”SacedObjectNamesFor3DsprayTool” sisällön objectlines-listaan ja tallentaa listan pituuden muuttujaan k.

```
      GameObject[] allObjects = FindObjectsOfType<GameObject>();
      string[] objectlines = System.IO.File.ReadAllLines(Sys-
tem.IO.Path.GetTempPath()+"SavedObjectNamesFor3DsprayTool.txt");
      k = objectlines.Length;
```

Esimerkkikoodi 31. Ote laaditusta koodista, jossa haetaan kirjainjonot tekstitiedostosta.

Latausfunktion while-silmukkassa vähennetään muuttujan k arvoa yhdellä, haetaan objectlines-listan kirjainjono listan muuttujan k-arvon kohdasta ja katkaistaan kirjainjono kohdasta ”_3DMspray_”. Ensimmäinen osa kirjainjonosta tallennetaan uudeksi kirjainjonoksi ”oldName”, joka saa kirjainjonokseen peliobjektin alkuperäisen nimen. Toinen osa on ”_3DMspray_”, jota ei käytetä mihinkään. Kolmas osa kirjainjonosta tallennetaan IdName-muuttujaan, joka saa kirjainjonokseen peliobjektin sarjanumeron.

```
      while (k > 0)
      {
            k--;
            String gameobjectFullString = System.Convert.ToString(object-
lines[k]);
            String[] splitName = gameobjectFullString.Split("_3DMspray_"[0]);
            String oldName = splitName[0];
            String IdName = splitName[2];
            String localId = " ";
```

Esimerkkikoodi 32. Ote laaditusta koodista, jossa kirjainjonot paloitellaan objektin nimeksi ja sen sarjanumeroksi.

Seuraavaksi niin kauan kuin-silmukan (eng. while) sisällä on toisto-lause (eng. for), joka käy kaikki allObjects-listan kirjainjonot vuorotellen läpi ja vertaa niitä ”IdName”-kirjainjonoon. Samanlaisten jonojen löytyessä kutsutaan findId-funktiota, jolla haetaan allObjects-listan objektin sarjanumero ja tallennetaan se localId-muuttujaan. Seuraavassa ehto-lauseessa verrataan localId-muuttujaa IdName-muuttujaan, eli etsitään muokatun objektin sarjanumeroa vastaava objekti.

```
            for(int i = 0; i < allObjects.Length; ++i)
            {
            if(allObjects[i].name == oldName)
                  {
                  localId = findID(allObjects[i]).ToString();
                  if (localId == IdName)
                        {
```

Esimerkkikoodi 33. Ote laaditusta koodista, jossa etsitään oikea objekti sen nimen ja sarjanumeron avulla.

Oikean objektin löytyessä etsitään objektin sarjanumerolla nimetyn materiaalin tiedostopolku ja asetetaan oikea materiaali sen avulla objektille. Funktion lopuksi jaetaan vielä

materiaalin nimi sarjanumeron lopusta kahtia ja tallennetaan materiaalille uusi nimi objektin alkuperäisen nimen mukaan, johon lisätään loppuliite ”_PaintedMaterial.mat”. Uuden materiaalin nimen tarkoituksena on helpottaa materiaalien löytämistä projektin kansioista.

```
                    String assetPathMat = "Assets/Resources/3D-SprayMateri-
als/" + IdName + "_PaintedMaterial.mat";
                    allObjects[i].GetComponent<Renderer>().sharedMaterial =
(Material)AssetDatabase.LoadAssetAtPath(assetPathMat, typeof(Material));
                    String[] splitMatName = gameobjectFull-
String.Split("_"[0]);
                    String oldMatName = splitName[0];
                    String newName = (oldName + "_PaintedMaterial.mat");
                    AssetDatabase.RenameAsset(assetPathMat, newName);
                }}}}
```

Esimerkkikoodi 34. Ote laaditusta koodista, jossa oikealle objektille asetetaan sen uusi materiaali.

Sarjanumeron hakeminen funktiolla

Muuttumattoman uniikin sarjanumeron hakeminen koodissa on tehty Unity-pelimoottorissa vaikeaksi ja objektien ainoa tällainen numero on ”Local Identfier In File” -sarjanumero. findID-funktio palauttaa sille annetun objektin ”Local Identfier In File” -sarjanumeron. TheLackey3326-nimimerkin ratkaisussa haetaan ensin ominaisuustiedoista (eng. PropertyInfo) sarjallistetun objektin käyttöliittymätilan (eng. inspectorMode) tiedot, jotka tallennetaan inspectorModeInfo-muuttujaan.

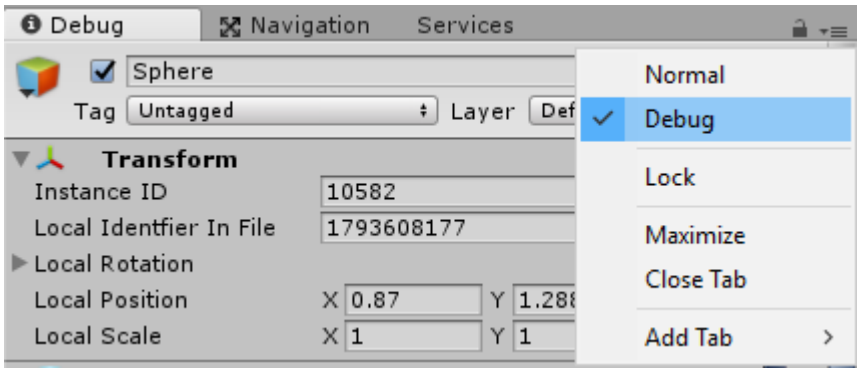
```
public int findID(GameObject gameobjectID)
    {
     PropertyInfo inspectorModeInfo = typeof(SerializedObject).GetProp-
erty("inspectorMode", BindingFlags.NonPublic | BindingFlags.Instance);
```

Esimerkkikoodi 35. Ote laaditusta koodista, jossa haetaan sarjallistetun objektin ominaisuustiedot.

