Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Emil Dewald

Tarinankerronnan työkalut pelimoottoreissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

20.5.2019

Tekijä Otsikko Sivumäärä Aika	Emil Dewald Tarinankerronnan työkalut pelimoottoreissa 37 sivua 20.5.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Antti Laiho

Insinöörityössä perehdyttiin pelinarratiivin historiaan ja sitä tukeviin pelinkehitystyökaluihin. Pelinkehitystyökalun tuotantoprosessia tutkittiin syventymällä pelinkehitystyökalun käyttäjäkokemuksen suunnittelemiseen ja sen parhaisiin käytäntöihin. Työn teknisen toteutuksen tavoitteena oli kehittää pelinarratiivin kehitystä tukeva työkalu. Työkalun tavoitteena on lisätä työskentelyn tehokkuutta vähentämällä Scriptable Object -olioiden luomiseen kuluvaa aikaa ja oppimisprosessin kestoa.

Pelinkehitystyökalu suunniteltiin Unity-pelieditoriin erityisesti käyttäjäkokemussuunnitteluun ja käyttöliittymään painottaen. Suunnitellulle pelinkehitystyökalulle luotiin kuvakäsikirjoitus, jossa käsiteltiin työkalun käyttökulkua. Insinöörityössä tutkittiin pelinkehitystyökalun kehityksessä huomioonotettavia käyttäjäkokemussuunnittelukäytäntöjä ja ratkaisuja yleisiin käyttäjäkokemusongelmiin. Myös projektin toteutuksessa käytettäviin työkaluihin ja lähdeaineistoihin perehdyttiin.

Työkalun ohjelmoimiseen varattiin rajattu määrä aikaa projektin laajuuden rajaamiseksi. Aikaa jätettiin myös raportoimiseen ja kriittisten ongelmien korjaamiseen. Ohjelmoidessa tuli vastaan ongelmatilanne, jossa ilmeni projektityöhön tehtävän taustatutkimuksen tärkeys.Tutkimus Scriptable Object -olioista ja niiden ohjelmoimisesta oli puutteellinen, ja siksi työkalun tietorakenne jouduttiin suunnittelemaan uudelleen ja ohjelmointiin tuli viive. Työkalu julkaistiin GitHub-sivustolla avoimen lähdekoodin projektina.

Insinöörityön teknisen toteutuksen tuloksena syntyi toimiva Unity Editor -työkalu, joka tukee Scriptable Object -olioiden luomista vähentämällä käyttäjän ohjelmointiosaamisen tarvetta. Toteutetun työkalun käyttöä verrattiin nopeusvertailukokeella Scriptable Object -olion luomiseen koodaamalla. Tulosten perusteella työkalu tarjoaa noin 50 prosentin tehokkuusparannuksen Scriptable Object -olioiden luomisnopeudessa.

Työkalu on saatavilla GitHub-sivustolla.

Avainsanat	narratiivi, pelinkehitystyökalut, Unity, käyttäjäkokemus, UX

Author Title Number of Pages Date	Emil Dewald Narrative tools in game engines 37 pages 20 May 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructor	Antti Laiho, Senior Lecturer

This thesis is a study of the history of storytelling, game narrative and the game development tools that support narrative design. The development process of game development tools is researched, focusing on User Experience- and User Interface design and best practices. The goal of the technical part of the final year project was to create a game development tool that supports narrative design work. The goal of the tool is to increase the efficiency of Scriptable Object creation by reducing the time spent creating them, and the learning curve of working with Scriptable Objects.

The tool was designed for the Unity Editor, focusing especially on UX- and UI-design work. A storyboard that visualizes the user flow of the designed tool was created. Studies of UX-design methodologies in the creation of game development tools, and solutions to common UX problems were used as development guidelines. The tools and sources used in the development of the Scriptable Object Editor tool were also listed, from the Unity game engine and Microsoft Visual Studio programming platform to the main literary sources used as guidelines for the design and programming work.

A predetermined amount of time was reserved for programming the designed tool to limit the scope of the project. During programming, a major problem concerning the data structure of the tool and how Scriptable Objects work delayed the project completion, forcing a redesign of the structure from the ground up. This problem showed the importance of background research in software projects.

The result of the final year project is a working Unity Editor extension that supports the creation of Scriptable Objects by reducing the programming skill requirements for the user. A speed comparison test was run between using the Scriptable Object Editor tool and creating a scriptable object through programming. The results showed an estimated 50% efficiency increase in time spent creating a Scriptable Object when using the tool.

The Scriptable Object Editor tool is available on the GitHub-site.

Keywords	Narrative, Game Development Tools, Unity, UX

Sisällys

1 Johdanto

Videopelien historian alkuvuosista lähtien narratiivi on ollut tärkeä osa suuressa osassa pelejä. Jopa ennen kuin videopelit olivat suosittu viihdemuoto, tarinankerrontaa käytettiin pelimuodossa pelikirjoissa, kuten Choose Your Own Adventure -kirjasarjassa.

Videopeleissä tarinankerrontaa voi hyödyntää usealla tavalla, kuten visuaalisella, auditiivisella tai kirjallisella tavalla tai jollain näitten yhdistelmällä. Esimerkkejä tarinankerronnan eri muodoista videopeleissä ovat välianimaatiot, dialogit ja ääninäyttely.

Videopelin tuotannon aikana pelinkehittäjät käyttävät useita erilaisia työkaluja, jotka on suunniteltu helpottamaan pelin tuotantoprosessia. Pelimoottorit ovat selkein esimerkki pelinkehitystyökaluista, ja usein olennaisin niistä. Pelimoottorit ovat useimmiten kokoelma erilaisia työkaluja ja valmiita mekaniikkoja ja pelirakenteita, kuten fysiikkamoottori tai renderöintiteknologiat.

Tämän opinnäytetyön tarkoituksena on perehtyä pelinkehityksessä käytettäviin työkaluihin, jotka toimivat joko itsenäisesti tai pelimoottorin sisällä, etenkin Unity-pelimoottorin sisäisiin kolmannen osapuolen työkaluihin, jotka on suunniteltu avustamaan pelinkehitysprosessin tarinankerronnallisissa osuuksissa. Alussa perehdytään tarinankerronnan historiaan peleissä ja sitä tukeviin työkaluihin. Opinnäytetyössä tutkitaan myös pelinkehitystyökalun kehitysprosessia, siinä usein ilmeneviä ongelmia ja parhaat käytännöt kehitysprosessin sujuvuuden takaamiseksi. Samalla perehdytään myös pelinkehitystyökalun käyttäjäkokemussuunnitteluprosessiin.

Luvuissa 3 ja 4 raportoidaan insinöörityön osana tehdyn pelinkehitystyökalun kehitysprosessi. Luvuissa käydään läpi projektin tavoitteet, vaatimukset ja mahdolliset esteet, minkä jälkeen käsitellään pelinkehitystyökalun käyttäjäkokemusta ja käyttöliittymän suunnittelua. Viimeiseksi käydään läpi toteutukseen valitut ohjelmistot ja menetelmät sekä aineisto, jota hyödynnettiin tukena kehityksessä, kuten kirjat ja internetlähteet. Lopulta raportoidaan teknisen toteutuksen prosessi, sen aikana ilmaantuneet ongelmat, esteet ja niiden ratkaisut sekä lopputuloksen vastaavuus alkusuunnitelmaan.

Viimeiseksi pohditaan opinnäytetyön tuloksia ja sitä, kuinka paljon työssä kehitetty Scriptable Object Editor -työkalu avustaa tarinankerronnassa. Tutkitaan, nopeuttaako työkalu kehitysprosessia ja minkälaisia muita hyötyjä työkalusta voi olla kehittäjälle. Pohditaan myös projektin jatkokehitysmahdollisuuksia insinöörityön jälkeen.

Opinnäytetyöraportti on suunnattu kaikille, jotka ovat kiinnostuneet tarinankerronnallisesta pelinkehityksestä sekä pelinkehitystyökaluista ja niiden kehittämisestä. Koska työssä otetaan lähtökohtaisesti käyttäjäkokemuspainotteinen suunnitteluasenne, työ saattaa myös kiinnostaa käyttäjäkokemussuunnittelijan urasta kiinnostuneita.

2 Pelien narratiivi ja sitä tukevat työkalut

Tarinankerronta on yksi ihmiskunnan kehityksen kulmakivistä. Jo ennen kirjattua historiaa ihmiset ovat kertoneet tarinoita toisilleen. Varhaisin löydetty tarinankerronnan muoto on eteläisessä Ranskassa, Lascaux'n luolissa Pyreneiden vuoristossa. Luolan seinämille on maalattu kuvia erilaisista eläimistä ja ihmisistä. Luolamaalaukset kuvastavat tarinoita rituaaleista ja metsästyksestä (kuva 1). Luolamaalauksien arvioidaan olevan jostain vuosien 15 000 ja 13 000 eaa. väliltä. (A Very Brief History of Storytelling 2012.)

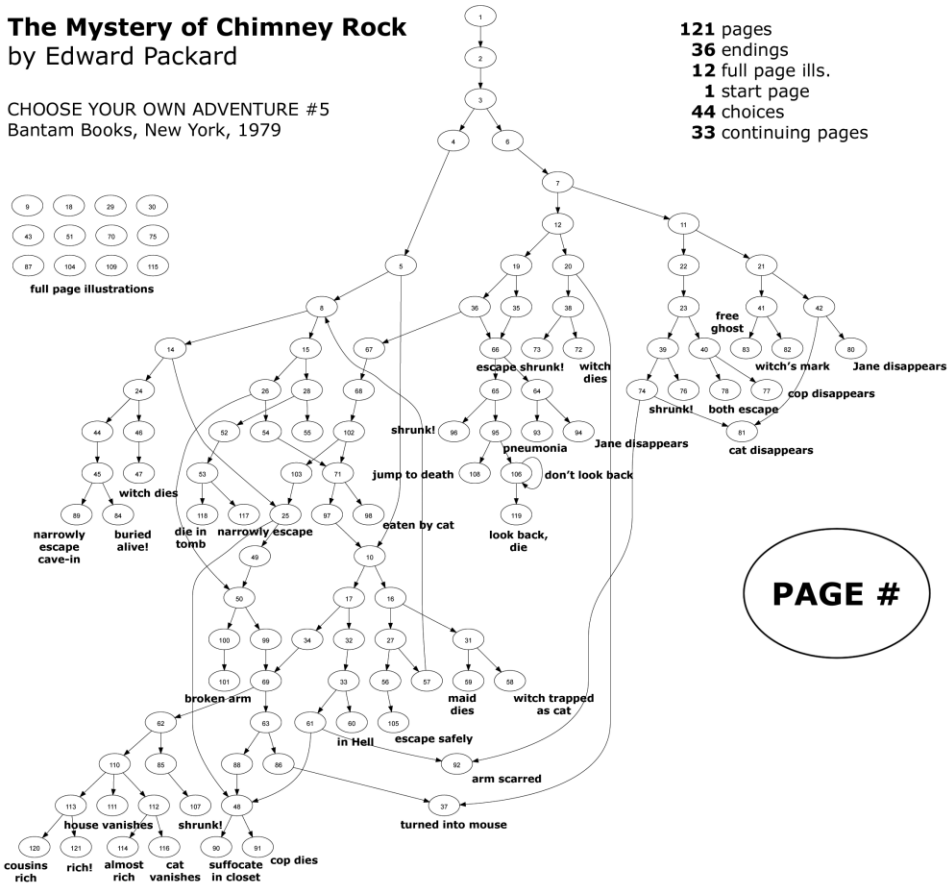
Kuva 1. Lascaux’n luolamaalauksesta tehty jäljennös (Beauregard 2017).

Vuosien edetessä ja teknologian kehittyessä olemassa olevat tarinankerronnan muodot kehittyivät ja uusia muotoja ilmaantui: kirjat ja runot, musiikki, teatteri, elokuvat ja viimeisimpänä tarinankerronnan muotona videopelit.

2.1 Tarinankerronta peleissä

Tarina on pitkään ollut monen videopelin tärkeimpiä elementtejä. Ensimmäiset esimerkit pelimuotoisista tarinoista ilmaantuivat vuonna 1979, kun Bantam Books aloitti vuonna 1976 julkaistun alkuperäisen konseptin pohjalta uuden, nuorille ja lapsille suunnatun julkaisuosaston. Interaktiivinen pelikirjasarja nimeltä ”Choose Your Own Adventure” sai tällöin alkunsa, ja vuosien 1979 ja 1999 välillä sarja myi yli 250 miljoonaa kappaletta maailmanlaajuisesti ja kirjoja oli käännetty 38 kielelle. (History of CYOA.)

Kuvassa 2 Choose Your Own Adventure -teoksen tarinankulkua on visualisoitu vuokaaviolla. Kuvasta näkee selkeästi, miten lukijan päätökset muuttavat tarinan kulkua ja miten kirjat ovat ensimmäisiä interaktiivisen narratiivin muotoja.



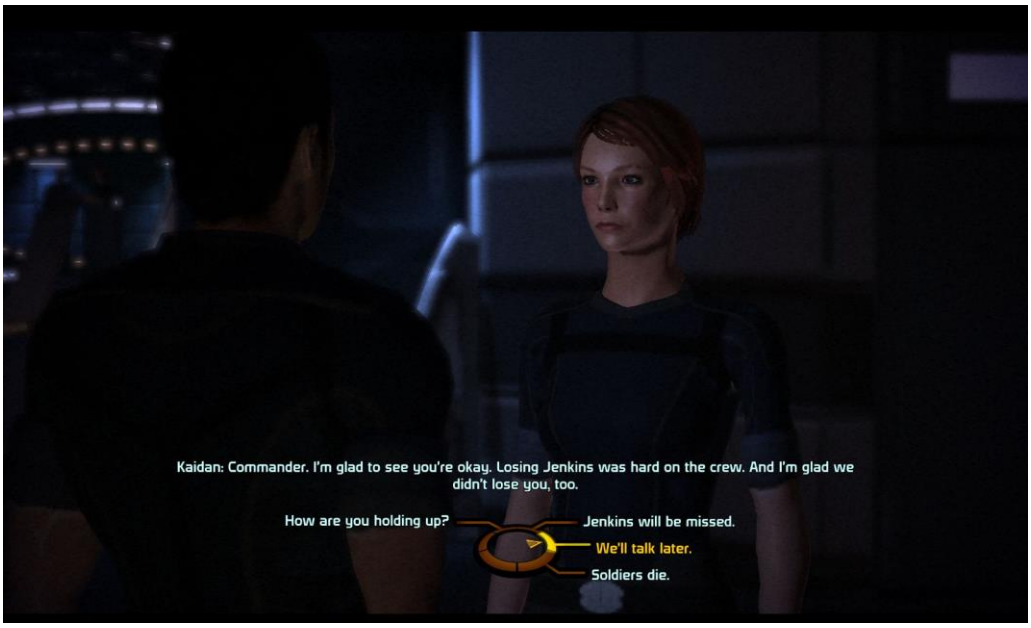
Kuva 2. Choose Your Own Adventure #5 -kirjan vuokaavio. Kuvassa näkyvät kaikki tarinan haarat ja lukijan valintojen seuraukset (Lamilami 2018).

