SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

SAVONIA-PELI ANDROIDILLE

TEKIJÄ: Milla Koivisto

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Milla Koivisto	
Työn nimi Savonia-peli Androidille	
Päiväys 22.5.2015	Sivumäärä/Liitteet 24/0
Ohjaaja(t) lehtori Jussi Koistinen, lehtori Keijo Kuosmanen	
Toimeksiantaja/Yhteistyökumppani(t) Savonia-ammattikorkeakoulu, viestintäpäällikkö Petteri Alanko	

Tiivistelmä

Opinnäytetyön aiheena oli tehdä peliprototyyppi Savonia-ammattikorkeakoululle. Projektin tavoitteena oli saada aikaan peliprototyyppi, jota voitaisiin tulevaisuudessa mahdollisesti käyttää Savonian markkinointiin.

Projekti aloitettiin pelin ideoimisella ja suunnittelulla, minkä jälkeen koodattavat ominaisuudet jaettiin kahdeksi erilliseksi opinnäytetyöksi, joista toisen teki Sini Mustonen. Projektin alussa valittiin tarvittavat työkalut, kuten Unity, Visual Studio sekä Git, ja tutustuttiin niiden käyttöön, minkä jälkeen aloitettiin itse projektin työstäminen.

Lopputuloksena saatiin aikaan peliprototyyppi, jonka kehitystä aiotaan jatkaa vielä lähitulevaisuudessa. Lisäksi suunnitteilla on projektin dokumentointi mahdollisia muita jatkokehittäjiä varten.

Avainsanat
Unity, Peli, Android, Versionhallinta, Git, C#

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Milla Koivisto			
Title of Thesis Savonia Game for Android			
Date	22 May 2015	Pages/Appendices	24/0
Supervisor(s) Mr. Jussi Koistinen, Lecturer and Mr. Keijo Kuosmanen, Lecturer			
Client Organisation /Partners Savonia University of Applied Sciences			

Abstract

The purpose of this thesis was to make a game prototype for Savonia University of Applied Sciences. In addition to making a prototype that could be later used in marketing Savonia, the goal of this project was to learn how games can be made.

The beginning of the project included brainstorming for ideas and planning the game. After that, it was split into two separate theses written by Milla Koivisto and Sini Mustonen, respectively. Next, tools were chosen for the project, including Unity, Visual Studio and Git. Time was spent on learning how to use the tools in question and then the practical part of the project was started.

As a result of this thesis a game prototype was created. There are plans to continue its development in the near future, along with writing documentation for the benefit of possible future developers.

Keywords
Unity, Game, Revision control, Git, C#

ESIPUHE

Haluan kiittää Savonia-ammattikorkeakoulua ja viestintäpäällikkö Petteri Alankoa mielenkiintoisesta opinnäytetyön aiheesta. Kiitokset myös lehtori Jussi Koistiselle neuvoista ja kärsivällisestä ohjauksesta.

Kuopiossa 22.5.2015
Milla Koivisto

SISÄLTÖ

TERMIT JA LYHENTEET

Asset

Projektin sisältämät grafiikat, äänet ja skriptit joita pelimoottori pystyy hyödyntämään.

GameObject

GameObjectilla tarkoitetaan Unityssa käytännössä mitä tahansa pelin objektia, esimerkiksi vaikkapa hahmoa. Toimiakseen se tarvitsee sisälleen komponentteja kuten esimerkiksi skriptejä.

IDE (Integrated Development Environment)

Integroitu ohjelmointiympäristö, joka sisältää ohjelmistojen suunnitteluun ja toteutukseen käytettäviä työkaluja.

NPC (Non-player-character)

Pelin sisäinen tekoälyllä varustettu ystävällinen hahmo, jota pelaaja ei voi ohjata.

Quest

Pelaajalle annettava tehtävä, joka voi sisältää yhden tai useamman tavoitteen.

1 JOHDANTO

Työn tilaaja on Savonia-ammattikorkeakoulu, jonka yhteyshenkilönä toimii markkinointipäällikkö Petteri Alanko. Opinnäytetyön aiheena on luoda Android-alustalle Savonia-aiheinen peliprototyyppi, jota voisi myöhemmin mahdollisesti käyttää koulun markkinointiin uusille oppilaille. Työn toissijaisena tavoitteena on opetella pelien tekemistä ja siihen tarvittavien työkalujen käyttöä.

Työ pohjautuu aiemmin mobiiliprojektikurssilla aloitettuun Android-peliin, joka jäi ajan puutteessa raakileeksi. Projektista tehdään kaksi opinnäytetyötä. Tässä työssä keskitytään hahmojen, quest-systeemin sekä dialogien suunnitteluun ja toteutukseen. Toisen opinnäytetyön tekijä on Sini Mustonen ja hänen osa-alueensa ovat käyttöliittymä, inventory-systeemi, pelin tallennus sekä hahmojen kontrollointi.

2 TYÖKALUT JA TEKNIIKAT

2.1 Unity

Unity 5 on Unity Technologiesin pelinkehitysympäristö, jolla on mahdollista luoda sekä kaksi- että kolmiulotteisia pelejä useille eri alustoille. Siitä on olemassa maksullisen Professional Editionin lisäksi ilmainen Personal Edition, jota tässä projektissa käytettiin. Molemmat sisältävät kaikki pelimoottorin ominaisuudet, kuten grafiikoiden, animaatioiden, fysiikoiden ja äänten käsittelyn. Pelin rakentaminen tapahtuu käytännössä Unity Editorilla, jota pystyy kustomoimaan sekä koodaamalla itse käyttäen apuna siihen tarkoitettua rajapintaa että lataamalla Asset Storesta joko ilmaisia tai maksullisia laajennuksia. Pelimoottori tukee kolmea eri skriptikieltä: C#:a, UnityScriptiä ja Boo:ta. (Unity Technologies a. 2015-05-22.)

Kehitysympäristöksi valittiin Unity, koska se on ilmainen ja laajalti käytössä, erityisesti indie- ja harrastelijapiireissä. Tämän vuoksi sillä on kattavan dokumentaation lisäksi internetissä laaja yhteisö, mistä voi olla paljon apua ongelmien ilmaantuessa. Myös julkaisualustojen runsaus luettiin suureksi hyödyksi, ja erityisesti mahdollisuus julkaista useille mobiililaitteille vaikutti päätökseen.

2.1.1 GameObject

Pelien luominen Unityssa pohjautuu GameObjecteihin. Ne ovat rakennuspalikoita, jotka eivät itsessään sisällä funktionaalisuutta, vaan tarvitsevat komponentteja toimiakseen. Esimerkiksi Transform-komponentti määrittää objektin sijainnin ja orientaation. Kaikki pelin objektit, kuten hahmot, ympäristö ja valaistus, ovat GameObjecteja. (Unity Technologies b. 2015-05-22.)

2.1.2 MonoBehaviour

MonoBehaviour on Unityn oma luokka, jonka kaikki Unity Editorissa luodut skriptit perivät automaattisesti, ellei sitä manuaalisesti muuteta. Se tuo perivälle luokalle lisäominaisuuksia, kuten julkisten muuttujien arvojen muokkaamisen Unityn Inspector-editorissa sekä Unityn tapahtumafunktiot. MonoBehaviouria tarvitaan, kun halutaan liittää skripti komponentiksi GameObjectiin. (Unity Technologies c. 2015-05-22.)

