Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Simo Ollonen

Tekstikonsoliliitännäinen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

13.5.2019

Tekijä Otsikko Sivumäärä Aika	Simo Ollonen Tekstikonsoliliitännäinen 32 sivua 13.5.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaajat	Lehtori Miikka Mäki-Uuro Toimitusjohtaja Roni Jokinen

Insinöörityön tarkoituksena oli kehittää peliyritykselle pelien ja ohjelmien kehitysprosessia auttava, ohjelman kulkua seuraava kirjaustyökalu. Sen tehtävänä on auttaa ohjelmistokehittäjää saamaan tietoa ohjelman eri tapahtumista. Kirjaustyökalulle tehtiin Unity3D-pelimoottoriin käyttöliittymä, jota kehittäjät voivat hyödyntää osana arkipäivästä pelinkehitystä. Ohjelman oli tarkoitus parantaa Unity3D-pelimoottorin alkuperäistä tekstikonsolikäyttöliittymän käyttökokemusta ja sen puutteita.

Insinöörityön aikana käytiin läpi eri tekstikonsolityökaluja ja niiden tarjoamia toimintoja. Tekstikonsoliliitännäistä varten vertailtiin asiakkaan käyttämiä pelimoottorien sisältämiä tekstikonsolityökaluja, joita käytettiin perusteena luoda parempi käyttökokemus ja kattavampi tapahtumien suodatus kehitettävään tekstikonsoliliitännäiseen.

Insinöörityöhön kehitettiin tapahtumien suodatusjärjestelmiä, jotka tarjoavat kehittäjälle paremman hallinnan tapahtumien tutkimisessa. Osana työtä kehitettiin Unity3D-pelimoottooriin graafinen käyttöliittymä, jonka avulla kehittäjät voivat hallinnoida kirjaustyökalun eri toimintoja. Graafisen käyttöliittymän suunnittelussa otettiin huomioon yksinkertaisuus ja selkeys. Sen toteutuksessa käytettiin Unity3D-pelimoottoriympäristön, käyttöliittymien rakentamiseen tarkoitettua, IMGUI-järjestelmää.

Insinöörityön tuloksena syntyi käytettävä tekstikonsoliliitännäinen. Sitä ei ole toistaiseksi päästy hyödyntämään asiakkaan projekteissa niiden luonteen vuoksi. Insinöörityön kehityksessä tutustuttiin MVC-ohjelmistoarkkitehtuurimalliin ja sen hyötyihin ohjelman logiikan ja käyttöliittymän erittelyssä. Kehityksen tuloksena saatiin myös ymmärrystä IMGUI-järjestelmän hyödyistä nopeassa käyttöliittymien rakentamisessa ja haitoista monimutkaisten käyttöliittymien toteutuksissa.

Avainsanat	Unity3D, loki, lokitiedosto, käyttöliittymä, MVC, IMGUI

Author Title Number of Pages Date	Simo Ollonen Console Plugin 32 pages 13 May 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructors	Miikka Mäki-Uuro, Senior Lecturer Roni Jokinen, Chief Executive Officer

The goal of the thesis was to create a log analysis tool for a game company LunarByte to assist game developers in their daily tasks.

The log analysis tool was created as a plugin to Unity3D game engine and it aims to improve user experience of a similar existing solution in the game engine. The purpose of the tool is to assist game developers to get information from different events occurring in the developed game or program which can vary from generic information to errors.

As a part of the thesis, different solutions to log analysis in other game engines are compared to create an overall understanding of the features that are included in the produced tool. The thesis covers parts of the design and production process of the log analysis tool and its user interface in the Unity3D game engine.

Keywords	Unity3D, Logging, Log File, UI, MVC, IMGUI

Sisällys

Lyhenteet

API	Application Programming Interface eli ohjelmointirajapinta on määritelmä ohjelman kyvystä kommunikoida toisen ohjelman kanssa.
CLI	Command line interface eli komentoliittymä on tapa kommunikoida ohjelman ja käyttäjän välillä suoritettavilla komennoilla.
GUI	Graphical User Interface eli graafinen käyttöliittymä on käyttöliittymä, jossa kommunikointi tapahtuu graafisten elementtien avulla.
IDE	Integrated development environment eli ohjelmointiympäristö on ohjelma, jolla ohjelmoija luo ohjelmia.
IMGUI	Immediate Mode Graphical User Interface on koodipohjainen käyttöliittymien rakennusjärjestelmä.
MVC	Model-View-Controller on ohjelmistoarkkitehtuuri, jota käytetään käyttöliittymän ja muun ohjelman logiikan erottelussa.

1 Johdanto

Insinöörityön tarkoituksena oli kehittää tekstikonsolityökalu LunarByte Oy -nimiselle peliyritykselle. LunarByte on perustettu vuonna 2017, ja se on julkaissut yhden pelin ja kehittää pelejä, joita ei ole vielä julkaistu. Nykyisin yritys keskittyy enemmän pelialan konsultointiin omien pelien julkaisujen sijasta.

Tekstikonsoli on ohjelmien kehityksessä käytettävä työkalu, joka yhdistää ohjelman syötteiden kirjaamisen ja niiden näyttämisen käyttöliittymässä. Tekstikonsolia käytetään ohjelmien kehityksessä niin, että ohjelman kehityksen aikana ja ohjelman suorittamisen aikana voidaan kerätä erilaisia tapahtumatietoja ja virheilmoituksia.

Insinöörityön tavoitteena oli luoda ohjelmien kehityksessä käytettävä tekstikonsolityökalu, jonka tehtävänä on auttaa ohjelmistokehittäjää saamaan tietoa ohjelman eri tapahtumista. Työkalulle oli tarkoitus tehdä Unity3D-pelimoottoriin käyttöliittymä, jota kehittäjät voivat hyödyntää osana pelinkehitystä. Työkalun käyttöliittymän oli tarkoitus parantaa Unity3D-pelimoottori editorin vastaavanlaisen työkalun käyttökokemusta ja sen puutteita.

Työssä perehdytään eri tekstikonsoleiden tapoihin ilmaista ohjelman tapahtumien viestintää käyttäjälle ja käsitellään ominaisuuksia, joita haluttiin hyödyntää ja kehittää kehitettävässä tekstikonsolityökalussa. Lisäksi työssä käydään läpi tekstikonsolin suunnittelua ja kehitystä ja pohditaan kehitystyön tuloksia.

2 Viestien kirjaaminen eri ympäristöissä

Isojen tietokoneohjelmien kehityksen aikana luodaan paljon uusia ominaisuuksia. Ohjelmien kehittämiseen liittyy tyypillisesti jonkin verran virheenjäljitystä. Yksi virheenjäljityksen tavoista on lokitiedoston analysointi, jota käsitellään tässä työssä. Muita virhejäljitysmenetelmiä ovat mm.

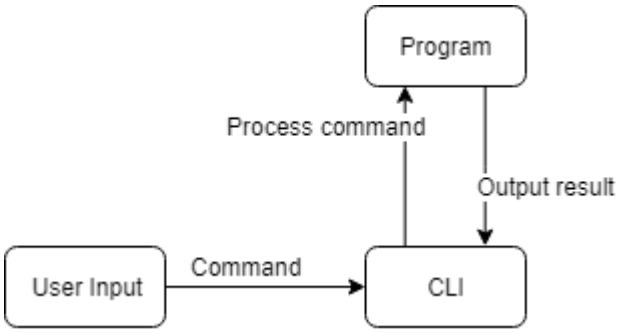
- yksikkötestaaminen (engl. unit testing)
- integraatiotestit (engl. integration testing)
- interaktiivinen virheenjäljitys.

Kirjaaminen (engl. logging) on tietotekniikassa käytetty määritelmä automaattiselle ohjelman viestinnän seuraamiselle. [1.] Kirjaamisessa ohjelmasta tulevat tulosteet ohjataan tiedostoon, jota kutsutaan lokitiedostoksi tai lokiksi (engl. log file). Lokitiedostossa olevia tulosteita kutsutaan lokimerkinnöiksi. [2.]

Kirjaamisella on liiketoiminnassa monia eri sovellutuksia ja käyttötarkoituksia. Yksi sovellutuksista on tietoturvaan liittyvät tapahtumat, jotka voivat auttaa ehkäisemään mm. tietoturva- ja operationaalisia ongelmia. [2.] Kirjaamista käytetään myös ohjelman suorituksen seurantaan ja vianetsintään. Kun ohjelma kirjaa lokitiedostoon sen eri tapahtumia ja virheitä, tämä auttaa jälkeenpäin ohjelman suorituksen analysoimisessa ja normaalin suorituksen varmistamisessa.

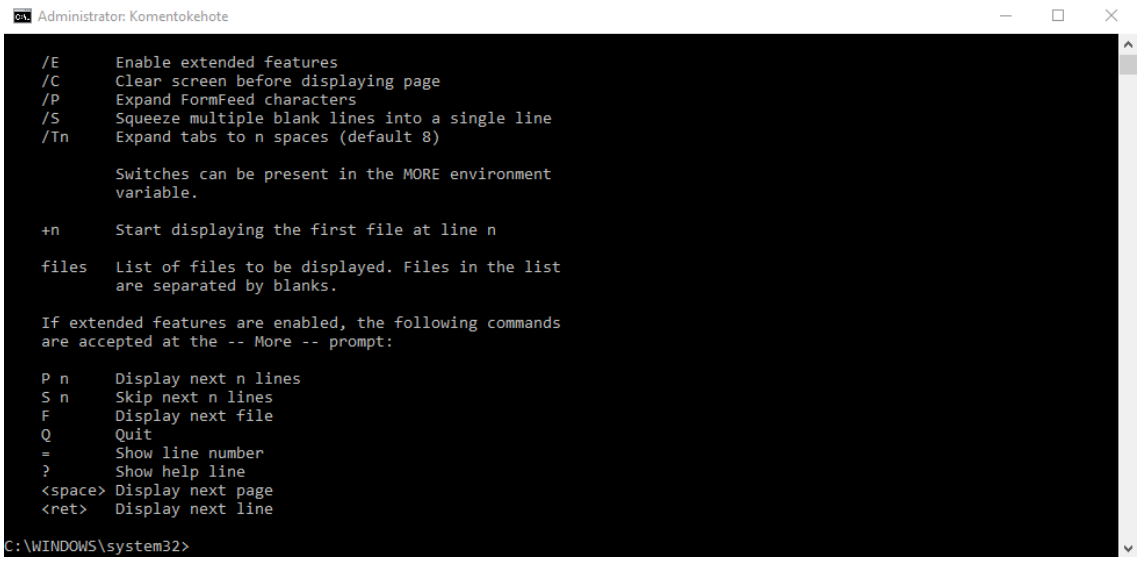
2.1 Komentoliittymä

Komentoliittymä eli CLI (engl. Command Line Interface) on tietokoneissa tekstipohjainen käyttöliittymä, joka ottaa vastaan syötteitä ja tulostaa tuloksia käyttäjän antamien syötteiden pohjalta erilaisten komentojen avulla. Komennot voivat olla suoritettavia ohjelmapätkiä, joissa käsitellään annettu syöte. Kun ohjelma on suoritettu, siitä saadut tulosteet tulostetaan komentoliittymässä. Tuloste riippuu ohjelmasta, joka suorittaa komennon. Kaikki suoritettavat ohjelmat eivät välttämättä luo tulosteita, jolloin annetulle syötteelle ei tule ohjelmalta tulostetta. Kuvassa 1 havainnollistetaan syötteen ja tulostuksen käsittelyä komentoliittymässä.