Choose Your Own Adventure -kirjasarjalla oli selkeä vaikutus peleihin. Etenkin roolipelit, kuten Dungeons & Dragons -pöytäroolipeli, saivat paljon vaikutteita Choose Your Own Adventure -kirjoista. Varhaisissa videopeleissä Choose Your Own Adventure -kirjojen vaikutuksen huomaa etenkin Visual Novel -peligenressä ja sellaisissa peleissä kuin Final Fantasy, jotka molemmat ovat lähtöisin Japanista. Visual Novel -genressä suurimman vaikutteen saivat seikkailupainotteiset pelit ja treffisimulaattoripelit, etenkin Japanissa suosittu bishōjo-alagenre, jossa pelaajat pyrkivät saavuttamaan kauniin nuoren naisen huomion. (History of CYOA.)

Vuosien varrella pelien tarinankerronta on kehittynyt. Vanhemmissa peleissä tarinat olivat rajoitettuja hyvinkin rajallisen teknologian takia yksinkertaisiin tarinakehikkoihin, kuten Super Mario Bros -pelin tarina putkimiehestä, joka pelastaa prinsessan pahan kilpikonnamaisen olennon kourista.

Nykyään taas nähdään joka vuosi suurempia ja mahtavampia saavutuksia pelien tarinankerronnassa, kuten Mass Effect -pelisarjassa, jossa on vaikuttavia 3D-välianimaatioita ja ajankohtaan nähden edistyksekäs dialogi- ja valintajärjestelmä, joka vaikuttaa tarinan kulkuun. Näiden lisäksi Mass Effect -peleissä on myös joukko sivuhahmoja, joiden kanssa pelaaja voi seikkailla ja keskustella ja joilla on eriävät mielipiteet pelaajan valinnoista. Sivuhahmoista voi oppia uusia asioita, ja heidän kanssaan voi rakentaa läheisiä suhteita, tai pelaajan ja sivuhahmon välinen suhde voi hajota riippuen pelaajan valinnoista. Hyvä esimerkki näistä Mass Effect -sarjan ominaisuuksista on Mass Effect 2 ja etenkin sen loppuosa, missä pelin aikana tehdyt valinnat ja pelaajan suhteet sivuhahmoihin vaikuttavat dramaattisesti tarinan lopputulokseen ja päähahmon johtaman avaruusaluksen, SSV Normandyn, ja sen miehistön kohtaloon.

Mass Effect -pelisarjan dialogijärjestelmä (kuva 3) muistuttaa paljon Choose Your Own Adventure -kirjojen narratiivisia vuokaavioita (kuva 2) siinä mielessä, että jokainen pelaajan valitsema vaihtoehto dialogissa muuttaa dialogin kulkua ja lopullista päätöspistettä. Pelaaja saattaa esimerkiksi suututtaa dialogin toisen osapuolen sanomalla jotain väärää, joka sulkee pois mahdollisuuden pyytää häneltä apua myöhemmin, ja saattaa jopa johtaa taistelukohtaukseen toista osapuolta vastaan, jos pelaaja ei ole varovainen jatkossa.



Kuva 3. Mass Effect -pelin dialogisysteemin visuaalinen esitys keskustelun aikana. Jokainen valinta vaikuttaa tarinaan ja/tai sivuhahmon ja pelaajahahmon väliseen suhteeseen (Giant Bomb 2011).

Jennifer Brandes Hepler mainitsee teoksessa pelinarratiivin suunnittelusta, kuinka pelialalla mikään ei ole standardisoitua. Tämän vuoksi ihmisiä, joiden tehtävä on narratiivinen suunnittelu, kutsutaan eri työnimikkeillä eri peliyhtiöissä. (Heussner & al. 2015: chapter 1.) Tämä näkemys pätee myös tarinan luonteeseen peleissä. Tarina ei peleissä koostu pelkästään käsikirjoituksesta ja ääniraidoista, vaan useasta osasta, kuten kirjoitetusta narratiivista ja visuaalisista ja auditiivisista efekteistä. Myös tarinankerrontaa tukevat pelinkehitystyökalut voidaan tulkita laajakäsitteisesti tämän seurauksena. Animaatiotyökalutkin voitaisiin laskea tarinankerrontaa tukevien pelinkehitystyökalujen joukkoon.

2.2 Tarinankerrontaa tukevat pelinkehitystyökalut

Pelinkehityksessä käytetään monia erilaisia ja erikokoisia työkaluja, kuten pelimoottoreita, tekstuurinkäsittelytyökaluja ja projektinhallintatyökaluja. Tässä insinöörityössä tutkitaan pelialalla käytettäviä työkaluja ja selvitetään, mikä tekee työkalusta suositun. Ensin käydään pelinkehitystyökaluja yleisluontoisesti läpi, ja sitten siirrytään tarinankerronnan työkaluihin tarkemmin.

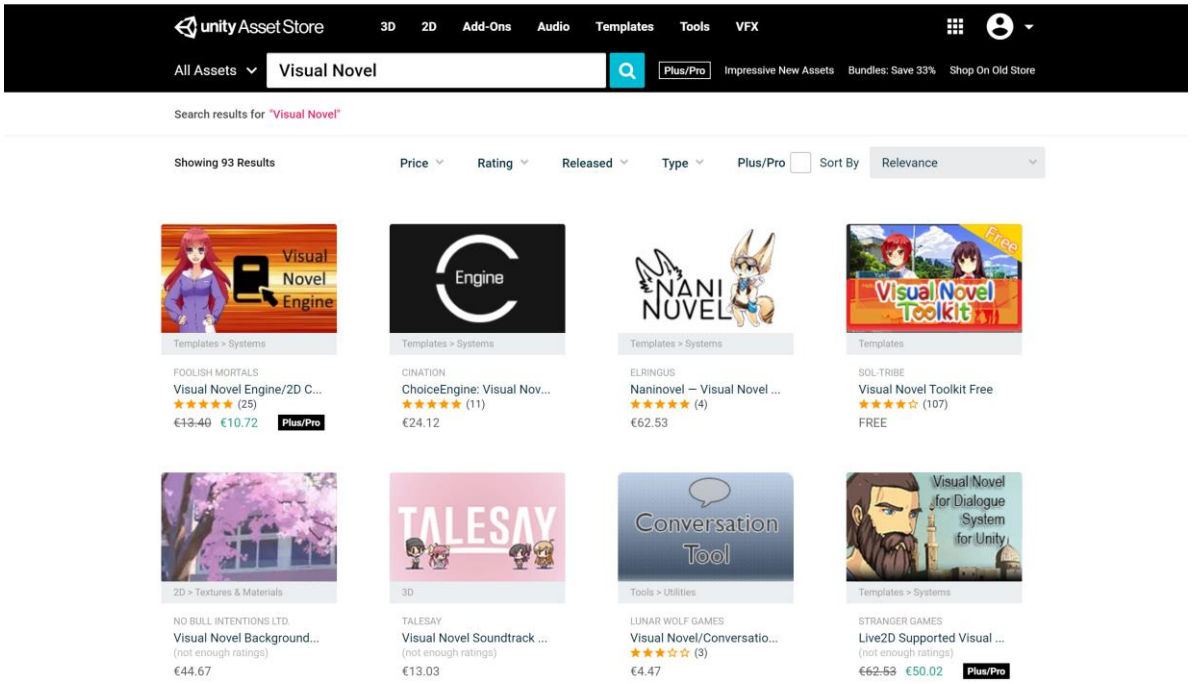
Pelimoottori on yleisimmin käytetty ja tunnetuin pelinkehityksen työkalu. Pelimoottori sisältää pohjarakenteen, jonka varassa peli toimii. Se hallitsee pelin komponentteja, kuten fysiikkamoottoria, tekstuurien, 3D-mallien ja animaatioiden käsittelyä sekä pelaajan syötteitä. Pelialan historian alkuvaiheessa pelistudiot kehittivät omat pelimoottorinsa peliensä vaatimusten mukaan. Osa pelistudioista kehittää edelleen omat pelimoottorinsa, mutta nykyään on saatavilla pelimoottoreiden valmistamiseen keskittyneiden yhtiöiden tarjoamia pelimoottoreita, jotka ovat vapaasti kaikkien halukkaiden saatavilla ja suurin osa pelimoottoreista on jopa ilmaisia. Näistä vapaasti saatavilla olevista pelimoottoreista Unity Technologiesin kehittämä Unity on yksi tunnetuimmista. Unityssä mielenkiintoinen ominaisuus on käyttäjän mahdollisuus ohjelmoida Unity Editorin sisäisiä skriptejä, joiden tarkoituksena on tukea pelin kehitystä. Skriptit voivat olla pieniä, kehitysprosessia nopeuttavia, tai ne voivat olla suurempia ohjelmistokehyksiä, jotka tuovat uusia ominaisuuksia pelin kokonaisuuteen, kuten verkkomoninpelikehys Photon Unity Networking (Photon Unity Networking).

Adobe Photoshop -kuvankäsittelytyökalu on toinen tunnettu pelinkehityksessä käytetty työkalu, mutta Photoshopin päätarkoitus ei ole pelinkehityksen tukeminen, vaan kuvankäsittely yleisesti. Sama koskee Blender Foundationin Blender-3D-mallinnusohjelmaa ja Autodeskin 3ds Max- ja Maya-ohjelmia. Nämä kaikki tunnetaan pelialalla, ja ne ovat usein myös alan työpaikkojen osaamisvaatimuksissa, mutta ne eivät lähtökohtaisesti ole vain pelialaa varten rakennettuja. Ainoat suoranaisesti pelien kehittämistä varten luodut työkalut ovat pelieditoreita ja -moottoreita, kuten aiemmin mainittu Unity, Epic Gamesin Unreal-pelimoottori, suoraan verkossa toimiva Twine-narratiivipelieditori ja Degican julkaisema RPG Maker -pelimoottorisarja. (Konik 2018.)

Pelialalla käytetään myös paljon verkkopohjaisia työkaluja, kuten projektinhallintaan ja tiimikommunikaatioon erikoistuneita ohjelmia. Esimerkkejä näistä ovat Atlassianin Trello- ja Jira-projektinhallintaohjelmat sekä Slack- ja Discord-kommunikaatiosovellukset. Nämä ohjelmat tukevat projektin rakennetta ja etenemistä sekä vahvistavat tiimidynamiikkaa ja yhteistyötä.

Koska tarinankerrontaa ei ole määritelty pelialan laajuisesti, opinnäytetyön puitteissa tulkinta tarinankerronnasta peleissä on seuraava: Kaikki pelin komponentit, jotka ovat suoranaisesti vaikutussuhteessa pelin narratiivin kanssa, lasketaan pelin tarinankerronnan komponenteiksi, esimerkkinä animaatiojärjestelmä, joka tukee ääninäyttelijöiden työtä tai välianimaatioita. Unity-pelimoottorin animaatioikkuna on tämän tulkinnan mukaan tarinankerrontaa tukeva työkalu.

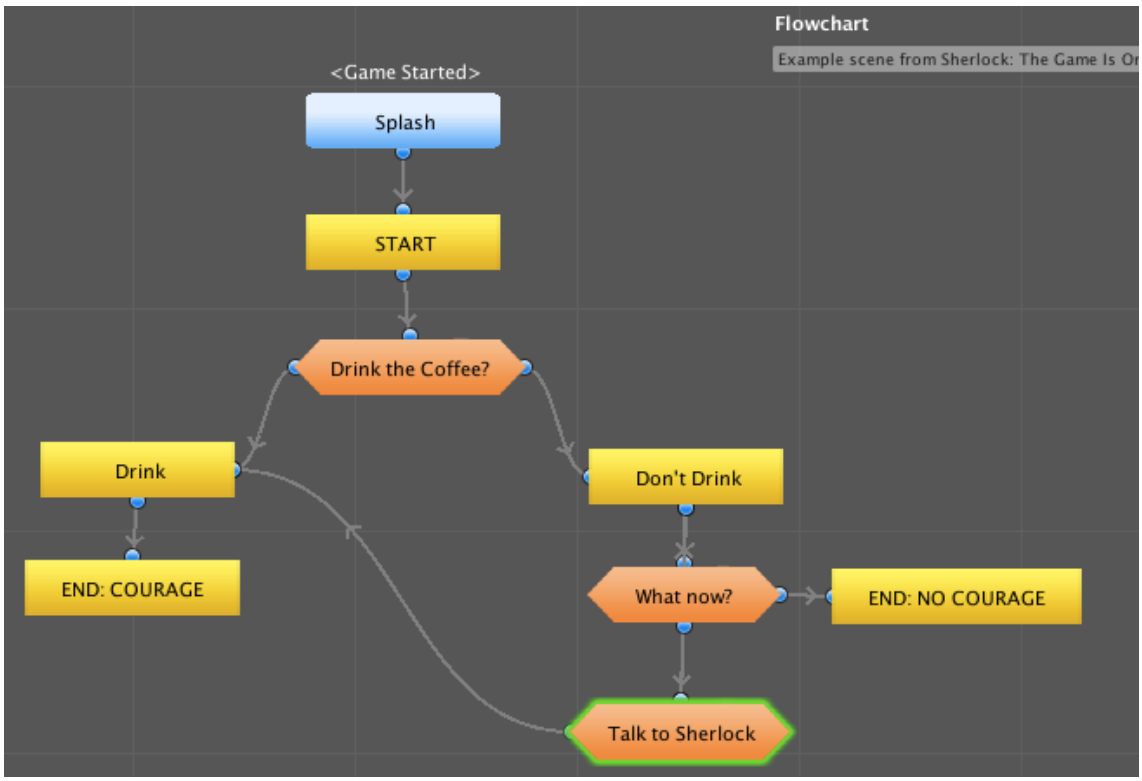
Tarinankerrontaa tukevia työkaluja on useita, etenkin Visual Novel -pelieditoreita ja -moottoreita, kuten avoimen lähdekoodin Ren’Py, Strikeworksin TyranoBuilder ja Degican Visual Novel Maker. Unityn sisäisiä tarinankerrontaa tukevia työkaluja löytyy Unity Asset Storesta, Unityn omasta kauppapaikasta, josta pelinkehittäjät voivat löytää pelinkehitystään tukevia kolmannen osapuolen työkaluja ja graafista tai auditiivista sisältöä peliinsä. Esimerkiksi Asset Storesta löytyy yli 90 osumaa hakusanalla ”Visual Novel” (kuva 4).



Kuva 4. Unity Asset Storen listanäkymä Visual Novel -hakusanalla (Unity Technologies 2019).

