10.8.2020

Daniel Juola
daaajuol@student.jyu.fi
Pääaine: Tietotekniikka

TIEA306 Ohjelmointityö: Itsearviointi & Jälkiselvitys

Sisällysluettelo

Tiivistelmä

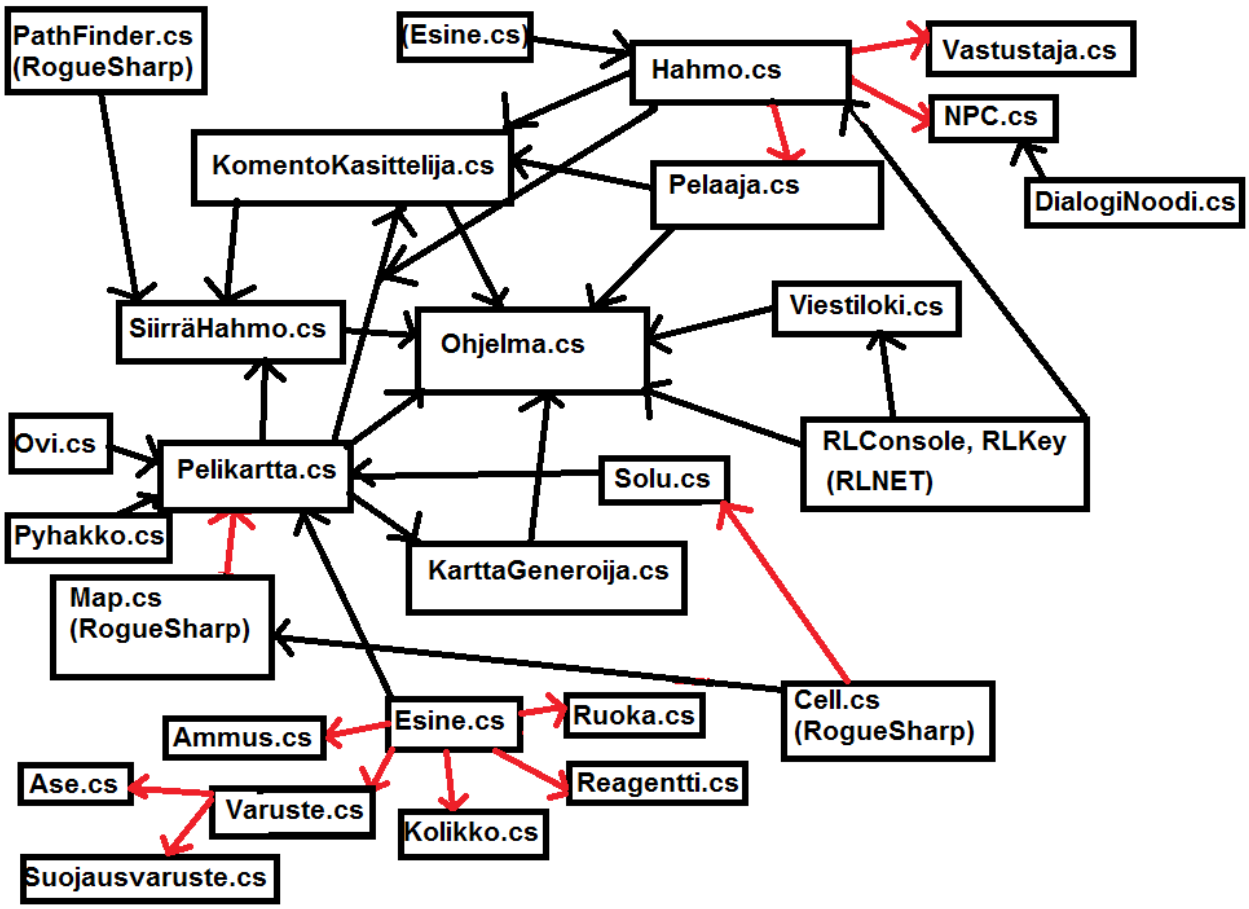
Ohjelmointityöni oli Ultima-kloonin luominen, ja mielestäni onnistuin tässä, vaikka en saanutkaan toteutettua aivan kaikkia mahdollisia ominaisuuksia. Lopputuloksena on siis yksinkertainen ASCII-grafiikkaa käyttävä tietokoneroolipeli, joka perustuu 1980-luvun Ultima-peleihin. Siinä on monia samoja ominaisuuksia, kuten NPC-hahmojen kanssa keskustelu, interaktio esineiden kanssa, taikasysteemi ja karma-moraalimittari. Pelin saa läpäistyä noin tunnissa.