Kuva 1. Komentoliittymän syötteen ja tulostuksen käsittely [3].

1960-luvulla komentoliittymät olivat ainoa tapa kommunikoida käyttäjän ja käyttöjärjestelmän välillä. Komentoliittymiä käytettiin, ja käytetään vielä nykyisinkin, hallinnoimaan ohjelmien asentamista ja konfiguroimista tapauksissa, joissa graafinen käyttöliittymä ei tarjoa tiettyjä toimintoja tai sitä ei ole. Nykyisin graafisia käyttöliittymiä hyödynnetään enemmän kuin tekstipohjaista komentoliittymää. [3.] Kuvassa 2 on Windows 10:n tekstipohjainen komentoliittymä.

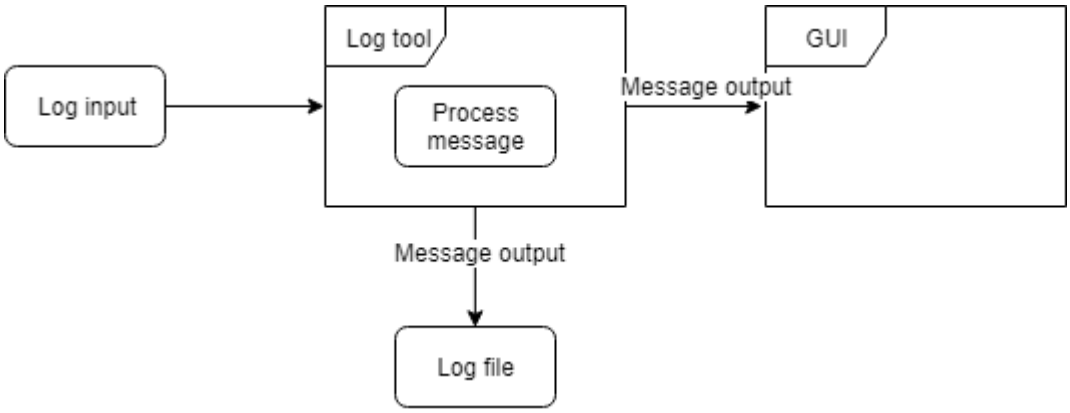


Kuva 2. Windows 10 -komentoliittymä [4].

2.2 Tekstikonsoli

Tekstikonsoli on työkalu, joka yhdistää kirjaamisen ja komentoliittymän ominaisuuksia. Tekstikonsolista käytetään eri ympäristöissä erilaisia nimiä, mutta niitä yhdistävät yleensä samat toiminnot. Toimintoihin sisältyvät mm. tulosteen tallentaminen ja näyttäminen. Monesti tekstikonsolilla on myös graafinen käyttöliittymä, joka sisältää toimintoja lokianalyysin helpottamiseen.

Viestit ovat ohjelman toimintaa ilmaisevia tulosteita, jotka kertovat ohjelman eri vaiheista. Viestit luodaan ohjelman lähdekoodissa määritellyistä syötteistä, joiden avulla tekstikonsoli luo viestin tulosteen. Tekstikonsoli voi ohjata syötteen sen määrittelemään paikkaan, kuten lokitiedostoon tai käyttöliittymään. Kuvassa 3 on tekstikonsolin toiminnan yleiskuva, jossa näkyvät syötteen ja tulosteen kulku kirjaustyökalun kautta.



Kuva 3. Tavanomainen tekstikonsolin toiminta.

Tekstikonsoli eroaa komentoliittymästä siten, että se ei yleensä tarjoa erillistä tekstisyötekenttää komennoille. Tekstikonsolille annetut syötteet ohjataan kirjaustyökalun eri kohteisiin, joista yksi on graafinen käyttöliittymä, jossa syötteistä luodut tulosteet näytetään. Tekstikonsoli sisältää usein myös lokin hallinnan, toisin kuin komentoliittymä.

2.3 Ohjelmointiympäristö

Ohjelmointiympäristö, tai IDE (engl. integrated development environment), on ohjelma, joka tarjoaa kehitysympäristön ja työkaluja ohjelmien kehittämiseen. Insinöörityössä käytettiin tekstikonsolin luomiseen Visual Studio -ohjelmointiympäristöä, jota asiakas käyttää myös jokapäiväisessä pelienkehityksessä.

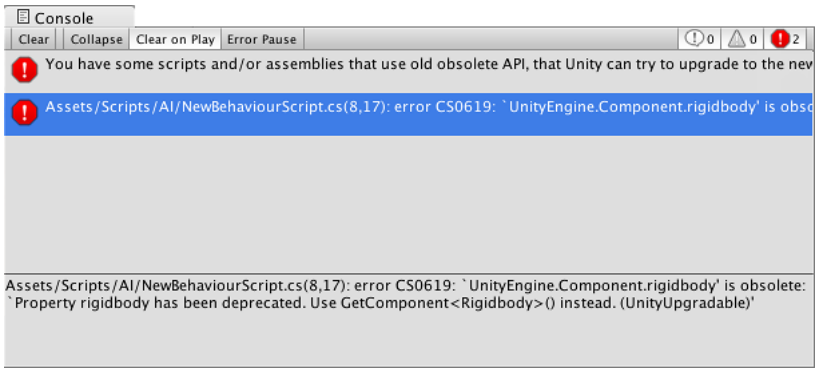
2.4 Unity3D-konsoli

Unity3D on pelien kehitykseen tarkoitettu pelimoottori. Pelimoottorit yhdistävät pelinkehityksessä yleisesti käytettyjä toiminnallisuuksia, kuten fysiikkamallinnus, renderöinti ja syötteenkäsittely. Unity3D-editori sisältää reaaliaikaisen kehitysympäristön (myöh. editori), jonka vahvuuksina ovat pelien nopea kehitys, joustavuus ja laajennettavuus. [5; 6.]

Unity sisältää pelienkehitykseen oman tekstikonsoli-ikkunan (console), jonka ominaisuuksiin kuuluu:

- viestien näyttö listassa
- vakavuustasojen suodatus
- kutsupinon näyttäminen
- pelimoottorin pysäyttäminen virheen tapahtuessa
- lokitiedostoon kirjaamisen.

Kuvassa 4 näkyy Unity3D-editorin Console-ikkuna, jonka kautta voidaan tarkastella viestejä, jotka ovat syötetty Unity3D-pelimoottorin tarjoaman Debug-ohjelmointirajapinnan (Application Programming Interface) kautta. [7.]



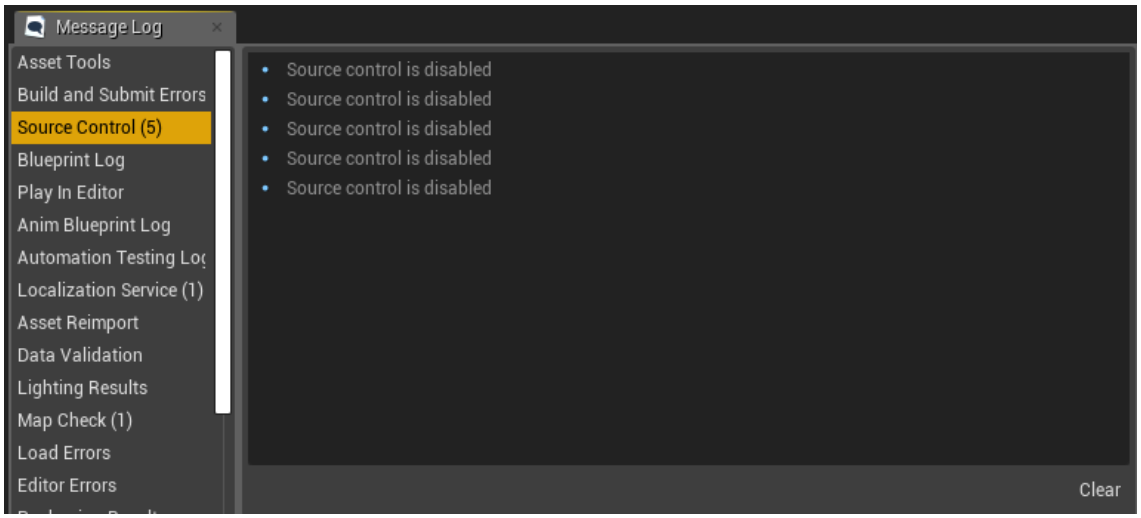
Kuva 4. Unity3D:n konsoli-ikkuna [7.]

2.5 Unreal Engine 4 -ohjelmistopaketti

Unreal Engine 4 on reaaliaikaisen teknologian kehitystyökalujen ohjelmistopaketti. Sitä käytetään yritysohjelmien, elokuvien ja pelien tuotannossa. Unreal Engine 4:ssä on paljon työkaluja pelinkehitykseen, kuten erilaiset työkalut fotorealismin ja uskottavien animaatioiden luontiin. Työkalujen joukossa on myös lokianalyysiin liittyviä työkaluja, kuten profiloija ja tekstikonsolityökaluja. [8.]

Unreal Engine 4:ssä on kaksi lokianalyysityökalua, Output Log ja Message Log. Output Log toimii enemmän tavanomaisen komentoliittymän tapaan, kun taas Message Log toimii viestien näyttämisessä ryhmitellysti.

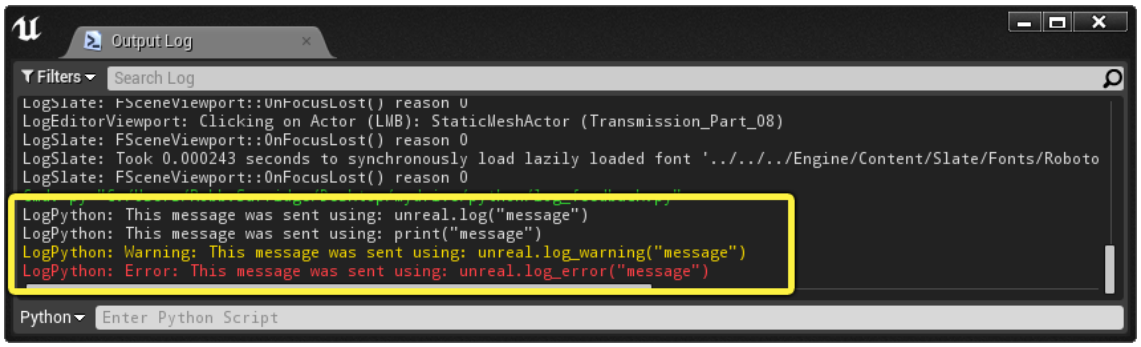
Unreal Engine 4:n Message Log -työkalussa on hyödynnetty kategorioita, joiden tarkoitus on ryhmitellä viestejä kontekstuaalisesti. Kategoria kertoo tyypillisesti viestin lähettämästä ympäristöstä ja lähteestä. Kuvassa 5 on Unreal Engine 4:n Message Log -työkalu, jossa on eriteltynä eri kategorioita ja niihin liittyviä viestejä.



