KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Joonas Niinistö

Monogame – Yleiskatsaus ohjelmistokehyksen toimintaan

Opinnäytetyö
Joulukuu 2019

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Joulukuu 2019
Tietojenkäsittelyn koulutus
Alempi korkeakoulututkinto
Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä
Joonas Niinistö

Nimeke
Monogame – Yleiskatsaus ohjelmistokehyksen toimintaan

Toimeksiantaja
-

Tiivistelmä

Opinnäytetyössä tutustutaan pelikehitykseen tarkoitetun ohjelmistokehyksen, Monogamen, toimintaan. Työn tavoitteena ei ollut luoda Monogame-sovellus vaan tutustua tämän ympärillä vaikuttaviin teknisiin tekijöihin ja kokonaisuuteen. Työ jakautuu kolmeen osaan:

Ensimmäisenä on teoria osio, johon sisältyy Monogamen yleinen sekä historiallinen määrittely. Osiossa esitellään myös Monogamen toiminnallisia tekijöitä, kuten erilaisia grafiikka rajapintoja sekä ”välikerroksia”.

Käytännönosuudessa tutustutaan kirjoittajan kehittämään pelisovellukseen ja sen toimintaan Monogamea hyödyntäen. Osion tavoitteena on tutustua ja seurata sovelluksen suorittamista eri alustoilla, kuten Debian Linux- ja Android(mobiili)-ympäristöissä. Pelisovelluksen lähdekoodi on luettavissa työn liiteosiossa.

Viimeisessä osiossa tutustutaan hieman uudenlaisiin grafiikkarajapintoihin, joiden voidaan olettaa yleistyvän lähivuosien aikana.

Lähteinä ovat pääasiallisesti erilaiset dokumentaatiot sekä hieman kirjallisuutta. Historiaan liittyvän tiedon hankinnassa on hyödynnetty myös aikansa verkkouutisia. Käytännönosuus sisältää myös hieman kirjoittajan omia tulkintoja esimerkkisovelluksen toiminnasta.

Kieli
suomi

Sivuja 121
Liitteet 1
Liitesivumäärä 30

Asiasanat
Sovelluskehitys, ohjelmistokehys, rajapinta, alusta riippumaton, mobiilikehitys, grafiikka

THESIS
December 2019
Degree programme in Business IT
Bachelor's Degree
Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author
Joonas Niinistö

Title
Monogame – Overview of the Software Framework's Functionality

Commissioned by
-

Abstract

This thesis introduces a programming framework, Monogame, which is used for video game application development. The aim of the thesis was not to create a Monogame based application but to study its technical factors and wholeness around it. The thesis is divided into three sections:

The first part is a theoretical section, introducing a general and historical definition of Monogame. In addition, the section introduces the functional elements of Monogame, such as various graphics interfaces and "wrappers".

The practical section covers a videogame application developed by the author and how it works with Monogame. The section aims to introduce and observe how to execute the application on various platforms, such as Debian Linux and Android (mobile)phone. The source code of the example application is readable from the appendices section of the thesis.

The last section introduces some modern graphics interfaces which could be assumed to become more common in the near future.

Information references mostly include different software documentation and literature. Web articles and news of the time are also utilized with sources related to Monogame's history. The practical section also includes writer's own interpretations related to the example application's functionality.

Language
Finnish

Pages 121
Appendices 1
Pages of Appendices 30

Keywords

Software development, framework, interface, cross -platform, mobile, graphics

Sisältö

Liitteet

Lyhenteet

Android Googlen kehittämä mobiilijärjestelmä. Arkikielessä sanalla tarkoitetaan älypuhelimia, jotka sisältävät kyseisen järjestelmän.

Alustariippumattomuus (Cross-platform)

Ohjelmoinnissa käytettävä termi, jolloin kehittäjän käyttämät ohjelmistot ja työkalut eivät ole sitoutuneet mihinkään tiettyyn laitteistoon tai järjestelmään. Alustariippumattomuus pyrkii tekemään kehitystyöstä mahdollisimman samanlaista kaikissa järjestelmissä. Se ei kuitenkaan takaa automaattista kykyä kehittää sovellusta mille tahansa järjestelmälle ilman asianmukaista ohjelmistoa.

Avoin lähdekoodi (Open source)

Ohjelmakoodi on jaettu tarkastelu ja muokkauskäyttöä varten. Lähdekoodiin liittyy aina myös lisenssi, joka määrittelee miten koodia saa määritellä ja jakaa.

Kehitystyökalu (Software Development Kit, SDK)

Ohjelmiston kehityksessä apuna käytettävä työkalu tai laitteisto. Se sisältää ominaisuuksia, joiden avulla kehittäjä voi helpottaa kehitystyötään työkalun tarjoamille alustoille.

Kuvapuskuri (FrameBuffer)

Puskuri, johon näytöllä oleva kuva tallentuu kokonaisuudessaan.

Kääntäjä Ohjelmisto, joka kääntää halutun ohjelmakoodin tietokoneen ymmärtämään muotoon.

MAC ja iOS-ympäristöt

Apple Inc–yrityksen kehittämät ohjelmistoympäristöt. MAC termillä tarkoitetaan Applen kehittämiä Macintosh-tietokoneita, joiden

käyttöjärjestelmänä on MacOS. iOS on käyttöjärjestelmä, jota löytyy Applen mobiililaitteista.

Nuget — Microsoftin dotNet kehitysympäristön pakettienhallintaohjelma, jonka avulla käyttäjä voi luoda omia ohjelmistopaketteja, sekä ladata ja käyttää muita paketteja.

Ohjelmisto spesifikaatio

Määritelmä, kuinka ohjelmiston on tarkoitus toimia.

Ohjelmistorajapinta (Application Programming interface, API)

Tarkoitetaan yleisesti ohjelmaa tai järjestelmää, jonka avulla käyttäjä voi suorittaa haluttuja toimintoja laitteistoon, joka suorittaa käytössä olevaa rajapintaa.