Yksi Unity Asset Storesta saatavilla olevista tarinankerrontaa tukevista työkaluista on Snozbotin tekemä avoimen lähteen Fungus, jonka tavoitteena on saada käyttäjät tekemään narratiivisia pelejä pelkällä visuaalisella skriptauksella, kuten narratiivisilla vuokaavioilla. Tämä ominaisuus on hyödyllinen sellaisille kehittäjille, joilla ei ole ohjelmointitaustaa.

Funguksen narratiivisella vuokaaviolla on huomattava etu siinä, että käyttäjät voivat visualisoida pelinsä narratiivisen virtauksen reaaliajassa. Tällä toiminnolla (kuva 5) on huomattavia samanlaisuuksia sen kanssa, miten Choose Your Own Adventure -kirjojen narratiivi toteutuu (kuva 2).



Kuva 5. Unity Editor -ikkuna, joka sisältää Fungus-vuokaaviotyökalun (Snozbot 2015).

Unity Asset Store on Unity-lisäosien kehittäjien suosima julkaisupaikka, koska Unity Editorin käyttäjät voivat helposti sovelluksen sisältä etsiä lisäosia omiin tarpeisiinsa. Lisäosat on myös helppo asentaa Asset Storen kautta vain yhtä painiketta painamalla.

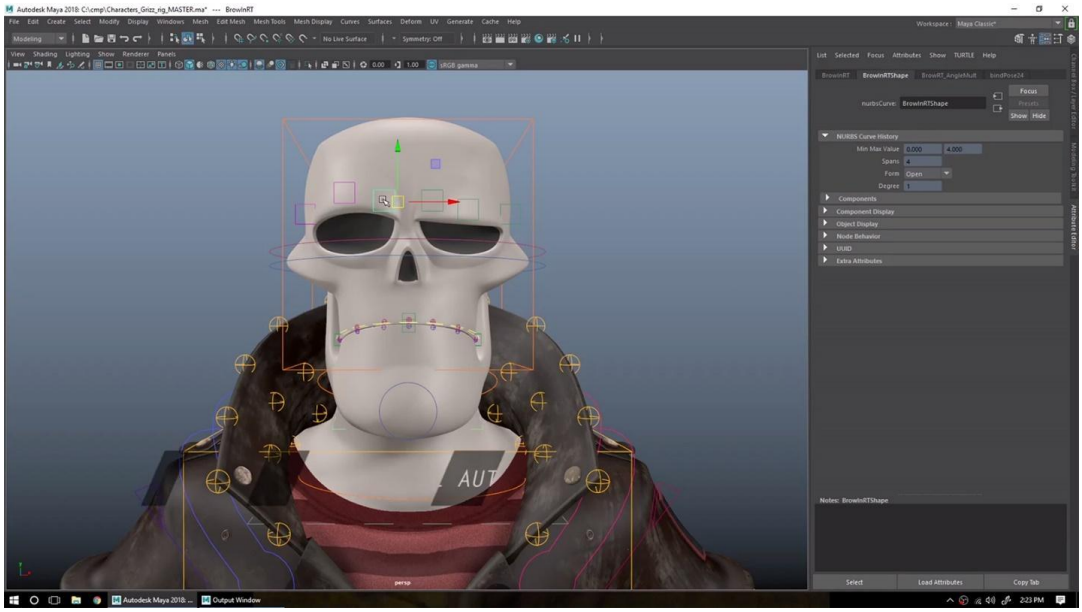
2.3 Unity Editor -työkalun käyttäjäkokemussuunnitteluprosessi

Unity Editorissa on mahdollista kehittää editorin sisäisiä lisäosia, joilla avustetaan pelin kehitysprosessia. Tätä ominaisuutta, nimeltään Editor-skriptaus (engl. Editor Scripting), käytetään tämän insinöörityön teknisessä toteutuksessa. Editor-skriptausta käytetään mm. ajan säästöksi esimerkiksi ohittamalla yksitoikkoiset, usein toistuvat tehtävät. Joskus Unity tai muut kolmannen osapuolen työkalut eivät riitä projektin tarpeisiin ja kehittäjien on keksittävä omia ratkaisuja ongelmiinsa (Sohail 2017).

Insinöörityön teknisen toteutuksen tavoitteena on tehdä editoriin ikkuna, jossa voi luoda esimerkiksi Non-player-character-olioita (NPC), joihin käyttäjä voi varastoida kaiken haluamansa datan, jonka käyttäjä itse määrittelee. Työkalun tavoitteena on helpottaa hahmopohjaisten narratiivisten pelien kehittäjien työtä vähentämällä ohjelmointitaitojen tarvetta ja nopeuttamalla olioiden luomista huomattavasti. Luvuissa 3 ja 4 syvennytään tarkemmin työkaluprojektin ominaisuuksiin. Tarkoituksena on perehtyä Unity Editor -skriptauksen mahdollisuuksiin, haasteisiin ja rajoitteisiin, ennen kuin projektin toteutus aloitetaan.

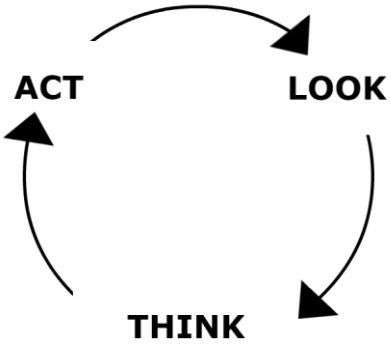
Projektissa otetaan käyttäjäkokemuspainotteinen (engl. User Experience, UX) suunnittelunäkökanta, joten opinnäytetyössä perehdytään syvällisesti pelinkehitystyökalun käyttäjäkokemussuunnitteluprosessiin ja parhaisiin menetelmiin. Opinnäytetyössä perehdytään myös työkalun ohjelmointiprosessiin sekä julkaisumahdollisuuksiin ja -prosessiin.

Käyttöliittymää suunnitellessa on otettava huomioon monta asiaa käyttäjäkokemussuunnittelijan näkökulmasta. Teoksessaan ”Designing the User Experience of Game Development Tools” David Lightbown (2015: 4) painottaa kahteen käyttäjäkokemuksen aspektiin: ”opittavuus” (engl. learnability) ja ”tehokkuus” (engl. efficiency). Esimerkiksi, jos käyttää pelkkiä kuvakkeita käyttöliittymän painikkeissa, saattaa säästää tilaa ja käyttäjä näkee enemmän omasta työstään, mutta ilman tekstiselitteitä painikkeille ohjelmisto on aluksi vaikeasti opittava (kuva 6). (Tokarev 2018).



Kuva 6. Autodeskin Maya-ohjelma, jonka käyttöliittymä jättää tilaa käyttäjän työn näyttämiselle ja moni painikkeista on ilmaistu pelkillä kuvakkeilla. Esimerkki tehokkaasta, mutta vaikeasti opittavasta käyttöliittymästä (Tokarev 2018).

Käyttäjäkokemussuunnittelijan työnkuvaan kuuluu usein tutkimustyötä ja testejä, kuten käyttäjähaastatteluja, A/B-testejä ja käyttötestejä. Käyttäjäkokemussuunnittelun alalla käytetään myös paljon toimintatutkimuksen alalla yleisiä käsitteitä, kuten Lightbownin teoksessa mainittu Kurt Lewinin alkuperäiseen konseptiin pohjautuva Look, Think, Act -sykli (kuva 7). Käyttäjäkokemussuunnittelijat pyrkivät optimoimaan käyttäjäkokemusta lisäämällä käyttäjän kuluttamaa aikaa act-vaiheessa sykliä ja vähentämällä look- ja think-vaiheissa käytettyä aikaa (Lightbown 2015: 43–45 & kuva 4.1.)



Kuva 7. Look, Think, Act sykli.

Kuvassa 7 kuvattu Look, Think, Act -sykli on Lightbownin teoksessa usein mainittu kuvio ja käyttäjäkokemuksen alue, joka otetaan toistuvasti huomioon. Look, Think, Act -sykliin tutustuminen ja sen optimoiminen omassa ohjelmistoprojektissa ovat käyttäjäkokemussuunnittelijan ensimmäisiä prioriteetteja.

3 Scriptable Object Editor -työkalun suunnitteluprosessi

Osana opinnäytetyötä suunniteltiin ja ohjelmoitiin pelinkehitystyökalu Unity-pelimoottorin sisäisesti käyttämällä Unity Editor -skriptausta. Toteutuksessa keskityttiin enemmän järjestelmän suunnitteluun, etenkin käyttäjäkokemussuunnitteluun.

3.1 Projektin vaatimukset ja tavoitteet

Scriptable Object Editor -työkalun konsepti syntyi tarpeesta tehdä hahmoja tulevaan peliprojektiin. Suunnitelmana oli toteuttaa peli Unityssä käyttämällä Fungus-työkalua insinöörityössä toteutetun työkalun ja käsin ohjelmoinnin rinnalla. Työkalun tavoitteena oli tehdä pelihahmojen luomisprosessista nopeampaa ja yksinkertaisempaa sekä helposti opittavaa mahdollisille uusille tiimin jäsenille. Työkalun käyttöliittymän suunnittelussa oli priorisoitava sen tehokkuus ja opittavuus minimoimalla turhia elementtejä käyttöliittymästä ja lisäämällä vain tarvittavat elementit.

Työkalu tehtiin Unitylle, koska peliprojekti oli tarkoitus toteuttaa Unityllä ja pelitiimillä oli eniten kokemusta Unityn kanssa työskentelystä. Tällöin työkalu on mahdollisimman hyödyllinen pelinkehitysprosessissa. Peliprojektissa oli myös tarkoitus käyttää muita Unitylle tehtyjä työkaluja, kuten Fungus-työkalua. Työkalun oli toimittava Funguksen rinnalla, mutta sen oli oltava tarpeeksi itsenäinen, ettei työkalun ainoa käyttöympäristö olisi Fungus-peliprojekteissa. Tavoitteena oli, että voisi tukea Fungus-vuokaavioikkunaa (kuva 5) mahdollistamalla hahmojen muuttujiin vaikuttaminen vuokaavioikkunassa. Työkalun toteutuksessa hyödynnettiin Unityn Scriptable Object -ominaisuutta, jolla voi tehdä olioita, jotka säilyttävät kaiken tarvittavan datan. Esimerkiksi jos käyttäjä haluaa tehdä örkkityyppisen vihollisen, hän voi työkalua käyttämällä luoda Scriptable Objectin, joka sisältää taulukon 1 mukaiset arvot.

Taulukko 1. Esimerkki Scriptable Objectin sisältämistä muuttujista. Taulukko kuvitteellisesta örkkityyppisestä viholliseta, jota kehittäjä voi tuottaa useita nopeasti tarvittaessa vain viittaamalla kyseiseen Scriptable Objectiin.

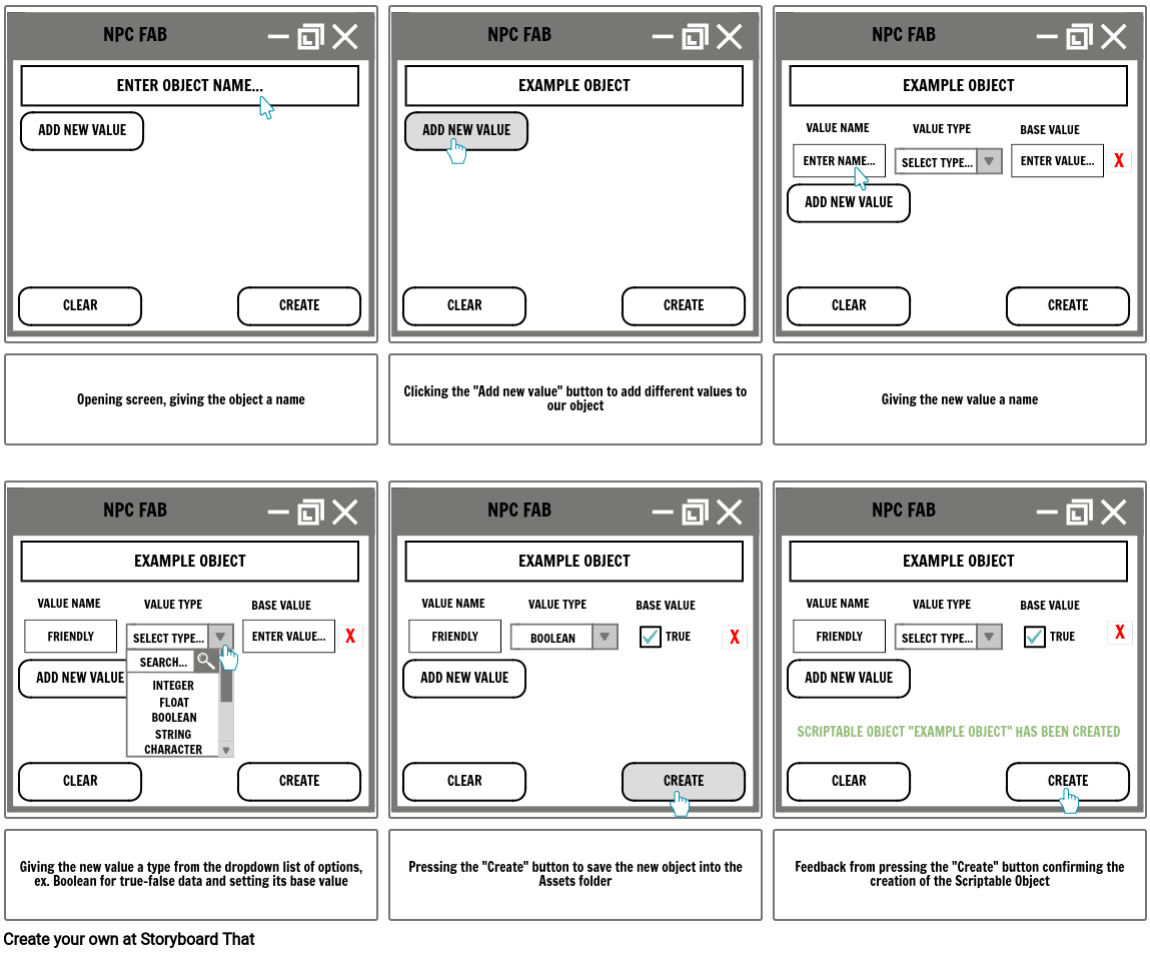
Muuttujan nimi	**Muuttujan tyyppi**	**Muuttujan arvo**
Nimi	String	Örkki
Elämät	Integer	100
Hyökkäyksen vahinko	Integer	10
Vihollinen	Boolean	Tosi

Ohjelmointikielenä toimi C#, koska se on ainoa Unityn tukema kieli. Unity lakkautti UnityScript- ja Boo-kielien tukemisen vuonna 2017 (Fine 2017). C#-kielen avulla voi käyttää Unityn valmiita luokkia, kuten EditorWindow-luokkaa, josta perimällä käyttäjä saa suoraan käyttöönsä kaikki Unity Editor -ikkunoille tarpeelliset muuttujat ja metodit. EditorWindow-luokka on joustava, ja sen avulla voi tehdä yksinkertaisimmista Unity Editor -ikkunoista monimutkaisimpiinkin työkaluihin, kuten Fungus-vuokaaviotyökalun (kuva 5).