Kuva 5. Unreal Engine 4 Message Log -työkalu [9].

Output Log

Output Log on Unreal Engine 4:n lokianalyysityökalu. Se näyttää listassa muotoiltuja tulosteita, jotka kirjataan myös erilliseen lokitiedostoon. Output Log -työkaluun kirjataan ohjelman tapahtumia Unreal Engine 4:n tarjoaman rajapinnan kautta, mikä mahdollistaa tulosteen muotoilun, kuten värin valinnan ja kategorian valinnan. Output Log -työkalu pystyy suodattamaan tulosteita niiden kategorioiden perusteella ja tekstin sisällön perusteella. Se sisältää myös komentoliittymän, jonka avulla kehittäjä voi syöttää komentoja Unreal Engine 4:n editorille. Kuvassa 6 on kuvankaappaus Output Log -työkalusta.



Kuva 6. Unreal Engine 4 Output Log -työkalu [10].

3 Tekstikonsolin suunnittelu

Insinöörityössä luotiin helppokäyttöinen ja selkeä tekstikonsoli Unity3D-pelimoottoriin. Suunnitteluaikana keskityttiin ongelmiin, joita Unity3D-editorin tarjoamassa tekstikonsolissa oli, ja ominaisuuksiin, joita haluttiin lisätä helpottamaan arkipäiväistä työskentelyä ohjelmien kehittämisessä. Kehitettävän tekstikonsolin työnimeksi annettiin LunarConsole, jota käytettiin työkaluun viitattaessa.

Työn aloittamisessa toimeksiantajan kanssa suunniteltiin työkalun vaadittuja ominaisuuksia ja käyttöliittymän ulkoasua. Toimeksiantajan projekteissa käytettyjen pelimoottoreiden tarjoamia tekstikonsolityökaluja käytettiin vertailussa mukana.

Yrityksen kehittämissä projekteissa huomattiin kirjaukseen liittyviä ongelmia ja puutteita pelinkehityksessä. Yritys käytti eräässä Unity3D-pelimoottorilla tehdyssä projektissa analytiikkatyökalua, jonka tarkoituksena oli lähettää tietoja pelaajan tekemistä toiminnoista peleissä. Jokainen analytiikkakutsu haluttiin kirjata varmistukseksi siitä, että tiedot muodostettiin oikein ja lähetettiin eteenpäin. Analytiikan lisääntyessä Unity3D:n Console-ikkuna täyttyi analytiikkakutsuista ja muut viestit olivat niiden seurauksena vaikeasti löydettävissä. Unity3D:n Console-ikkunasta tuli viestien määrän vuoksi vaikeasti hyödynnettävä lokianalyysityökalu, ja sitä käytettiin vain virheilmoituksien seurantaan.

Tekstikonsolissa haluttiin painottaa mahdollisimman paljon viestien löytämiseen liittyvien ongelmien vähentämistä. Viestejä haluttiin näyttää kehittäjän käsiteltävän tehtävän osalta, ja niiden täytyi olla suodatettavissa joustavasti, jotta halutun tiedon tai virheen etsintään kului mahdollisimman vähän aikaa.

3.1 Viestit

Viesti on tekstikonsolissa käsite, joka kuvaa kirjaamisesta luotua tietoa. Viesti sisältää ohjelmassa kehittäjän määrittelemien syötteiden lisäksi myös muuta dataa, jonka avulla saadaan yksityiskohtaisempaa tietoa viestin kirjaustapahtumasta. Taulukko 1 kuvaa viestin datan rakennetta. Viestin dataan on sisälletty myös metadata. Se ei ole viestin sisältöön liittyvää tietoa, vaan tietoa, joka kuvaa viestin ympäristöstä saatua tietoa.

Taulukko 1. Viestin datan rakenne.

Viestin tietue	**Selite**
Teksti	Kehittäjän syöttämä syöte
Vakavuustaso	Viestille annettu vakavuustaso
Kategoria	Viestin dynaamisesti määritetty kategoria
Metadata	Tietoa viestistä, kuten aika ja kutsuympäristö

Jokainen viesti sisältää metadataa, jota hyödynnetään viestin lisätietona. Kirjaustyökalu luo jokaiselle luodulle viestille metadatatietueen, joka voidaan kirjata lokitiedostoon ja/tai näyttää käyttöliittymässä. Metadatan tarkoitus on erotella ja sisältää viestiin liittyvää tietoa, joka ei ole viestin sisällölle olennaista, vaan viestiin liittyviä käsitteitä. Taulukossa 2 on esitetty kirjaustyökalun käyttämä metadatan rakenne ja data.

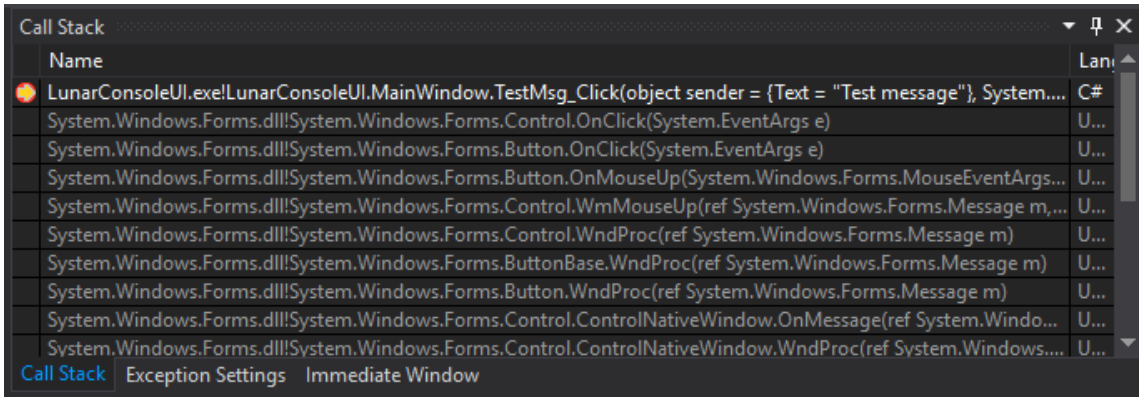
Taulukko 2. Metadatan rakenne, jota tekstikonsolissa hyödynnetään.

Metadata tietue	**Selite**
Kutsuja	Viestin luoneen luokan instanssi
Kutsupino	Ohjelman kutsupino ohjelman viestin luontihetkellä
Aikaleima	Päivämäärä ja aika, jolloin viesti luotiin
Staattinen kategoria	Viestiin liitetty staattinen kategoria

Kutsupino

Tietokoneohjelmat ovat yhdistelmä dataa ja funktioita, jotka suorittavat erilaisia toimintoja. Funktiot ovat toimintoja, joita tietokoneen prosessori suorittaa muokatakseen ohjelman tilaa. Monimutkaisissa ohjelmissa on tyypillistä, että funktiot kutsuvat muita funktioita muodostamalla erilaisia toimintoja. Tämäntyyppisestä funktioiden kutsujen rakenteesta muodostuu näennäinen funktiokutsujen ketju, jota kutsutaan kutsupinoksi (engl. Call stack). [11.]

Kutsupino kuvaa ohjelmassa aktiivisten funktiokutsujen pinoa. Funktiokutsun yhteydessä funktio lisätään pinoon, ja funktion palatessa funktio poistetaan kyseisestä pinosta. Monimutkaisissa ohjelmissa, kuten peleissä, kutsupinot voivat olla todella pitkiä. Kuvassa 7 näkyy ohjelman kutsupino tietyllä ajan hetkellä. Siinä näkyvät kutsupinon viimeisimpään funktioon johtaneet funktiokutsut. [2.2 – 11.]



Kuva 7. Ohjelmasta otettu kutsupino tietyllä ajan hetkellä Visual Studio -ohjelmointiympäristössä [12.]

Kutsupinojen suunniteltu tehtävä kirjaustyökaluissa on kertoa viestin luontiin johtaneen ohjelman lähde. Työkalun haluttiin näyttävän jokaisen viestin lähde, jotta lähdekoodin ja tiedoston löytäminen olisi helpompaa. Kutsupinon avulla viesteistä voidaan varmistaa ohjelman haluttu toiminnallisuus tai vianetsinnässä seurata kutsupinoa mahdollisten virheellisten viestien ilmaantuessa.

Kirjaaminen

Kirjaustyökalun keskeinen ominaisuus on viestien kirjaaminen lokitiedostoon. Tiedostokirjaus päätettiin luoda yhteen lokitiedostoon, joka luodaan käyttäjän omalle koneelle työkalun käytön yhteydessä. Kirjaustyökalulla on ennalta määritetty polku lokitiedostoon, johon viestit menevät oletuksena. Kirjaustyökaluun haluttiin myös tuki päättää tiedostopolku, johon viestit tallennetaan, riippuen käyttöympäristöstä.

Lokitiedostoa ei yleensä tarvitse käyttää, mutta se auttaa vianetsinnässä jälkeenpäin. Viestien näyttäminen käyttöliittymässä on täysin kehittäjän hallinnassa, mikä tarkoittaa,

että viestit eivät ole käyttöliittymässä välttämättä näkyvissä koko aikaa. Tämän seurauksena, jos kehittäjä haluaa ohjelman suorituksesta tietoa jälkeenpäin, se ei ole välttämättä saatavilla käyttöliittymässä. Tätä varten kehittäjä voi etsiä viestejä jälkeenpäin lokitiedostosta, johon jokainen viesti kirjataan.

Lokitiedostot mahdollistavat myös vianetsinnän jakamisen muille kehittäjille. Jos ohjelmassa tapahtuu normaalista toiminnasta poikkeavia toimintoja, voidaan nämä ongelmat löytää lokitiedostosta. Lokitiedosto voidaan lähettää ohjelman kehittäjälle, joka tietää ohjelman suorituksesta paremmin, ja hän voi etsiä epäsäännöllistä toimintaa, kuten virheitä lokitiedostosta, ja korjata niitä.

3.2 Suodatus

Tässä luvussa käsitellään tekstikonsoliin suunniteltuja suodatustoimintoja. Suodattimien avulla tekstikonsolista voidaan suodattaa viestejä eri periaatteiden perusteella. Suodattimilla kehittäjä pystyy hallinnoimaan haluttujen viestien näyttämistä.

Kategoriat

Kategorioiden tarkoitus oli antaa käyttäjälle kontekstuaalista tietoa viestin syöttäneestä ympäristöstä ja auttaa kehittäjää ymmärtämään tulostetun viestin lähde. Viestit liittyvät aina johonkin ohjelman osaan, kuten fysiikkamoottoriin, pelin alustukseen tai kolmannen osapuolen kirjastoihin.