Tehtävän kuvaus

Esisuunnitelmassani esittelin tehtäväni, Ultima-kloonin tekemisen, yksityiskohtaisemmin. Jaottelin tarvittavat ominaisuudet neljään eri vaiheeseen. Ensimmäinen vaihe koostui seuraavista perusominaisuuksista: Tiilipohjainen 2D-kartta, jossa on liikuteltava pelihahmo, ASCII-grafiikka, karttojen luonti ja tallennus, siirtyminen kartasta toiseen ja viestiloki. Toinen vaihe, interaktio, koostui vain yksinkertaisesta keskustelusta ja taistelusta muiden pelihahmojen kanssa sekä kerättävistä esineistä. Kolmannessa vaiheessa täytyi lisätä Ultima-pelien ominaisuuksia, kuten taikasysteemi, karma-mekaniikka, kumppanit, monimutkaisempi keskustelu, ja tehtävät joita voi suorittaa. Viimeinen vaihe oli vähiten tärkeä ja koostui ainoastaan ”bonus”-ominaisuuksista: Kustomoidut pikseligrafiikat tiileille, yö/päivä sykli, enemmän esineitä ja interaktiota pelimaailman kanssa, päätarina ja ääniefektit.

10.8.2020

Käytännön toteutus



Yllä on epätäydellinen kuvaus ohjelman rakenteesta. Laatikot ovat eri luokkatiedostoja, ja mustat nuolet osoittavat mikä luokka on erityisen riippuvainen mistäkin luokasta. Punaiset nuolet osoittavat luokkien perintää. System-kirjaston luokkia ei mainita tässä eikä alla. RLNET on Travis M. Clarkin käsialaa kun taas RogueSharp on Faron Bracyn tekemä. Itse ohjelma toteutettiin Visual Studio 2019-IDEä käyttäen C#-kielellä.

Ohjelman ytimenä on Ohjelma.cs, joten ohjelman rakenteen selvittäminen on paras aloittaa siitä. Ohjelma.cs hyödyntää seuraavia luokkia: RLConsole (RLNET-kirjasto), RLRootConsole (RLNET-kirjasto), NPC, Vastustaja, SiirraHahmo, KarttaGeneroija, PeliKartta, Pelaaja, KomentoKasittelija, Viestiloki, Esine, Ammus, RLKeyPress (RLNET-kirjasto), RLKey (RLNET-kirjasto), Ase, Suojausvaruste ja RLColor (RLNET-kirjasto). Ohjelma on ainoa luokka joka vastaanottaa dataa ulkopuolelta näppäimistön kautta. Se tulkkaa pelaajan syötettä ja hallinnoi kaikkea mihin pelaajan syöte vaikuttaa. Jos pelaaja ei tee mitään, Ohjelma-luokka ei myöskään tee mitään. Tämän lisäksi se lukee tallenuustiedostoja ja luo niitä (JSON-muodossa), vaikkakin jälkimmäisen kanssa on ongelmia objektien serialisoinnin kanssa. RLRootConsole-luokka piirtää ohjelman datan näytölle Ohjelma-luokan kautta.

Seuraava tärkeä luokka on Pelikartta.cs, joka hallinnoi pelin karttaa. Se perii RogueSharpin Map-luokan ja onkin vain sen laajennos. Sillä on viisi listaa, joissa ovat kaikki kartan Esine-, NPC-, Vastustaja-, Ovi- ja Solu-luokan objektit. PeliKartta-luokka käskee kaikkia kartalla olevia ja pelaajahahmon näköpiirissä olevia objekteja piirtämään itsensä karttakonsoliin, asettaa pelihahmojen sijainnin sen muuttuessa, päivittää pelaajan näkökentän, avaa tiellä olevia ovia ja lopuksi vielä aktivoi pyhäkön jos pelaaja on sen luona.