Tulkki — Toisin kuin kääntäjä, ohjelmistotulkki ei käännä ohjelmakoodia järjestelmän ymmärtämään muotoon, vaan suorittaa sen välittömästi. Tämä voi joissain määrin olla hitaampaa, sillä tulkin on analysoitava valittu koodi ennen sen suorittamista.

Välikerros/”kietoja” (Wrapper/Binding)

Välikerros on sovellus, joka ”kietoo” muuta dataa tai sovelluksia, tehden niistä yhteensopivia kohdealustalle. Monogame käyttää välikerroksia ”SharpDX” ja ”SDL”.

Xamarin — Opinnäytetyössä termillä viitataan yritykseen ja sen samannimiseen kehitysalustaan. Sana on tiedettävästi väännös apinalajista ”Tamarin” (tamariini) sekä X-kirjaimesta, jolla viitataan sanaan ”cross-platform”.

1 Johdanto

Videopeli- ja ohjelmistoala ovat kehittyneet, laajentuneet ja yleistyneet huomattavasti viimevuosien aikana. Tämä on saavutettu niin kysynnän, kehittyneen koulutuksen sekä erilaisten alustojen ja kanavien, kuten Steam, Epic Games Store tai Google Play, ilmestyessä. Näiden avulla kehittäjät voivat itsenäisesti käyttää sovellustensa julkaisemiseksi.Yleistymiseen on vaikuttanut myös erilaiset pelimoottorit, joita suositaan helpon käyttöönoton, kattavan tarjonnan sekä verkostovaikutuksen seurauksina. Maineikkaimpina esimerkkeinä näistä ovat Unity, Godot sekä Epic Gamesin Unreal Engine.

Opinnäytetyössä keskitytään kuitenkin avoimen lähdekoodin ohjelmistokehykseen (*framework*) nimeltään Monogame, jonka tavoitteena on olla mahdollisimman avoin ja alustariippumaton kehitystyökalu sovellusten ja etenkin pelien kehittämiseen.
Päädyin käyttämään Monogamea sattumalta tutkiessani ilmaisia ja avoimia sovelluksia sekä työkaluja, joiden avulla voisin toteuttaa harrastustani ohjelmoida vapaa-ajallani pienimuotoisia pelejä. Työkalun vaatimuksina oli, että se olisi mahdollisimman vapaakäyttöinen, mutta siitä löytyisi valmiita ominaisuuksia, kuten sovellusikkunan ja kuvien piirto, hiiri ja näppäinkomentoja sekä ääniominaisuuksia. Suunnitteilla olleet projektini olivat sisällöltään melko yksinkertaisia, joten työkalun tuli olla myös mahdollisimman ”kevyt” ja ”ohjelmointipainotteinen”. Valinta oli selvä viimeistään silloin, kun sain tietää Monogamen käyttävän jo minulle entuudestaan tuttua C#-ohjelmointikieltä.

Minkä takia sitten päätin kirjoittaa aiheesta opinnäytetyön?

Vaikka olinkin käyttänyt Monogamea jo jonkin aikaa, sen sisäinen toiminta oli minulle vielä varsin tuntematon. Se käyttää C#-ohjelmointikieltä ja pyrkii olemaan alustariippumaton, mutta miten se tarkalleen ottaen saavutetaan? Näistä syistä opinnäytetyön päätavoite on tutustua Monogamen toiminnallisuuteen, tutkia mitä kaikkea se sisältää saavuttaakseen alustariippumattomuutensa ja miten käyttäjä voi hyödyntää niitä sovellusten toteuttamisessa.

Tutustumisen kohteena ovat käytössä olevat rajapinnat, välikerrokset sekä nimiavaruudet, joista saatava tieto on ripoteltuna eri dokumentaatioihin ja verkkopalveluille. Tästä syystä opinnäytetyön osatavoitteena on myös löytää ja kerätä aiheista kaikkein oleellisin tieto ja koostaa niistä kattava kokonaisuus.

Työ sisältää myös esimerkkisovelluksen, joita lukija voi käyttää mallina omissa toteutuksissaan. Esimerkin tehtävänä on myös havainnollistaa Monogamen käyttöä perustasolla. Pyrin kirjoittamaan kokonaisuuden riittävän kattavaksi, jotta lukijan olisi helppo jatkaa Monogamen käyttöä ja tutustumista itsenäisesti. Pohjimmiltani haluan kuitenkin innostaa muita käyttämään ja tutustumaan sovellusten ja työkalujen sisältöön ja toiminnallisuuteen, mikäli siihen on mahdollisuus.

2 Monogamen historia

2.1 XNA – Monogamen juuret

Monogamen juuret ovat lähtöisin Microsoftin kehittämästä, ilmaisesta dotNET kehitysympäristöön pohjautuvasta ohjelmistokirjastosta, XNA:sta, joka tarkoitettiin oliopohjaiseen peliohjelmointiin. Sen avulla käyttäjä pystyi kehittämään ja hyödyntämään koodia kaikille Microsoftin tukemille alustoille, kuten Windows käyttöjärjestelmiin ja Xbox360 -pelikonsolille. Viimeisimmässä versiossa 4.0 se sai myös tuen Windows pohjaisiin puhelimiin.

XNA:n kehityksestä ilmoitettiin ensimmäisen kerran 24.3.2014 Kaliforniassa järjestetyssä ”Game Developers Conference” tapahtumassa (Microsoft 2004). 14.3.2006 ensimmäinen beta-versio julkaistiin nimellä ”XNA Game Studio Express”, josta lopullinen versio saatiin vasta 11.12.2006. Kirjaston kehitystä jatkettiin nimellä ”XNA Game Studio”, jota päivitettiin vuosien mittaan muutamaan kertaan, kunnes vuoden 2013 alussa Microsoft ilmoitti lakkauttavansa XNA:n kehittämisen (Promit 2013).