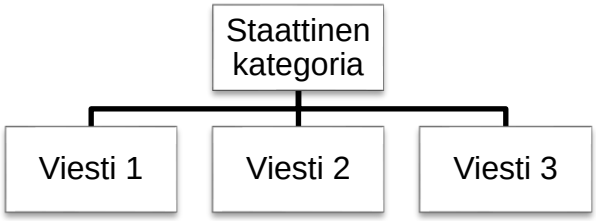
Kirjaustyökaluun suunniteltiin kaksi kategoriatyyppiä, staattinen ja dynaaminen kategoria. Kategoriatyyppien suurin eroavaisuus on niiden määritteleminen ja käyttäminen. Staattisten kategorioiden tehtävä on määritellä viesteille ympäristö, jossa viestit ovat sidottuja kyseiseen kategoriaan. Dynaamiset kategoriat ovat puolestaan viestisidonnaisia ja määritellään viestien luonnin yhteydessä.

Eri kategoriatyyppeihin päädyttiin pohtimalla, miten kategorioita käytettäisiin ohjelman kehityksessä. Kategorian oli tarkoitus määritellä viestin ympäristö, jossa luodut viestit ryhmittyisivät tiettyyn kontekstiin automaattisesti. Automatisoitu ryhmittely nähtiin isona

hyötynä, mutta kehittäjälle haluttiin myös tarjota mahdollisuus sitoa viesti kategoriaan hallitusti.

Staattiset kategoriat

Staattiset kategoriat ovat viestiin sidottua metadataa. Staattisten kategorioiden ja viestien välinen sidonnaisuus määritellään viestin luoneen ympäristön perusteella. Ympäristöt voivat olla esimerkiksi ohjelmassa esiintyviä koodiluokkia, joissa viestejä luodaan luokan jäsenfunktioiden avulla. Kuvassa 8 on havainnollistettu viestien ja staattisten kategorioiden suhde. Jokaisella viestillä voi olla vain yksi staattinen kategoria, mutta yhdellä kategorialla voi olla useampai viestejä.



Kuva 8. Staattisen kategorian ja viestien suhde.

Dynaamiset kategoriat

Dynaamiset kategoriat ovat kehittäjän vapaasti määrittelemiä kategorioita viestien luonnin yhteydessä. Dynaamisten kategorioiden tarkoituksena on lisätä kontekstuaalisia suodatusmahdollisuuksia viesteihin staattisten kategorioiden lisäksi. Dynaamiset kategoriat tarjoavat kehittäjälle hallittavuutta isojen kontekstien viestien järjestelyyn.

Esimerkiksi peleissä pelin latausta käsittelevälle järjestelmälle voidaan määritellä yksi staattinen kategoria, mutta pelin kehityksen edetessä latausjärjestelmän käsiteltävä sisältö kasvaa ja sen seurauksen pelin lataukseen liittyvät viestit lisääntyä. Tässä tapauksessa dynaamisten kategorioiden käyttö on hyödyllistä lisäämään pelin lataukseen liittyvien viestien kategorisointia, sillä latausta käsittelevä järjestelmä voi hallita paljon toisistaan riippumattomia asioita.

Vakavuustasot

Vakavuustasot (engl. Verbosity) kuvaavat kirjattujen viestien vakavuutta. Vakavuustasoilla pyritään kuvamaan viestin luonnetta ja sen tarkoitusta viestin ympäristössä. Vakavuustasoja suunniteltiin käytettäväksi seitsemän, ja ne näkyvät taulukossa 3. Taulukko 3 kuvaa myös kirjaustyökalun ohjeistettua käyttöä vakavuustasoille viestien yhteydessä.

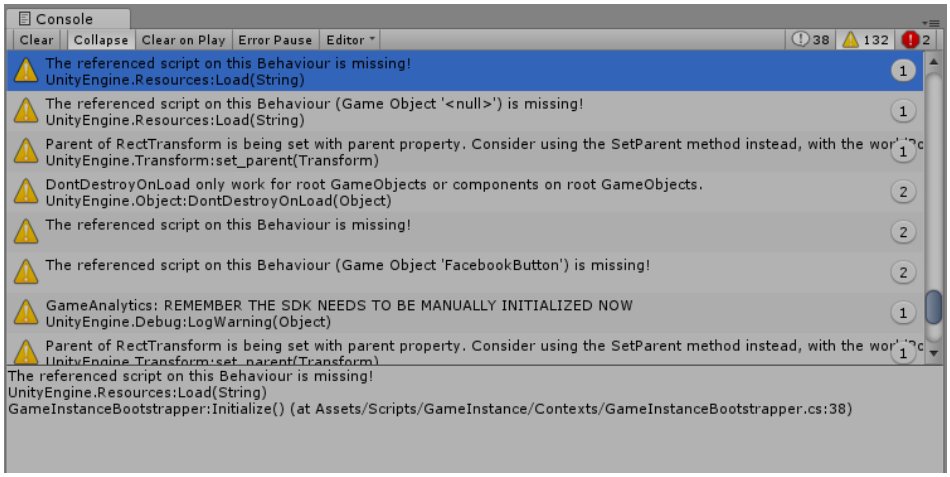
Taulukko 3. Vakavuustasojen käyttö

Vakavuus-taso	Kuvaus
Fatal	Ohjelmaa kohdanneet virheviestit, jotka lakkauttavat ohjelman toiminnan.
Error	Virhe. Ohjelmassa on tapahtunut jokin virhe ja ohjelma ei välttämättä toimi normaalisti.
Warning	Varoitus. Ohjelma saattaa toimia, mutta sen toiminnassa saattaa olla puutteita.
Log	Normaali taso. Oletustaso kaikille viesteille, joille ei ole erikseen määritelty vakavuutta.
Verbose	Ohjelman ajautumista kuvaavat viestit, kuten järjestelmien alustus ja sulkeminen.
VeryVerbose	Ohjelmaa vapaammin kuvaavat viestit, kuten järjestelmän jatkuvaa syötettä kuvaavat viestit.
Debug	Virheen etsintään tarkoitetut ohjelman koodiin sijoitetut viestit.

Vakavuustasojen määrässä haluttiin suunnitteluvaiheessa ottaa huomioon joustavuus. Unityn tarjoamassa Console-työkalussa mahdollisuutena on valita vain kolme vakavuustasoa. [7.] Yritys oli aikaisemmissa projekteissa Unityn kanssa törmännyt vakavuustasojen vähyyden ongelmaan ja tarpeeseen hyödyntää viestejä toiminnoissa, joissa viestisyötteitä saatettiin käyttää suunnattomasti.

Kehityksen aikana voi ilmaantua satunnaisia varoituksia ja virheitä, jotka näytetään samassa tulostusikkunassa. Kuvassa 9 havainnollistetaan viestien määrää ja niihin

sidottua vakavuustasoa. Kuvan esimerkissä Console-ikkunassa suodatetaan vain viestit, joiden vakavuustaso vastaa varoitusta.



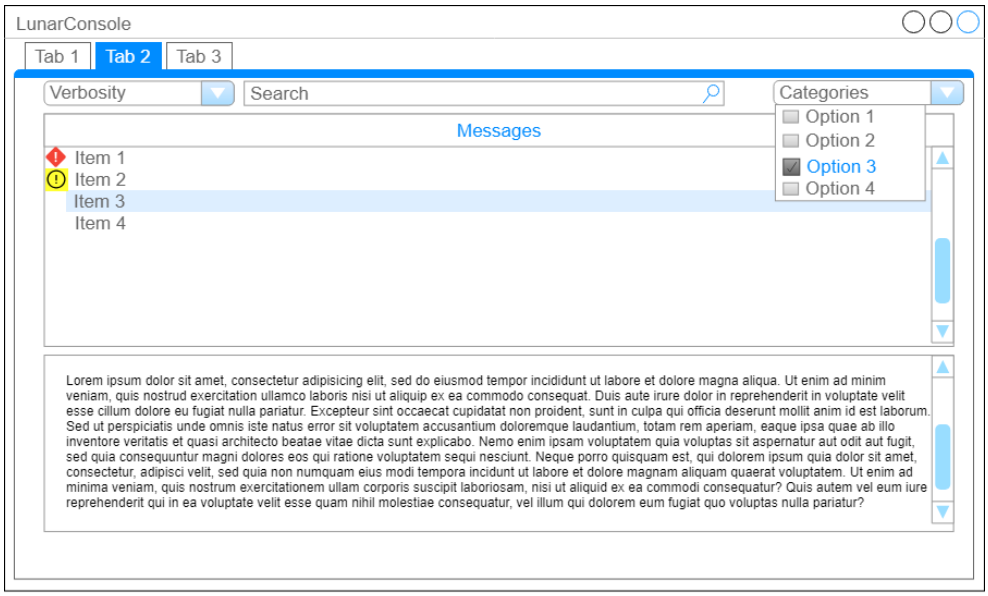
Kuva 9. Unity3D-konsoli pelin suorituksen aikana. Oikeassa yläreunassa näkyvät Unity3D-konsolin suodatustasot ja niihin kuuluvat viestien määrät [6].

Etsintä

Viestien etsintää haluttiin helpottaa sisällönhaulla. Sisällönhaun suunniteltiin tapahtuvan tekstipohjaisella hakupalkilla, jolla kehittäjä voisi suodattaa viestejä niiden sisällön perusteella. Koska kirjaustyökalu ei sisällä itsessään käyttöliittymää, tulisi etsintä toimimaan käyttöliittymätasolla. Suodatuksen tuli toimia niin, että jokainen viesti, joka ei sisällä annettua merkkijonoa hakupalkilla, suodatetaan pois.

3.3 Käyttöliittymä

LunarConsolen käyttöliittymästä oli tavoitteena saada selkeä ja yksinkertainen näkymä. Yksi käyttöliittymän haasteista oli rajoitettu ikkunan koko. Käyttöliittymän täytyi olla toiminnallinen pienessä ikkunakoossa, jotta Unity3D-editorin muut kehitystyökalut olivat näkyvissä samanaikaisesti. Ikkunakoon rajoitetun tilan seurauksena monet käyttöliittymäelementit pyrittiin pitämään mahdollisimman pienikokoisina, mutta selkeinä, kuten kuvassa 10 on esitetty.



Kuva 10. Alustava suunnitelma kehitetystä LunarConsole-tekstikonsolista.

Viestimerkintä

Viestimerkintä on graafinen elementti, joka esittää kirjaustyökalusta tulleen viestin sisältöä. Viestimerkinnät esitetään viestilistassa riveinä, ja ne koostuvat kahdesta osasta, kuvake ja viestimerkinnän sisältö. Jokaisella kirjaustyökalusta näytettävällä viestillä on oma viestimerkintäelementti.

Kuvake on visuaalinen vihje viestin vakavuustasosta. Sen avulla kehittäjä erottaa viestit vakavuustasoiltaan paremmin toisistaan. Jokaiselle vakavuustasolle näytetään kuvake, mutta vain virheillä ja varoituksilla on omat kuvakkeet.

Viestimerkinnän sisältö koostuu viestin tiedosta. Sisällössä näytetään viestin ennalta määriteltyjä tietoja, jotka ovat yleisesti merkityksellisiä kehittäjälle. Sisältö on myös ohjelmallisesti konfiguroitavissa, jos oletussisältö ei ole tyydyttävä kehittäjälle. Oletuksena viestimerkinnän sisältö koostuu seuraavista tiedoista:

- aikaleima
- kategoria

- vakavuustaso
- viestin teksti.