10.8.2020

KarttaGeneroija.cs rakentaa erilaisia pelikarttoja, ja sitä varten käyttää samoja luokkia kuin Pelikartta. Se myös vaihtaa karttaa kun pelaajan hahmo menee nykyisen kartan ulkopuolelle ja asettaa samalla sen sijainnin.

KomentoKasittelija.cs nimensä mukaisesti käsittelee pelaajan komentoja, ja toimii ikäänkuin Ohjelma-luokan laajennoksena. Se määrittää suunnat, ja sen pääasiallisena tehtävänä onkin siirtää pelaajaa eri suuntiin (peliKartta hoitaa itse uuteen paikaan sijoittamisen) ja käsittellä interaktio ammuksen ja vastustajan sekä pelaajan ja vastustajan välillä (ts. taistelumekaniikan) käyttäen RogueSharp-kirjaston Dice-luokkia satunnaisuutta varten. Kaikki interaktio luetellaan Viestilokiin.

SiirraHahmo on samankaltainen luokka, mutta siirtää hahmoja jotka eivät ole Pelaajan hahmo. Se käyttää RogueSharp-kirjaston PathFinder-luokkaa jotta hahmot tietävät miten lähestyä pelaajaa ja päivittää hahmojen näkökentän (ei pelaajan). Se myös päivittää pelikarttaa, sillä pelaaja ei voi mennä samaan tiileen kuin muut hahmot.

Viestiloki.cs on yksinkertainen luokka ja vain kirjoittaa halutut viestit konsoliin. Samalla tavalla DialogiNoodi on myös yksinkertainen ja käytetään NPC-hahmojen dialogin rakennetta varten. Rajapinnat.cs puolestaan määrittää kolme rajapintaa, joita monet muut luokat käyttävät: yksi hahmoja varten, toinen konsoliin piirrettäviä tiilejä varten ja kolmanneksi esineitä varten.

Seuraavaksi ovat peliobjektit-kansioon lajitellut tiedostot. Ne ovat erilaisia objekteja joita kaikki yllä mainitut luokat hallinnoivat, ja kaikki niistä näkyvät pelikartalla, ja siksi käyttävät rajapintoja ja RLNET-kirjaston RLColor-luokkaa. Ensimmäiseksi Solu.cs on RogueSharp-kirjaston Cell-luokan laajennos jota käytetään kartan tiilien ulkonäön määrittämistä varten, koska Cell-luokkaan ei voi tallentaa tiilen käyttämää merkkiä tai väriä. Toiseksi ovat Hahmo- ja Esine-luokat, jotka monet muut luokat perivät. Ne määrittävät kaikille periville luokille yhteisiä oletusominaisuuksia, kuten metodin sille että pelaaja ottaa esineen ja metodin kartalle piirtämiselle. Hahmo-luokan perivät Pelaaja, Vastustaja ja NPC, eli pelaajan hahmo, vihollishahmo ja ystävällinen hahmo. Näistä kolmesta Pelaaja on laajin, koska kyseessä on pelin ”päähenkilö” jota ohjataan suoraan. Hahmoilla on esine-inventaario ja pelaajalla on sen lisäksi lista puetuista varusteista. Pelaaja-luokassa on metodeja näiden kahden käsittelylle. Muut metodit päivittävät konsoleita joissa luetellaan pelaajan hahmon tiedot, ja käsittelevät pelaajan kuoleminen, esineiden ostamisen NPC-hahmoilta sekä erilaiset metodit joilla muutetaan Pelaaja-luokan muuttujia. NPC-luokassa on esineen myymisen toinen puolisko, kuten myös metodi dialogia varten jonka avulla pelaaja voi keskustella NPC-hahmon kanssa. Vastustaja-luokka ei tee muuta kuin määrittää mitä tapahtuu kun pelaaja tappaa vihollisen. Esine-luokan perivät Varuste-, Ammus-, Kolikko-, Reagentti- ja Ruoka-luokat. Varusteet ovat esineitä joita pelaaja voi pukea tai käyttää aseena, ja siksi sen vielä perivätkin Suojausvaruste- ja Ase-luokat. Ammus-luokkaa käytetään vain erottaakseen pitkän kantaman aseen ammukset muista esineistä. Kolikko-esinettä käytetään kaupankäyntiin pelaajan ja NPC-hahmojen välillä, kun taas Reagentti-esineitä käytetään loitsuissa. Viimeiseksi Ruoka-luokan esineet lisäävät pelaajan nälkämittaria kun niitä käytetään.