2.1.3 ScriptableObject

ScriptableObject on luokka, josta voidaan periä vaadittaessa objekteja, joita ei tarvitse liittää GameObjectiin. GameObjecteihin liittämisen sijaan tämän luokan tyyppisistä olioista voidaan tehdä assetteja. Näiden assettien luomiseen ei ole valmiita editoreja, vaan ne täytyy luoda aina itse.

ScriptableObjectia kannattaa käyttää luokissa, joiden tarkoitus on pitää tallessa paljon dataa, joka ei riipu ScriptableObjectia käyttävän skriptin instanssista. Tämä säästää muistia käyttämällä serialisoidessa kopioiden sijasta referenssejä. Toisin sanoen, jos useassa skriptissä käytetään samaa

ScriptableObjectin instanssia (eli assettia) samalla datalla, skripteihin tallennetaan vain viittaus kyseiseen assettiin sen sijaan, että sama asset kopioitaisiin jokaiseen sitä käyttävään skriptiin erikseen. (Unity Technologies d. 2015-05-22.)

2.2 Kenney Game Assets

Kenney Game Assets on joukko pelistudio Kenneyn tekemiä assetteja, jotka ovat yhteensopivia Unity-pelimoottorin kanssa. Tässä projektissa on käytetty vain kyseisen paketin ilmaisia, Creative Commons CC0 -lisenssillä varustettuja osia, jotka ovat ladattavissa Kenneyn verkkosivuilta. Pelin visuaalinen ilme on luotu näiden assettien avulla. (Kenney 2015-05-26.)

2.3 Visual Studio 2012

Visual Studio on Microsoftin lisensoima ohjelmistojen kehitysympäristö (IDE), jonka avulla voidaan luoda ohjelmia esimerkiksi työpöytä-, palvelin-, mobiili- ja pilviympäristöihin. Visual Studio tukee C++, C# ja VisualBasic -ohjelmointikieliä. (Microsoft b. 2015-05-20.)

Projektin alussa käytettiin Unityn mukana tulevaa MonoDevelop-ympäristöä, mutta se vaihdettiin nopeasti Visual Studioon, kun selvisi, että sen integrointi Unityyn ei ole vaikeaa. Tärkein syy tähän ohjelmointiympäristön muutokseen oli se, että MonoDevelopin automaattinen koodin täydennys ei toiminut kovin hyvin. Välillä valikko ei auennut ollenkaan, ja silloin kun se aukesi, olivat täydennysvaihtoehdot usein kontekstiin sopimattomia, mikä hidasti koodin kirjoittamista. Ongelmia aiheuttivat myös koodin sisennykset, joten oli järkevää vaihtaa työkaluun, jonka tiedettiin toimivan paremmin.

2.3.1 C#

C# (*C sharp*) on vahvasti tyypitetty oliopohjainen ohjelmointikieli, joka on suunniteltu yksinkertaisuutta ja tehokkuutta silmällä pitäen. Se muistuttaa syntaksiltaan ja rakenteeltaan C:tä, C++:aa sekä Javaa. Tämän vuoksi sitä on nopea oppia, jos osaa jo jotakin edellä mainituista kielistä. (Microsoft a. 2015-05-21.)

Unity tukee C#:n lisäksi myös Boo- ja UnityScript (tunnetaan myös nimellä JavaScript for Unity) -kieliä, joten ennen opinnäytetyön aloittamista oli valittava, mitä näistä kolmesta tullaan käyttämään.

Boo on oliopohjainen ohjelmointikieli, jonka syntaksi perustuu Pythoniin. Se ei ole kovin laajalle levinnyt, joten sen opettelemisesta tuskin olisi hyötyä tulevaisuudessa. Myös koodiesimerkkejä on hyvin vähän. Näin ollen Boo jätettiin tämän projektin skriptivalinnoista kokonaan pois.

UnityScript puolestaan oli varteenotettavampi vaihtoehto kuin Boo. Koodiesimerkkejä oli paljon ja syntaksikin oli tuttu. UnityScriptin huonoksi puoleksi osoittautuivat kuitenkin tyypittämättömät muuttujat. Vahvasti tyypitettyihin ohjelmointikieliin tottuneena on helppo unohtaa kyseiseen ominaisuu-

teen liittyvät ongelmat ja niiden välttämiseen tarvittavat varokeinot, mikä puolestaan johtaa turhiin virheisiin, joiden korjaaminen vie ylimääräistä aikaa.

Lopulta päädyttiin C#-kieleen, joka oli projektin jäsenille jo entuudestaan tuttu. Virheiden löytäminen vahvasti tyypitetystä ohjelmointikielestä oli helpompaa, ja koodiesimerkkejä löytyi runsaasti. Lisäksi C# on yksi Visual Studion tukemista ohjelmointikielistä.

2.3.2 UnityVS

UnityVS, joka tunnetaan myös nimellä Visual Studio Tools for Unity, on Microsoftin omistama ilmainen työkalu, joka mahdollistaa Visual Studion ja Unityn välisen kommunikaation. Tämän avulla Unity-projektin virheenkorjaukseen voi käyttää Visual Studion debuggeria. (SyntaxTree 2015-04-17.)

2.4 Versionhallinta

Versionhallintajärjestelmän tarkoituksena on pitää kirjaa projektiin kuuluviin tiedostoihin tehdyistä muutoksista, uusien tiedostojen lisäyksistä ja turhien poistamisista. Tällöin on mahdollista palata tarpeen mukaan mihin tahansa aiempaan versioon joko yksittäisten tiedostojen tai koko projektin kannalta. Jos useampi henkilö työskentelee saman työn parissa, versionhallinnasta voi nähdä myös, kuka on tehnyt muutoksia, minne ja milloin. Siten projektin jäsenillä on ainakin jonkinlainen kuva siitä, mitä muut tekevät. (Chacon, Straub 2015-04-18.)

Versionhallintaa on olemassa kolmea eri tyyppiä: paikallinen, keskitetty ja jaettu. Yksinkertaisimmillaan paikallinen versionhallinta tarkoitta tiedostojen kopioimista eri kansioon, mutta sen vähemmän virhealtis muoto on eri versioiden tallentaminen yksinkertaiseen tietokantaan. Paikallisen versionhallinnan ongelma on tietenkin se, ettei tiedostoihin pääse käsiksi muuten kuin paikallisesti. Keskitetyssä versionhallinnassa puolestaan tallennus tapahtuu palvelimelle, mikä mahdollistaa tiedostojen saatavuuden usealle henkilölle samanaikaisesti. Tällöin ongelmia voi kuitenkin ilmetä vaikkapa palvelimen kaatuessa tai tiedostojen kadotessa palvelimelta. Jos näin käy, projektin edistyminen pysähtyy tai pahimmillaan se voidaan joutua aloittamaan kokonaan alusta. Jaettu versionhallinta sen sijaan tarkoittaa, että kaikki projektiin kuuluvat palvelimelle tallennetut tiedostot kopioidaan kokonaisuudessaan paikallisille tietokoneille, jolloin edellä mainittua ongelmaa ei ole. Vaikka palvelin kaatuisi, voidaan työskentelyä jatkaa paikallisella koneella, tai vaikka palvelimella olevat tiedostot katoaisivat, voidaan miltä tahansa paikalliselta koneelta kopioida tiedostot takaisin palvelimelle. (Chacon, Straub 2015-04-18.)