Seuraavaksi luodaan uusi sarjallistettu objekti ”serializesObject”, jolle annetaan parametrina funktiolle annettu peliobjekti. inspectorModeInfo-muuttujalle annetaan parametreina sarjallistettu objekti, käyttöliittymätilan vianmääritys välilehden (eng. debug) ja tyhjän muuttujan. Näin päästään koodin kautta käsiksi kuvan 26 objektin käyttöliittymän vianmääritys-välilehteen, josta ”Local Identfier In File”-sarjanumeron arvo on haettavissa [41].

```
SerializedObject serializedObject = new SerializedObject(gameobjectID);
inspectorModeInfo.SetValue(serializedObject, InspectorMode.Debug, null);
```

Esimerkkikoodi 36. Ote laaditusta koodista, jossa luodaan uusi muuttujan jolle annetaan objektin sarjanumero.



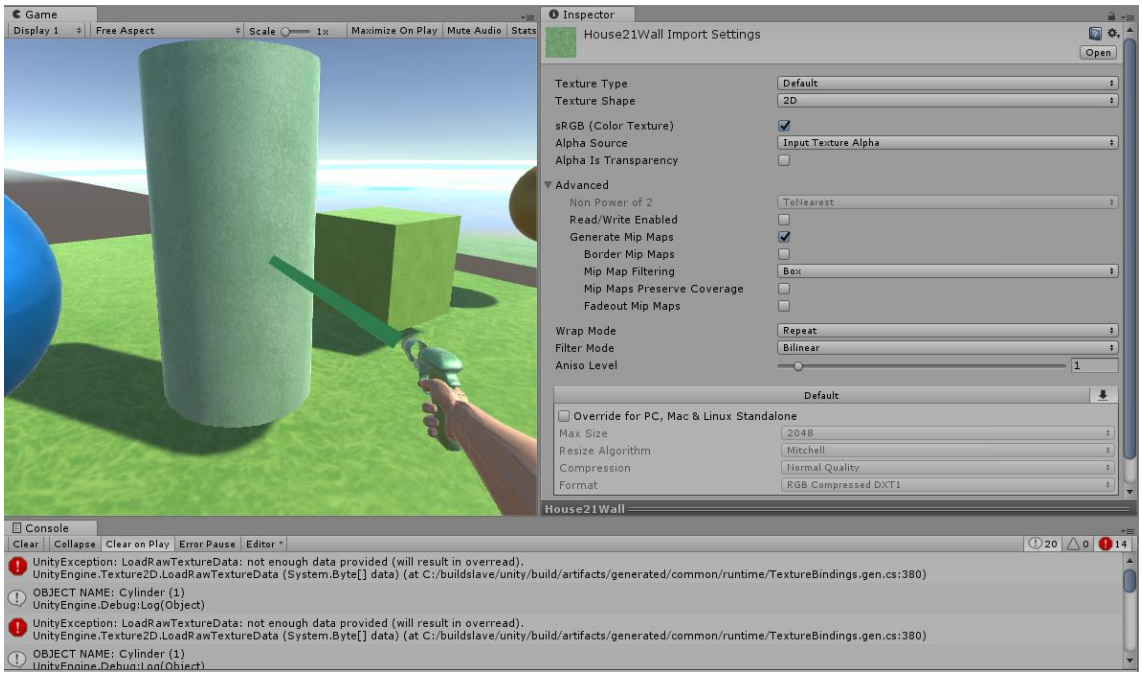
Kuva 26. Objektin käyttöliittymän tilaa voi vaihdella normaalitilan ja vianmääritystilan välillä.

Tämän alustuksen jälkeen päästään käsiksi ”Local Identfier In File” -sarjanumeroon, joka tallennetaan sarjallistettuna ominaisuutena (eng. SerialzedProperty) ”localIdPorp”-muuttujaan. Tästä muuttujasta sarjanumero voidaan muuttaa kokonaisluvuksi (eng. integer) ja tallentaa localId-muuttujaan, jonka funktio palauttaa.

```
SerializedProperty localIdProp = serializedObject.FindProperty("m_LocalI-
dentfierInFile");   //note the misspelling!

int localId = localIdProp.intValue;
return localId;
}
```

Esimerkkikoodi 37. Ote laaditusta koodista, jossa localIdProp-muuttujaan tallennetaan objektin yksilöllinen sarjanumero.

Virheilmoitukset ja käyttäjävirheiden ehkäisy



Kuva 27. Kuvakaappaus virheilmoituksesta ja siihen johtaneen objektin alkuperäisestä tekstuurista sekä virheellisestä formaatista.

Mikäli 3DM-spray-työkalulla yritetään maalata objektia, jota ei ole muutettu oikeaan formaattiin, se antaa kuvan 27 mukaisen virheilmoituksen LoadRawData-metodista. Tähän tehtiin helposti ratkaisu kopioimalla tekstuurin kopiointimetodista tekstuurin muuttaminen muokattavaan muotoon ja sen formaatin muuttaminen. Tekstuurin kopiointifunktioon lisätään ehto-lause, jossa tarkistetaan, täsmääkö vanhan tekstuurin formaatti valittua tekstuuria ja väärien tekstuurien kohdalla se kutsuu ChangeTextureFormat-funktiota. Samalla CopyTexture-funktio laitettiin hyödyntämään uutta ChangeTextureFormat-funktiota kaikissa formaatin muuttamista vaativissa tilanteissa.

```
if (oldTex.format != TextureFormat.RGBA32 || oldTex.format != TextureFor-
mat.RGB24)
      {
      Debug.Log("Old texture format changed");
      ChangeTextureFormat(oldTex);
      }
```

Esimerkkikoodi 38. Ote laaditusta koodista, jossa tekstuurin formaatti tarkistetaan ja muutetaan tarvittaessa.