XNA:han kuului erilaisia pelikehitykseen liittyviä nimiavaruuksia, kuten 2D- ja 3D-grafiikan piirron, äänen toiston sekä syötelaitteiden, kuten Xbox360:n peliohjaimen hallinta. Oletuksena kirjastoon ei sisältynyt erillisiä työkaluja, mutta aktiiviset kehittäjät täydensivät tätä omilla sovelluksillaan, kuten FireFly Inc:n ”XNA 3D Level Editor” (CodePlex Archive 2019) sekä Colin Vellan ”tIDE”. Mainitsemisen arvoisena tIDE:ä käytetään yhä pelin ”Stardew Valley” karttaeditointiin (Vella 2016).

XNA:lla toteutettuja sovelluksia pystyi julkaisemaan Xbox360 konsolin ”Live Arcade”-palveluun erillisen ”Microsoft XNA Creator's Club”-ryhmän kautta, johon pystyi liittymään kuukaudeksi hintaan 49- tai vuodeksi 99.dollaria (Pelaaja-lehden artikkelin mukaan hinnat olivat euroissa samat (Pelaaja-lehti 2006). Englannin punnissa hinnat olivat 30/60 GBP (Gibson 2006)). Creators Club julkaistiin samalla XNA Game Studio Expressin kanssa (Microsoft 2006a). Julkaisun yhteydessä Microsoft julkisti ”Dream–Build-Play”-kilpailun, johon pystyi osallistumaan peliprojekteillaan. Ensimmäisen kerran kilpailu alkoi tammikuussa 2007 (Microsoft 2006b), josta pystyi voittamaan palkintoja, kuten kymmenen tuhannen dollarin pääpalkinnon sekä sovelluksen julkaisuoikeudet Xbox360 Live Arcade palveluun (Microsoft 2007). Alun perin kilpailua järjestettiin vuoteen 2012 saakka. Seuraavan kerran se järjestettiin vuonna 2017, jossa palkintona sovellus julkaistaisiin ”Universal Windows Platform”-alustalla. (Parsons 2017.)

9.10.2015 Microsoft ilmoitti Creators Clubin lakkauttamisesta ja esti palveluun rekisteröitymisen (Charla & Dunn 2015). Vuotta myöhemmin sovellusten julkaisu palvelussa päättyi. Ryhmän toiminta lakkautettiin lopullisesti seuraavana vuonna, lokakuussa 2017. Creator’s Club lopetti toimintansa vain nimellisesti, sillä sen korvasi samoihin aikoihin olemassa ollut, $99 vuosimaksulla toiminut ”Microsoft App Hub”. Alustan avulla kehittäjille mainostettiin sovelluskehitystä Windows 7 käyttöjärjestelmässä, sekä Windows puhelimissa. Kehittäjät pystyivät julkaisemaan sovelluksensa esimerkiksi ”Windows Phone Marketplace”-sovelluskaupassa. (Orland 2010.)

2.2 XNA Touch – Monogamen synty

Monogamen tarina ei suinkaan alkanut heti XNA:n kehityksen lopetuksen jälkeen 2013, sillä vuosien varrella XNA:sta on sen olemassaolon aikana pyritty kehittämään avoimempaa versiota eri kehittäjien toimesta. Varhaisin näistä on José Antonio Leal de Fariasin lokakuussa 2009 aloittama avoimen lähdekoodin projektin nimeltä ”XNA Touch”, jonka tavoitteena oli mahdollistaa sovellusten kehittäminen mobiililaitteille. Projektissa käytettiin apuna Bill Reissin kehittämää Silver Sprite-kääntäjää sekä hieman MonoXNA-kirjastoa. XNA Touch sai ensimmäisen julkaisunsa joulukuussa 2009 iPhonelle. (Monogame Team 2019c.)

2.3 SilverSprite & MonoXNA – XNA Touchin pohjakivet

XNA Touchin kehitys ei suinkaan alkanut tyhjästä vaan se pohjustettiin kahden jo olemassa olevan, avoimen lähdekoodin ohjelmiston avulla.

SilverSprite on kirjasto, jonka avulla XNA-sovelluksia pystyi suorittamaan käyttämällä Microsoftin Silverlight-ohjelmointiympäristöä ilman tarvittavia muutoksia koodiin. Tämän avulla XNA sovelluksia pystyi ajamaan kaikissa ympäristöissä, jotka tukivat Silverlightia. Työkalun kehitys jatkui Silverlightin viimeiseen 5. versioon saakka. (Microsoft/CodePlex 2018.) Silverlightin tukea jatketaan 12.10.2021 asti (Microsoft 2019b).

MonoXNA on ensimmäisiä yrityksiä toteuttaa yhteensopiva ohjelmistokehys XNA ympäristöön. Projektin alustava tavoite oli kattaa Windows PC -ympäristön lisäksi Linux ja Mac ympäristöt. OpenGL rajapintaa hyödynnettiin toteutuksen 3D ominaisuuksissa. (Google Code 2013.) Projektin Github sivun perusteella projektin oletetaan alkaneen vuoden 2006 aikana ja päättyneen vuoden 2013 aikoihin. Väitettä tukee projektin nettisivujen kaatuminen, XNA:n lakkauttaminen samana vuonna ja varsinaisen Monogame projektin yleistyminen. Mono.XNA:lla on tiedettävästi toteutettu Bomberman pelisarjasta-klooni, ”FERI Bomberman”, joka pelattavuudeltaan vastaa esikuvaansa. (MonoXna 2009.)