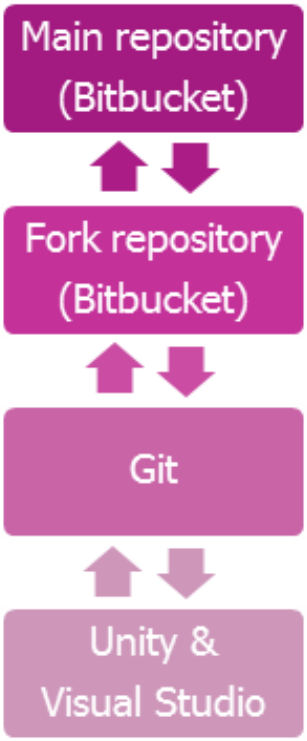
2.4.1 Git

Git on avoimen lähdekoodin jaettu versionhallintajärjestelmä. Se valittiin projektiin, koska sen käyttö on lyhyen opettelun jälkeen hyvin nopeaa ja vaivatonta. Lisäksi Git-osaamisesta voi olla hyötyä tulevaisuudessa, jos haluaa kehittää omia taitojaan osallistumalla projekteihin, joissa kehitetään avointa lähdekoodia.

2.4.2 Bitbucket

Bitbucket ja GitHub ovat jaettua versionhallintaa varten luotuja sivustoja. Ne ovat hyvin samankaltaisia, sillä molemmat on tarkoitettu lähdekoodin tallentamiseksi tietovarastoon (repository) ja sen jakamiseen versionhallinnan avulla. Suurin ero näiden kahden välillä ja opinnäytetyöprojektiin valinnan ratkaiseva tekijä oli se, että Bitbucketissa voi luoda maksutta yksityisen tietovaraston. GitHubissa vain julkiset tietovarastot ovat ilmaisia, joten tämän projektin kannalta Bitbucket on parempi vaihtoehto.

2.4.3 Työnkulku versionhallintaa käyttäen



KUVIO 1. Työnkulun vaiheet versionhallintaa käyttäen

Kuviossa 1 esitellään projektin työnkulkua versionhallintaa käyttäen. Kaksi ylintä laatikkoa kuvaavat Bitbucket-palvelussa sijaitsevia tietovarastoja. Main repository on nimensä mukaisesti projektin pääasiallinen tietovarasto. Fork repository puolestaan on henkilökohtainen kopio Main repositorysta. Työskentelyn alkaessa on syytä tarkistaa Bitbucketissa, onko Fork repository ajan tasalla. Jos näin ei ole, synkronoidaan Main repositoryyn tallennetut muutokset Fork repositoryyn.

Seuraavaksi projektiin kuuluvat tiedostot haetaan paikalliselle tietokoneelle käyttäen Gittiä, minkä jälkeen projektia voidaan muokata Unityssa ja Visual Studiossa. Sitten muutokset valmistellaan (stage) Gitissä committia eli pysyvää tallennusta varten. Kun kaikki halutut muutokset on valmisteltu, suoritetaan commit ja kirjoitetaan sitä kuvaileva viesti, jossa kerrotaan, mitä muutoksia kyseinen commit sisältää. Näitä vaiheita voidaan toistaa useita kertoja ennen seuraavaan vaiheeseen siirtymistä.

Kun vähintään yksi commit on suoritettu, voidaan muutokset työntää (push) Bitbucketissa sijaitsevaan Fork repositoryyn. Tällöin tehdyt muutokset ovat saatavilla myös silloin, kun tekijä vaihtaa tietokonetta, jolla työskentelee. Ne eivät kuitenkaan ole vielä projektin muiden tekijöiden saatavilla, vaan Fork repositoryssa on tehtävä pull request. Jos pull request hyväksytään Main repositoryssa, muutokset viedään myös sinne.

3 PROJEKTI

Toimeksiantajalla oli hyvin vähän toiveita pelistä, joten suunnittelu oli erittäin vapaata alusta loppuun. Pelityypiksi valittiin roolipeli, sillä ongelmanratkaisuun liittyviin peleihin ei ollut kovin helppo keksiä uudenlaisia tai mielenkiintoisia mekaniikoita. Roolipeleihin sen sijaan pystytään helposti lisäämään paljon erilaista sisältöä, jos taustalla olevat toiminnot toteutetaan tätä tukevalla tavalla.

Pelin tapahtumapaikkana oli tarkoitus olla jokin Savonian kampus, joten projektiin otettiin mukaan kaksi muotoilun opiskelijaa piirtämään vastaavat grafiikat. Yhteistyö osoittautui kuitenkin ongelmalliseksi, koska kommunikaatio ohjelmoijien ja graafikoiden välillä ei toiminut. Valmiita kuvia saatiin ainoastaan hahmoista eikä niitäkään voitu hyödyntää pelissä, koska niiden kanssa yhteen sopivia taustoja ei ollut. Lopulta päädyttiin käyttämään ilmaisia osia Kenney Game Asseteista.

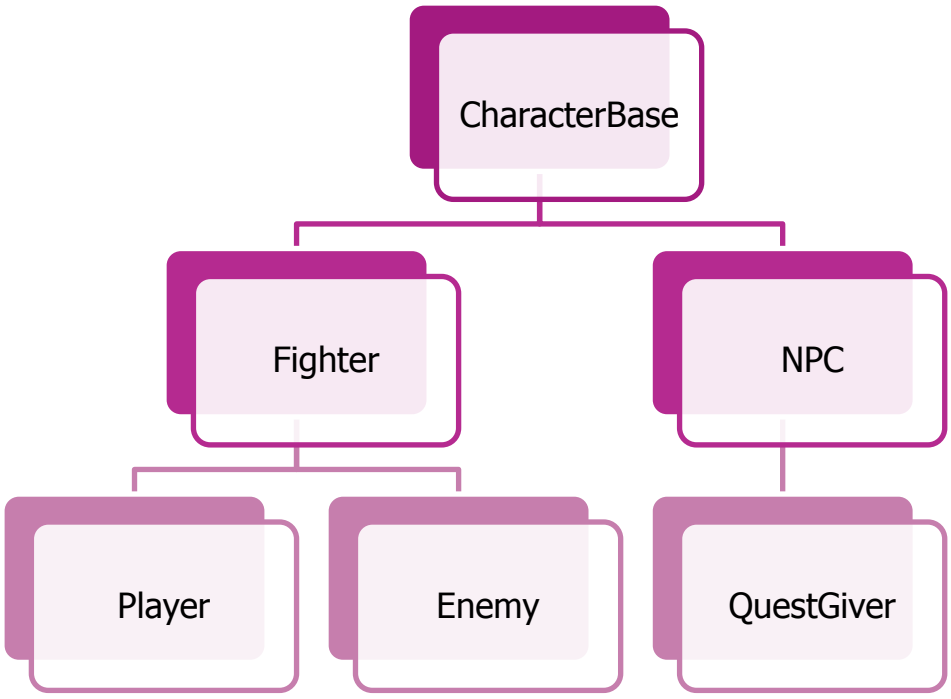
Projektisuunnitelmaan listattiin kaikki peliin tarvittavat osa-alueet. Nämä puolestaan jaettiin kahteen osaan, jotta saataisiin aikaan kaksi erillistä opinnäytetyötä. Toisen opinnäytetyön tekijä on Sini Mustonen, ja hänen osa-alueensa ovat käyttöliittymä, inventory-systeemi, pelin tallennus ja hahmojen kontrollointi. Projektin alkupuolella tehtiin myös käsikirjoitus, jossa kerrotaan pelin juoni ja suurin piirtein mitä milläkin tasolla tapahtuu.