Samaan objektiin useilla eri ohjelman käynnistyskerroilla aiheuttaa virheen, koska kopioitu tekstuuri on samanniminen kuin ensimmäisellä kerralla, vaikka välissä olisi tallennettu. Tallennus-funktiossa tekstuurin uudelleennimeäminen aiheuttaa muutosten katoamisen tekstuurista.

5 3DM-Spray:n suoritustehon testaus, testitulokset ja tulevaisuus

5.1 3DM-Spay-työkalun suoritustehon testaus

Taulukossa 1 on testissä käytetyn tietokoneen tiedot. Kyseessä oli msi G51 -merkkinen kannettava tietokone, joka oli testin aikana kytkettynä verkkovirtaan. Tietokoneessa on Intel Core i7-6700HQ -prosessori ja yksi 8 gigabitin muisti, jonka nopeus on 2133 megahertsiä. Käyttöjärjestelmänä on Windows 10 Home, joka on 64-bittinen käyttöjärjestelmä. Näytön ohjaimen tietokoneessa on Nvidia:n GTX960M. Voidaan todeta, että kyseessä on keskiverto kannettavaa tietokonetta tehokkaampi kone, joka soveltuu hyvin pelinkehitykseen.

Taulukko 1. Testissä käytetyn tietokoneen tekniset tiedot

Tietokone	**Msi G51**
Prosessori	Intel® Core™ i7-6700HQ CPU @ 2.60Ghz 64-bittinen
Muisti (RAM)	1 x 8 GB, 2133 MHz
Käyttöjärjestelmä	Windows 10 Home, 64-bittinen käyttöjärjestelmä
Näytönohjain	Nvidia GTX960M

Taulukossa 2 on testissä käytetyn Unity-version asetukset ja projektin tiedot. Asetuksista huomioitavaa on kuvataajuuden optimointi (eng. V sync count) asetuksen muuttaminen ei rajoitettuun (don’t sync) asetukseen. Projektista poistettiin 3DM-Spray-työkalua, lattiaa simuloivaa neliötä ja kuution muotoista kohdeobjektia lukuun ottamatta kaikki muut objektit, mukaan lukien valot. Kohdelaatikon materiaalin tekstuuriksi valittiin 1024 x 1024 pikseliä kokoinen tekstuuri, koska sen resoluutio on riittävän tarkka pelimaailman objekteihin ja työn tilanneen yrityksen projektien suurimmat tekstuurit ovat sen kokoisia.

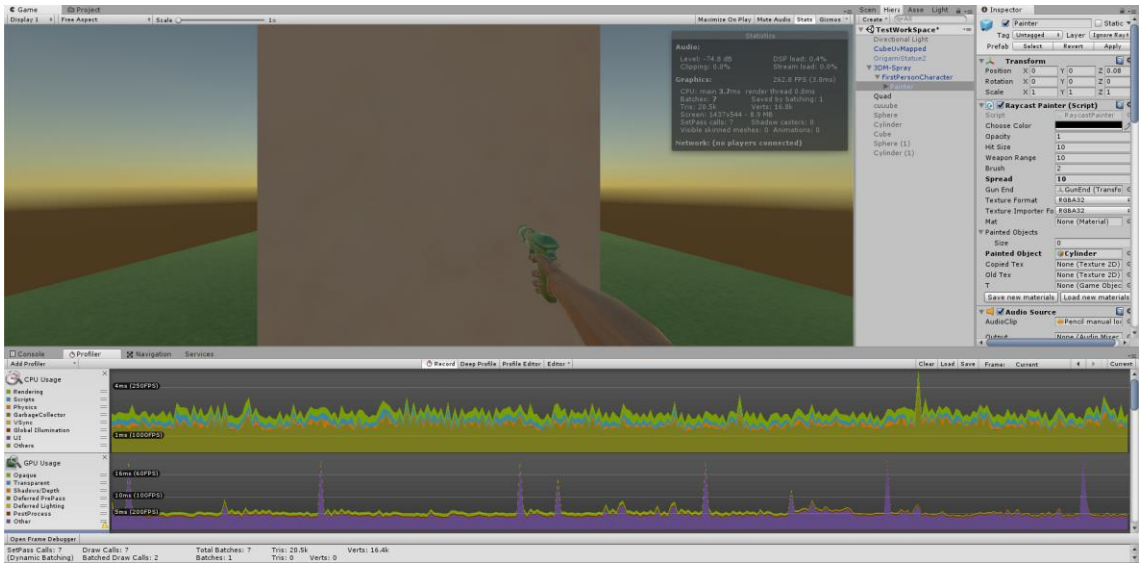
Taulukko 2. Unity-versio ja sen asetukset sekä projektin sisältö

Unity versio	**Unity 2017.3.1f1**
V sync count	Don’t sync
Projektin sisältö	
3DM-Spray	Maalaustyökalu, joka toimii myös pelihahmona
15x15 Neliö (eng. quad)	Toimii tasona, jonka päällä pelihahmo seisoo
4x4x4 Laatikko (Blender)	Blenderissä tehty laatikko, jonka tekstuuriin maalataan.
Tekstuuri	1024 x 1024 pikselin kokoinen tekstuuri
Directional light	Kaikki valot on otettu pois päältä