2.4 Mono ja Xamarin – Laajentamassa dotNET käyttöä

Monogamen nykyiseen toimintaan vaikuttaa suuresti Mono (Linux) ja Xamarin (mobiili) ohjelmistot, jotka ovat laajentaneet dotNET rajapinnan käyttöä Microsoft alustojen ulkopuolelle. Niiden historiaan on kuulunut monenlaisia tilanteita erilaisten yrityskauppojen ja omistajuuksien suhteen. Historia alkaa vuodesta 1999, kun Miguel de Icaza ja Nat Friedman perustivat Xamarinin edeltäjän, ”Ximianin”, joka kehitti, myi ja tuki Linux ja Unix pohjaisia, GNOME alustalla toteutettuja sovelluksia. Yritys tunnettiin aluksi myös nimillä ”Helix Code” sekä ”International Gnome Support”. Vuoden 2000 aikana, kesäkuussa Bill Gates ilmoitti Forum 2000 tapahtumassa Microsoftin ”seuraavan sukupolven Windows palveluista”, jotka brändättäisiin nimellä ”dot-net” (Fontana 2000). Kun ensimmäiset dokumentaatiot projektista julkaistiin joulukuussa 2000, Miguel de Icaza kiinnostui siitä ja alkoi tutkia projektin kehittämisestä Linuxille. Dokumentaatiosta puuttui kuitenkin Meta-dataan liittyvää tietoa, jota De Icaza kysyi dotNETin postituslistalta (eräänlainen foorumeita edeltänyt viestinnän muoto internetissä). De Icaza sai projektin kohdalta yhteydenottoja muilta kehittäjiltä, kuten Rhys Weatherlytä, joka oli ”takaisinmallintanut” (reverse engineering) dotNET alustan tiedostomuodon sekä koodia tiedostojen lataamiseen. Projektia voitiin jatkaa. Mono-projektin julkaisua suunniteltiin 19.7.2001 O’Reilly Conferenssiin (De Icaza 2003), mutta nykyisin julkaisupäiväksi on merkitty 30.6.2001 Mono-projektin sivuilla.

4.8.2003 Novell ostaa Ximianin, mutta Mono-projektin kehitystä jatketaan normaalisti. Projektin kehitysaskeleet julkaistaan ensimmäisen kerran vuonna 2003 (tarkka päivämäärä todennäköisesti on 1.11.2003). (De Icaza 2003.)

Nykyisin Linuxilla ja Unity pelimoottorissa käytettyä MonoDevelop IDE:ä aloitettiin kehittämään vuoden 2003 aikoihin ja sen ensimmäinen versio 0.1 julkaistiin 4.2.2004. Kehitys aloitettiin Mike Krugerin Windowsille tarkoitetusta C# IDE:stä, SharpDevelop. (Li 2019.) Ensimmäinen virallinen versio 1.0 julkaistiin 14.3.2008 ja Versio 2.0 julkaistiin 6.10.2008 (Novell Inc 2008).

14.9.2009 Xamarin.iOS (aiemmin MonoTouch) julkaistaan ensimmäisen kerran (De Icaza 2009). Työkalun avulla dotNET sovelluksia pystyi kääntäjään Applen kehittämille laitteille.

22.11.2010 Attachmate ostaa Novellin 2.2 miljardin dollarin hintaan. Oston yhteydessä satoja työntekijöitä irtisanotaan, laittaen Mono-projektin tulevaisuuden vaakalaudalle. (Waters 2011.)

16.5.2011 Xamarin perustettiin ja Miguel de Icaza ilmoitti Mono-projektin kehityksen jatkuvan. Alkuun ilmoitus loi kysymyksiä liittyen Monon lisensointiin ja kilpailukykyyn, mutta myöhemmin Novell antoi jatkuvan lisenssin Xamarinille projektin jatkamiseen. (De Icaza 2011.)

15.9.2014 Microfocus International ostaa Attachmaten 1.2 miljardin dollarin hintaan (Ballard & Gallivan 2014).

24.2.2016 Microsoft ostaa Xamarinin ja Mono-projektin (Guthrie 2016). Noin kuukausi myöhemmin 31.3.2016 Mono-projekti asetetaan MIT-lisenssin alaisuuteen (Friedman 2016). Nykyisin Mono-projektia ja Xamarin-sovellusta päivitetään säännöllisesti Microsoftin sponsoroimana.

2.5 Avoimempi Microsoft

Vuoden 2012 aikana Microsoft perusti avoimen lähdekoodin projekteihin keskittyvän organisaation nimeltä ”Microsoft Open Tech”, jonka alaisuudessa erilaiset Microsoft ohjelmistot saivat uuden suunnan.

DotNETin kääntäjiä työstettiin avoimempaan suuntaan projektilla ”Roslyn”, jonka tavoitteena oli kehittää uudenlaiset kääntäjät Visual Basic ja C#-kielille. Toteutus tapahtuisi kokonaan kääntäjän omassa koodikielessä (eli C# kääntäjä kirjoitetaan kokonaan C#-ohjelmointikielellä). (Kunk 2012.)

Vaikka projektin pohjustukset ja ideat avoimesta kääntäjästä alkoivat jo vuonna 2009, ei Microsoft ollut silloin vielä valmiina viemään sitä eteenpäin. Ajan kuluessa ja avoimien kääntäjien yleistyessä projektin suunta kuitenkin kääntyi. Huhtikuussa 2014 Anders Hejlsberg esitteli Roslynia avoimen lähdekoodin sovelluksena Microsoftin “Build” konferenssissa. Lopullinen julkaisu tapahtuikin 3.4.2014 Apache 2.0 lisenssillä Microsoftin “CodePlex”-palvelussa. (Torgersen 2018.)