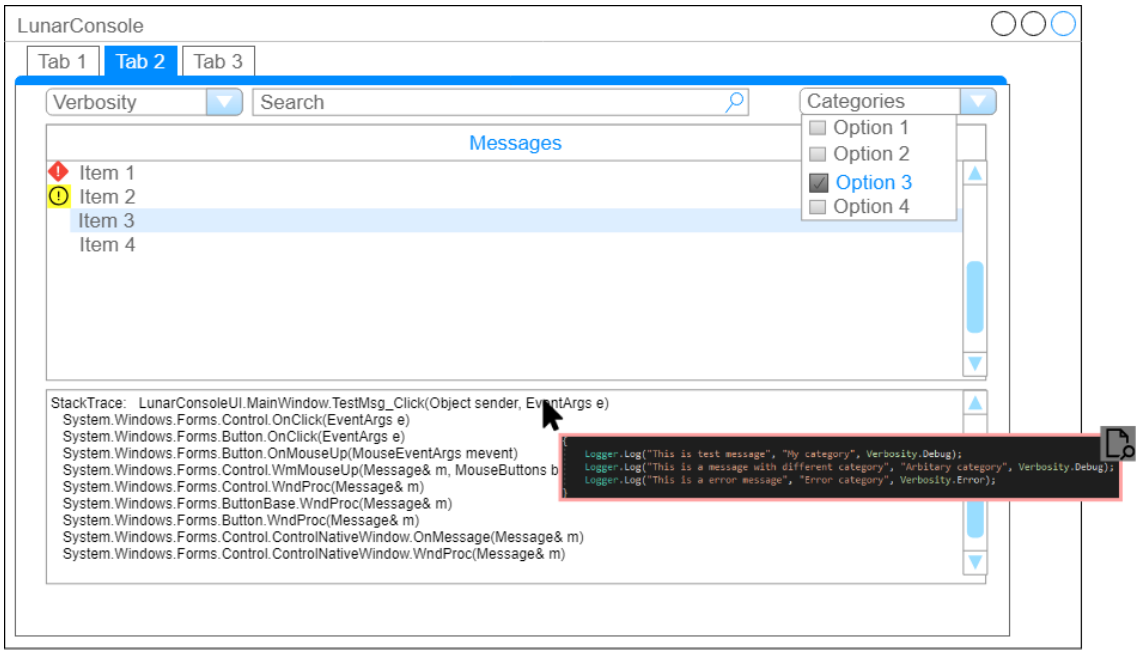
Koodinavigointi

Koodinavigointi on ominaisuus, jolla kehittäjä voisi siirtyä tekstikonsolin käyttöliittymästä ohjelmointiympäristöön viestin lähteen perusteella. Viestin kutsupinon avulla voitaisiin selvittää lähdekoodin tiedosto ja avata tiedosto ohjelmointiympäristössä. Kutsupinosta voitaisiin myös selvittää kutsupinossa esiintyvien muiden funktioiden lähdekoodit ja vastaavasti siirtyä ohjelmointiympäristöön lähdekoodin muokkaamiseen, jos lähdekoodit ovat saatavilla.

Suunnitteluvaiheessa haasteeksi osoittautui eri ohjelmointiympäristöjen valitseminen ja avaaminen ohjelmointiympäristössä. Kehittäjät saattoivat myös käyttää eri ohjelmointiympäristöjä, joille pitäisi luoda vastaavanlainen tuki. Ominaisuus päätettiin mainittujen syiden takia toistaiseksi jättää kehittämättä tekstikonsoliin.

Koodin esikatselu

Käyttöliittymään suunniteltiin ominaisuudeksi kutsupinossa esiintyvien funktioiden esikatselua. Esikatseluikkunan suunniteltiin näkyvän, kun hiiri viedään funktion päälle kutsupinolistassa ja painetaan näppäimistön CTRL-näppäintä pohjaan. Esikatseluikkunassa kehittäjä pystyisi vierittämään kooditiedoston sisältöä nopeasti samassa käyttöliittymässä ilman siirtymistä koodin ohjelmointiympäristöön. Kuvassa 11 esitetään koodin esikatseluominaisuutta.



Kuva 11. Luonnostelu koodin esittämisestä esikatseluikkunassa, kun hiiri on kutsupinon funktion päällä.

Kutsupinon muuttujat

Kutsupinon muuttujat ovat kutsupinossa esiintyvien funktiokutsujen yhteydessä annettuja arvoja. Toimeksiantajan kanssa mietittiin, kuinka kutsupinoja voitaisiin hyödyntää paremmin vianetsinnässä. Toimeksiantajan käyttämät tekstikonsolit, kuten Unity3D-Console ja Unreal Engine 4:n Log-työkalut, näyttivät funktiokutsujen määrittelyn, mutta eivät funktiokutsujen yhteydessä syötettyjä parametrien arvoja. Toiminnallisuudesta nähtiin paljon hyötyä vianetsintään käytetyn ajan lyhentämisessä, jolloin erillistä debuggeria voitaisiin tarvita vähemmän virheiden etsintään.

Suunnittelun aikana kävi kuitenkin ilmi, että työkalusta pitäisi luoda eräänlainen debuggeri, jotta ohjelmassa ajettavien funktioiden arvoja voitaisiin automatisoidusti tarkkailla. Toiminnallisuuden katsottiin lisäävään työkalun laajuutta ja monimutkaisuutta liikaa, minkä takia ominaisuus jätettiin toistaiseksi kehityksestä pois.

4 Tekstikonsolin toteutus

Kirjaustyökalu kehitettiin C#-ohjelmointikielellä ja .NET Foundationin, aikaisemmin Microsoftin, kehittämän .NET-ohjelmistokomponenttikirjaston avulla. .NET on ilmainen avoimen lähdekoodin kirjasto, joka toimii monella eri ohjelmointikielellä, kuten C#, F# ja Visual Basic. [13.]

C#-ohjelmointikieltä käytetään Unity3D-pelimoottorissa pelien ohjelmointiin ja editorin käyttöliittymän laajentamiseen. Kirjaustyökalun ja sen käyttöliittymän kehittäminen samalla ohjelmointikielellä vähensi kielien ja kirjastojen eroavaisuutta ja samalla niiden yhdistämiseen vaadittavia rajapintamuutoksia.

4.1 Käsitteitä

Tyyppijärjestelmä

Kirjaustyökalun kehityksessä hyödynnettiin C#-kielen tyyppijärjestelmää. C#-kieli on vahvasti tyypitetty kieli, mikä tarkoittaa, että kaikilla muuttujilla ja lausekkeilla on jokin määritelty tyyppi. [14.]

Reflektio

Reflektio on tapa tutkia ohjelmassa esiintyviä tyyppejä suorituksen aikana. C#:ssa reflektiota käytetään .NET Frameworkin Reflection-kirjastolla, joka tarjoaa rajapinnan C#-tyyppien käsittelemiseen. Kirjastolla voidaan ohjelman suorituksen aikana tutkia ohjelmassa esiintyviä tyyppejä ja niiden tietueita, kuten luokan muuttujia ja funktioita. Reflektio perustuu C#:n tyyppijärjestelmän hyödyntämiseen. [15.]

Attribuutit

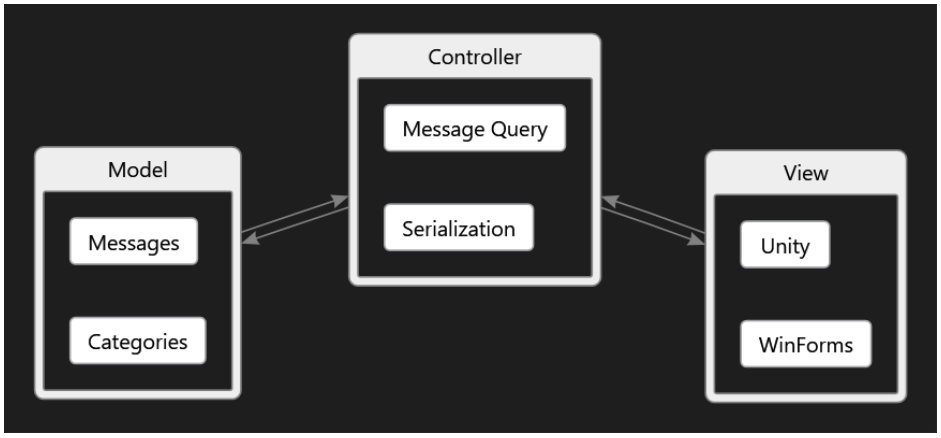
Attribuutit ovat tapa yhdistää metadataa koodissa esiintyviin eri tyyppeihin. Metadata on mielivaltaista tietoa ohjelmassa esiintyvistä tyypeistä. Attribuutteja käyttämällä voidaan

yhdistetystä tyypistä kysyä määritettyä attribuuttia ohjelman suorituksen aikana käyttämällä reflektiota. [16.]

4.2 MVC-arkkitehtuuri

Kehityksessä haluttiin ottaa huomioon kirjaustyökalun ja käyttöliittymän erottelu toisistaan, jotta ohjelman rakenne pysyy selkeänä. Erottelu mahdollistaisi kirjaustyökalun käytön ilman käyttöliittymää ja myös tulevaisuudessa useamman käyttöliittymä implementaation luonnin eri ympäristöihin.

MVC (Model-View-Controller) on arkkitehtuurimalli, jota käytetään käyttöliittymien erotteluun ohjelman logiikasta. Arkkitehtuurimallit pyrkivät parantamaan ohjelman jäsentelyä tarjoamalla ratkaisuja usein toistuviin ongelmiin. [17.] MVC jakaa ohjelman kolmeen loogiseen osaan, malli, kontrolleri ja näkymä. Yksi MVC:n hyödyistä on koodin riippuvuuksien vähentäminen, minkä seurauksena koodista tulee selkeämpää ja uudelleenkäytettävämpää. MVC on yksi käytetyimmistä arkkitehtuurimalleista web-kehityksessä. [18.] Kuvassa 12 havainnollistetaan MVC-arkkitehtuurimallia ja sen soveltamista tekstikonsolissa.



Kuva 12. MVC-arkkitehtuurimalli kirjaustyökalussa.

Malli-osa on MVC-arkkitehtuurissa ohjelman osa, joka kattaa ohjelman dataan liittyvän logiikan. Malli käsittää kaiken datan, jota ohjelma voi käyttää sen muissa osissa. Kirjaustyökalussa malli-osa käsittää viestien ja kategorioiden datan hallinnan. [18.]