Hahmo- ja Esine-luokkiin liittyvien luokkien lisäksi on vielä Ovi- ja Pyhakko-luokat, jotka ovat pelikartalla sijaitsevia paikallaan pysyviä objekteja. Oven läpi pelaaja ja viholliset eivät voi nähdä kunnes se avataan, kun taas pyhäkkö on pelin progression kannalta tärkeä. Pelaaja saa siitä karma-pisteitä ja voi saavuttaa uuden tason, mitä vaaditaan pelin voittamista varten.

Itsearviointi

Toiveenani alun perin oli ainakin toteuttaa kaikki mahdollinen vaiheista 1-3 ja ehkä myös jotain vaiheesta 4. Valitettavasti korona-tilanne ja muut kurssit hidastivat työtä, ja vain vaiheet 1-2 menivät jotenkuten aikataulun mukaan. Vaiheen 3 ominaisuuksiin menikin enemmän aikaa, kuten myös itse pelin sisällön luomiseen mekaniikkojen rakentamisen lisäksi. Käyttämäni kirjastot eivät

myöskään soveltuneet kaikkeen haluamaani (esim. pelin tallentaminen ei toimi täydellisesti). Ensimmäisestä kahdesta vaiheesta sain kaiken toteutettua, mutta vaiheesta kolme puuttuvat kumppanit. Vaikka olisi ollut melko helppoa toteuttaa ystävällisiä hahmoja jotka seuraavat pelaajaa, se ei olisi riittänyt vaan Ultima-tyyliset kumppanit olisi vaatinut kaikkien ryhmän jäsenten manageerointia. Tämä olisi ollut työläämpää ja siksi päätin keskittyä muiden ominaisuuksien toteuttamiseen ensin. Lopuksi kumppanit jäivät tekemättä, mutta peli on pelattavissa yhdenkin hahmon kanssa, ja Ultima-peleissä oli muutenkin vaihtoehtona pelata ilman kumppaneita. Toinen asia mikä ei ihan toteutunut täydellisesti on dialogi-systeemi. Tekemälläni systeemillä on helppoa tehdä monimutkaisia dialogipuita, kyllä, mutta dialogin pitäisi olla muutakin kun vain tekstiä, vaan sen pitäisi vaikutta muihinkin asioihin. Samoin dialogilla voisi olla vaatimuksia joten tietyt dialogivaihtoehdot näkyisivät vain kun ne on täytetty. Sain lisättyä yksinkertaisen kaupankäynnin, mutta dialogi voisi toimia paremmin ja olla vähemmän köyhä ominaisuuksiltaan. Vaiheen neljä ominaisuuksia en oikeastaan toteuttanut ollenkaan, koska mielummin keskityin hiomaan muiden vaiheiden ominaisuuksia ja varmistamaan että ne toimivat kuten pitääkin, ja että pelissä oli tarpeeksi sisältöä eikä vain erilaisia mekaniikkoja joilla ei pystynyt saavuttamaan mitään.

Jos pitäisi valita muutama ominaisuus joka oli hankala toteuttaa, niin valitsisin pelin tallentamisen/lataamisen, dialogisysteemin ja pelikartan. Ongelmat johtuivat usein siitä että käyttämäni kirjastot olivat vaikea soveltaa haluamaani asiaa varten enkä kyennyt keksimään siihen hyvää ratkaisua. Oletinkin tätä ongelmaa esisuunnitelmaa tehdessäni ja mainitsin sen. Tästä huolimatta sain taivutettua kirjastot tekemään aika paljon asiota, vaikka RogueSharp oli alun perin tarkoitettu yksinkertaisia roguelike-pelejä varten. Tarpeen kautta opin kuitenkin käyttämään ominaisuuksia jotka olivat minulle uusia, kuten delegaatteja, serialisointia, override-metodeja, rajapintoja sekä monimutkaisten asioiden piirtämistä konsoliin.