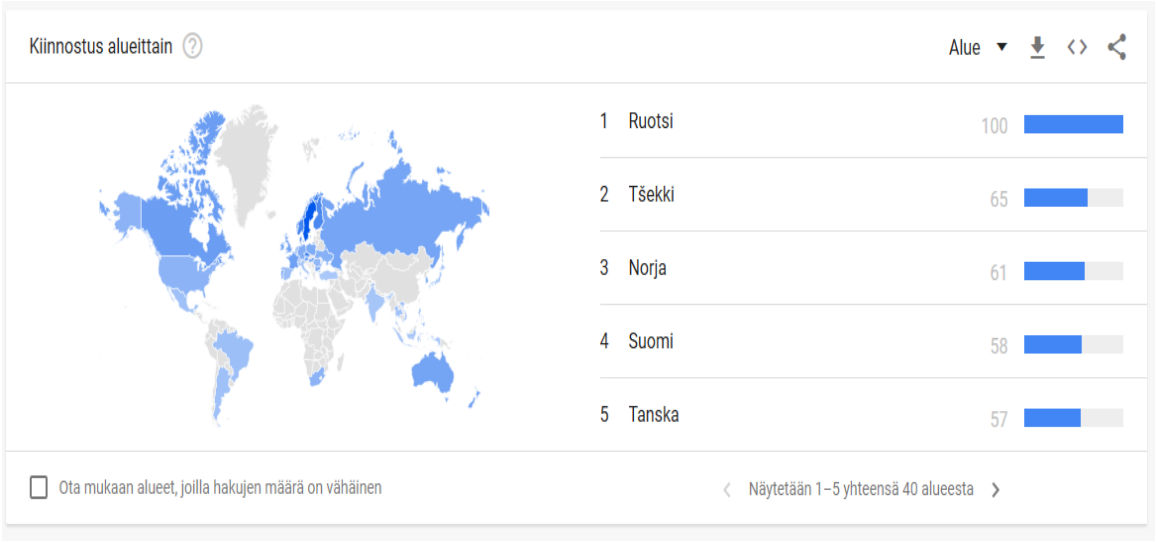
12.11.2014 Microsoft ilmoitti tekevänsä dotNET ympäristöstä ja sen ytimestä (NET.Core) avointa lähdekoodia. Päätöksen taustalta löytyi useita tekijöitä, kuten kilpailukyvyn säilyttäminen avoimiin ohjelmointikieliin ja ympäristöihin, kuten Javaan (Krill 2014) sekä laajentaa dotNETin käyttöä muille yleistyneille alustoille, kuten Linux, MacOS sekä mobiili -ympäristöille. Päätöksen jälkeen ympäristöä on kehitetty läheisessä yhteistyössä Mono -projektin ja Xamarinin kanssa. (Protalinksi 2014.) Microsoft Open Tech lakkautettiin vuoden 2015 aikana, sillä se saavutti tavoitteensa yleistää avoimen lähdekoodin projekteja (Paoli 2015).

2.6 Kehittyminen ja yleistyminen

Maaliskuussa 2011 XNA Touch projekti otti käyttöönsä nykyisen nimensä ”MonoGame”. Samana vuonna se sai tuen Android mobiililaitteille, kuten myös Windows, Mac ja Linux käyttöjärjestelmille. Projektin läpimurto tapahtui kuitenkin vasta vuosien 2012- ja 13 aikaan, kun Windows tukea jatkokehitettiin ja laajennettiin myös Windows puhelimiin. (Monogame Team 2019c.) Lisäksi vuonna 2013 ensimmäiset Monogamea hyödyntävät pelit ilmoitettiin julkaistavaksi PlayStation 4-pelikonsolille, joihin lukeutui muun muassa TowerFall (Matt Makes Games), Transistor (SuperGiant Games) sekä Mercenary Kings (Tribute Games). Kyseiset pelit ovat sittemmin julkaistu vuosien saatossa useille muillekin nykyaikaisille konsoleille, kuten Nintendo Switch tai XboxOne. (Monogame Team 2019c.) Läpimurtoon saattoi vaikuttaa myös XNA:n kehityksen päättyminen, jolloin alustaan tottuneet siirtyivät Monogamen pariin. Läpimurto on nähtävissä myös Google Trends palvelussa, jossa sanaa ”Monogame” on haettu eniten vuosien 2013 ja 2017 välillä (kuvio 1).

Kuvio 1. Google Trends -palvelun tulokset sanalle ”Monogame”

Monogamea on haettu Tšekin lisäksi kaikista eniten Pohjoismaissa (kuvio 2).



Kuvio 2. Google Trends palvelun keräämää tietoa mistä maista ”Monogamea” on haettu eniten.

Vuodesta 2014 lähtien projektista ovat vastanneet Tom Spilman sekä Steve Williams. Monogame on tutkittavissa GitHubissa ja sitä päivitetään säännöllisesti.

2.7 Monogamella toteutetut pelit

Monogamea on käytetty jo jonkin verran erilaisten sovellusten toteuttamiseen. Jokaisella niistä on ollut oma tarinansa ja syynsä käyttää Monogamen tarjoamia mahdollisuuksia. Esimerkiksi Monogamen pääkehittäjän Tom Spilmanin pelistudio, Sickhead Gamesin, toimintaperiaate perustuu videopelien kääntämiseen eri alustoille. Pelien joukossa on ollut paljon XNA toteutuksia, jotka ovat käännetty Monogamea hyödyntäen. Näihin peleihin lukeutuu muun muassa “Axion Verge” (Acevedo, 2016a), ”Celeste” (Matt Makes Games 2018) sekä ”Tooth and Tail” (Pocket Watch Games 2017). Seuraavaksi tutustaan tarkemmin muutamaan peliin ja niiden yhteydestä Monogameen.

2.7.1 Fez (POLYTRON)

Yksi varhaisimpia tunnettuja pelejä, jotka käyttivät Monogamen edeltäjää, XNA:ta. Peli on tullut tunnetuksi osittain sen pitkän kehitysajan, kehittäjästä Philippe Poissonista (Phil Fish), Indie Game The Movien sekä vuonna 2012 Independent Games Festival tapahtuman, ”Seumas McNally Grand Prize”-palkinnon voittajana (Independent Games Festival 2012). Pelin päämekaniikka on 3D-ulotteinen maailma, jota voi pyörittä ja tutkia sivu kerrallaan 2D-ympäristönä, jonka pohjalta peli yhdistelee tasohyppelyä ja pulmanratkontaa. Alun perin Xbox360 konsolille, Xbox Live Arcade palveluun 13.4.12 julkaistu peli, joka sittemmin käännettiin PC/MAC ja Linux alustoille käyttäen Monogamea pelin ohjelmoijan Renaud Bédardin toimesta. PC-käännös julkaistiin 1.5.13, MAC ja Linux -käännösten seuratessa 11.10.13 (Bédard 2014). Peli käännettiin myös PlayStation 3, 4 ja Vita konsoleille sekä iOS mobiililaitteille, ulkoistamalla tehtävä BlitWorks-pelistudiolle. Studion haastattelun mukaan peliä kääntäessä PlayStation Network-palveluun (PSN) pelin lähdekoodi ja Monogame jouduttiin kääntämään ”C++”-ohjelmointikielelle C#:n tuen puutteen vuoksi. Mikäli kehittäjät olisivat jatkaneet C#:n käyttöä ja kääntäneet vain Monogamen ajonaikaisen ympäristön, olisi sovellus ajautunut suorituskyky ongelmiin. (Wawro 2014.)