Tässä opinnäytetyössä esitellään tarkemmin hahmojen, quest-systeemin sekä dialogien suunnittelua ja toteutusta. Kaikki kolme osa-aluetta liittyvät tiiviisti toisiinsa ja siksi yhden työstäminen vaatii hyvin usein myös toisen ja ehkä kolmannenkin osa-alueen muokkausta. Aina kun projektin edetessä opittiin jotain uutta, jokin parempi tapa toteuttaa jokin ominaisuus, jouduttiin tekemään suuriakin muutoksia muihin osa-alueisiin. Jotkin näistä muutoksista vaativat jopa toisen osan uudelleensuunnittelua.

Edellä mainittujen osa-alueiden lisäksi tässä osuudessa kerrotaan Unity Editorin kustomoinnista tämän opinnäytetyöprojektin kannalta.

3.1 Hahmot

Roolipeleissä on yleisesti ottaen kolmea eri hahmotyyppiä: pelaajat, NPC:t ja viholliset. Pelaajahahmo on nimensä mukaisesti hahmo, jonka tekemisiä pelaaja ohjaa. NPC eli non-player-character on pelin sisäinen ystävällinen hahmo, jonka kanssa pelaajahahmo voi halutessaan olla vuorovaikutuksessa. Esimerkiksi kauppiaat, joilta pelaaja voi ostaa varusteita, kuuluvat tähän ryhmään. Viholliset puolestaan ovat vastustajia, joita vastaan pelaaja taistelee. Ne voidaan lajitella karkeasti kahteen eri tyyppiin: sellaisiin, jotka hyökkäävät heti huomatessaan pelaajan ja sellaisiin jotka hyökkäävät vain, jos pelaaja hyökkää ensin. Jälkimmäistä tyyppiä ei joka pelissä ole ollenkaan, mutta usein sitä edustavat esimerkiksi villieläimet.



KUVIO 2. Hahmoluokkien perintä

Alkuperäisessä projektisuunnitelmassa pelaajahahmo, NPC:t ja viholliset määriteltiin erillisiksi osa-alueiksi, mutta koska ne sisältävät samoja ominaisuuksia, niiden luokkarakenteet oli järkevää yhdistää käyttäen hyödyksi perintää, mitä havainnollistetaan kuviossa 2. Ylimpänä näkyvä abstrakti CharacterBase-luokka sisältää kaikille hahmoille yhteisiä ominaisuuksia, kuten esimerkiksi nimen. Fighter-luokkaan on puolestaan sisällytetty taisteluissa tarvittavia asioita, kuten abstrakti Die-metodi, joka tekee eri asioita sen mukaan onko kuollut hahmo Player- vai Enemy-tyyppiä. Player-hahmon kuollessa kysytään pelaajalta kuvan 1 mukaisesti haluaako hän aloittaa uuden pelin, ladata tallennuksen vai lopettaa pelaamisen. Enemyn kuollessa puolestaan tehdään useita eri asioita, kuten siirretään sen kantamat tavarat Playerille ja ilmoitetaan kuolemasta pelin muille kyseistä tietoa tarvitseville osille, esimerkiksi quest-systeemille. Player ja Enemy eroavat toisistaan myös Attack-metodin osalta, sillä pelaajahamoa ohjataan, kun viholliset puolestaan hyökkäävät käyttäen tekoälyä.

KUVA 1. Pelaajahahmon kuollessa näytettävä valikko

NPC- ja QuestGiver-luokat sen sijaan eivät tarvitse edellä mainittuja taisteluihin liittyviä ominaisuuksia, sillä niiden tyyppisten hahmojen tehtävänä on puhua pelaajahahmon kanssa. Niimpä ne sisältävät muun muassa listan keskustelujen aloituksista sekä metodin, jonka avulla siirrytään seuraavaan keskustelun alkuun. Edellä mainittujen ominaisuuksien lisäksi QuestGiver antaa pelaajalle suoritettavia tehtäviä eli questejä. Tällöin kyseisen luokan on syytä tietää, mikä sen antamista questeistä on milloinkin meneillään, jotta keskustelu ei etenisi seuraavaan ennen kuin kyseinen tehtävä on suoritettu.

Hahmoluokat on suunniteltu niin, että jatkokehityksessä niiden laajentaminen edelleen on helppoa. Esimerkiksi NPC-luokasta voitaisiin johtaa vaikkapa Shopkeeper, joka tarvitsisi perittyjen ominaisuuksien lisäksi myös kaupantekoon tarvittavia muuttujia ja metodeja. Myös Player ja Enemy -luokkiin on helppo kuvitella samankaltaisia lisäyksiä, kuten eri kulttuurit, lajit tai rodut.

CharacterBase, ja siten kaikki muutkin hahmot, perii MonoBehaviour-luokan. Tämä tarkoittaa sitä, että hahmoluokkia käytetään lisäämällä peliin uusi GameObject, jolle lisätään yhdeksi komponentiksi haluttu hahmoluokkaskripti. Koska hahmot tarvitsevat useita muitakin GameObjectiin liitettäviä komponentteja, ei ole järkevää tehdä hahmoluokista erillisiä assetteja.

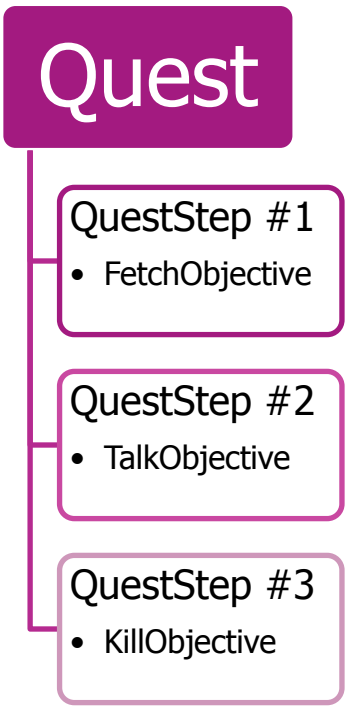
3.2 Questit

Quest tarkoittaa yksinkertaisesti pelaajalle annettua tehtävää, jollaisen loppuun suorittamisesta pelaajahahmo saa yleensä jotakin vastineeksi: pelin sisäistä rahaa, varusteita, uusia taitoja, jonkinlaisia pisteitä tai vaikkapa pääsyn seuraavalle tasolle. Yleisesti ottaen questien tarkoitus on edistää juonta kertomalla pelaajalle, mitä pitää tehdä seuraavaksi. Laajemmissa peleissä on usein sekä pääquestejä, jotka on pakko suorittaa päästäkseen pelissä eteenpäin, että vapaaehtoisia sivu-questejä, jotka tuovat lisää tekemistä tai vaikkapa antavat pelaajalle lisää tietoa pelin maailmasta. Näin ollen quest-systeemi toimii eräänlaisena pelaajan ohjeistus- ja palkitsemiskeinona ja on hyvin tärkeä osa mitä tahansa roolipeliä.