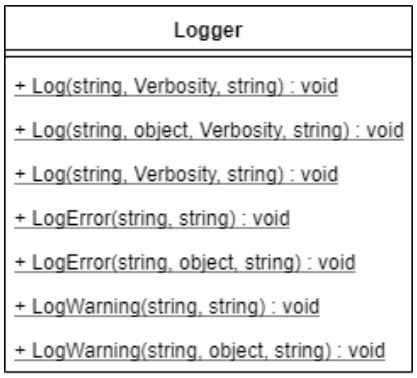
Näkymä on ohjelman osa, joka sisältää käyttöliittymän implementaation. Käyttöliittymän erottelulla pyritään erottelemaan näkymän implementaatio ohjelman muusta logiikasta. Erottelu mahdollistaa erilaisten käyttöliittymäimplementaatioiden käyttämisen ohjelman kanssa. Kirjaustyökalussa käyttöliittymiä voi olla useita, ja ne voivat mukautua ympäristön mukaan. Kirjaustyökalun näkymän tarkoitus on näyttää saatua tietoa viesteistä ja tarjota käyttöliittymäratkaisu hallinoimaan ohjelman määrittelemiä ominaisuuksia. [18.]

Kontrolleri on MVC-arkkitehtuurissa käsite, joka toimii mallin ja näkymän välisenä rajapintana. Kontrolleri käsittelee kaiken ohjelman logiikan, muuntaa mallin datan näkymälle vaadittavaan muotoon ja käsittelee näkymästä saatuja käskyjä. Kirjaustyökalussa kontrolleri suorittaa kirjaamisen lokitiedostoon, viestien luonnin ja kategorioiden käsittelemisen. Kontrollerin avulla voidaan käsitellä kirjaustyökalun eri ominaisuuksia, kuten käyttöliittymän sitomista, viestien vastaanottamista ja kategorioiden hakemista. [18.]

4.3 Kirjausrajapinta

Kirjausrajapinta haluttiin säilyttää yksinkertaisena, sen haluttiin poikkeavan mahdollisimman vähän muista käytetyistä kirjaustyökaluista. Vaikutteita otettiin mm. C#:n standardivirtoja käsittelevän Console-luokan, Unityn Debug-luokan ja Unreal Engine 4:n rajapinnoista, joissa rajapintafunktioita on vain muutamia.

Kirjaustyökalun viestin syötteet luodaan Logger-luokalla. Se on staattinen luokka, joka sisältää useita funktioita viestien sisällön syöttämiseen. Logger-luokan rajapinta on kuvattu kuvassa 13. Logger-luokka välittää kehittäjän syöttämät parametrit kirjaustyökalun kontrollerille, joka muodostaa datasta viestin.



Kuva 13. Logger-luokan rajapinta, jonka kautta kehittäjä luo viestejä kirjaamiseen.

Optionaalinen parametri on C#-kielessä funktion parametri, jolla on oletuksena alustettu arvo. Tavallisesti funktiokutsun yhteydessä jokainen funktion parametri on syötettävä, mutta optionaaliset parametrit voidaan vaihtoehtoisesti jättää syöttämättä, jolloin niiden arvona käytetään funktion määrittelyssä määritettyä arvoa. Optionaalisten parametrien täytyy olla aina parametrilistan lopussa kaikkien pakollisten parametrien jälkeen. [19.]

CallerFilePath-attribuutti on optionaalisille merkkijono parametreille määriteltävä attribuutti, joka on osa C#-kompilaatiopalvelukirjastoa (engl. CompilerServices). Lähdekoodin kääntämisvaiheessa jokaiselle funktiokutsulle, jossa parametrinä on CallerFilePath-attribuutilla merkitty merkkijono, syötetään funktiokutsun tekevän tiedoston lähdepolku. [20.]

Jokainen Logger-luokan funktio sisältää optionaalisen parametrin, jolla on CallerFilePath-attribuutti. Lähdetiedoston polku tallennetaan osana viestiä ja sen avulla voidaan jälkeenpäin viitata suoraan viestin luoneen koodin tiedostoon. Kompilaatiopalvelu tarjoaa myös attribuutit *CallerLineNumber* ja *CallerMemberName*, joita voidaan käyttää samalla tavalla kuin CallerFilePath-attribuuttia saamaan funktiokutsun tekevän lähdetiedoston rivinumero ja funktion nimi. CallerLineNumber- ja CallerMemberName-attribuutteja ei kuitenkaan hyödynnetä, joten ne jätettiin Logger-luokan rajapinnasta pois.

4.4 Staattiset kategoriat

Staattisen kategorian määrittelyyn käytetään LogCategory-attribuuttia. Se on attribuutti, joka ottaa vastaan parametrinä merkkijonon, joka on käyttäjän antama kategorian nimi. LogCategory-attribuutti määritellään luokkaan, johon halutaan sitoa staattinen kategoria. Määrittelyn perusteella luokan funktioista luodut viestit sidotaan automaattisesti funktion määrittelemän luokan kategorian mukaan. Esimerkkikoodi 1 näyttää miten staattisen kategorian määrittely tehdään LogCategory-attribuutilla. Esimerkissä kirjattu viesti on sidottu luokassa määriteltyyn staattisen kategoriaan.

```
[LogCategory("My Category")]
public partial class MainWindow : Form
{

    private void TestMsg_Click(object sender, EventArgs e)
    {
        Logger.Log("This is a test message", Verbosity.Debug);
    }
}
```

Esimerkkikoodi 1. LogCategory-attribuutti luokan määrittelyssä.

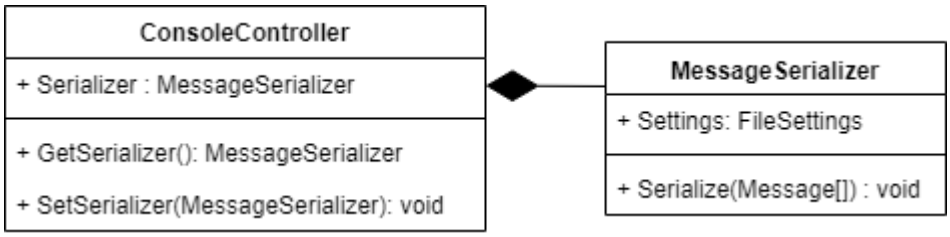
Kirjaustyökalu osaa automatisoidusti hyödyntää attribuutissa määriteltyä kategorian nimeä, ilman että käyttäjän täytyy tehdä erillisiä toimenpiteitä. Attribuutin lukeminen ohjelman suorituksen aikana on toteutettu reflektiolla, jonka avulla kirjaustyökalun alustuksen yhteydessä luetaan kaikki saatavilla olevat luokat ja niissä määritetyt LogCategory-attribuutit. Luokka ja siinä määritelty LogCategory-attribuutti tallennetaan kirjaustyökalun välimuistiin. Viestin luonnin yhteydessä luodun kutsupinon avulla voidaan selvittää funktio, jossa viesti on luotu, ja funktion määrittelemä luokka. Funktion määrittelemää luokkaa vertaillaan viestin luonnin yhteydessä välimuistissa oleviin kategorioihin, ja näin saadaan tietoon kategoria, johon viesti kuuluu.

4.5 Lokitiedostoon kirjaaminen

Serialisointi on tapa tallentaa luokan instanssin tietoja ja luoda instanssi uudelleen tallennetun tiedon pohjalta. Serialisointia hyödynnetään peleissä esimerkiksi pelin eri vaiheiden tallentamiseen. Tallennettu pelivaihe voidaan siten jälkeenpäin palauttaa serialisoitujen olioiden ja tiedon pohjalta, ts. deserialisoida (engl. deserialize).

Serialisointia hyödynnetään kirjaamisessa viestin olion tietojen tallentamisessa lokitiedostoon. Jokainen kirjaustyökalun viesti tallennetaan serialisointia hyödyntämällä tekstimuodossa määritettyyn lokitiedostoon. Yksinkertaisuuden vuoksi kirjaustyökalu tallentaa saapuvat viestit automaattisesti lokitiedostoon.

MessageSerializer on luokka, joka käsittää viestien datan serialisoinnin määritettyyn lokitiedostoon. Kirjoitettavan lokitiedoston tiedostopolku määritetään *FileSettings*-luokan instanssin kautta, jonka MessageSerializer-luokka sisältää. Serialisoinnissa käytettävän lokitiedoston voi tarvittaessa vaihtaa FileSettings-luokan kautta. Vaihtoehtoisesti perimällä MessageSerializer-luokka voidaan toteuttaa oma serialisointilogiikka. Kontrolleri sisältää kuvan 14 mukaisesti kompositio suhteen MessageSerializer-luokan ilmentymään ja rajapinnan sen käsittelemiseen.



Kuva 14. Kontrollerin suhde MessageSerializer-luokkaan, joka tallentaa annetut viestit tekstitiedostoon.

5 Tekstikonsolin käyttöliittymän toteutus

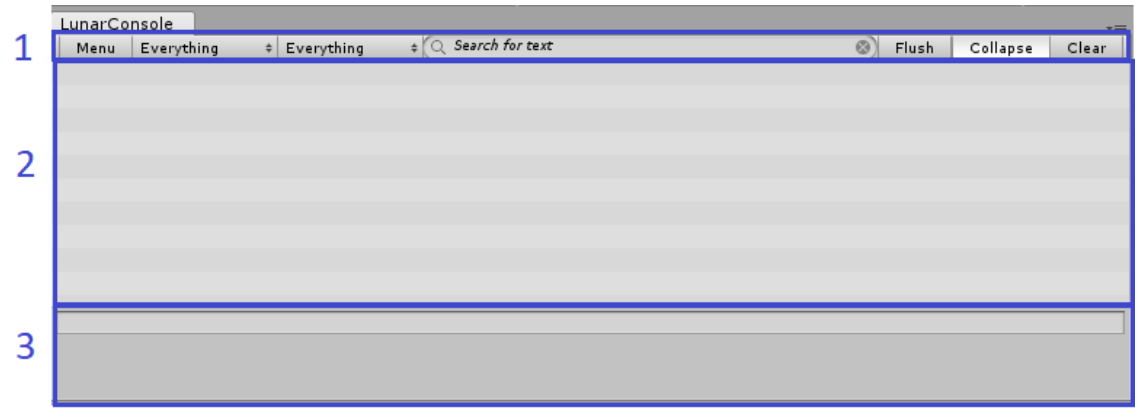
Kehityksen alkaessa lähdettiin suunnittelun pohjalta luomaan käyttöliittymää, jossa tekstikonsoli toimisi. Ohjelman oli tarkoitus toimia Unity3D-editorissa, jossa tekstikonsolia käytettäisiin Unity3D:n oman Console-työkalun sijasta.

Unity3D-editorissa käytetään IMGUI:a (engl. Immediate mode graphical user interface) sen käyttöliittymän laajentamiseen. IMGUI on graafisten käyttöliittymien rakentamiseen tarkoitettu järjestelmä, jossa käyttöliittymiä luodaan koodipohjaisesti. [21.]

Käyttöliittymät rakennetaan IMGUI:lla ohjelmallisesti funktiokutsujen avulla, jotka toteuttavat erilaisia toimintoja käyttöliittymässä. Toiminnot voivat olla mm. erilaisten käyttöliittymäelementtien piirtämistä, elementtien asetteluun tai muihin käyttöliittymässä suoritettaviin toimintoihin liittyviä funktiokutsuja.