2.7.2 Stardew Valley (Chucklefish Games)

Soolokehittäjä Eric Baronen toteuttama avoin maatilasimulaattori, joka julkaistiin ensimmäisen kerran PC:lle 26.2.16. Pelin tavoitteena on hoitaa maatilaa viljelemällä ja kasvattamalla kasveja sekä hoitamalla eläimiä samalla yrittäen laajentaa maatilan toimintaa. Työskentelyn lomassa pelaaja voi tehdä tuttavuutta pelialueella sijaitsevan kylän asukkaisiin, mahdollisesti löytäen kumppanin ja perustaa perhe.

Alkuperäinen PC -julkaisu kehitettiin käyttämällä XNA:ta, mutta Mac ja Linux käännökset käyttävät Monogamea apunaan. Pelin konsoliversiot ulkoistettiin Sickhead Gamesille, jotka käänsivät pelin Xbox One, PlayStation 4 ja Vita sekä Nintendo Switch alustoille. (Barone 2016.) Android ja iOS -versioista vastaa pelistudio ”The Secret Police” (The Secret Police 2018). Peliä päivitetään yhä ja sen moninpeliominaisuus on tulossa Xbox ja PS4 versioihin (Barone 2019).

2.7.3 Infinite Flight -Flight Simulator (Infinite Flight LLC)

Lentokone simulaattori mobiililaitteille, jonka toteutuksessa on pyritty simuloimaan lentokoneen käyttöä mahdollisimman tarkasti tarjoamalla käyttäjälle useita oikeisiin lento- ja sotakoneisiin pohjautuvia malleja, kaikki suurimmat lentokentät, johon laskeutua, tutustuminen ilma-alusten moottorin käynnistykseen ja sammuttamiseen sekä paljon muuta (Infinite Flight LLC 2019). Monogame astui mukaan pelin kehitykseen, kun iOS-version työstäminen aloitettiin kääntämällä sovellus Windows puhelimista (Windows Phone). Kehittäjät kohtasivat alkuun jonkin verran haasteita järjestelmien käyttämien työkalujen suhteen, mutta päätyivät käyttämään Xamarinin tarjoamaa ”MonoTouch”–rajapintaa (eli Xamarin.iOS) (”DevToolsGuy” 2013). Lopulta kehittäjät löysivät Monogamen, jolla sovelluksen 2D-ominaisuudet toimivat

moitteetta. Sen toiminta muutenkin tuki projektia Windows pohjansa (XNA) ansiosta ja sisälsi tuen kehittäjien aiemmin käyttämään MonoTouchiin. Kehitystä päätettiin jatkaa kääntämällä sovelluksen 3D-ominaisuudet yhteensopivaksi (Infinite Flight LLC 2011). Sovellus on nykyisin saatavilla Apple App Storesta sekä Google Play kaupasta hintaan 5,49 euroa. Se sisältää myös erillisen tilaus mahdollisuuden (Infinite Flight Pro subscription), joka tarjoaa lisäsisältöä halukkaille.

3 Pelimoottori vai ohjelmistokehys

Ohjelmistona XNA:sta ja sittemmin Monogamesta puhutaan virallisesti ”ohjelmistokehyksenä”, runkona, jonka päälle sovellus kehitetään valmiiksi määritellyillä ehdoilla. Kuten pelimoottoritkin, ohjelmistokehyksen tehtävänä on helpottaa ja nopeuttaa sovelluksen kehittämistä (Rouse 2015), mutta suosittuihin pelimoottoreihin verrattuna Monogame ei tarjoa erillistä käyttöliittymää tai työkaluja, kuten kartta- tai animaatioeditoria. Onko siis Monogamen kutsuminen pelimoottoriksi oikein? Mikä erottaa ohjelmistokehyksen ja pelimoottorin toisistaan?

3.1 Pelimoottorin määritelmä

Pelimoottorit ovat kokonaisia sovelluksia, joilla kehittäjät voivat toteuttaa, yleensä videopeliaiheisen sovelluksen, käyttämällä valmiita ominaisuuksia, kuten grafiikan piirtäminen, äänentoisto, törmäystunnistus tai apuvälineet tuoda sovellus useille alustoille. Ominaisuudet ovat kokonaisia valmiiksi kehitettyjä työkaluja, joiden tavoitteena on helpottaa kehittäjien työtä ja antaa mahdollisuus keskittyä muihin projektin osiin. (Ward 2008.) Esimerkkinä Epic Gamesin Unreal Engine-pelimoottorin sisältämä visuaalinen työkalu ”Blueprints Visual Scripting”,

jolla voi toteuttaa projektin ohjelmointia graafisessa ympäristössä käyttämällä valmiiksi määriteltyjä toimintoja (”nodeja”).