Suunnitteluvaiheessa päätettiin, että yksi quest voi sisältää useita tavoitteita ja että tavoitteita on useita eri tyyppejä. Jo suunnittelun alussa otettiin huomioon mahdollisen jatkokehityksen tarpeet niin, että erilaisia tyyppejä on helppo lisätä tarvittaessa vielä enemmänkin. Tätä tukee abstrakti QuestObjective-luokka, jonka jokainen tavoitetyyppiluokka perii.

Quest-luokan muuttujiin tallennetaan perustietoja tehtävästä, kuten sen vaiheet ja tila. Lisäksi luokalla on metodeja, joita muut luokat voivat käyttää kyseisten muuttujien käsittelyyn. QuestStep-luokassa puolestaan on tarkempaa tietoa tehtävän yksittäisistä vaiheista, mukaan lukien vaiheen tavoite, joita on kolmea eri tyyppiä. Nämä tyypit ovat

- FetchObjective, jossa pelaajan on löydettävä tietty määrä jotakin tavaraa
- TalkObjective, jossa pelaajan täytyy puhua tietyn NPC:n kanssa ja
- KillObjective, jossa pelaajan täytyy lyödä tietty määrä joitakin vihollisia.

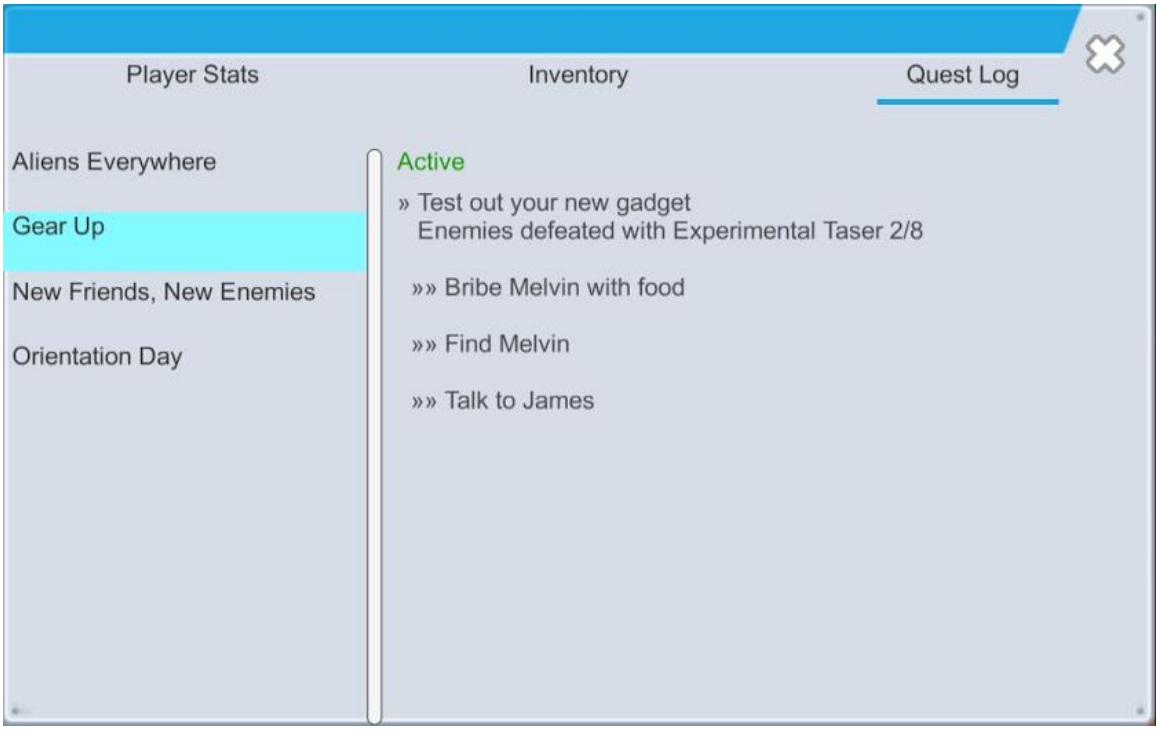


KUVIO 3. Esimerkki Questin rakenteesta

Kuviossa 3 näkyvässä esimerkissä on käytetty kaikkia kolmea tavoitetyyppiä. Pelaajan ensimmäinen haaste, FetchObjective, voisi olla vaikkapa aseen etsiminen. Tämän jälkeisessä TalkObjective-tavoitteessa pelaajan pitäisi esimerkiksi löytää NPC, joka kertoisi, kuinka edellä mainittua asetta käytetään. Questin viimeinen osuus, KillObjective, voisi tässä tapauksessa olla harjoitus, jossa pelaaja opettelee taistelua käytännössä.

Luokat Quest, QuestStep ja kaikkien tavoitetyyppien pohjana toimiva QuestObjective periytyvät ScriptableObject-luokasta. Näin ollen kyseisistä luokista voidaan tehdä assetteja sen sijaan, että ne liitettäisiin tyhjiin GameObjecteihin. Toinen syy ScriptableObject-luokan käyttöön on se, että questien tietoja ei yleensä tarvitse muokata sen jälkeen, kun ne on luotu itse tehdyn luontieditorin avulla.

Lisäksi quest-systeemiin kuuluu singleton-tyyppinen QuestManager-luokka, jonka tarkoitus on pitää kirjaa pelaajan saamista tehtävistä. Questit lisätään QuestManagerin sisäiseen listaan, kun pelaaja hyväksyy annetun tehtävän dialogissa. Tämän jälkeen listassa olevien tehtävien tilaa ja vaiheita päivitetään tarvittaessa. Pelin aikana listan sisältöä voi seurata kuvassa 2 näkyvästä Quest Logista. Ikkunan vasemmassa reunassa on luettelo kaikista pelaajan hyväksymistä questeista. Kun pelaaja valitsee niistä jonkin, oikeanpuoleisesta paneelista näkee tehtävän tilan, nykyisen vaiheen nimen ja edistymisen sekä listan jo suoritettujen vaiheiden nimistä.



KUVA 2. Ruutukaappaus Quest Log -ikkunasta

3.3 Dialogi

Peleissä dialogilla tarkoitetaan pelaajahahmon ja NPC:n välistä keskustelua. Nykyaikaisissa peleissä tämä on yleensä toteutettu niin, että keskustelun alkaessa näytölle ilmestyy ikkuna, jossa lukee, mitä NPC sanoo. Lisäksi siinä näkyy yksi tai useampi painike, joista pelaaja voi valita mieleisensä vastausvaihtoehdon. Tällä tavoin pystytään antamaan pelaajalle mielikuva, että hän itse päättää, kuinka pelin tarina etenee, ja monissa peleissä näin myös todella tapahtuu. Dialogisysteemiä käytetään myös pelaajan ohjeistamiseen ja questien antamiseen.