En enää saa peliä kaatumaan, mutta se kaatui silloin tällöin pelin tekemisen aikana. Kaatuilun aiheuttamat ongelmat on sittemmin korjattu, mutta on mahdollista että peli voi vielä jostain syystä kaatua enkä ole vain löytänyt sitä. Kun peli kaatui, se johtui usein siitä että hahmo yritti liikkua kartan ulkopuolelle enkä ollut lisännyt mitään käsittelemään tätä. Peli myös kaatuili kun taulukkojen id:t olivat liian suuria, mikä tapahtui kun numerot menivät päässäni sekaisin.

Oman pelin kehittäminen itse kyllä eroaa kurssien töistä siten, että ohjausta ja opastusta on vähän tai ei ollenkaan. Kaikki pitää löytää, ymmärtää ja keksiä itse. Kirjastoista on hyötyä mutta vain rajoitettu määrä, ja nekin pitää oppia tuntemaan jotta niitä voi käyttää kunnolla. Lisäksi ohjelmointiprojektit joista olen lukenut yleensä tehdään ryhmätyönä, jossa eri ihmiset keskittyvät tiettyihin osa-alueisiin. Jos haluaa tehdä kaiken itse, niin pitää tehdä paljon enemmän työtä että saa edes yksinkertaisen roolipelin tehtyä.

Ylläpitäjän/Jatkokehittäjän ohjeet

ks. README.md-tiedosto.

Jatkokehitys

Peli on nyt hyvä ydin laajemmalle pelille, johon on helppo lisätä sisältöä, kuten uusia karttoja, loitsuja, varusteita, vihollisia jne. Perusmekaniikat ovat paikoillaan ja toimivat hyvin. Joitain asioita voisi kuitenkin parantaa. Tallennus pitäisi saada toimimaan täydellisesti, dialogi-systeemin voisi varmaankin toteuttaa paremmin jotta se olisi joustavampi eri asioita varten, ja oviin voisi lisätä lukkoja jotka aukeaisivat avaimilla.Vihollisille voisi helposti tehdä ominaisuuksia kuten pitkän kantaman hyökkäys tai loitsiminen. Kumppaneita voisi myös lisätä, mutta se vaatisi hyvin paljon työtä, ja siksi olisi varmaankin parasta jättää ne tekemättä. Itse koodia voisi varmaankin joku kokeneempi organisoida, tiivistää ja puhdistaa turhasta.

10.8.2020

Yhteenveto

Loppujen lopuksi olen ylpeä tuotoksestani, vaikka se ei täydellinen olekaan. Annoin sen yhdelle kaverille testattavaksi ja vaikka hän löysikin monia bugeja joita sitten korjasin, hän piti pelistä ja sai sen läpäistyä. Paragon of Virtue on siis toimiva, jonkin verran viihdyttävä ja yksinkertainen roolipeli jossa on roolipelin perusominaisuudet sekä monia viittauksia Ultima-pelisarjaan. Se on pelattavissa alusta loppuun vaikkakin kokemus on vain tunnin mittainen. Opin myös monia uusia asioita ohjelmoinnista ja sain paljon käytännön kokemusta, josta on hyötyä muussakin kuin pelien tekemisessä.

Lähteet

RogueSharp-kirjasto

https://github.com/FaronBracy/RogueSharp

RLNET-kirjasto

https://bitbucket.org/clarktravism/rlnet/src/master/

RogueSharp-tutoriaalit ja koodiesimerkit joihin peli osittain perustuu

https://roguesharp.wordpress.com/

The Codex of Ultima Wisdom, Ultima-wiki josta löytyy paljon tietoa Ultima-peleistä joihin tämä peli perustuu

https://wiki.ultimacodex.com/wiki/Main_Page