Unity3D-editorin IMGUI-järjestelmä sisältää monia valmiita graafisia elementtejä, kuten nappuloita ja tekstikenttiä. Valmiita graafisia elementtejä käytettiin tekstikonsolin kehittämisessä nopeuttamaan kehitysaikaa ja luomaan Unity3D-editorin ympäristöön mukautuvaa käyttöliittymää.

Käyttöliittymä koostuu kolmesta pääelementistä, työkalupalkki, viestilista ja infotekstialue. Työkalupalkin avulla kehittäjä voi hallinnoida viestien suodattamista ja tekstikonsolin eri toimintoja. Kuvassa 15 on kehitetyn käyttöliittymän näkymä ja merkittynä käyttöliittymän pääelementit.



Kuva 15. Tekstikonsolin käyttöliittymä. Merkittynä tekstikonsolin pääelementit: työkalupalkki, viestilista ja infotekstialue.

5.1 Viestilista

Viestilista on tekstikonsolissa lista ohjelmasta tulleista viesteistä. Viesteistä, jotka ovat viestilistassa, käytetään nimitystä viestimerkintä. Viestilista on toteutettu hyödyntäen Unity3D-editorin tarjoamaa TreeView IMGUI -elementtiä. TreeView-elementtiä käytetään näyttämään dataa hierarkkisessa suhteessa. Sitä voi käyttää myös listojen näyttämisessä, jossa hierarkkinen suhde ei ole relevanttia. Elementillä voidaan myös hallita viestimerkinnän näyttämistä listassa, kuten kokoa ja piirtotyyliä. [22.]

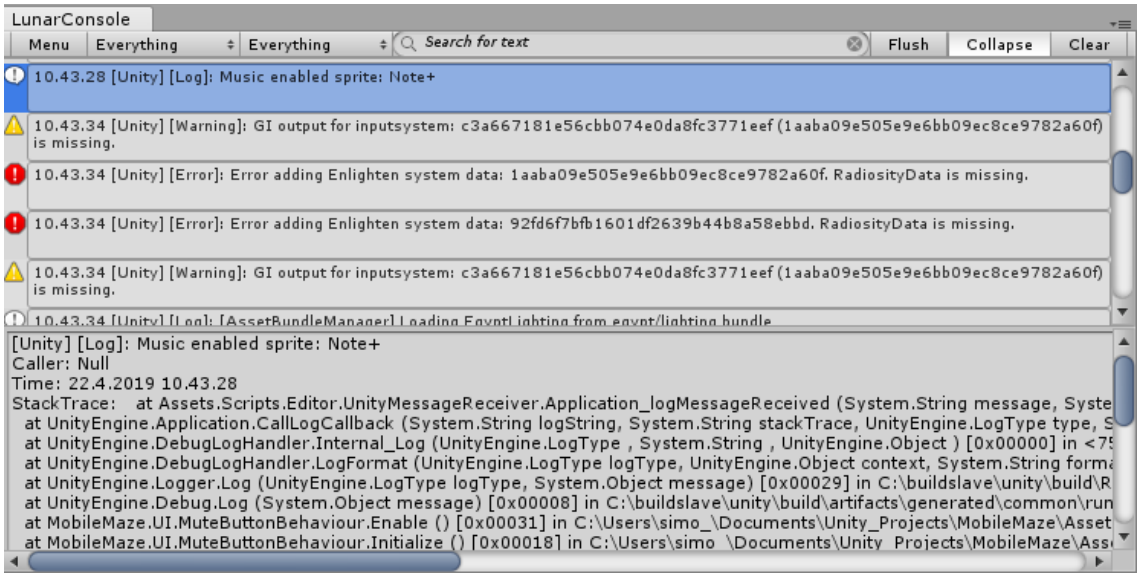
Viestilistaan kehitettiin joustava suodatusjärjestelmä, jonka avulla kehittäjät voivat luoda omia suodattimia, joiden perusteella viestilistassa näytetään viestejä. Viestilista tarjoaa rajapinnan, jonka avulla kehittäjä voi ohjelmallisesti määrittää suodatinfunktion, jota viestilista käyttää päättääkseen, näytetäänkö yksittäinen viesti. Suodatinfunktion rajoitteena on funktion määritelmä, jonka pitää palauttaa boolean-tyyppinen palautusarvo ja ottaa parametrinä vastaan Message-luokan instanssi. Viestilista suorittaa suodatinfunktiot yhtä monta kertaa, kuin viestejä on tekstikonsolissa saatavilla.

Viestilistan viestimerkintöjen piirtäminen toteutettiin ohjelmallisesti laajennettavaksi. Viestilista-luokka paljastaa C#-eventin, johon voidaan kytkeä viestin piirtämiseen tarkoitettu funktio. Viestin piirtäminen toteutettiin myös IMGUI-järjestelmän avulla.

Viestimerkinnän piirtämisen räätälöinti antaa kehittäjille mahdollisuuden luoda halutunlainen visualisointi.

5.2 Infotekstikenttä

Infotekstikenttä on tekstikonsolissa graafinen tekstielementti, joka näyttää valitun viestin tietoja. Infotekstikentän tehtävä on esittää kehittäjälle viestin sisältö kokonaisuudessaan. Se näyttää myös viestissä olevaa metadataa, kuten viestin lähteen kutsupinon avulla. Kuvassa 16 on valittuna eräästä yrityksen pelistä saatu viesti, jonka tietoja näytetään tekstikonsolin alaosassa sijaitsevassa infotekstikentässä.



Kuva 16. Tekstikonsoli, jossa valittuna on viesti, jonka tietoja näytetään tekstikonsolin alaosassa olevassa infotekstikentässä.

Metadatassa oleva kutsupino on yleensä tekstikonsolin infotekstikentässä liian pitkä näytettäväksi, minkä takia päädyttiin vieritettävään ikkunaan sivu- ja pystysuunnassa. Vaihtoehtona vieritettävän ikkunan sijasta olisi ollut tekstin kääriminen näytettävään osaan, jolloin tekstin rivit olisivat jakautuneet useammalle riville. Tekstin käärimistä haluttiin kuitenkin välttää, koska se teki kutsupinojen lukemisesta hankalaa. Kuvassa 17 on Unity3D-editorin Console-työkalusta otettu kuva viestin kutsupinosta, joka on käärityn useammalle riville, jotta teksti mahtuu sivusuunnassa tekstikenttään.

Kuva 17. Unity3D-editorin Console-työkalun tekstin kääriminen.

5.3 Suodattimet

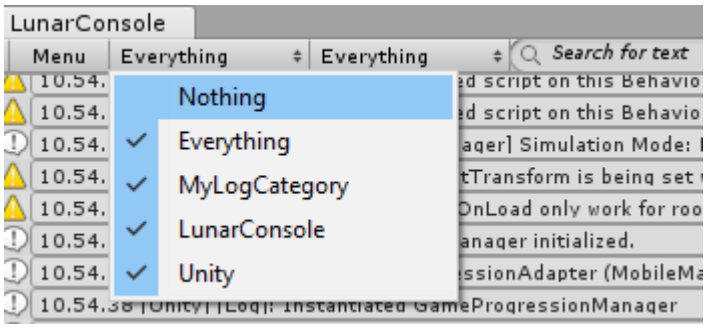
Suodattimet ovat tekstikonsolin käyttöliittymässä tapoja suodattaa viestejä. Tekstikonsoliin suunnitteluvaiheessa päätettiin luoda kolme suodatinta, joiden perusteella viestejä voidaan suodattaa.

Osassa suodattimia hyödynnetään Unity3D:n IMGUI-järjestelmän MaskField-elementtiä. Se sisältää listan merkkijonoista, joiden tila voi päällä tai pois. Elementissä on nappula, jota painamalla merkkijonolista avautuu. Kuvassa 18 on MaskField-elementin merkkijonolista avattuna. Normaalissa elementin tilassa piirretään nappula ilman merkkijonolistaa.

Kuva 18. MaskField-elementin merkkijonolista avattuna. Kuvassa näkyvät valitut merkkijonot [23].

Kategoriasuodatin

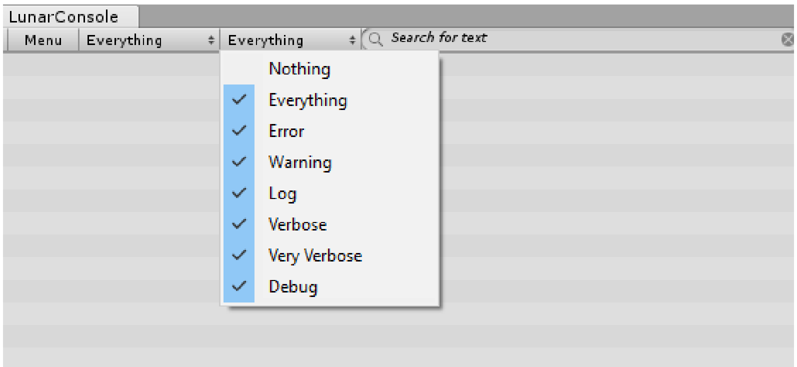
Kategoriasuodatin on tekstikonsolin käyttöliittymässä graafinen elementti, joka mahdollistaa staattisten kategorioiden suodattamisen. Suodattimessa käytetään Unity3D:n IMGUI-järjestelmän MaskField-elementtiä. Tekstikonsolin käyttöliittymä hyödyntää kontrollerin rajapintaa selvittääkseen saatavilla olevat staattiset kategoriat, joiden perusteella käyttöliittymä täyttää MaskField-elementin merkkijonolistan. Tekstikonsolin käyttöliittymä päivittää viestilistan näytettävät viestit kategoriasuodattimen avulla. Kuvassa 19 on merkkijonolista kategorioista, joiden perusteella viestilistasta suodatetaan viestit, joiden kategoriaa ei ole valittu.



Kuva 19. Kategorioiden suodatus MaskField-elementtiä hyödyntämällä.

Vakavuustasojen suodatin

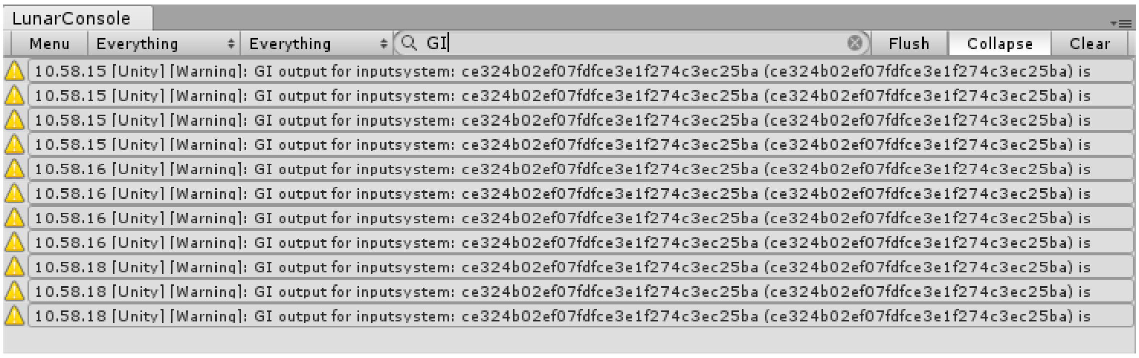
Vakavuustasojen suodatin on kategoriasuodattimen tapaan graafinen elementti, joka hyödyntää MaskField-elementtiä. Vakavuustasojen suodattimella voidaan määrittää viestilistaan näkyväksi vain viestit, jotka on suodattimessa määritelty. Vakavuustasot saadaan kirjaustyökalun kontrollerilta, joka määrittää mahdolliset vakavuustasot. Kuvassa 20 on MaskField-elementin merkkijonolista avattuna, joiden perusteella viestilistasta suodatetaan pois viestit, joiden vakavuustaso ei ole valittuna listassa.