Dialogi liittyy läheisesti sekä hahmo- että quest-luokkiin. Siksi kyseisten luokkien rakenteita on jouduttu muokkaamaan useita kertoja dialogisysteemin muuttuessa. Tässä projektissa dialogi on suunniteltu niin, että keskusteluihin tarvittavat tiedot on tallennettu DialogData-tyyppisiin assetteihin. Näiden yksittäisten assettien luominen onnistuu helposti Unityssa, koska ScriptableObjectista perivälle DialogData-luokalle on projektin aikana tehty oma luontieditori.

Toinen keskusteluihin liittyvä luokka on nimeltään Dialog. Se sisältää metodin, joka määrää, milloin kuvan 3 kaltainen keskusteluikkuna avataan ja suljetaan. Se päättää myös, siirrytäänkö seuraavaan keskusteluun vai alkaako sama keskustelu alusta, jos pelaaja päättää jutella NPC:n kanssa uudelleen.

KUVA 3. Ruutukaappaus dialogi-ikkunasta

3.4 Unity Editorin kustomointi

Projektin aikana Unity Editoriin tehtiin kolmentyyppisiä laajennuksia helpottamaan pelin testausta ja sisällön luontia. Nämä laajennustyypit ovat:

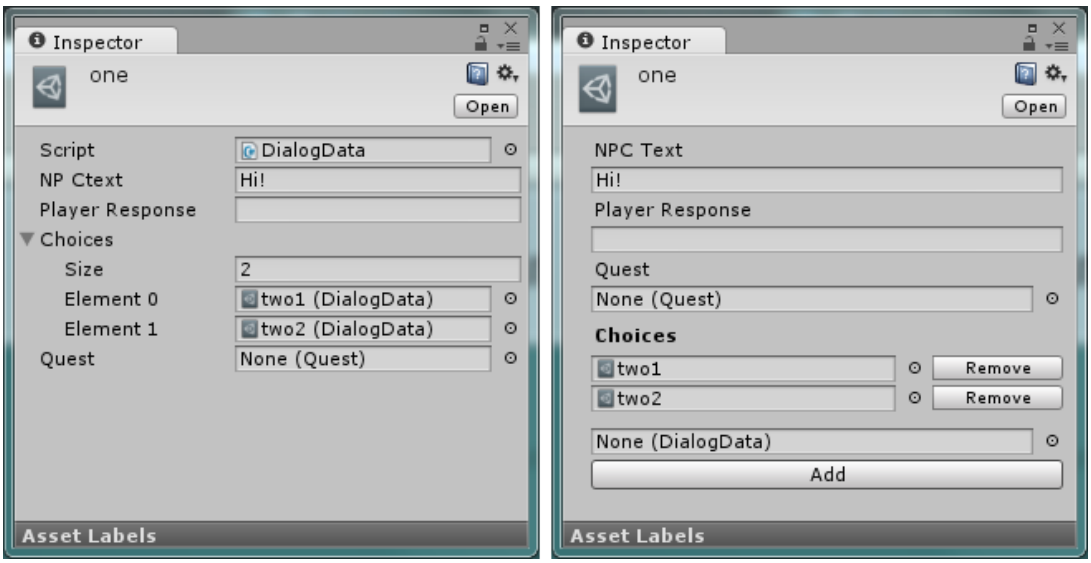
- Inspectorien kustomointi
- assettien luontieditorit
- puutietorakenne-editori.

3.4.1 Inspector-editorien kustomointi

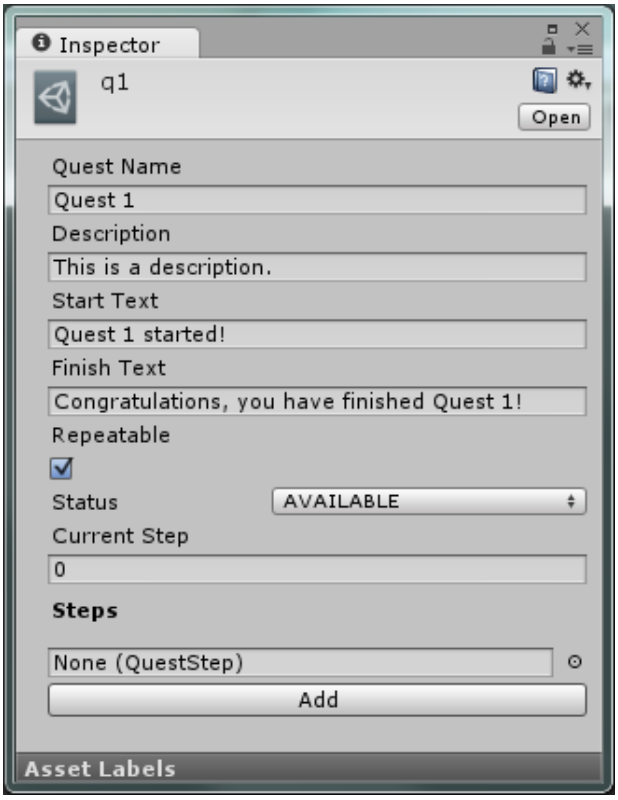
Unity 5 luo automaattisesti Inspector-editorin myös itse tehdyille luokille. Joillekin luokille tehtiin tästä huolimatta oma editoriskripti, koska automaattisesti luodulla Inspectorilla erikoisten tietotyyppien, kuten List ja Dictionary, hallinta on työläämpää.

Erikoisten tietotyyppien hallinnan työläys käy ilmi kuvassa 3, jossa on asetettu vierekkäin DialogData-tyyppisen assetin automaattisesti luotu ja kustomoitu Inspector-editori. Choices on List-tyyppinen muuttuja, jonka sisältöä pitäisi pystyä muuttamaan rakennettaessa hahmojen välisiä keskusteluja. Kuvan 3 vasemmanpuoleisessa, automaattisesti luodussa Inspectorissa uuden elementin lisääminen listaan tapahtuu muuttamalla listan kokoa manuaalisesti. Uusi elementti kuitenkin saa automaattisesti saman arvon kuin edellinen elementti, ja se täytyy muistaa muuttaa manuaalisesti jälkeenpäin. Myös elementtien poistamisessa ilmenee ongelma, sillä sekin täytyy tehdä listan kokoa muuttamalla, mutta näin voidaan poistaa vain listan loppupäässä olevia elementtejä. Kustomoinnin avulla voidaan selvittää molemmat ongelmat yksinkertaisesti lisäämällä edellä mainittuja elementin lisäys- ja poistotoimintoja varten napit, kuten kuvan 4 oikeanpuoleisessa, kustomoidussa Inspectorissa.