Testauksen tavoite

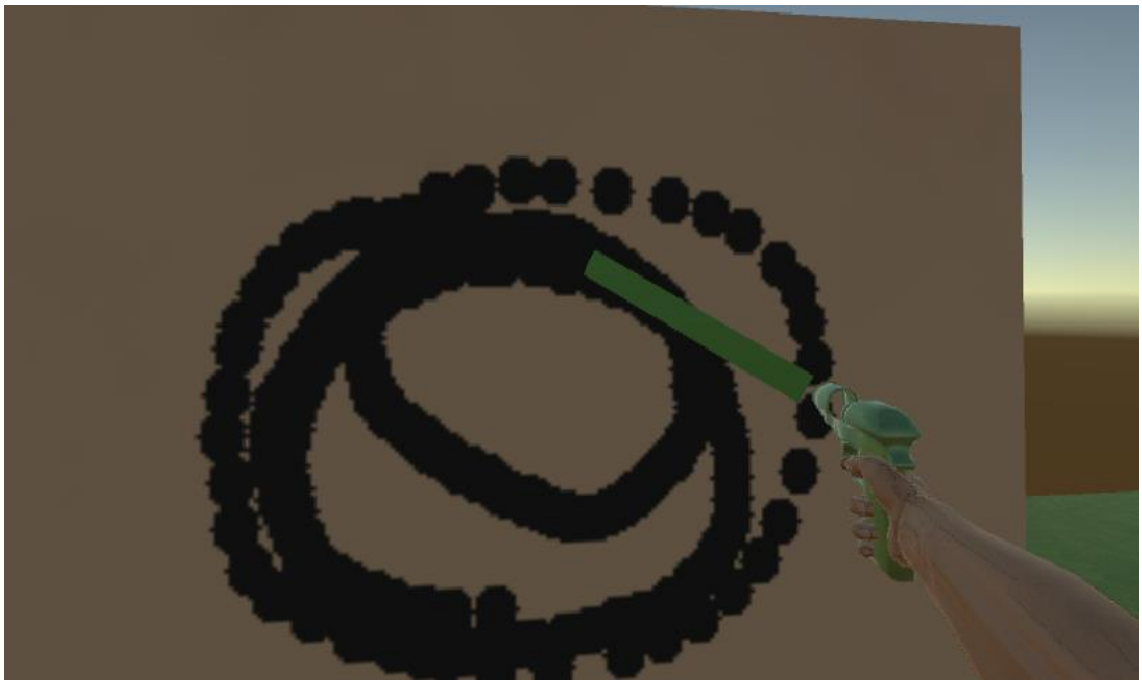
Suoritustehon testaamisen tavoite on selvittää mahdollisia ongelmakohtia maalaamisen kannalta. Tallennus- ja latausominaisuutta ei tarvitse testata, sillä niissä tapahtuvat viiveet eivät vaikuta käyttäjäkokemukseen negatiivisesti ja ovat siksi hyväksyttäviä. Testauksessa tarkastellaan kuvassa 28 näkyvästä Unityn Profiler-ikkunan antamia lukuja prosessorin (eng. CPU) tehosta ja grafiikkaprosessorin (eng. GPU) tehosta. Lisäksi Unityn peli-ikkunasta tarkkaillaan statistiikka ikkunasta pelin kuvataajuuden tehoa sekunnilta (fps) arvoa. Testin tarkoitus on antaa suuntaa-antava arvio työkalun suorituskyvystä ja sen takia mittaustuloksissa tapahtuneita virheitä ei tarvitse ottaa huomioon.

Mittaustulokset laskettiin Unityn dataikkunoissa juoksevien lukujen keskiarvoina, mikä aiheuttaa mahdollisuuden inhimillisiin virheisiin ja täten voidaan todeta, että testitulokset eivät ole tarkkoja arvoja. Testauksen tavoitteiden kannalta suurella virhemarginaalilla ei kuitenkaan ole merkitystä, sillä mittauksesta saadaan selville hyvä kokonaiskuva työkalun toiminta tehosta.



Kuva 28. Kuvakaappaus Unityn näkymästä testausvaiheessa. Peli-ikkunan oikeassa yläreunassa näkyy statistiikka, josta löytyy ruudun päivitys sekunnilta arvo. Alhaalla on kaksi aaltoliikettä tekevää käyrää, jotka kuvaavat grafiikkaprosessorin ja prosessorin suoritusta.

Testissä testataan kaikkien neljän siveltimen, sekä vertailuna ilman maalausta, tehokkuutta yksittäisen kuvion maalauksessa ja useamman kuvan maalaamisessa. Testi suoritetaan yhden, kymmenen, sadan ja viiden sadan pikselin kokoisilla siveltimillä. Ensimmäinen osa testistä suoritettiin ampumalla yksittäinen laukaus keskelle kuvassa 28 näkyvää kuution muotoista objektia ja tarkkailemalla laukauksen aiheuttamia poikkeamia arvoissa. Toinen osa testistä suoritetaan pitämällä hiiren nappulaa pohjassa ja liikuttamalla sitä ympyrässä, eli maalaamalla kohde objektiin ympyröitä kuva 29 mukaisesti.



Kuva 29. Kuvakaappaus testin kaksi toisesta vaiheesta, jossa siveltimellä kolme maalataan ympyröitä.