Työkalulle suotavin julkaisumuoto oli avoimen lähdekoodin projektina GitHub-sivustolla, mistä moni avoimen lähdekoodin ohjelmistoprojekti löytyy. GitHub-sivuston kautta löydettävää projektia on myös helppo käyttää työnäytteenä mahdollisille työnantajille. Avoimen lähdekoodin projektina työkalu on kaikille ilmainen ja vapaasti muokattava käyttäjän omien tarpeiden mukaisesti, ja GitHub-sivustolla työkalun käyttäjät voivat antaa ehdotuksia uusiksi lisäyksiksi tai ongelmien korjauksiksi. Työkalua ei alustavasti ajateltu julkaista Unity Asset Storessa, koska opinnäytetyön puitteissa keskityttiin työkalun suunnittelemiseen. Jatkokehityksen myötä Unity Asset Store -julkaisua voidaan harkita uudelleen.

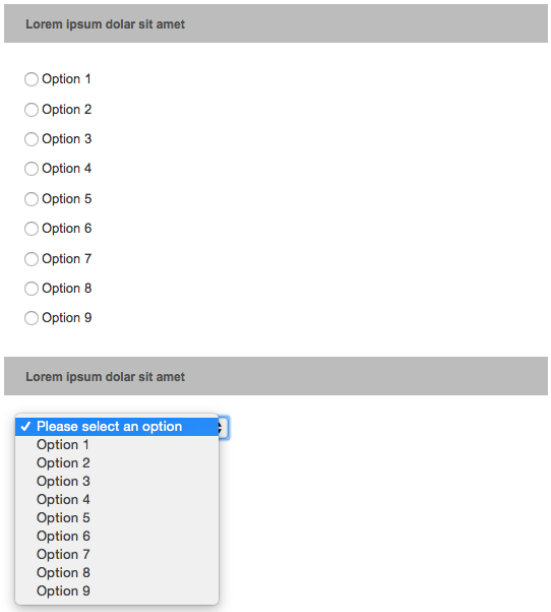
3.2 Käyttäjäkokemuksen ja käyttöliittymän suunnittelu

Käyttäjäkokemuksen ja käyttöliittymän suunnittelu aloitettiin luomalla ohjelmalle kuvakäsikirjoitus työkalun käyttäjäkokemuksesta (kuva 8), josta ilmenevät ohjelman ja käyttöliittymän pääominaisuudet ja tavoitteet. Kuvakäsikirjoitus visualisoi käyttöliittymän ilmettä ja toimintaa sekä tuo ilmi suunnitellut käyttäjäkokemusominaisuudet.



Kuva 8. NPC Fab (myöh. Scriptable Object Editor) -ohjelmakonseptin suunniteltua käyttökulkua (engl. user flow) kuvaava kuvakäsikirjoitus (Storyboard That 2019).

Ohjelman käyttöliittymää ja -kokemusta suunnitellessa on suositeltavaa, että suunnittelija seuraa alan suunnitteluohjeita ja parhaita käytäntöjä. Tällaisia voivat olla esimerkiksi Microsoftin asettamat ohjeet käyttöliittymän suunnitteluun (Lightbown 2015: 79). Esimerkiksi, jos haluaisi tehdä listan vaihtoehtoja, joista voi valita yhden vaihtoehdon kerrallaan, Microsoft ohjeistaa käyttämään valintanappeja, kun vaihtoehtoja on yhdestä seitsemään. Kun vaihtoehtoja on yli seitsemän, suositellaan avattavia luetteloita (kuva 9). (Kennedy & Satran 2018.)



Kuva 9. Valintanappien ja avattavan luettelon vertailukuva. Kuvan kontekstissa Microsoftin ohjeiden mukaan avattava luettelo olisi parempi vaihtoehto, koska vaihtoehtoja on listassa yhdeksän (UX Stack Exchange 2015).

Teoksensa luvussa 5 Lightbown käsittelee pelinkehitystyökalun käyttäjäkokemussuunnittelun aspekteja syvällisemmin. Luvussa oli paljon hyödyllisiä termejä ja käytäntöjä suunnitellun työkalun käyttäjäkokemuksen suunnittelun kannalta, kuten samanaiheisten asioiden ryhmittämistä yhteen (engl. grouping). Esimerkiksi ohjelmien työkalupalkeissa vaihtoehdot on ryhmitetty teemoittain, kuten tiedostoon liittyviin työkaluihin (tallennus, avaaminen, sulkeminen jne.). (Lightbown 2015: 97–102.)

Koska pelinkehitystyökalua tehtiin Unity-pelimoottorin sisäisesti, suunniteltiin käyttäjäkokemus Unityn käytäntöjen mukaisesti, mutta koska Unityn suunnitteluohjeita ja parhaita käytäntöjä ei ollut opinnäytetyön kirjoittamisen ajankohtana julkisesti saatavilla, jouduttiin erikseen tulkitsemaan jokaisen ongelmatilanteen kohdalla, miten Unityn kehittäjät toimisivat. Taustatukena käytettiin Microsoftin suunnitteluohjeita sellaisiin tilanteisiin, joihin ei heti löytynyt ratkaisua.

Suunnitteluohjeita seurattaessa on oltava tarkkana. Kuvan 8 ruudussa 4 käytetään avattavaa luetteloa muuttujan datatyypin valitsemiseen. C#-ohjelmointikieli tukee kuutta datatyyppiä, ja Microsoftin ohjeissa suositellaan radiopainikkeita alle kahdeksan vaihtoehdon kohdalla, jos ohjeita katsotaan radiopainikkeiden kohdalla. Mutta jos katsoo avattavan luettelon ohjeita, luettelon käyttöä suositellaan, jos tilan säästäminen on tärkeää. Koska käyttäjän on tärkeää saada selkeästi lisättyä useampia muuttujia oliolleen helposti luettavalla tavalla, päätettiin kuvakäsikirjoituksessa käyttää avattavaa luetteloa tilan säästämiseksi.

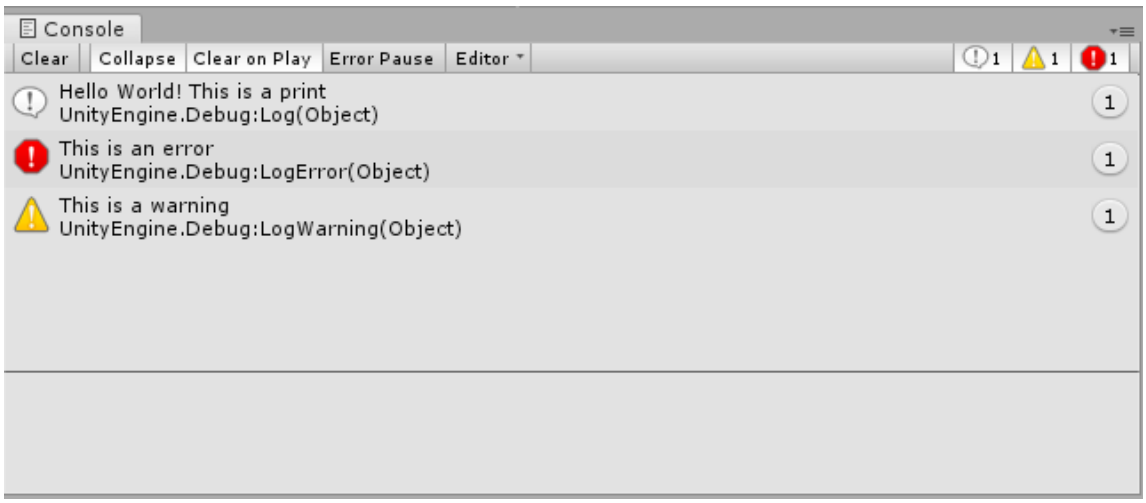
Kuvakäsikirjoituksen ei ole tarkoitus toimia lopullisena suunnitelmana työkalun käyttöliittymälle, vaan alustavana visualisoinnin avustajana. Esimerkiksi kuvan 8 ruudussa 6 ohjelma antaa käyttäjälle palautetta, kun käyttäjä painaa ”Create”-painiketta tallentaakseen olion. Palaute näkyy kuvassa vihreää tekstiä, joka ilmestyy ruudulle, kun tallennus on onnistunut, mutta se ei välttämättä ole selkein ja odotettavin tapa antaa palautetta ohjelman toiminnasta Unity-käyttäjälle. Unityssä voi C#-tiedoston kautta tulostaa Unityn Console-ikkunassa virheilmoituksia käyttämällä mm. esimerkkikoodin 1 koodirivejä. Näiden rivien tulokset näkyvät kuvassa 10.

```
void Start()
{
//Tekstin tulostus esimerkki
Debug.Log("Hello World! This is a print");

//Suorituksenaikaisen virheen tulostus esimerkki
Debug.LogError("This is an error");

//Suorituksenaikaisen varoituksen tulostus esimerkki
Debug.LogWarning("This is a warning");
}
```

Esimerkkikoodi 1. C#-koodirivit, joilla saa Unityn Consolen tulostamaan tavallisen tulostuksen (esim. ”Hello World”), suorituksenaikaisen virheen ja -varoituksen.



Kuva 10. Unityn Console-ikkuna, jossa näkyy erilaisia esimerkkitulostuksia, jotka on tuotettu koodin kautta (esimerkkikoodi 1) (Unity Technologies 2019).

Moni Unity-käyttäjä saattaa olla tottunut saamaan palautetta Unityn sisäisestä toiminnasta Unity Editorin Console-ikkunasta, mihin Unity kirjaa ohjelman suorituksenaikaiset virheet ja varoitukset sekä kehittäjän itse koodin kautta asettamat tekstin tulostukset (kuva 10).

3.3 Toteutuksessa hyödynnetyt ohjelmistot, menetelmät ja aineisto

Työkalu toteutettiin Unityn sisäisesti, koska Unity tarjoaa C#-kielelle monta valmista luokkaa ja toimintoa, jotka tukevat työkalun tuotantoa ja sen vaatimuksia. Esimerkiksi EditorWindow-luokka on Unityn sisäinen C#-luokka, joka auttaa käyttöliittymän tuottamisessa. Myös Scriptable Object -tyyppinen olio on Unity-pelimoottorille ainutlaatuinen ominaisuus.

Ohjelmointiin käytettiin Microsoftin Visual Studio -ohjelmankehitysympäristöä. Toinen vaihtoehto, Mono Develop, menetti Unity-tuen versiossa 2018.1 (Unity 2018). Visual Studio tukee kolmannen osapuolen lisäyksiä, jotka voivat olla hyödyllisiä tietyissä ohjelmointitilanteissa ja -tarpeissa, esimerkiksi JetBrainsin ReSharper-lisäosa, joka tehostaa koodin laaduntarkastusta ja varoittaa virhetilanteista koodissa reaaliajassa (JetBrains 2019).

Kirjallisena aineistona ja opasteena käyttäjäkokemuksen ja käyttöliittymän suunnittelemisessa käytettiin David Lightbownin teosta ”Designing the User Experience of Game Development Tools” ja Donald A. Normanin teosta ”Miten avata mahdottomia ovia? Tuotesuunnittelun salakarit”. Etenkin Lightbownin teoksesta oli hyötyä, koska se käsittelee juuri pelinkehitystyökalun käyttäjäkokemuksen suunnittelua.

Unity Editor -skriptauksen opasteena käytettiin Unityn dokumentaatiota aiheesta, kuten Editor Window -verkkokäsikirjaa. Videolähteinä käytettiin mm. Unityn esittelyvideota Scriptable Objecteista ja Youtube-kanavan Renaissance Coders -soittolistaa Unity Editor -skriptauksen alkeista. Ongelmatilanteissa käytettiin myös tukisivustoja, kuten Stack Overflow -sivustoa, jossa käyttäjät voivat esittää kysymyksiä ohjelmistotuotantoon liittyvistä asioista ja kokeneemmat käyttäjät voivat vastata kysymyksiin vapaasti. Näitä lähteitä yhdistämällä voi tuottaa toimivan työkalun alkuperäisen suunnitelman mukaisesti.

4 Scriptable Object Editor -työkalun tekninen toteutus

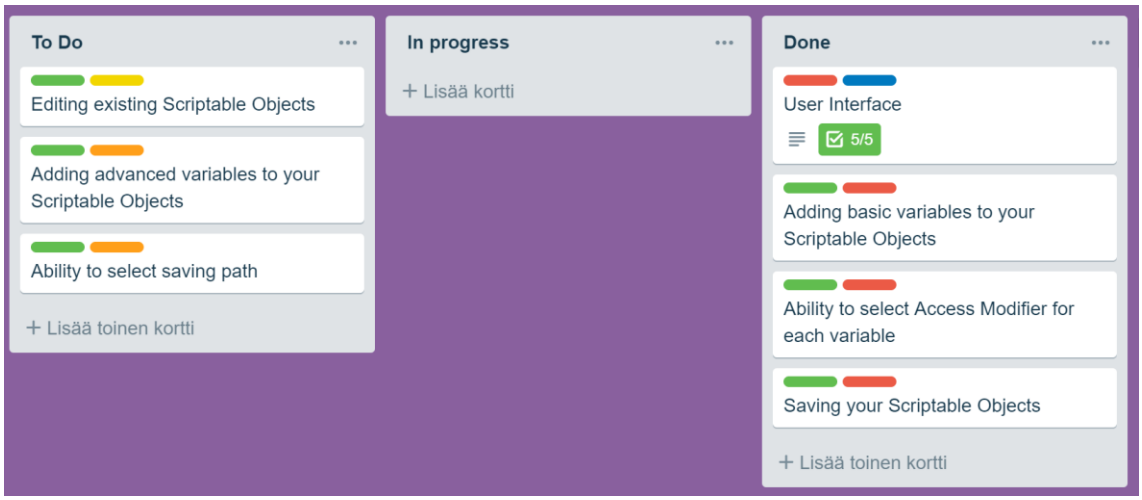
Projektisuunnitelman mukaan varattiin yksinomaan työkalun ohjelmoimiselle rajattu määrä aikaa. Tavoitteena oli rajoittaa projektin laajuutta ja lieventää riskiä, että projektiin lisätään opinnäytetyön viime hetkillä ominaisuuksia, jotka rikkovat työkalun toiminnallisuuden. Varaamalla aikaa yksinomaan ohjelmoimiselle varmistetaan myös se, että suurin osa ohjelmointityöstä saavutetaan varattuna aikana, mutta aikaa jää myös ongelmien korjaamiselle ja raportoimiselle.