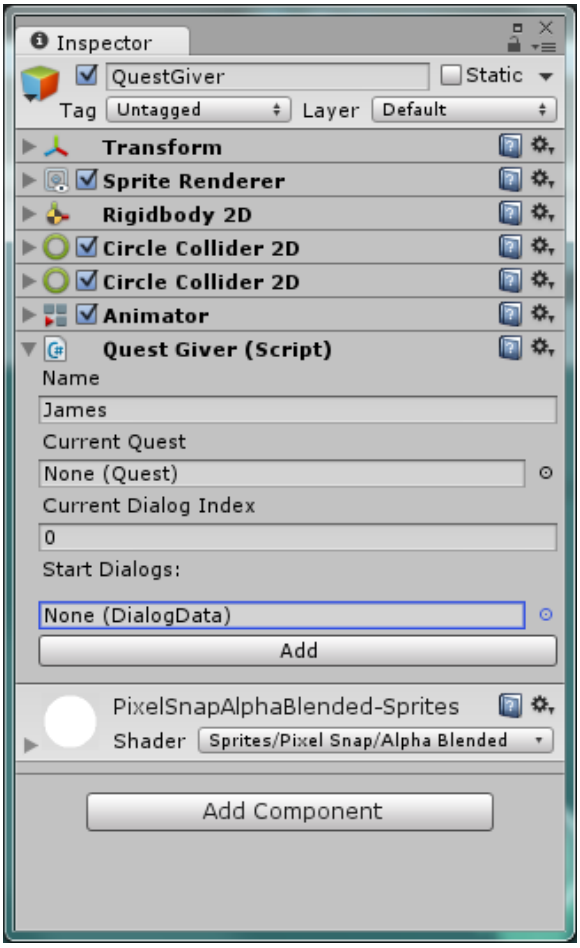
Kustomoidut Inspector-editorit on tehty DialogData-assettien lisäksi myös Quest-asseteille (kuva 5) ja hahmoluokille (kuva 6).



KUVA 4. Vasemmalla DialogData-assetille automaattisesti luotu ja oikealla kustomoitu Inspector-editori



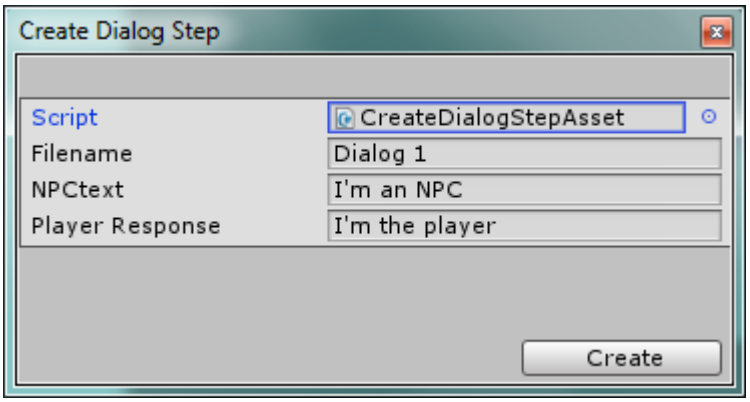
KUVA 5. Quest-assetin kustomoitu Inspector-editori



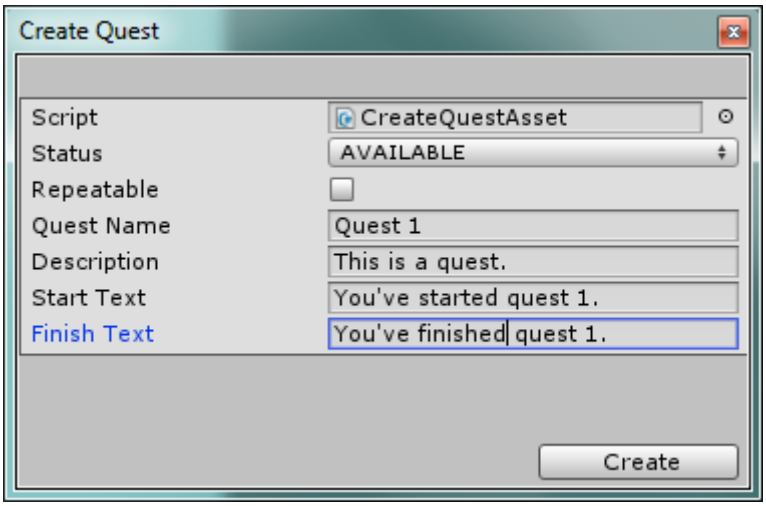
KUVA 6. QuestGiver-skriptin kustomoitu Inspector avattuna GameObjectin muiden komponenttien joukossa

3.4.2 Assettien luontieditorit

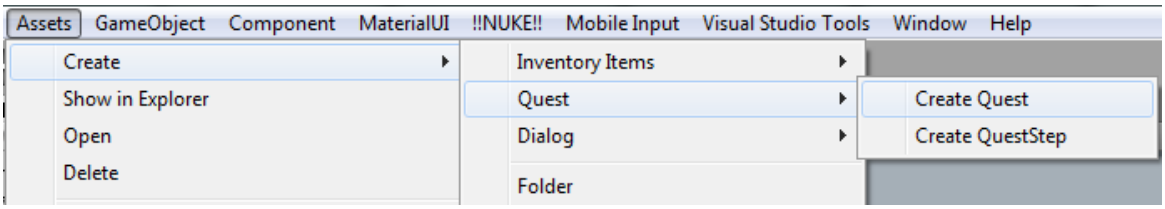
Jotta ScriptableObjectin perivistä luokista voitaisiin instantioida assetteja, niille tarvitaan omat luontieditorit, kuten kuvassa 7 näkyvä Create DialogStep ja kuvassa 8 näkyvä Create Quest. Koska Unity ei osaa tehdä luontieditoreja automaattisesti, ne on koodattava itse. Tässä projektissa kyseiseen tarkoitukseen käytetään ScriptableWizard-luokkaa, josta perivät omatekoiset editorit voidaan laittaa avautumaan kuvan 9 kaltaisesti Unity Editorin valikosta.