3.2 Ohjelmistokehyksen määritelmä

Ohjelmistokehys ei ole kokonainen sovellus, vaan se on kokoelma erilaisia sovellusosia, kuten kirjastoja, kääntäjän tai tulkin sekä tuen OpenGL:n tai DirectX:n kaltaisille grafiikkarajapinnoille. Yleisesti ohjelmistokehykset sisältävät vain perustoimenpiteitä, kuten sovelluksen ikkunan ja grafiikan piirto, syötteet, äänentoiston perustoimia ja näitä ympäröivä runko, johon käyttäjän tulee kehittää sovelluksensa. Näiden tehtävänä on helpottaa käyttäjän työtä, mutta samalla antaa tarvittava vapaus jatkaa kehittämistä käyttäjän haluamaan suuntaan. Ohjelmistokehysrakenteessa kehys hallinnoi sovelluksen ohjausta ja tekee tarvittaessa kutsuja käyttäjän tekemään koodiin. (Computer Hope 2019.) Monogamen tapauksessa kutsuja tehdään aina kun pelin ruutua päivitetään. Ohjelmistokehyksen ansiosta sovelluksen lopullinen sisältö ja arkkitehtuuri saattaa olla helpommin hallittavissa, riippuen kehyksen avoimuudesta.

3.3 Väliohjelmiston määritelmä

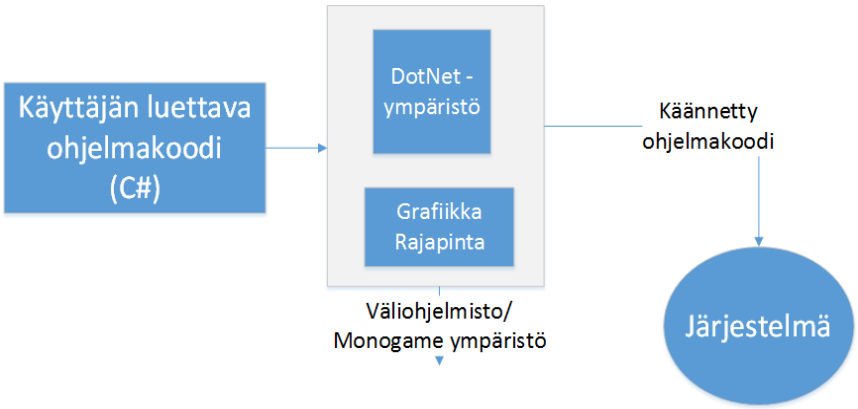
Kun puhutaan pelimoottoreista ja ohjelmistokehyksistä, on hyvä esitellä lyhyesti myös termi ”väliohjelmisto” (middleware). Terminä se on melko vapaamuotoinen, mutta yleisesti sillä tarkoitetaan erillistä, kokonaista ohjelmistoa, joka toimii sovelluksien välissä palveluna, yhdistäen ne ja mahdollistaen niiden välisen kommunikoinnin. Väliohjelmistot voidaan jakaa eri tyyppeihin, joilla on tietyt käyttötarkoitukset. (Sydeek 2017.)

Esimerkiksi tietokantojen määrittelyssä voidaan käyttää tietojen integrointiin liittyvää väliohjelmistoa, joka yhdistää eri lähteistä olevan tiedon yhteen näkymään. Käyttäjät voivat sittemmin tutustua ja käsitellä tätä näkymää (Red Hat 2019). Väliohjelmiston ja ohjelmistokehyksen toiminta on hyvin samanlainen,

mutta väliohjelmiston toiminta kattaa tietyn osa-alueen toteutuksen kokonaan sisältäen myös valmiiksi kehitetyt työkalut, eikä käyttäjän tarvitse jatkokehittää tai täydentää sen sisältöä lainkaan.

Pelimoottoritkin ovat jonkinlaisia väliohjelmistoja, sillä niiden toimintaperiaatteena on muuttaa käyttäjän tekemiä toimintoja järjestelmän ymmärtämään muotoon. Niiden yhteydessä väliohjelmistolla tarkoitetaan moottorin alla toimivaa osajärjestelmää (sub-system). (Sydeek 2017.)

Tunnettuja osajärjestelmiä ovat mm. Havok-fysiikkamoottori, FMOD-audiomoottori ja SpeedTree-grafiikkamoottori, joiden tehtävänä on tarjota valmiiksi määriteltyjä toimintoja pelimoottoriin. Tom Spilman on kuvaillut Monogamea väliohjelmistoksi Nintendo Switchille (Spilman 2017). Sen toiminta vastaa hyvin paljon väliohjelmiston käytänteitä, jossa väliohjelmisto yhdistää käyttäjän kirjoittaman ohjelmistokoodin haluttuun alustaan (kuvio 3) vaatimatta erityisiä toimenpiteitä käyttäjältä.



Kuvio 3. Esimerkki väliohjelmiston toiminnasta (kuvio: Joonas Niinistö).

3.4 Monogame - ohjelmistokehys

Monogame toimii ohjelmistokehyksen periaatteiden mukaisesti tarjoamalla valmiita ominaisuuksia ja funktioita eri toimintoihin, kuten grafiikan piirtämiseen ja äänentoistoon, jotka hyödyntävät käyttäjän valitsemaa rajapintaa. Jokaisen sovelluksen taustalla toimii Monogamen määrittelemä ohjelmakoodi (runko), joka suorittaa käyttäjän tekemää sovelluskoodia. Valmiin rungon lisäksi käyttäjällä ei

ole käytössä muita ominaisuuksia, kuten rungosta ajettua käyttöliittymää pelimaailman tai animaatioiden editoimiseen.
Monogame ei siis ole pelimoottori, vaan ohjelmistokehys, joka kattaa sovelluksen tekoon tarvittavat perusominaisuudet, joiden jatkokehitystä käyttäjä voi toteuttaa itsenäisesti.
Kehitystoimenpiteisiin voi sisältyä myös Monogamen kehittäminen pelimoottorimaisempaan suuntaan, kuten avoimen lähdekoodin ”Monofoxe”-projekti (”GnFur” 2019).