4.1 Projektinhallinta ja etenemisen seuranta

Projektinhallinta on tärkeä osa ohjelmistotuotantoa. Insinöörityöprojektissa käytettiin Atlassianin Trello-projektinhallintaohjelmaa ja ylläpidettiin blogia projektin etenemisestä. Projektinhallinnan käyttö on projektille tärkeää, koska sitä käyttämällä kehittäjät voivat varmistaa, että projektin laajuus pysyy kehitystiimille sopivalla tasolla.

Trelloon kirjattiin maanantaina 8.4.2019 ohjelmointitehtävät. Trellossa on tarjolla useita ominaisuuksia, joilla projektin laajuutta voi pitää ajan tasalla ja etenemistä voi seurata. Trellossa luodaan projekteille tauluja, joihin koko tiimi voi lisätä listoja. Listoihin lisätään kortteja, joissa on esimerkiksi tehtäviä tai suunniteltuja ominaisuuksia. Kortit siirretään listasta toiseen, kun kortin sisältämä ominaisuus saadaan valmiiksi tai ominaisuuden tila muutetaan.

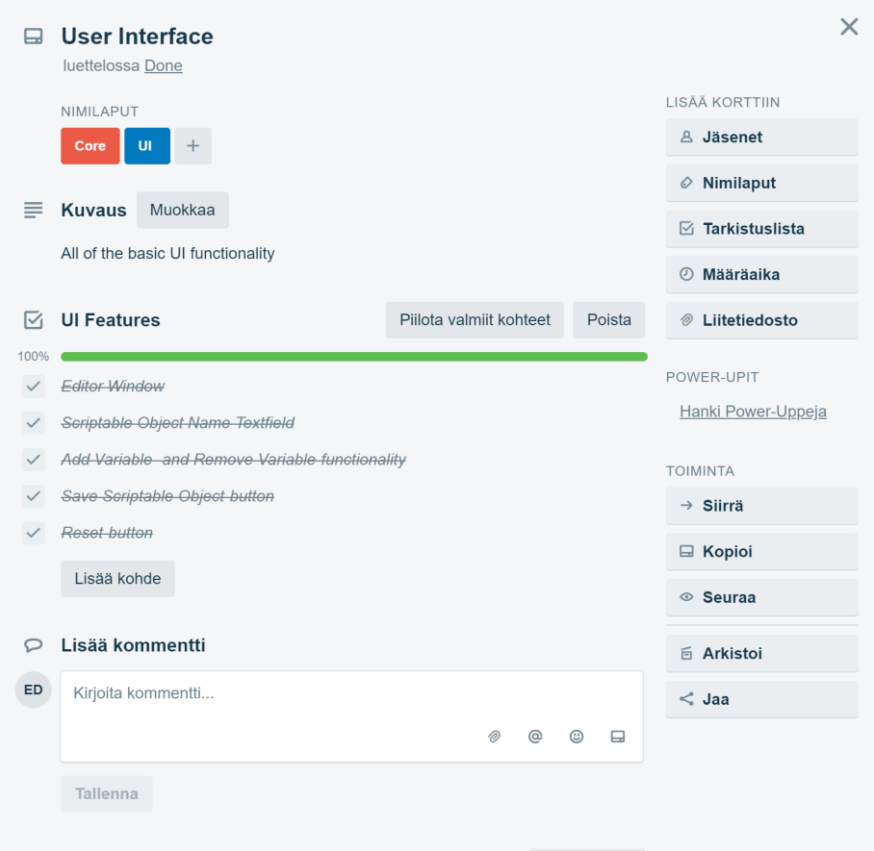
Projektin Trello-tauluun (kuva 11) lisättiin kolme listaa: To Do -lista tehtäville ominaisuuksille, In progress -lista työn alla oleville ominaisuuksille ja Done-lista valmiille ominaisuuksille. To Do -listalle luotiin kortteja, jotka edustavat suunniteltuja ominaisuuksia. Kun jokin tietty ominaisuus on työn alla, kortin voi siirtää To Do -listalta In progress -listalle ja sieltä Done-listalle, kun ominaisuus on saatu valmiiksi. Tämä auttaa pysymään ajan tasalla siitä, miten projekti etenee ja mitä on vielä tehtävissä. Vaikka tämä projekti oli ominaisuuslistaltaan ja tiimikooltaan hyvin pieni, projektinhallinta on tärkeä osa hyvää projektia.



Kuva 11. Scriptable Object Editor -työkalun Trello-taulu. Taulussa on listat tehtäville, työn alla oleville ja valmiille ominaisuuksille (Trello 2019).

Korteille voi antaa värikoodeja, jotka auttavat esimerkiksi priorisoimaan ominaisuuksia tärkeysjärjestyksessä. Projektin kortit värikoodattiin kahdella eri merkillä: Yhdet värimerkit kuvastivat ominaisuuden prioriteettitasoa. Punainen oli Core-ominaisuus eli elintärkeä ohjelmalle, oranssi oli Extended-ominaisuus eli tärkeä muttei pakollinen, ja keltainen oli Wishlist-ominaisuus eli jotain, mitä olisi hyvä lopulta saada ohjelmaan, mutta luultavasti ei ehdi valmiiksi opinnäytetyön aikana. Toiset värimerkit erottelivat ominaisuudet kategorioihin. Vihreä kuvastaa toiminnallisuutta ja sininen kuvastaa käyttöliittymää.

Trello-kortit itsessään sisältävät ominaisuuksia, jotka voivat olla hyödyllisiä, kuten mahdollisuus käyttää tarkistuslistaa, määräajan asettaminen tai tiettyjen tiiminjäsenten liittäminen korttiin. Tarkistuslista on hyödyllinen etenkin silloin, kun suunniteltuja ominaisuuksia on paljon, mutta ne voitaisiin tiivistää pienempään määrään kortteja käyttämällä tarkistuslistoja (kuva 12).



Kuva 12. Trello-kortti. Kortti edustaa työkalun käyttöliittymää ja sisältää nimilaput Core- ja käyttöliittymäluokitteluille sekä tarkistuslistan, joka erottelee kortin pienempiin käyttöliittymän ominaisuuksiin (Trello 2019).

Trellon hyödyllisyys tulee eniten ilmi suurissa tiimeissä, joissa tiimin jäsenet voivat löytää itselleen tehtäviä ja pysyä ajan tasalla projektin etenemisestä.

Versionhallintatyökaluna käytettiin Unityn sisäänrakennettua Collab-versionhallintalisäosaa. Versionhallintatyökalua käyttämällä pystyttiin työskentelemään usealla päätelaitteella siirtämällä päivittäiset muutokset ja lisäykset Unityn pilvipalvelimille. Versionhallinta on tarpeen etenkin suurissa projekteissa, joissa on paljon tiimijäseniä. Sitä käyttämällä projektitiimi voi varmistaa, että jokainen työskentelee ohjelman viimeisimmän version parissa.

4.2 Ohjelmoinnin vaiheet

Ohjelmointi aloitettiin Trello-taulun tekemisen jälkeen. Ensimmäisenä keskityttiin Editor-ikkunan luomiseen ja toiminnallisuuden varmistamiseen. Kun Editor-ikkuna oli sommiteltu, siirryttiin ohjelmoimaan työkalun olennaisin toiminnallisuus: muuttujien lisääminen ja Scriptable Object -olion tallentaminen. Lopulta keskityttiin työkalun julkaisemiseen GitHub-sivustolla.

4.2.1 Editor-ikkuna

Työkalun tuottamisen ensimmäinen vaihe oli Editor-ikkunan tekeminen. Unity Technologies on tehnyt Editor-ikkunoita varten valmiiksi C#-luokan, nimeltä EditorWindow. Tästä luokasta perimällä ohjelmoija voi helposti tuottaa Editor-ikkunoita omiin tarpeisiinsa. Käyttämällä EditorWindow-luokan sisäistä funktiota OnGUI ohjelmoija saa luotua käyttöliittymäelementtejä kuten painikkeita ja liukusäätimiä. Unityn GUI-järjestelmää käyttämällä voi myös sommitella käyttöliittymää eri ryhmiin käyttämällä Rect-tyyppistä muuttujaa (esimerkkikoodi 2).

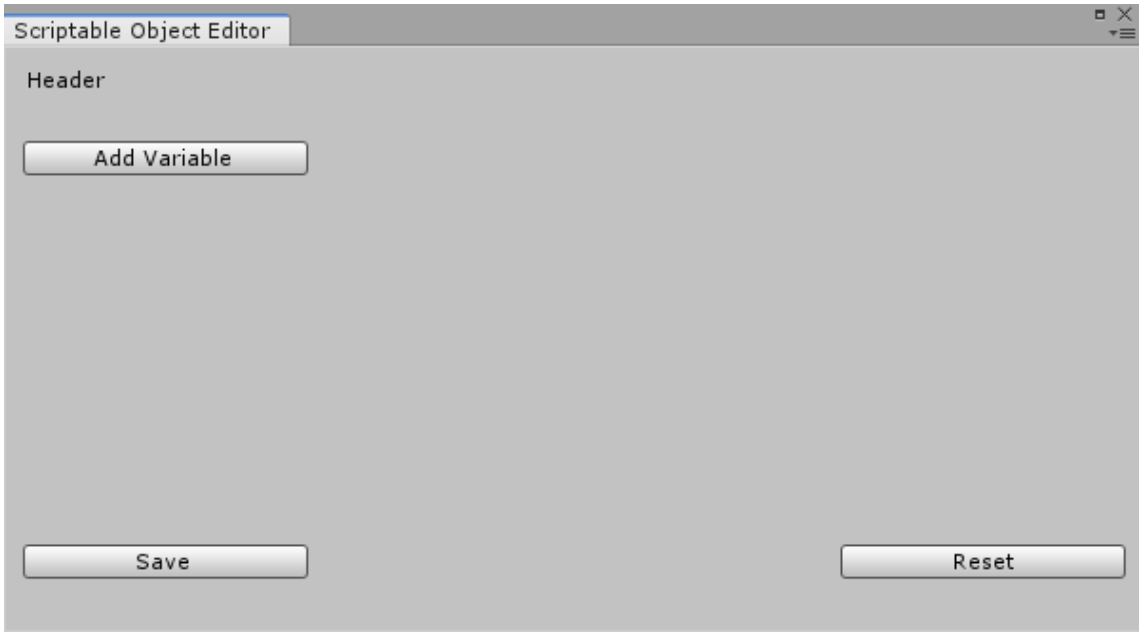
```
private void DrawLayouts()
    {
        headerSection = new Rect(10, 10, position.width-20, 40);

        bodySection = new Rect(10, 50, position.width-20, position.height-100);

        footerSection = new Rect(10, position.height-50, position.width-20, 40);
    }
```

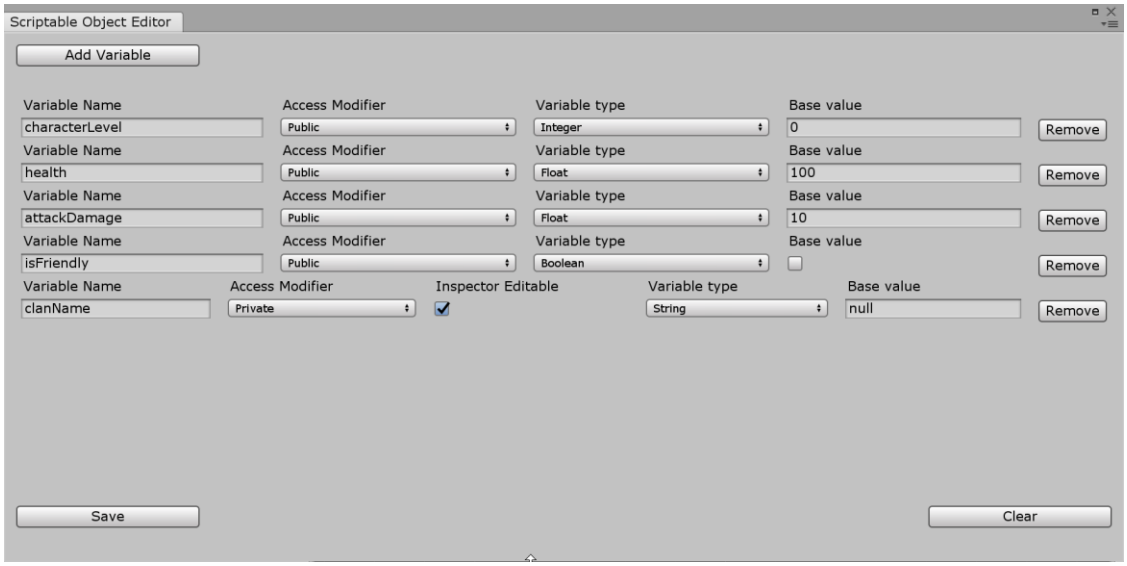
Esimerkkikoodi 2. Käyttämällä Rect-muuttujaa jaettiin Editor-ikkuna kolmeen osaan. Tämän avulla ikkunan koosta riippumatta näille ikkunan osille on aina varattu samankokoinen alue.

OnGUI-funktion sisällön selkeyttämiseksi tehtiin omia metodeja, jotka erottelevat OnGUI-funktion toiminnallisuuksia. Tämän jälkeen luotiin DrawLayouts-funktio, jonka avulla rajataan ikkuna kolmeen eri sektoriin, joihin pystyisi tulevaisuudessa lisäämään sektorille olennaiset käyttöliittymäelementit (kuva 13).



Kuva 13. Unity Editor -ikkuna ensimmäisen ohjelmointipäivän lopulla. Pääosin sommittelu on valmis, mutta painikkeissa ei ole toiminnallisuutta.

Ikkuna sommiteltiin yläryhmään, keskiryhmään ja alaryhmään. Yläryhmään tuli painike, jonka avulla käyttäjä voi lisätä uuden muuttujan. Tämä eroaa ensimmäisen päivän jälkeisestä tuloksesta (kuva 13) siinä, että Add Variable -painike on kuvassa keskiryhmässä. Keskiryhmään tuli lopulta työkalun tavoiteltu toiminnallisuus. Muuttujakenttä sisältää osiot muuttujan nimeämiselle, suojausmääreen valitsemiselle, tyypin valitsemiselle ja pohja-arvon asettamiselle ja lopuksi painike, jonka avulla muuttujan voi poistaa listalta. Suojausmääreen valinnasta riippuen muuttujakenttään voi tulla näkyviin osio, joka sisältää valintakentän, joka määrää sen, onko muuttuja muokattavissa Scriptable Object -instanssin Inspektorissa (kuva 14).



Kuva 14. Lopullinen Unity Editor -ikkuna, jossa on lisätty muutama esimerkkimuuttuja visualisoimaan työkalun toimintamallia.