Testitulokset ovat taulukoissa kolme, neljä, viisi ja kuusi. Vasemmanpuoleisissa sarakkeissa on ylhäältä alas testin numero ja testissä käytetyt siveltimet. Sinisellä pohjalla on kaikki testien ensimmäisten osien tulokset ja vihreällä pohjalla toisien osien tulokset. Testeissä testatut arvot prosessorin tehosta, kuvataajuudesta sekunnilta ja grafiikkaprosessorin tehosta on listattu kunkin lyhenteen alle. Selvyyden vuoksi testituloksien taulukoissa on käytetty englanninkielisiä lyhenteitä, eli lyhenteellä ”CPU” (central processing unit) tarkoitetaan prosessoria, lyhenteellä ”FPS” (frames per second) tarkoitetaan kuvataajuutta sekunnilta ja lyhenteellä ”GPU” (graphics processing unit) tarkoitetaan grafiikkaprosessoria.

Ensimmäisenä testattiin yhden pikselin kokoisten siveltimien tehokkuutta. Tämä on ainoa testi, johon sivellin 1 voi osallistua, koska siihen ei ole vielä tehty koon muuttamista. Ensimmäisessä testissä oli myös mukana ilman 3DM-Spray-työkalun käyttöä suoritettu testi, jossa liikutettiin hiirtä samalla tavalla kuin siveltimien kanssa.

Taulukko 3. Testi yhdellä pikselillä. Testattiin ilman sivellintä ja siveltimillä 1-4.

Testi 1: 1px	**Yksittäislaukaus**			**Kuvion maalaaminen**		
Sivellin	**CPU**	**FPS**	**GPU**	**CPU 2**	**FPS 2**	**GPU 2**
Ei sivellintä	3 ms	350 fps	3 ms	3 ms	350 fps	5 ms
Sivellin 1	3 ms	350 fps	3 ms	20 ms	50 fps	16 ms
Sivellin 2	5 ms	200 fps	3 ms	14 ms	70 fps	10 ms
Sivellin 3	5 ms	160 fps	3 ms	17 ms	60 fps	10 ms
Sivellin 4	5 ms	180 fps	4 ms	15 ms	60 fps	12 ms

Taulukossa 3 on näkyvissä testin yksi tulokset, joista voidaan päätellä, että siveltimen 1 käyttö yksittäislaukauksella ei käytä näkyvää määrää mitään tehoja, kun taas kuviota maalatessa siveltimen 1 käyttämä aseta pikseli -toiminto on raskain kaikista siveltimistä. Siveltimien 2 ja 3 käyttämä aseta pikselit -toiminto on raskaampi maalatessa yhden pikselin kokoisella siveltimellä kuvioita, koska se luo jokaisella pikselin maalaus kerralla listan kaikista tekstuurin pikseleistä. Kaikissa neljässä testissä grafiikkaprosessori ei rasittunut yhtään yksittäislaukaustesteissä. Testin 1 tulokset olivat positiiviset, eikä mikään sivellin aiheuttanut ongelmia kuvion maalaamisessakaan ja kuvataajuuskin pysyi jokaisen siveltimen kohdalla yli 60 kuvapäivityksessä sekunnilta.

Taulukko 4. Testi kymmenellä pikselillä. Testi tehtiin siveltimillä 1-3.

Testi 2: 10px	**Yksittäislaukaus**			**Kuvion maalaaminen**		
Sivellin	**CPU**	**FPS**	**GPU**	**CPU 2**	**FPS 2**	**GPU 2**
Sivellin 2	6 ms	160 fps	3 ms	15 ms	60 fps	16 ms
Sivellin 3	6 ms	150 fps	3 ms	19 ms	60 fps	20 ms
Sivellin 4	9 ms	100 fps	3 ms	90 ms	10 fps	30 ms

Taulukossa 4 näkyvässä testissä kaksi ilmaantui ensimmäinen kuvataajuuden tippuminen huonolle tasolle, kun siveltimellä neljä maalattiin ympyräkuviota. Siveltimen 4 kuvion maalaaminen hidasti myös prosessorin toiminnan vasteajan 90 millisekuntiin, joka johtui epäilemättä raskaista aseta pikseli -toiminnosta ja satunnaisluvun generoimisesta. Siveltimet 2 ja 3 sen sijaan selvittivät kuvion maalaamisen täysin ongelmitta ja rasittamatta

prosessoreita huomattavasti. Yksittäislaukauksissa ei ollut mainittavia viiveitä tai tehojen käyttöä.

Taulukko 5. Testi sadalla pikselillä. Testi tehtiin siveltimillä 1-3.

Testi3:100px	**Yksittäislaukaus**			**Kuvion maalaaminen**		
Sivellin	**CPU**	**FPS**	**GPU**	**CPU 2**	**FPS 2**	**GPU 2**
Sivellin 2	6 ms	170 fps	3 ms	13 ms	70 fps	16 ms
Sivellin 3	6 ms	150 fps	3 ms	18 ms	50 fps	20 ms
Sivellin 4	400 ms	0 fps	-	-	-	-

Testissä 3 maalattiin sadan pikselin kokoisella siveltimellä, joka aiheutti huomattavaa hajontaa suorituskyvyssä. Kuten taulukosta 5 voi huomata, siveltimestä 4 ei ole dataa kuvion maalaamisesta, koska sillä kesti yksittäisen kuvion maalaamisessa useita sekunteja, ja se nosti prosessorin vasteajan yli 400 millisekuntiin. Siveltimen 4 yksittäislaukaus toimi kuitenkin halutulla tavalla, vaikkakin kuvataajuus tippui jokaisessa laukauksessa useaksi sekunniksi nollaan näytönpäivitykseen sekunnissa. Testi 3 osoitti myös siveltimien 2 ja 3 välillä ensimmäiset erot kuviota maalatessa, kun siveltimen 3 grafiikkaprosessorin vasteaika nousi yli 20 ms ja kuvataajuus laski 50 näytönpäivitykseen sekunnissa.