Kuva 20. Vakavuustasojen suodatus MaskField-elementtiä hyödyntämällä.

Tekstinhaku

Tekstinhakupalkissa käytetään visualisoinnin apuna Unity3D-editorin SearchField-elementtiä. Se tarjoaa hakupalkille tyypillisen visuaalisen ilmentymän. Hakupalkki toimii pääpiirteittäin tavallisena tekstikenttänä, johon käyttäjä voi kirjoittaa tekstiä. Kuvassa 21 on tekstikonsolin hakupalkkiin annettu merkkijono, jonka avulla viestilistasta suodatetaan viestit sisällön perusteella.



Kuva 21. Viestien suodattaminen tekstihakupalkilla viestin sisällön perusteella.

6 Tulosten arviointi

Insinöörityön tuloksena saatiin aikaan toimiva testiversio tekstikonsolista. Joistakin tekstikonsolin ominaisuuksista jouduttiin aikarajoitteiden ja työn laajuuden takia luopumaan.

Monet karsituista ominaisuuksista olivat helppokäyttöisyystoimintoja tai visuaalisia tekijöitä, jotka eivät itsessään vaikuttaneet tekstikonsolin tarkoitettuun toimintaan.

Tekstikonsolin käytön testeissä huomattiin kategorioissa ongelmia, joita pitäisi vielä korjata. Kategorioiden jakaminen staattisiin ja dynaamisiin tyyppeihin toimii hyvin, kun halutaan luoda nopeasti kategoria uusiin viesteihin tuomaan lisäkontekstia. Kategorioiden ongelma nykyisellään on, että viestillä voi olla kerrallaan vain kaksi kategoriaa, yksi kutakin tyyppiä. Tapauksissa, joissa staattinen kategoria on ohjelmassa korkealla tasolla, syntyi ongelma, jossa uuden staattisen kategorian määritteleminen syvemmällä ohjelmakoodissa aiheutti kategorian ylikirjoittamisen. Kategorian ylikirjoittaminen tarkoitti, että viestiin sidottu staattinen kategoria oli aina viimeisimpänä ilmestyvä staattinen kategoria, mikä vaikutti puolestaan käyttöliittymässä oletettujen kategorioiden häviämiseen viestien yhteydessä. Ongelman ratkaisuksi tekstikonsolissa staattiset kategoriat pitäisi esittää hierarkkisessa suhteessa toisiinsa, minkä avulla viesti voi kuulua useampaan kategoriaan.

MVC-arkkitehtuurimallin hyödyntäminen onnistui tekstikonsolissa hyvin. Arkkitehtuurimallin implementoinnin seurauksena tekstikonsolista käyttöliittymän erittely onnistuu todella helposti kirjaustyökalusta. Kirjaustyökalusta tuli myös modulaarinen, minkä seurauksena sen eri ominaisuuksia voi laajentaa hyvin. Kehittäjä pystyy esimerkiksi luomaan oman datarakenteen viesteille tai luomaan kokonaan uuden viestintallentamisjärjestelmän eri lähteisiin. Tallentamisjärjestelmän muokkaus mahdollistaa esimerkiksi tallennuspaikan ohjaamisen moneen eri paikkaan ja implementaatiosta riippuen myös tietokantaan tai palvelimelle.

Unity3D-editoriin luotu käyttöliittymä koettiin suhteellisen onnistuneeksi. Käyttöliittymän IMGUI-järjestelmä on luonteeltaan työläs monimutkaisten käyttöliittymien rakentamiseen. IMGUI-järjestelmän tuomia ongelmia implementaatiossa havaittiin sen tavassa sekoittaa piirtokutsut käyttöliittymän muuhun logiikkaan. Sen seurauksena monimutkaisemmissa käyttöliittymän osissa on paljon sekoitettu piirtämiseen ja toiminnallisuuteen liittyvää logiikkaa, mikä aiheuttaa pahimmassa tapauksessa vaikeasti hallinnoitavaa koodia.

Tekstikonsolin modulaarisuuden ja käyttöliittymän erottelun myötä sitä voitaisiin hyödyntää myös erillisenä ohjelmana kehityksen yhteydessä. Erillinen tekstikonsoliohjelma voisi olla oma ohjelmaprosessinsa, jolla olisi käyttöjärjestelmässä toimiva käyttöliittymä. Ohjelmaa voisi hyödyntää ohjaamalla syötteet tekstikonsolin erilliseen prosessiin. Erillinen ohjelma vähentäisi kehitysympäristön käyttöliittymän integroimiseen käytettävää aikaa ja tekisi tekstikonsolin käyttämisen eri ympäristöissä helpommaksi.

LunarConsole-tekstikonsolissa yhdistettiin hyväksi todettuja toimintoja Unreal Engine 4:n ja Unity3D:n tekstikonsolityökaluista, kuten Unreal Engine 4:n kategoriat ja isompi valikoima vakavuustasoja. Unity3D:n Console-ikkunasta vaikutteita otettiin viestien sisällön ja käyttöliittymän suunnittelussa. Tekstikonsoliin luodut suodatustoiminnot, kuten kategoriat ja etsintä, toivat paljon lisäystä Unity3D:n tekstikonsoli-ikkunaan. Kehitetyillä ominaisuuksilla LunarConsole-tekstikonsolista saatiin kattava työkalu, joka auttaa kehittäjää lokianalyysiin hallinnassa Unity3D:n pelimoottoriympäristössä työkalun tarjoamilla ominaisuuksilla.

7 Yhteenveto

Insinöörityössä luotiin LunarByte Oy:lle Unity3D-pelimoottoriin tekstikonsoliliitännäinen, jota hyödynnetään peli- ja ohjelmakehityksissä. Tekstikonsolin tehtävä oli viestiä kehittäjälle kehitettävän pelin tai ohjelman eri tapahtumia ja toimia apuna vianetsinnässä.

Suunnittelun aikana käytiin toimeksiantajan kanssa läpi Unity3D-pelimoottorin tarjoaman Console-tekstikonsolityökalun toimintoja ja puutteita, joita voitaisiin parantaa paremman käyttökokemuksen saavuttamiseksi. Suunnittelun aikana pohdittiin paljon lisättäviä ominaisuuksia ja toimintoja, mutta jo suunnittelun aikana todettiin, että kaikkia ominaisuuksia ei ehditä välttämättä tekemään työn aikana.

Kehityksen aikana tutustuttiin enemmän MVC-arkkitehtuuri malliin, jonka avulla ohjelman ja sen käyttöliittymän logiikan erottaminen oli helpompaa. MVC-arkkitehtuurilla oli hyvät tulokset, ja sen seurauksena sitä ja vastaavia arkkitehtuurimalleja mietitään myös yrityksen omien projektien hyödyntämisessä enemmän. Myös Unity3D:n IMGUI-järjestelmään tutustuttiin työn aikana. Se todettiin toimivaksi ainakin pienissä käyttöliittymissä,

joissa käyttöliittymän koodi pysyy hallittavassa määrässä. Monimutkaisemmissa käyttöliittymissä IMGUI-järjestelmän käyttäminen on työlästä ja laajennettavauus hankalaa.

Työn lopputuloksena saatiin tekstikonsolista toimiva versio Unity3D-editoriympäristössä. Työssä luotua tekstikonsolia ei vielä ole yrityksessä päästy hyödyntämään projektien luonteen vuoksi, mutta tulevaisuudessa sen hyödyntäminen on todennäköistä. Tekstikonsolia todennäköisesti myös kehitetään projektien yhteydessä, sillä sen tuomat edut projektien kehityksessä olivat jo suunnitteluvaiheessa selvillä.

Lähteet

1 Zhang Ellen. 2014. What is log analysis? Verkkoaineisto. <https://whatis.techtarget.com/definition/log-log-file> Luettu 11.3.2019.

2 Kent Karent & Souppaya Murugiah. 2006. Guide to Computer Security Log Management. Verkkoaineisto. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf> Luettu 12.3.2019.

3 What is Comand Line Interface (CLI)? Verkkoaineisto. W3Schools. <https://www.w3schools.com/whatis/whatis_cli.asp> Luettu 20.4.2019.

4 Command Prompt. 2019. Microsoft Corporation.

5 What is a Game Engine? 2008. Verkkoaineisto. Game career guide. <http://www.gamecareerguide.com/features/529/what_is_a_game_.php> Luettu 18.3.2019.

6 Unity for gaming. Verkkoaineisto. Unity Technologies. <https://unity.com/solutions/gaming> Luettu 18.3.2019.

7 Console Window. 2017. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Manual/Console.html> Luettu 11.3.2019.

8 Unreal Engine 4 Features. Verkkoaineisto. Epic Games. <https://www.unrealengine.com/en-US/features> Luettu 24.3.2019.

9 Unreal Engine 4. 2019. Epic Games.

10 Scripting the Editor using Python. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/en-us/Editor/ScriptingAndAutomation/Python> Luettu 15.4.2019

11 McMaster Scott & Memon Atif. 2006. Call Stack Coverage for GUI Test-Suite Reduction. Verkkoaineisto. <http://www.cs.umd.edu/~atif/papers/McMasterMemonISSRE2006.pdf> Luettu 14.3.2019.

12 Visual Studio 2017. 2019. Microsoft Corporation.

13 What is .NET? Verkkoaineisto. Microsoft Corporation. <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet> Luettu 11.3.2019.

14 Types (C# Programming Guide). 2015. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types> Luettu 21.4.2019.

15 Reflection in the .NET Framwork. 2017. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection> Luettu 5.1.2019.

16 Attributes (C#). 2018. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/attributes> Luettu 5.1.2019.

17 Architectural Patterns and Styles. 2010. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v=pandp.10)> Luettu 17.3.2019.

18 MVC Framework – Introduction. Verkkoaineisto. Tutorialspoint. <https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm> Luettu 22.4.2019.

19 Named and Optional Arguments. 2015. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/named-and-optional-arguments> Luettu 26.3.2019.

20 CallerFilePathAttribute Class. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.compilerservices.callerfilepathattribute?view=netframework-4.7.2> Luettu 26.3.2019.

21 Immediate Mode GUI (IMGUI). 2018. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Manual/GUIScriptingGuide.html> Luettu 12.3.2019.

22 TreeView. 2018. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Manual/TreeViewAPI.html> Luettu 22.4.2019.

23 Unity3D Documentation MaskField. 2018 Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/ScriptReference/EditorGUI.MaskField.html> Luettu 15.2.2019.