Kuvista 13 ja 14 huomaa Save- ja Clear-painikkeiden sijoittelusta sen, että verrattuna kuvan 8 kuvakäsikirjoitukseen alkuperäisestä suunnitelmasta painikkeet ovat vaihtaneet paikkoja toistensa kanssa. Tämä sommittelupäätös tehtiin, koska silloin kuin Clear-painike on kaukana muista painikkeista, hiiren siirtämisessä painikkeelle on suurempi kognitiivinen kuorma. Clear-painikkeen kognitiivisen kuorman suurentamiseksi vielä enemmän painiketta painaessa ilmaantuu dialogi-ikkuna, joka kysyy, onko käyttäjä varma, että haluaa tyhjentää muuttujalistasta kaikki muuttujat. Tämä tarkoituksella hankaloittaminen pienentää käyttäjän virhemarginaalia ja estää käyttäjää turhautumasta ohjelmaan ehkäisemällä työn poistamisen vahingossa. Samalla Save-painike on lähellä sellaisia elementtejä, joiden kanssa käyttäjä työskentelee paljon. Tämä säästää aikaa ja lisää työkalun tehokkuutta, kun käyttäjän kuluttama aika act-vaiheessa look-think-act-syklissä (kuva 7) on suhteessa suurempi. (Lightbown 2015: 104–112.)

Kuvassa 14 myös Add Variable -painike on siirtynyt yläryhmään. Tämä päätös tuli sen jälkeen, kun työkalua testattiin ensimmäistä kertaa sellaisen henkilön kanssa, joka ei ollut työskennellyt työkalun parissa aikaisemmin. Testauksessa tuli ilmi, että käyttökokemuksen kannalta Add Variable -painikkeen paras sijainti olisi yläryhmässä, missä painike ei liikkuisi, kun muuttujalista kasvaa. Aikaisemmassa versiossa painike oli keskiosion viimeinen elementti, mikä tarkoitti sitä, että joka kerta, kun muuttujalistaan tuli uusi muuttuja, painike siirtyisi alemmas listassa. Tämä olisi ongelmallista työkalun tehokkuuden kannalta sellaisissa tapauksissa, joissa käyttäjä tietää ennalta, kuinka monta muuttujaa tarvitaan. Jos jokaisella painalluskerralla käyttäjä joutuu siirtämään hiirensä takaisin painikkeen ylle, tämä lisää kognitiivista kuormaa ja saattaa lisätä turhautumista työkaluun.

4.2.2 Työkalun toiminnallisuus

Ohjelman alkuperäisenä tavoitteena oli antaa käyttäjälle tapa luoda Scriptable Object -tiedostoja ilman ohjelmoimista. Ohjelman kehitysprosessin aikana opittiin, että Scriptable Object -tiedosto vaatii toimiakseen C#-tiedoston, jonka sisältämä luokka perii Unityn valmiista ScriptableObject-luokasta. Tämän seurauksena ilmeni, että työkalu ei voisi vain luoda Scriptable Object -tiedostoa suoraan, koska tiedosto vaatisi referenssin C#-tiedostoon. Ohjelman toiminnallisuus muuttui niin, että Save-painiketta painettaessa ohjelma generoi annetuista muuttujista C#-tiedoston, joka sisältää MenuObject-attribuutin, jonka avulla C#-tiedostoon referoivan Scriptable Object -tiedoston voi luoda suoraan Unityn Project-kansioon (esimerkkikoodi 3).

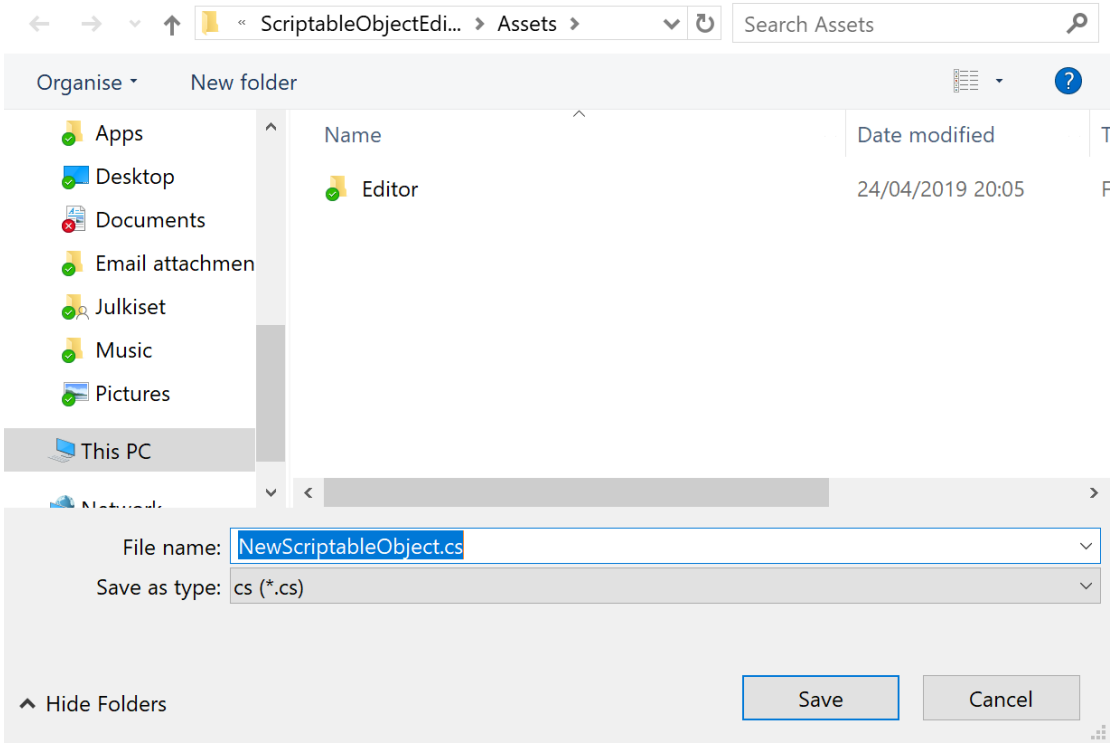
```
[CreateAssetMenu(fileName = "New New_Scriptable_Object", menuName = "New_Scrip-
table_Object")]
public class New_Scriptable_Object : ScriptableObject {

            public int ExampleVariable = 10;

}
```

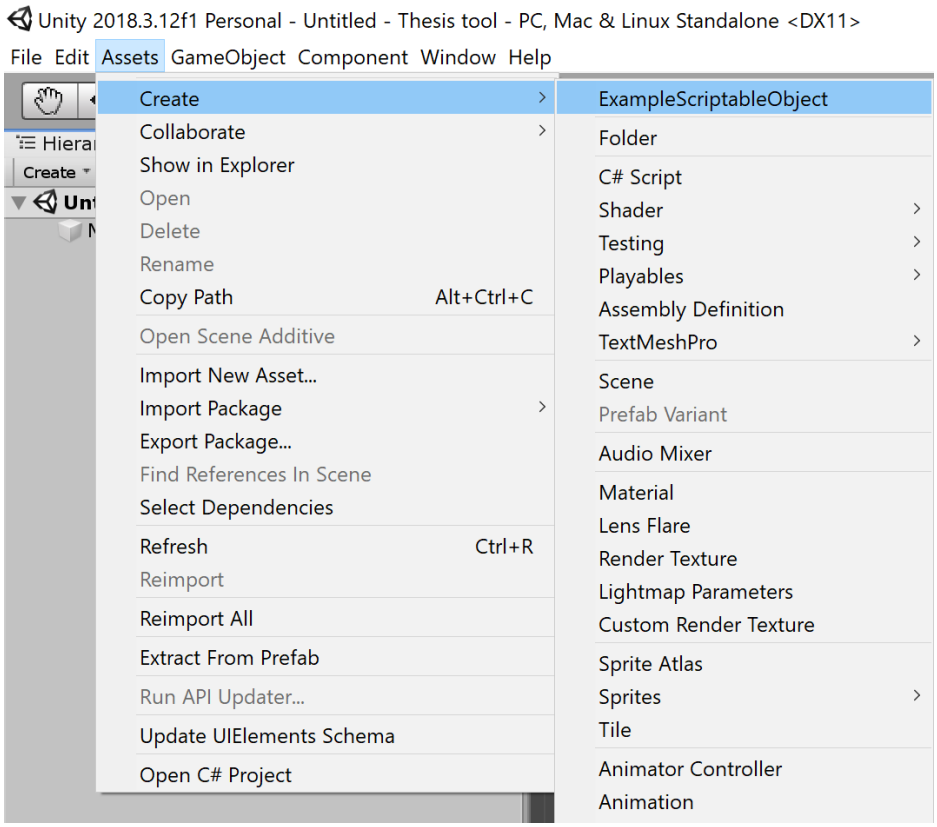
Esimerkkikoodi 3. Työkalulla luotu esimerkki-Scriptable Object -tiedosto, jossa on yksi esimerkkimuuttuja ja CreateAssetMenu-attribuutti Scriptable Object -instanssin luomiseksi.

Ohjelma myös avaa Save-painiketta painettaessa monelle tutumman dialogi-ikkunan (kuva 15), jossa tallennuspaikan voi valita ja lopullisen olion voi nimetä samassa ikkunassa. Tämä eroaa huomattavasti kuvan 8 kuvakäsikirjoituksesta, jossa alkuperäisen suunnitelman mukaan työkaluikkunan yläryhmässä olisi tekstikenttä olion lopulliselle nimelle. Käyttämällä yleisesti tunnettuja käyttöliittymäelementtejä vähennetään käyttäjän kokemaa rasitusta työkalua opetellessa.



Kuva 15. Työkalun tallennusdialogi-ikkuna.

Kun työkalun kautta luotu Scriptable Object -olio on tallennettu, on Unityn Asset-menussa uusi vaihtoehto luoda instanssi juuri luodusta Scriptable Object -oliosta. Tämän avulla käyttäjä voi esimerkiksi luoda Villager-olion työkalulla ja Asset-menusta (kuva 16) hän voi luoda Bob-, Rob- ja Hob-nimiset Villager-olion instanssit, joilla on omat ainutlaatuiset muuttujien arvot.



Kuva 16. Unity Editorin työkalupalkki. Esimerkki-Scriptable Object -tiedoston luomisvaihtoehto löytyy Assets/Create-valikosta (Unity 2019).

Työkalua käyttämällä voi yksinkertaistaa monimutkaisten oliohierarkioiden luomista. Esimerkiksi tietokoneroolipelien esinekirjaston luomisessa Scriptable Object -oliot voivat olla kovinkin hyödyllisiä pelinkehitystiimille. Usein toistuvalle oliotyypille, kuten esimerkiksi kyläläis-NPC:lle voi luoda Scriptable Object -olion, jonka kautta pelinkehittäjä voi helposti luoda tarvittaessaan kauppiaan tai tehtävänantajan, joka on Kyläläinen-tyyppiä mutta jonka arvot ovat erilaisia oletuskyläläiseen verrattuna. Esimerkiksi tehtävänantajan antaaTehtäviä-muuttuja voi olla tosi, kun oletuskyläläisen antaaTehtäviä-muuttuja on epätosi.

Tiedoston tallennuksessa käytetään Microsoftin StreamWriter-luokkaa, jota käytetään tekstitiedostojen luomiseen eri koodauksille ja tiedostopäätteille (Microsoft). Luokkaa käyttämällä täytetään kaikki Unitylle tarpeelliset tekstirivit manuaalisesti. C#-tiedoston kirjoittamiselle on muitakin menetelmiä, joista moni voi olla optimaalisempi, kuten System.Reflection.Emit-nimiavaruus, mutta puutteellisen ohjelmointikokemuksen ja rajallisen ohjelmoinnille varatun ajan vuoksi StreamWriter-luokka oli projektin tarkoitukseen sopivin.

Koska muutujatyyppejä oli useampia, ohjelman täytyy tarkistaa jokaisen muuttujan tietotyyppi ennen kirjoitusta. Tämä tehdään foreach-lausekkeella, joka käy läpi muuttujalistan ja listan jokaisen muuttujan kohdalla tekee switch-lausekkeen, joka tarkastaa muuttujan tyypin ja toimii muuttujatyypistä riippuen tietyllä tavalla (esimerkkikoodi 4).

```
using (StreamWriter outfile = new StreamWriter(copyPath))
{
    outfile.WriteLine("using System.Collections;");
    outfile.WriteLine("using System.Collections.Generic;");
    outfile.WriteLine("using UnityEngine;");
    outfile.WriteLine("");
    outfile.WriteLine("[CreateAssetMenu(fileName = \"New " + name + "\", menuName
= \"" + name + "\")]");
    outfile.WriteLine("public class "+name+" : ScriptableObject {");
    outfile.WriteLine(" ");

    //Check the variable type and whether it should be editable in the inspector
(if not public)
    foreach (var item in vars)
    {
        switch (item.type)
        {
        case VariableEnum.Integer:
        if (item.inspectorEditable)
        {
            outfile.WriteLine("\t[SerializeField]");
        }
        outfile.WriteLine("\t" + item.accessModifier.ToString().ToLower() + " int
" + item.name + " = " + item.intVal + ";");
        outfile.WriteLine("");
        break;
        //rest of switch…
        }
    }
outfile.WriteLine("}");
}
AssetDatabase.Refresh();
```

Esimerkkikoodi 4. Esimerkki StreamWriter-luokan käyttötavasta ohjelmassa. Sisältää yhden tyypin tapauksen switch-lausekkeesta esimerkkinä.