Taulukko 6. Testi 500 pikselillä. Testi tehtiin siveltimillä 1-3.

Testi4:500px	**Yksittäislaukaus**			**Kuvion maalaaminen**		
Sivellin	**CPU**	**FPS**	**GPU**	**CPU 2**	**FPS 2**	**GPU 2**
Sivellin 2	7 ms	180 fps	3 ms	25 ms	45 fps	20 ms
Sivellin 3	7 ms	130 fps	3 ms	45 ms	20 fps	28 ms
Sivellin 4	-	-	-	-	-	-

Taulukosta 6 nähdään testin 4 tulokset, joissa maalattiin 500 pikselin kokoisella siveltimellä yksittäislaukaukset ja yhtenäiset kuviot. Tämä onnistui siveltimillä 2 ja 3, mutta sivellin 4 aiheutti Unityn kuvan pysähtymisen, eikä Unity reagoinut komentoihin. Sivellintä 4 testattiin näissä olo suhteissa kolme kertaa ja jokainen testi päättyi usean minuutin

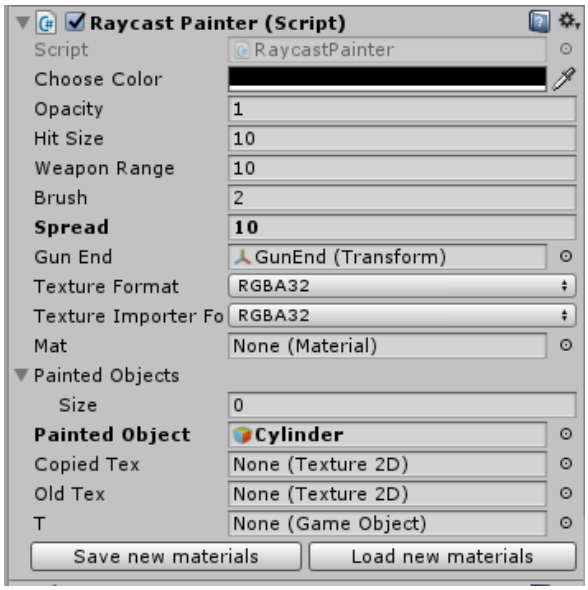
odottelun jälkeen Unityn sammutuksen pakottamiseen tietokoneen ohjauspaneelin kautta.

Sivellin 2 ja 3 suoriutuivat yksittäislaukauksista erittäin tehokkaasti, mutta siveltimen koko vaikutti kuvioiden maalaamiseen huomattavasti. Molemmat nostivat grafiikkaprosessorin ja prosessorin vasteaikaa yli 20 millisekuntiin. Sivellin kolme nosti prosessorin vasteajan jopa 45 millisekuntiin. Siveltimien 2 ja 3 kuvataajuudet laskivat myös huomattavasti, joka aiheutti katkonaisuutta maalatussa kuviossa. Siveltimen koon takia kuvion katkonaisuutta oli kuitenkin erittäin vaikea havaita. Siveltimen 3 kuvataajuus laski noin 20 kuvapäivitykseen sekunnilta, joka silti kaksi kertaa nopeampi kuin siveltimen 4 kuvataajuus testissä 2.

5.2 3DM-Spay-työkalun tulevaisuus

3DM-Spay-työkalun kehitystä jatketaan, ja se on edelleen tarkoitus saada julkaistavaan kuntoon. Seuraavat askeleet ovat siveltimien osalta niiden toimintojen kehittäminen ja monipuolistaminen. Eri siveltimille tulisi myös tehdä eri 3D-mallit niiden maaliruiskuista, jotta käyttäjä tietää aina mikä sivellin on käytössä. Käyttöliittymästä tulisi tehdä oma ikkunansa ja sen graafista ilmettä tulisi kohentaa, sillä tämän hetkinen ”Painter”-peliobjektissa olevan RaycastPainter-koodiin yhdistetty käyttöliittymä on graafisesti karu, kuten kuvasta 30 voi nähdä.

Maalausominaisuuksien monipuolistamiseksi suunnitteilla on siveltimien muotojen ja sattumanvaraisuuksien lisäämisen lisäksi uuden maalausominaisuuden lisääminen, joka toimisi Unity:n fysiikkamoottorin avulla. Tämän uuden työkalun tarkoitus on ampua käyttäjän määrittelemän verran partikkeleita, jotka saavat eri fysiikkavoimat lentää eteenpäin maaliruiskusta ja osuessaan objektiin maalaavat osumakohtansa valitulla kuviolla.



Kuva 30. Kuvakaappaus RaycastPainter-koodin käyttöliittymästä.

Siveltimien kannalta prioriteetteina on mahdollistaa siveltimen 1 toiminta muillakin siveltimien kokoina ja muuttaa sen toimintaperiaate toimimaan aseta pikselit -toiminnolla. Siveltimiin 2 ja 3 tulisi lisätä mahdollisuus maalata reunoilta häivytettyä kuviota. Testitulosten perusteella voidaan todeta, ettei sivellin neljä voi jäädä lopulliseen versioon sellaisenaan vaan se tulisi muuttaa käyttämään samaa metodia kuin sivellin 3, jossa luodaan ”array”-lista ja käytetään aseta pikselit -toimintoa.