KUVA 7. DialogData-assetin luontieditori



KUVA 8. Quest-assetin luontieditori

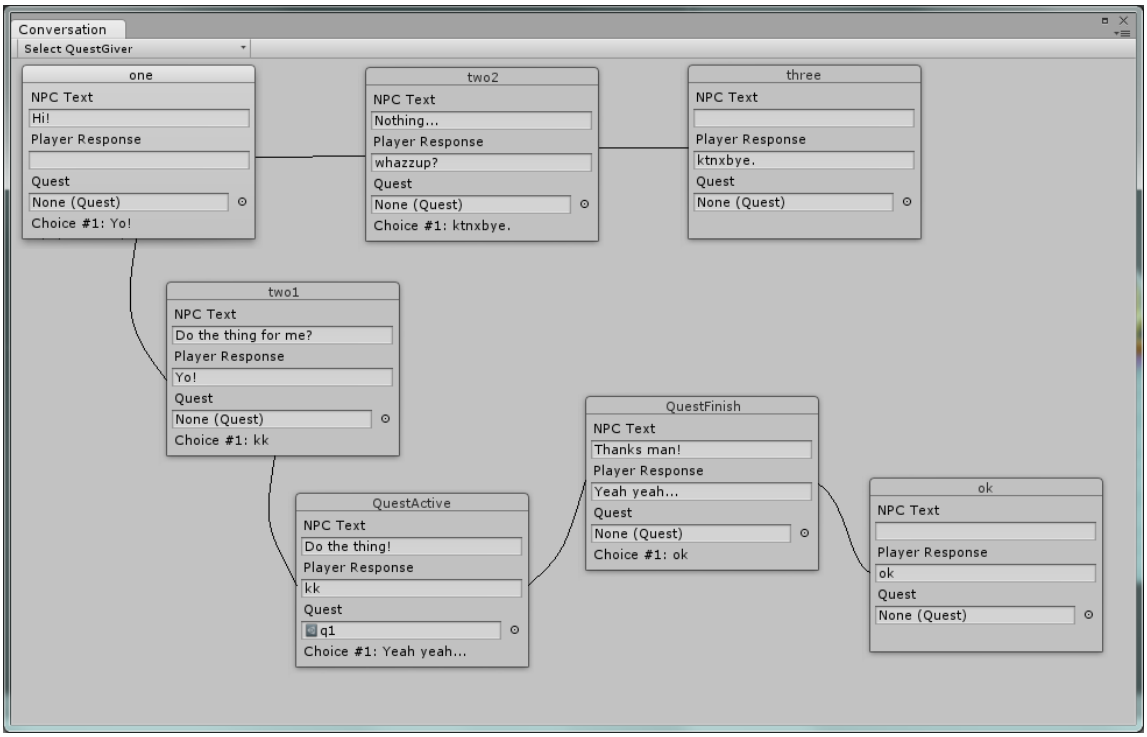


KUVA 9. Quest-assetin luontieditori Unity Editorin valikossa

3.4.3 Puutietorakenne-editori

Puutietorakenne-editorin tarkoitus tässä projektissa on helpottaa dialogien muokkausta. Tämä tapahtuu näyttämällä kaikki yksittäiseen NPC:hen liitetyt DialogData-assetit, kyseisiin assetteihin liitetyt DialogData-assetit ja niin edelleen. Näistä syntyy puurakenne, jota on hankala hallinnoida, jos yksittäisten assettien yhteyksiä toisiinsa ei ole visualisoitu.

Kuvassa 10 näkyvä keskusteluedi tori on suunniteltu niin, että jokainen DialogData-asset näkyy taustalla Inspectorin kaltaisena ikkunana, jonka arvoja pystyy muuttamaan tarvittaessa. Myös niiden välisiä suhteita voidaan muokata. Uusia assetteja voidaan luoda ja tarpeettomia poistaa hiiren oikeanpuoleisella painikkeella ilmestyvästä valikosta.



KUVA 10. Keskusteluedi tori-ikkuna

4 YHTEENVETO

Työn tavoitteena oli saada aikaan toimiva peliprototyyppi. Pelin taustatoiminnot ovat jo valmiina, mutta pelattavia kenttiä ei ole vielä tehty. Projektin jatkosuunnitelmiin kuuluu lisätä pelattavaa sisältöä, sekä kirjoittaa dokumentaatio mahdollisia muita jatkokehittäjiä varten.

Opinnäytetyön edetessä oppi lähes koko ajan jotain uutta. Opin, kuinka versionhallinta toimii ja miksi sitä kannattaa käyttää. Tiedän nyt myös, kuinka peli ehkä kannattaisi suunnitella ensi kerralla ja mitä sen toteuttamiseksi käytännössä vaaditaan.

Olisi ollut ehkä parempi tehdä ensimmäiseksi pelissä näkyviä asioita, minkä jälkeen olisi ollut helpompi miettiä, miten kaikki oikeasti toimii. Näin olisi saanut jo aikaisemmin realistisen kuvan siitä, mitä kaikkea tarvitaan taustalle. Jos aloittaisin samantapaisen peliprojektin samoilla työvälineillä, se olisi todennäköisesti valmis murto-osassa ajasta, joka tähän opinnäytetyöhön on käytetty.

Projektin suurin haaste oli ehkä aikataulu. Oli virhe ajoittaa projekti niin, että joutui samanaikaisesti käymään myös ansiotyössä. Ensinnäkin molempien tekeminen yhtä aikaa vei aivan liikaa energiaa. Toiseksi tuntui hankalalta keskittyä kumpaankaan kunnolla, sillä molemmat vaativat aika erilaisia ajattelutapoja. Aikatauluun olisi ehkä muutenkin kannattanut laatia hieman tarkempia välietappeja, joille jokaiselle olisi ollut oma deadlinensa. Siten projektin edistymistä olisi voinut seurata helpommin.

Koska jo pelin suunnittelussa otettiin hyvin huomioon mahdolliset jatkokehityksen tarpeet, pitäisi myös ominaisuuksien lisäämisen olla suhteellisen helppoa. Itse lähtisin viemään projektia perinteisten roolipelien suuntaan. Tätä varten voisi lisätä esimerkiksi pelattavia rotuja ominaispiirteineen. Hahmoilla voisi olla myös erilaisia ammatteja, joilla olisi kaikilla omat hyvät ja huonot puolensa, erikoiskykynsä ja rajoitteensa.

Questien ominaisuuksien lisäys vaatii ehkä hieman enemmän mielikuvitusta, mutta sekin on mahdollista. Esimerkiksi voitaisiin toteuttaa tavoitetyyppi LocationObjective, jossa pelaajan täytyy löytää tiensä johonkin tiettyyn paikkaan pelimaailmassa. Ainoa syy, miksei tätä ominaisuutta ole jo lisätty quest-systeemiin on se, että 2D-pelissä paikkojen löytäminen ei ole kovinkaan haastavaa.

Projektiryhmäläisten lisäksi myös toimeksiantaja voisi mahdollisesti jatkokehittää peliä. Tällä tavalla sen saisi vastaamaan paremmin Savonian tyyliä ja imagoa. Peliä voisi silloin käyttää myös koulun markkinointiin ja uusien opiskelijoiden houkuttelemiseen.

LÄHTEET

CHACON, Scott & STRAUB, Ben 2014. Pro Git. Apress. [Viitattu 2015-04-18.] Saatavissa: http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

KENNEY 2015. Kenney Game Assets. [Viitattu 2015-05-26.] Saatavissa: http://kenney.itch.io/kenney-donation

MICROSOFT 2015a. Introduction to the C# Language and the .NET Framework. [Viitattu 2015-05-21.] Saatavissa: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

MICROSOFT 2015b. Visual Studio With MSDN. [Viitattu 2015-05-20.] Saatavissa: https://www.visualstudio.com/products/visual-studio-with-msdn-overview-vs

SYNTAXTREE 2012-2014. Visual Studio Tools for Unity Features. [Viitattu 2015-04-17.] Saatavissa: http://unityvs.com/features/

THE CODEHAUS 2003-2006. Boo A wrist friendly language for the CLI. [Viitattu 2015-05-26.] Saatavissa: http://boo.codehaus.org/

UNITY TECHNOLOGIES 2015a. A Feature-Rich and Highly Flexible Editor. [Viitattu: 2015-05-22.] Saatavissa: http://unity3d.com/unity/editor

UNITY TECHNOLOGIES 2015b. GameObjects. [Viitattu: 2015-05-22.] Saatavilla:

http://docs.unity3d.com/Manual/class-GameObject.html

UNITY TECHNOLOGIES 2015c. MonoBehaviour. [Viitattu: 2015-05-22.] Saatavilla: http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

UNITY TECHNOLOGIES 2015d. ScriptableObject. [Viitattu: 2015-05-22.] Saatavilla: http://docs.unity3d.com/Manual/class-ScriptableObject.html