Tiedon väliaikaiseksi säilytysolioksi luotiin Variable-luokka, joka sisältää kaiken tarvittavan datan jokaisen työkalussa lisätyn muuttujan osalta (esimerkkikoodi 5). Luokasta tehdyt oliot säilytetään listassa, jonka kautta jokaista lisättyä muuttujaa käsitellään kerralla.

```
enum VariableEnum
{
    Integer,
    Float,
    Double,
    Boolean,
    Character,
    String
};

enum AccessModifierEnum
{
    Public,
    Private,
    Protected,
    Internal
};

private class Variable
{
    public AccessModifierEnum accessModifier;
    public bool inspectorEditable;
    public string name = "Enter name...";
    public VariableEnum type;
    public int intVal;
    public float floatVal;
    public double doubleVal;
    public bool boolVal;
    public char charVal = '0';
    public string stringVal = "Enter text...";
}
```

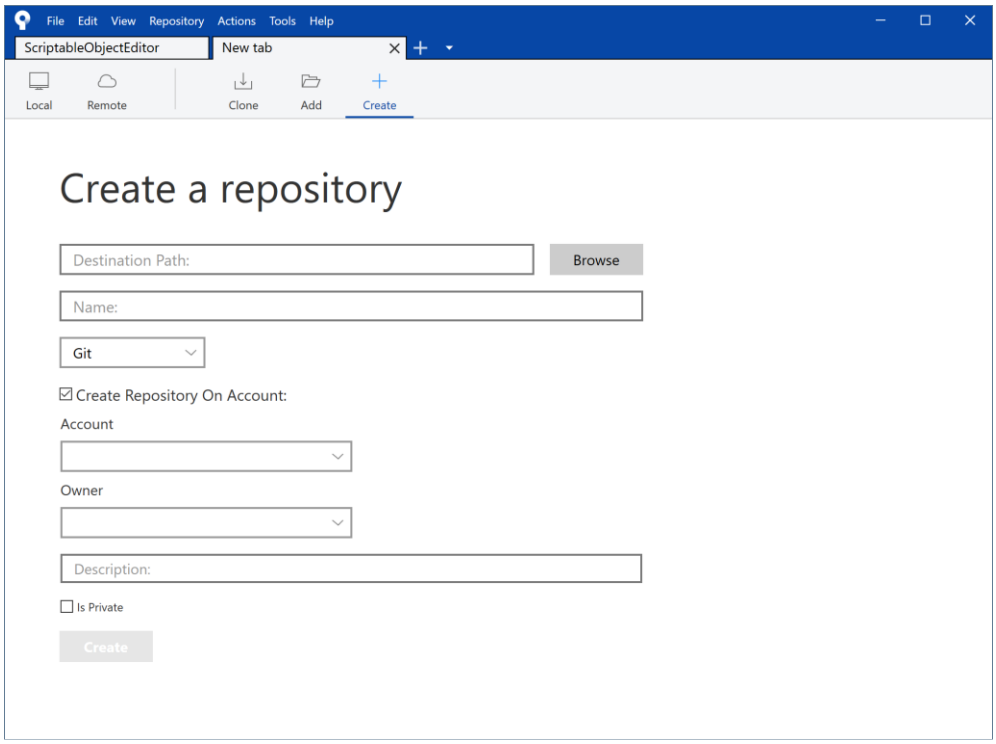
Esimerkkikoodi 5. Variable-muuttuja ja enumeraatiot muuttujan tyypille ja suojausmääreelle.

4.2.3 Työkalun julkaisu

Työkalun julkaisualustaksi valittiin GitHub-sivusto. Työkalu on ilmaiseksi saatavilla ja vapaasti käyttäjän muokattavissa. GitHub on ohjelmistotuotannossa usein käytetty versionhallintasivusto, jota tiimit käyttävät ohjelmavarastona. Se on myös tunnettu siitä, että ohjelmoijilla on henkilökohtaisia projekteja nähtävillä omassa GitHub-profiilissaan, jota he voivat käyttää portfolion tapaan työnhaussa. Samalla kun työkalu julkaistaan yleisesti tunnetulla ohjelmistotuotantoverkkosivulla, saadaan lisättyä uusi työnäyte omaan GitHub-profiiliin.

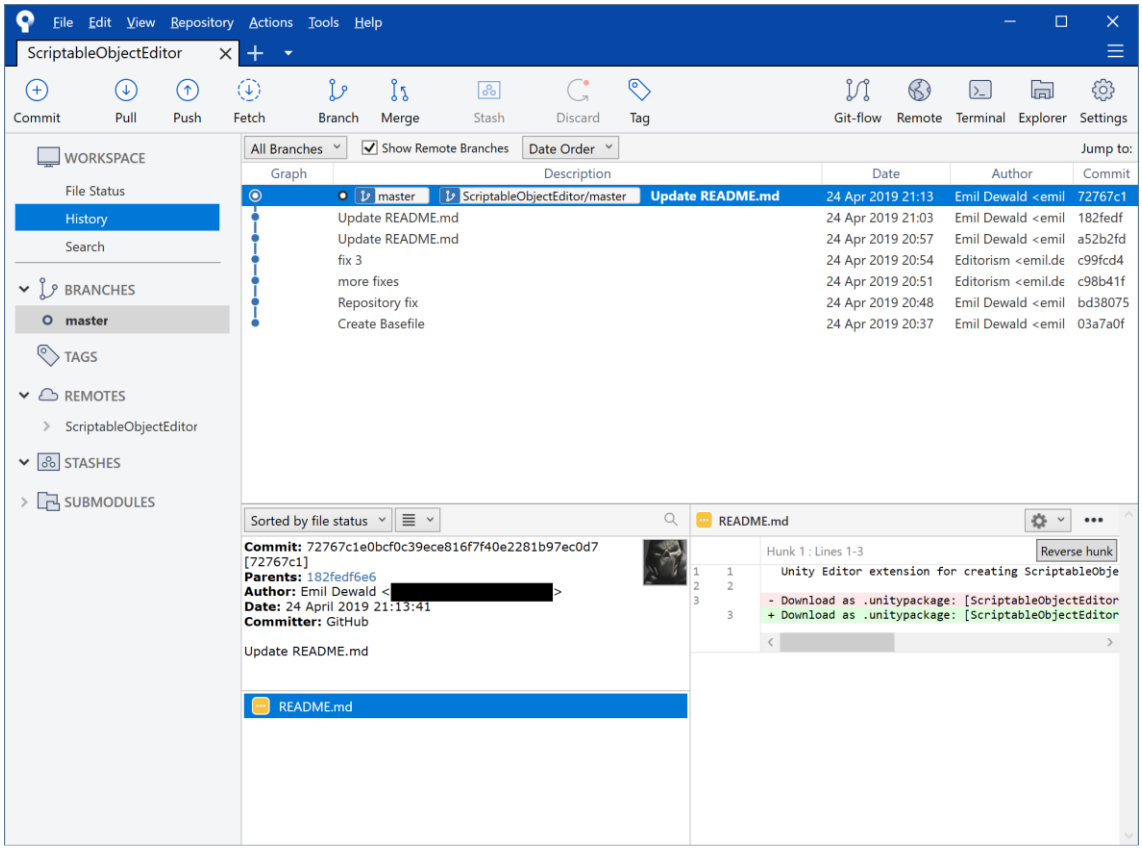
Unity-projektin julkaiseminen GitHubiin vaatii vain GitHub-profiilin ja Git-versionhallintaohjelman tai jonkin muun korvaavan versionhallintaohjelman. Git toimii komentoriviohjelmana, joten se vaatii tutustumista komentorivien toimintoon ja Gitin erilaisiin komentoihin. GitHubin kanssa toimivista versionhallintaohjelmista osassa on myös graafinen käyttöliittymä, esimerkiksi Atlassianin SourceTree-ohjelma. Projektissa päätettiin käyttää SourceTreetä, koska graafinen käyttöliittymä on selkeämpi käyttää.

SourceTreen asennus ei vaadi paljoa. Ohjelma on ladattavissa ilmaiseksi Atlassianin verkkosivustolla. SourceTree vaatii BitBucket-käyttäjätilin, jotta sen voi asentaa. BitBucket on Atlassianin oma versionhallintasivusto, mutta SourceTree sallii muiden vaihtoehtojen, kuten GitHubin, käytön. Asennuksen jälkeen käyttäjä voi lisätä esimerkiksi GitHub-käyttäjätilin. Tämän jälkeen luodaan repositorio (engl. repository, Gitin käyttämä nimitys datan säilytyspaikasta), joka yhdistetään GitHub-projektiin. SourceTreessä voi luoda verkkorepositorion samalla kun luodaan paikallinen repositorio (kuva 17). Repositorio luotiin Unity-projektin juurikansioon C:\Users\käyttäjänimi\Tiedostot\Unity Projects\ScriptableObjectEditor.



Kuva 17. Uuden repositorion luomistilanne SourceTreessä. Create Repository On Account -valintaa käytettäessä ohjelma luo myös halutulle käyttäjätilille repositorion verkkosivuille (esim. GitHub) (SourceTree 2019).

Kun repositorio on luotu, on aika julkaista ensimmäistä kertaa repositorion sisältö verkkoon käyttämällä Commit-vaihtoehtoa repositorionäkymässä (kuva 18). Kun Commit on tehty, on vielä siirrettävä vahvistetut muutokset verkkorepositorioon käyttämällä Push-vaihtoehtoa. Tämän jälkeen paikallinen (engl. local) ja etärepositorio (engl. remote) ovat ajan tasalla.



Kuva 18. ScriptableObjectEditor-projektin repositorionäkymä SourceTree-ohjelmassa (SourceTree 2019).

Lopuksi viimeisteltiin repositorion GitHub-sivu sopivaksi. Projektiin lisättiin kuvaus ja readme-tiedosto, johon lisättiin linkki, jonka kautta työkalun voi ladata unitypackage-tiedostomuodossa. Unitypackage-tiedostomuotoa käyttämällä saadaan tuotettua työkalusta helposti asennettava tiedosto. Unitypackage-tiedoston voi lisätä Unity-projektiin menemällä projektin työkalupalkissa [Assets>Import Package>Custom Package…] ja etsimällä lisättävä unitypackage-tiedosto tietokoneelta.

4.3 Haasteet ja esteet

Jokaisessa ohjelmointiprojektissa on omat haasteensa. Projektinhallintaohjelmia ja muita käytäntöjä, kuten dokumentaatiota tai projektinhallinnan viitekehyksiä (esim. Scrum), käyttämällä voi lieventää haasteiden vaikutusta projektin etenemiseen.

Opinnäytetyöprojektin aikana ilmeni useita haasteita. Suurin haaste oli projektisuunnitelmien ja -aikataulun tasapainottaminen taitotason kanssa. Suunnittelutyö keskittyi käyttäjäkokemukseen ja käyttöliittymään, ja tietorakenteen suunnittelu jäi puutteelliseksi. Tämän vuoksi, kun käyttöliittymä oli saatu ohjelmoitua ja oli aika ohjelmoida tietorakennetta ja toiminnallisuutta, projektissa tuli kriittinen ongelmatilanne ja ohjelmointityö ei edennyt yli vuorokauteen. Ongelman syy oli kokemattomuus Unityn ScriptableObject-luokan kanssa, etenkin tietämyksen puute Scriptable Object -olioiden toiminnallisuudesta ja niiden rajoitteista. Scriptable Object -olio Unityssä ei ole toiminnallinen ilman referenssiä C#-tiedostoon, jossa on luokka, joka perii ScriptableObject-luokasta.

Scriptable Object -olion luomisen sijaan täytyi siis luoda C#-tiedosto, johon laitetaan ScriptableObject-luokasta perivä oma luokka. Jouduttiin tutkimaan ratkaisua tälle tietorakenteen muutokselle, ja ongelman tutkimus lopulta johti C#-kielessä olevaan System.Reflection.Emit-nimiavaruuteen, jota käytetään metodien ja Type-luokkien dynaamisesti luomiseen (Microsoft). System.Reflection.Emit-nimiavaruutta käyttämällä yritettiin dynaamisesti luoda ScriptableObject-luokasta perivää Type-luokkaa, jonka pohjalta tehtäisiin Scriptable Object -olio Unityyn, mutta C#-ohjelmointitaitojen taso ja ymmärrys System.Reflection.Emit-nimiavaruudesta ei ollut tarpeeksi korkea, jotta sitä voisi käyttää toimivasti.

Ratkaisu tähän ongelmaan löytyi C#-kielen StreamWriter-luokasta, jota käyttämällä työkalu luo C#-tiedoston, joka täytettiin merkkijonoilla. Lopputulos on ScriptableObject-luokasta perivä luokka, joka sisältää kaikki käyttäjän antamat muuttujat ja niiden arvot.

Toiseksi haasteeksi ilmeni projektin etenemisen säännöllinen päivittäminen blogiin. Blogin kirjoittaminen oli tärkeä osa etenemisen seurantaa, ja se auttoi myös opinnäytetyön kirjoittamisessa varmistamalla, että jokaisen päivän tapahtumat olisivat tuoreesta muistista kirjoitettu blogiin ja sieltä otettavissa opinnäytetyöhön. Ohjelmointiprosessin alussa blogia saatiin päivitettyä säännöllisesti, mutta kun C#-luokan luomiseen liittyvä ongelma ilmaantui, päivittäminen jäi tekemättä usean päivän ajan. Lopulta saatiin päivitettyä kuluneista tapahtumista, mutta tieto aiemmilta päiviltä ei ollut enää tuoreessa muistissa. Onneksi ohjelmoimiseen varattu aika oli rajattu, joten tämän ongelman vaikutus oli lievä.

Haasteiden ilmeneminen oli välttämätöntä projektissa, mutta siihen oli valmistauduttu. Haasteista huolimatta työkalu saatiin julkaisuvalmiiksi. Jokaisen haasteen vaikutusta voi lieventää käyttämällä sopivia menetelmiä.

5 Projektityön tulokset

Scriptable Object Editor -työkalu julkaistiin GitHubiin, ja viimeinen opinnäytetyön aikainen päivitys repositorioon tehtiin 25.4.2019. Kaikki työkalun Trello-taulussa olevat Core-merkityt vaatimukset saatiin tehtyä, ja toiminnallisuus on varmistettu.

Scriptable Object Editor -työkalun avulla voi luoda joustavasti erilaisia Scriptable Object -pohjia sekä narratiivisiin että muihinkin peleihin. Koska työkalun tavoitteena on tukea narratiivisten pelien tuottamista, käsitellään tässä kohtaa työkalun hyödyt narratiivisten pelien tuotannossa. Työkalua käyttämällä voi esimerkiksi luoda pohjan hahmoille, joiden kanssa pelaaja voi keskustella ja jotka voivat esimerkiksi antaa pelaajalle tehtäviä, jotka vievät tarinaa eteenpäin. Työkalulla käyttäjä voi luoda Scriptable Objectin nimeltä storyNPC, jossa voi olla esimerkiksi int-tyyppiä oleva muuttuja ”playerOpinion”, joka muuttuu pelin sisällä sen mukaan, miten pelaaja käyttäytyy.

Toinen esimerkki työkalun käyttötavasta narratiivisessa peliprojektissa olisi luoda tarinakortteja. Hyvä esimerkki tarinakortista voisi olla Paradox Interactive -peliyhtiön historialliset strategiapelit, joissa pelin aikana näytölle saattaa ilmestyä tapahtuma, jossa on pelaajalle kiinnostavaa tarinaa, ja usein muutama vaihtoehto, jotka vaikuttavat pelaajan valitseman valtion tai hahmon tilanteeseen (kuva 19).

Kuva 19. Crusader Kings 2 -pelin tapahtumakortti, jossa pelaajalle tarjotaan useita vaihtoehtoja (Paradox Interactive 2018).

Scriptable Object Editor -työkalua käyttämällä voisi tehdä storyEventCard-nimisen Scriptable Object -olion, joka sisältää string-tyyppisen muuttujan kortin tarinatekstille ja string-tyyppiset muuttujat jokaiselle vaihtoehdolle.

Työkalu on saatavilla GitHub-sivustolla: <https://github.com/Editorism/ScriptableObjectEditor>.