6 Yhteenveto

Insinöörityön tarkoitus oli tutkia markkinoilta löytyviä 3D-mallien maalaustyökalujen ominaisuuksia ja toteuttaa 3D-mallien maalaustyökalu liitännäisenä Unity-pelimoottorille. Kilpailijoista erottumiseksi tähän maalaustyökaluun haluttiin tehdä ominaisuus tallentaa muutoksia Unityn pelitilasta, mutta kesken projektin kilpaileva maalaustyökalu Playtime Painter sai vastaavan tallennusratkaisun päivityksen yhteydessä [7]. Maalaustyökalu toteutettiin silti alkuperäisen suunnitelman mukaan ja sen tulevaisuuden suunnitelmaan lisättiin fysiikkapohjainen maalaaminen, joka antaisi sille kilpailijoihinsa nähden uniikin ominaisuuden.

Vaikka jo projektin alkuvaiheessa selvisi Unityn pelitilasta muutosten tallentamisen haasteellisuus ja aikataulusta tämän takia myöhästyminen, saatiin työkalun perustoiminnallisuus toteutettua ajoissa. 3DM-Spray -työkalun muokkaamien materiaalien ja tekstuurien tallentaminen pelitilasta saatiin toteutettua onnistuneesti samoin kuin lataustoiminto, jolla materiaalit ja tekstuurit asetetaan oikeille objekteille. Tallennuksen kanssa ilmennyt ongelma Unityn tekemän projektin varmuuskopion pelitilaan mentäessä ja sen varmuuskopion palauttaminen pelitilasta poistuttaessa ratkaistiin tallentamalla luotujen materiaalien ja niiden objektien tiedot tekstitiedostoon. Myöhemmin selvisi, että vastaavan ominaisuuden saaneen kilpailevan mallinnustyökalun kehittäjä oli ratkaissut tallentamisen lähes vastaavalla tavalla.

Työn yllättävistä haasteista johtuen käyttöliittymää ei ehditty toteuttaa ja osa siveltimiin halutuista toiminnoista jäi vajavaisiksi. Tähän versioon saatiin neljä sivellintä, joilla voi piirtää katkeamatonta viivaa, maalata neliöitä ja ympyröitä, sekä maalata ympyröitä, joista puuttuu sattumanvaraisesti pikseleitä. Kaikissa siveltimissä toimii värin valinta ja läpinäkyvyyden muuttaminen. Lisäksi kuviota tuottavissa siveltimissä toimii siveltimen koon määrittäminen. Neljännessä siveltimessä puuttuvien pikselien määrää vois säädellä omalla muuttujallaan. Neljästä toteutetusta siveltimestä kaksi ehdittiin muuttaa tehokkaasti toimivaan muotoon.

Projektin lopussa tehtiin myös suoritustehoa mittaava testi, jossa 3DM-Spary-työkalun siveltimien toimintaa testattiin. Testin perusteella voitiin todeta siveltimien välillä suuria eroja ja yksi toimintamalli hylättiin kokonaan, koska se aiheutti suurella sivellin koolla Unity-pelimoottorin kaatumisen. Tämä aiheuttaa tulevaisuuden työvaiheisiin kaikkien siveltimien muuttamisen tehokkaampaan muotoon.

Barracuda Disasterille tämä insinöörityö antoi arvokasta kokemusta ja tietoa 3D-maalaustyökalujen ja Unty-liitännäisten kehittämisestä. Työn tuloksena syntynyt ensimmäinen versio 3DM-Spary-työkalusta, joka on erinomainen pohja kehittää markkinoille uusi 3D-maalaustyökalu, jolla on uniikkeja ominaisuuksia ja potentiaali kilpailla muita 3D-mallinnustyökaluja vastaan.

Lähteet

1 Techopedia. 2018. RGB Color Model (RGB). Verkkoaineisto. <https://www.techopedia.com/definition/5555/rgb-color-model-rgb>. Luettu 03.11.2018.

2 Unity Documentation. 2017. Texture compression formats for platform-specific overrides. Verkkoaineisto. <https://docs.unity3d.com/560/Documentation/Manual/class-TextureImporterOverride.html>. Luettu 03.11.2018.

3 Teenotheque. 2015. Splat Painter. Verkkoaineisto. <https://assetstore.unity.com/packages/tools/painting/splat-painter-41837>. Luettu 03.11.2018.

4 Iurii Selinnyi. Quizcanners. Verkkoaineisto. <https://www.quizcanners.com/>. Luettu 03.11.2018.

5 Iurii Selinnyi. 2017. Playtime Painter. Verkkoaineisto. <https://assetstore.unity.com/packages/tools/painting/playtime-painter-91528>. Luettu 03.11.2018.

6 Alan Zucconi. 2015. Surface shaders in Unity. Verkkoaineisto. <http://www.alanzucconi.com/2015/06/17/surface-shaders-in-unity3d/>. Luettu 03.11.2018.

7 Iurii Selinnyi. 2017. Quizcanners tools. Verkkoaineisto <https://github.com/quizcanners/Tools/blob/master/Playtime_Painter/Texture%20Scripts/ImageData.cs>. Luettu 03.11.2018.

8 Unity Documentation. 2018. Application-presistentDataPath. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html>. Luettu 03.11.2018.

9 Carlos Wilkes. 2015. Carlos Wilkes Games. Verkkoaineisto. <https://sites.google.com/site/carloswilkes>. Luettu 03.11.2018.

10 Calos Wilkes. 2015. Paint in 3D. Verkkoaineisto. <https://assetstore.unity.com/packages/tools/painting/paint-in-3d-26286>. Luettu 03.11.2018.

11 Prized Goat. 2018. Editor Paint. Verkkoaineisto. <https://connect.unity.com/p/editorpaint>. Luettu 03.11.2018.

12 Prized Goat. 2018. Editor Paint. Verkkoaineisto. <https://assetstore.unity.com/packages/tools/painting/editorpaint-107540>. Luettu 03.11.2018.

13 Unity Documentation. 2018. Terrain textures. Verkkoaineisto. <https://docs.unity3d.com/Manual/terrain-Textures.html>. Luettu 03.11.2018.

14 Unity Technologies. 2018. Polybrush (Beta). Verkkoaineisto. <https://assetstore.unity.com/packages/tools/modeling/polybrush-beta-111427>. Luettu 03.11.2018.

15 Unity Asset Store kauppapaikka. Search results for ”paint”. Verkkoaineisto. <https://assetstore.unity.com/search?q=paint&k=paint>. Luettu 03.11.2018.

16 Unity kotisivut. Public relations. Verkkoaineisto. <https://unity3d.com/public-relations>. Luettu 03.11.2018.

17 Unity kotisivut. 2018. Polybrush. Verkkoaineisto. <https://unity3d.com/unity/features/worldbuilding/polybrush>. Luettu 03.11.2018.

18 Bora. 2017. What is a vertex color. Verkkoaineisto. <https://gamedev.stackexchange.com/questions/139059/what-is-a-vertex-color>. Luettu 03.11.2018.

19 Unity Documentation. Let’s try shooting raycasts (article). Verkkoaineisto. <https://unity3d.com/learn/tutorials/projects/lets-try-assignments/lets-try-shooting-raycasts-article>. Luettu 03.11.2018.

20 Unity Documentation. 2018. Standalone. Verkkoaineisto. <https://docs.unity3d.com/Manual/Standalone.html>. Luettu 03.11.2018.

21 Robert Attard. 2014. Unity 3d Blood Test. Verkkoaineisto. <https://www.youtube.com/watch?v=LlQwhmMmT1g>. Luettu 03.11.2018.

22 Prof. Pile jr, John. 2015. Unity3D – Save your in-play transform modifications 1 of 2. Verkkoaineisto. <http://prof.johnpile.com/2015/04/06/unity3d-save-your-in-play-transform-modifications-1-of-2/>. Luettu 30.10.2018.

23 User8469759. 2018. Change texture2D format in Unity. Verkkoaineisto. <https://stackoverflow.com/questions/50020051/change-texture2d-format-in-unity>. Luettu 03.11.2018.

24 MibZ. 2013. Changing texture import settings during runtime. Verkkoaineisto. <https://answers.unity.com/questions/382545/changing-texture-import-settings-during-runtime.html>. Luettu 03.11.2018.

25 Unity Documentation. 2018. TextureImporter.SetPlatformTextureSettings. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/TextureImporter.SetPlatformTextureSettings.html>. Luettu 03.11.2018.

26 Unity Documentation. 2018. TextureImporterPlatformSettings. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/TextureImporterPlatformSettings.html>. Luettu 03.11.2018.

27 SharperSoher. Conditional asset importer. Verkkoaineisto. <https://sarpersoher.com/unity3d/conditional-asset-importer/>. 03.11.2018.

28 Booger_sugar. 2010. Editor class ”texture importer” +apply import setting question. Verkkoaineisto. <https://answers.unity.com/questions/14246/editor-class-texture-importer-apply-import-setting.html>. Luettu 03.11.2018.

29 Unity Documentation. 2018. Object.GetInsanceID. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/Object.GetInstanceID.html>. Luettu 03.11.2018.

30 Unity Documentation. 2018. AssetDatabase.AssetPathToGUID. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/AssetDatabase.AssetPathToGUID.html>. Luettu 03.11.2018.

31 TheLackey3326. How to get the local identifier in file for scene objects. 2010. Verkkoaineisto. <https://forum.unity.com/threads/how-to-get-the-local-identifier-in-file-for-scene-objects.265686/>. Luettu 03.11.2018.

32 Brackeys. 2017. Shooting with raycasts – Unity tutorial. Verkkoaineisto. <https://www.youtube.com/watch?v=THnivyG0Mvo>. Luettu 03.11.2018.

33 Unity Documentation. 2018. Material.SetColor. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/Material.SetColor.html>. Luettu 03.11.2018.

34 Unity Documentation. 2018. Texture2D.SetPixels. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/Texture2D.SetPixels.html>. Luettu 03.11.2018.

35 Samuel. 2010. SetPixels ”Texture rectangle is out of bounds”. Verkkoaineisto. <https://answers.unity.com/questions/34659/setpixels-texture-rectangle-is-out-of-bounds.html>. Luettu 03.11.2018.

36 Unity Documentation. 2018. MonoBehaviour.Update(). Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html>. Luettu 03.11.2018.

37 Kaija Häkkinen ja Jyväskylän yliopisto. 2016. Matematiikan propedeuttinen kurssi. Verkkoaineisto. <http://www.math.jyu.fi/matyl/propedeuttinen/kirja/index-55.html>. Luettu 03.11.2018.

38 Unity Documentation. 2018. Texture2D.GetRawTextureData. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/Texture2D.GetRawTexture-Data.html>. Luettu 03.11.2018.

39 Unity Documentation. 2018. ImageConversion.EncodeToPNG. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/ImageConversion.Encode-ToPNG.html>. Luettu 03.11.2018.

40 Unity Documentation. 2018. Renderer-sharedMaterial. Verkkoaineisto. <https://docs.unity3d.com/ScriptReference/Renderer-sharedMaterial.html>. Luettu 03.11.2018.

41 Unity Documentation. 2018. InspectorOptions. Verkkoaineisto. <https://docs.unity3d.com/Manual/InspectorOptions.html>. Luettu 03.11.2018.