Työkalulla saavutettua tehokkuutta mitataan Scriptable Object -olioiden luomisessa tekemällä pienimuotoinen koe, jossa mitataan ennalta määritellyn olion luomiseen kulunut aika. Seuraavilla metodeilla saavutettua aikaa verrataan: Scriptable Object Editor -työkalua käyttämällä ja käsin luomalla C#-tiedoston.

Taulukossa 2 määritellään kokeessa luotava olio magicRing.

Taulukko 2. magicRing Scriptable Object -olion tietorakenne. “Muokattava editorissa” -kohdassa “-“ tarkoittaa, että kohta ei ole olennainen muuttujalle tai sitä ei ole saatavilla.

Muuttujan nimi	Suojausmääre	Muokattava editorissa	Muuttujan tyyppi	Oletusarvo
objectName	public	-	string	”null”
isEquippable	public	-	bool	true
lvlRequirement	public	-	int	1
strBonus	public	-	int	0
dexBonus	public	-	int	0
conBonus	public	-	int	0
intBonus	public	-	int	0
wisBonus	public	-	int	0
chaBonus	public	-	int	0
effectA	public	-	string	”null”
effectB	public	-	string	”null”
rarity	protected	kyllä	int	1
totalInWorld	private	ei	int	0

Vaikka koe, jonka avulla taulukon 3 tulokset saavutettiin, oli pienimuotoinen ja todellisten ja luotettavien tulosten saavuttamiseksi kokeita pitäisi ottaa laajemmalla otoksella, saavutetut tulokset näyttävät työkalun potentiaalin työn tehokkuuden nostamisessa. Kokeessa tuli ilmi, että Scriptable Object Editor -työkalun käyttäminen oli suurin piirtein 50 prosenttia tehokkaampi metodi luoda Scriptable Object -olioita kuin niiden käsin ohjelmoiminen.

Taulukko 3. Ajanmittauskokeen tulokset.

Metodi	Aika (min)
Scriptable Object Editor	02:20
Käsin ohjelmoiminen	03:31

Scriptable Object Editor -työkalun selkeimpiä rajoitteita on se, että työkalu tukee tällä hetkellä vain tavanomaisia muuttujia. Unity-pelimoottorissa on monia muuttujia ja komponentteja, joiden tukeminen lisäisi työkalun tehokkuutta. Näiden muuttujien ja komponenttien tuen lisääminen työkaluun ei onnistunut insinöörityölle asetetun aikarajan seurauksena, mutta se on jatkokehityksen prioriteettilistan huipulla. Työkalua rajoittava tekijä on myös se, että Scriptable Object -olioihin on joskus tarpeen tehdä funktioita, mutta työkalun avulla funktion tekeminen ei luultavasti lisäisi työkalun tehokkuutta.

Kuvan 19 tyyppistä oliota simuloivaa Scriptable Object -oliota luodessa työkalun rajoitteet ovat selkeät. Siinä tilanteessa työkalu hyötyisi edellä mainituista lisäyksistä, sillä lisäämällä funktioita Event-olioon voisi tapahtumakortista tehdä itsenäisesti toimivan olion, esimerkiksi vaihtoehdon valintafunktiolla ja lisäämällä muuttujan olioon, joka olisi tyyppiä Sprite. Sprite-muuttujan avulla voisi tarinakorttiin liittää kuvatiedoston, kuten kuvan 19 mallissa.

Jatkokehityksen mahdollisuudet

Vaikka projektin tulokset ovat positiiviset, Scriptable Object Editor -työkaluun on vielä paljon mahdollisia lisäyksiä ja parannuksia. Opinnäytetyön aikana ohjelmoin Trello-taulusta (kuva 10) vain Core-ominaisuudet. Tauluun jäi useita Extended- ja Wishlist-ominaisuuksia, joista moni olisi hyödyllinen, etekin muuttujien tietotyyppien tuen laajentaminen myös Unityn sisäisiin muuttujiin, kuten fysiikkamoottoriin (esim. Rigidbody), tai renderöintiin (esim. Material) liittyvien muuttujien tuen lisääminen.

Unity Asset Store -kauppapaikkaan julkaisemista voisi harkita jatkokehityksen myötä, riippuen osittain myös siitä, hyötyvätkö käyttäjät Scriptable Object Editor -työkalusta GitHubin kautta. Jos työkalu kerää huomiota, voisi olla hyödyllistä nähtävyyden kannalta lisätä se ilmaiseksi Unity Asset Store -kauppapaikkaan, josta Unity-käyttäjät voivat sen suoraan ladata ja lisätä projektiinsa.

6 Yhteenveto

Opinnäytetyössä perehdyttiin pelinarratiiviin ja pelinkehitystyökaluihin, etenkin narratiivista työtä tukeviin työkaluihin, kuten Visual Novel -pelieditoreihin. Pelinarratiivin määritelmää rajattiin opinnäytetyön puitteissa ja eroteltiin peleissä ilmenevän narratiivin eri muodot visuaalisiin, auditiivisiin ja kirjallisiin sekä niiden yhdistelmiin. Pelinarratiivin historiaan tutustuttiin lyhyesti Choose Your Own Adventure -kirjasarjasta Mass Effect -pelisarjan dialogisysteemin toteutukseen. Narratiivin määritelmän rajoittaminen oli haasteellista, koska narratiivin käsite on tulkinnan varassa. Narratiivin käsite rajattiin sellaiseen muotoon, jossa se olisi mahdollisimman relevantti opinnäytetyön kontekstissa.

Unity-pelieditoriin tulevan työkalun ohjelmointiprosessia tutkittiin eri näkökannoista, ja insinöörityössä perehdyttiin pelinkehitystyökalujen käyttäjäkokemuksen suunnitteluun olennaisimpana lähteenä David Lightbownin teos Designing the User Experience of Game Development Tools. Teoksesta otettiin neuvoa työkalun suunnittelussa tärkeisiin käyttäjäkokemukseen vaikuttaviin asioihin, ja teoksen sekä Microsoftin käyttöliittymäsuunnitteludokumentoinnin tukemana suunniteltiin työkalun käyttökulkua kuvaava kuvakäsikirjoitus. Kuvakäsikirjoitus näytti vain alustavasti, miten työkalu voisi toimia, ja lopulta työkalun käyttöliittymä muuttuikin, joskin vain hieman. käyttäjäkokemussuunnittelutyöstä opittiin huomattavan paljon käyttäjäkokemussuunnittelualan käsitteistä ja menetelmistä sekä käyttöliittymäsuunnittelun ansoista ja parhaista menetelmistä.

Työ toteutettiin C#-kielellä käyttämällä Unity-pelimoottorissa esiintyviä, valmiiksi tehtyjä luokkia, kuten EditorWindow-luokkaa ja ScriptableObject-luokkaa. Ohjelman toteutusvaiheessa tuli kriittinen ongelmatilanne, jossa ilmeni, että ohjelman suunniteltu tietorakenne ei olisi toiminut ScriptableObject-luokan kanssa. Työkalun tietorakennetta jouduttiin muokkaamaan niin, että ohjelma toimisi suunnitellusti. Tämän tuloksen saavuttamiseksi jouduttiin muuttamaan tapaa, jolla työkalu tallentaa käyttäjän syöttämät tiedot. Tämä toimi hyvänä muistutuksena siitä, että taustatutkimustyö on erittäin tärkeä osa ohjelmistotuotantotyötä. Jos taustatutkimusta olisi tehty syvällisemmin, olisi tämä ongelma huomattu, ennen kuin se pysäytti ohjelmoinnin etenemisen.

Insinöörityön tuloksena syntyi toimiva työkalu, jolla Scriptable Object -olioita voi luoda ilman, että käyttäjä joutuu itse ohjelmoimaan C#-kielellä. Tuloksia kokeiltiin pienimuotoisella nopeusvertailulla, jonka tulokseksi ilmeni suurin piirtein 50 prosentin tehokkuusparannus Scriptable Object -olion käsin ohjelmoimiseen verrattuna. Työkalu julkaistiin avoimen lähdekoodin projektina GitHub-sivustolle, josta se on jokaisen saatavilla ilmaiseksi ja vapaasti muokattavissa.

Lähteet

A Very Brief History of Storytelling. 2012. Verkkoaineisto. Big Fish Presentations. <https://bigfishpresentations.com/2012/02/28/a-very-brief-history-of-storytelling/>. Luettu 4.3.2019.

Beauregard, Mike. 2017. Left wall of the hall of bulls. Verkkoaineisto. <https://www.flickr.com/photos/31856336@N03/35992500056/>. Luettu 4.5.2019

Buckner, Adam. 2016. Introduction to Scriptable Objects. Verkkoaineisto. Unity Technologies ApS. < https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/scriptable-objects>. Luettu 16.5.2019.

Crusader Kings 2 tapahtumakortti. 2018. Verkkoaineisto. Paradox Interactive AB. <https://forum.paradoxplaza.com/forum/index.php?attachments/saints-ancestors-10-png.405694/>. Luettu 7.5.2019.

Fine, Richard. 2017. UnityScript's long ride off into the sunset. Verkkoaineisto. Unity Technologies ApS. <https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/>. Luettu 7.5.2019.

Fungus Unity Flowchart Window. 2015. Verkkoaineisto. Snozbot. <http://files.snozbot.com/games/fungus/docs/flowcharts/index.html>. Päivitetty 20.6.2015. Luettu 15.3.2019.

Heussner, Tobias & al. 2015. The game narrative toolbox. E-kirja. Focal Press.

History of CYOA. Julkaisuaika tuntematon. Verkkoaineisto. Chooseco LLC. <https://www.cyoa.com/pages/history-of-cyoa>. Luettu 16.5.2019.

Kennedy, John & Satran, Michael. 2018. Guidelines. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/fi-fi/windows/desktop/uxguide/guidelines>. Päivitetty 31.5.2018. Luettu 26.3.2019.

Konik, James. 2018. The Best Game Development Tools: How to Make Your Own Game. Verkkoaineisto. Cloudwards. <https://www.cloudwards.net/best-game-development-tools/>. Luettu 11.3.2019.

Lamilami, Eden. 2018. Create Your Own Adventure Book As Directed Graph Writing Flow Chart Flowchart. Verkkoaineisto. Arrowscan. <https://www.arrowscan.com/book-writing-flow-chart/2008-03-07-choose-your-own-adventure-book-as-directed-graph/>. Luettu 10.3.2019.

Lightbown, David. 2015. Designing the user experience of game development tools. E-kirja. CRC Press.

Mass Effect -pelin dialogijärjestelmä. 2011. Verkkoaineisto. Giant Bomb. <https://www.giantbomb.com/images/1300-1699009>. Luettu 6.3.2019.

Norman, Donald A. 1991. Miten avata mahdottomia ovia? Tuotesuunnittelun salakarit. Espoo: Weilin + Göös.

NPC Fab main function storyboard. 2019. Verkkoaineisto. Storyboard That. <https://www.storyboardthat.com/storyboards/emil96865/npc-fab-main-function-storyboard/>. Luettu 26.3.2019

Paczkowski, Lukasz. 2018. Replacing MonoDevelop-Unity with Visual Studio Community starting in Unity 2018.1. Verkkoaineisto. Unity Technologies ApS. <https://blogs.unity3d.com/2018/01/05/discontinuing-support-for-monodevelop-unity-starting-in-unity-2018-1/. Luettu 3.4.2019.

Photon Unity Networking: Photon Unity 3D Networking Framework SDKs and Game Backend. Verkkoaineisto. Exit Games, Inc. <https://www.photonengine.com/en/pun>. Luettu 15.5.2019.

ReSharper: The Visual Studio Extension for .NET Developers by JetBrains. Verkkoaineisto. JetBrains s.r.o. <https://www.jetbrains.com/resharper/>. Luettu 4.4.2019.

Scriptable Object Editor -ikkunan ensimmäinen versio. 2019. Unity Editor. 2005. Unity Technologies ApS.

Scriptable Object Editor -ikkunan toinen versio. 2019. Unity Editor. 2005. Unity Technologies ApS.

Scriptable Object Editor -työkalun tallennusdialogi-ikkuna. 2019. Unity Editor. 2005. Unity Technologies ApS.

Scriptable Object Editor -työkalun Trello-kortti. 2019. verkkoaineisto. Trello, Inc. <https://trello.com/c/cFot2iBv/1-user-interface>. Luettu 21.4.2019.

Scriptable Object Editor -työkalun Trello-taulu. 2019. verkkoaineisto. Trello, Inc. <https://trello.com/b/mPmnSsEv/scriptable-object-editor>. Luettu 21.4.2019.

Sohail, Asad. 2017. Unity Editor Scripting (A kick-starter guide) – Part 1. Verkkoaineisto. Informa PLC. <http://www.gamasutra.com/blogs/AsadSohail/20170502/297226/Unity_Editor_Scripting_A_kickstarter_guide__Part_1.php>. Luettu 13.3.2019.

SourceTreen Create a Repository-näkymä. 2019. SourceTree. 2013. Atlassian Corporation Plc.

SourceTreen Repository-näkymä. 2019. SourceTree. 2013. Atlassian Corporation Plc.

StreamWriter Class. Vuosi tuntematon. Microsoft Corporation. <https://docs.microsoft.com/fi-fi/dotnet/api/system.io.streamwriter?view=netframework-4.8>. Luettu 19.4.2019.

System.Reflection.Emit Namespace. Vuosi tuntematon. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/api/system.reflection.emit?view=netframework-4.8>. Luettu 24.4.2019.

Tokarev, Kirill. 2018. Autodesk Maya UI. Verkkoaineisto. 80 Level. <https://cdn.80.lv/80.lv/uploads/2018/07/image8-4.jpg>. Luettu 13.3.2019.

Tokarev, Kirill. 2018. User Experience of Game Development Tools Field. Verkkoaineisto. 80 Level. <https://80.lv/articles/user-experience-of-game-development-tools-field/>. Luettu 13.3.2019

Unity Asset Store. 2019. Verkkoaineisto. Unity Technologies ApS. <https://assetstore.unity.com/search?q=Visual&q=Novel&k=Visual%20Novel>. Päivitetty 28.1.2019. Luettu 15.3.2019.

Unity Console -ikkuna. 2019. Unity Editor. 2005. Unity Technologies ApS.

Unity Editor Scripting. Verkkoaineisto. Renaissance Coders. <https://www.youtube.com/watch?v=Osf1LUFGSvg&list=PL4CCSwmU04MiCnps1DRmwIEEH7gP9X3qq&index=1>. Luettu 16.5.2019.

Unity Editorin Assets/Create-valikko. 2019. Unity Editor. 2005. Unity Technologies ApS.

Valintanappien ja avattavan listan käyttötapausten erot. 2015. Verkkoaineisto. UX Stack Exchange. <https://i.stack.imgur.com/UCzsc.png>. Luettu 26.3.2019.