4 Monogame sovelluksen rakenne

Monogame mainostetaan ”alustariippumattomana” tapana kehittää sovelluksia, mutta miten tämä tarkalleen ottaen saavutetaan? Miten Microsoftin ohjelmistojen pohjalta kehittynyt ympäristö toimii esimerkiksi Linuxilla? Entä mitä Monogame sovellus pitää sisällään? Läpikäynti tutustuu Monogame-projektin runkoon ja toimenpiteisiin, joihin käyttäjä tulee ensisijaisesti törmäämään. Tämän jälkeen tutustutaan sovelluksen kääntämiseen erilaisille yleisille alustoille.
Työssä ei käydä läpi sovelluksen kääntämistä pelikonsoleille, joka edellyttäisi käyttäjältä rekisteröitymistä konsolin vaatimaan kehittäjäpalveluun ja hankkimaan maksulliset kehitystyökalut (development kit). Alustavasti aiheeseen voi kuitenkin tutustua Monogamen kehittäjien toteuttaman ”Brute” kääntäjän kautta, joka muuntaa C# -ohjelmakoodin konsolilla suoritettavaan muotoon (SickHead Games 2018). Projekti on suljettua lähdekoodia, mutta rekisteröityneet kehittäjät voivat ottaa sen käyttöön ilmaiseksi (Spilman 2019).

4.1 Parade – Esimerkkisovellus

Opinnäytetyössä käyttämäni esimerkkisovellus on aiemmin luomani peli, ”*Parade*”, jota päivitin hieman opinnäytetyötä varten. Alun perin toteutin projektin opetellakseni Monogamen perusteita sekä kerratakseni C#-osaamistani. Sen

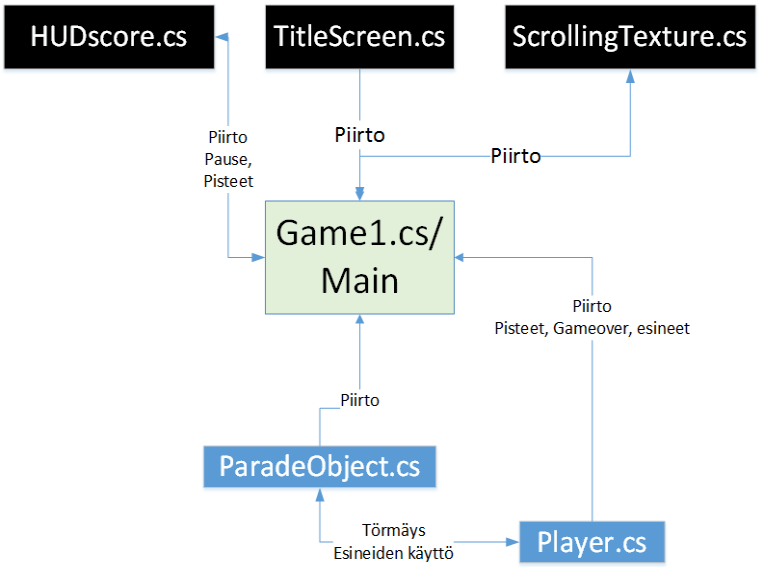
grafiikkarajapintana toimi DirectX rajapinta, jonka ”valinta” tapahtui suuremmin pohtimatta Youtubesta katsottujen opasvideoiden kautta.
Peli itsessään on ”loputon”, jonka tavoitteena on kerätä mahdollisimman paljon pisteitä ja väistellä vastaantulevaa paraatia. Törmääminen paraatiin tai joutuminen pelialueen ulkopuolelle johtaa pelin päättymiseen ja pisteiden nollautumiseen. Pelaajalla on käytössään myös esineitä, joilla voi vaikuttaa eri tavoin pelaamiseen. Yksinkertaisen luonteensa vuoksi projekti toimii hyvin esittelemään Monogamen ja perusteita, kuten projektin runko ja tiedoston hallintaa suorittavan ”Content Pipelinen” sekä näppäin- ja piirtokomennot.

4.1.1 C#-ohjelmointikieli

C# on moderni korkean tason ohjelmointikieli, jonka käyttäminen vaatii dotNET ohjelmistokomponenttikirjaston. Alkujaan kielen käyttö oli rajoitettu Windows-ympäristöihin, kunnes Microsoft teki C#:n kääntäjistä avoimempia. (.Net Foundation 2019.) Kielenä C# on olio-ohjelmointi pohjainen ja sisältää käyttäjän työtä helpottavia toimintoja, kuten roskien keruu (garbage collection). Jäsentelynsä ansiosta sovelluskoodin voi kirjoittaa ”hajotetusti”, tehden sen muokkaamisesta ja tulkitsemisesta helpompaa. Sen suunnittelussa on myös huomioitu turvallisuustoimintoja, kuten automaattinen muistinhallinta ja tyyppiturvallisuus, joka estää sovelluskoodia käyttämästä sille tarkoittamatonta muistipaikkaa. (Samual 2018.)
Ohjelmoinnin helpottamiseksi ja nopeuttamiseksi kielelle on tarjolla myös erilaisia kirjastoja. Esimerkkinä ”Base class Library” sekä Monogamen käyttämä XNA:n ”Microsoft.XNA”-kirjasto, joka sisältää pelin toteuttamiseen tarvittavia luokkia ja toimintoja. Sitä voi käyttää myös web-palvelujen kehittämisessä hyödyntämällä Microsoftin ASP.NET -rajapintaa (Microsoft 2019c).
C#-kielen kehysluokkakirjasto (Framework Class Library, FLC) on kokoelma erilaisia kirjastoja ja nimiavaruuksia, joita käyttäjä voi hyödyntää ohjelmoinnissa (Microsoft 2015a). ”System”-nimiavaruus kattaa suurimman osan luokkakirjaston sisällöstä ja siihen tullaan törmäämään vielä myöhemmin. C# on tuttu myös Unity (Unity Technologies 2019) ja GoDot (Ignacio Roldán Etcheverry 2017) pelimoottoreista. Syntaksiltaan kieli muistuttaa paljon Oraclen Java-kieltä.

4.1.2 Esimerkki sovelluksen rakenne

Pelisovelluksen sisältö on kirjoitettu eri luokkiin, jotka vastaavat tietyistä tehtävistä ja kommunikoivat muiden luokkien kanssa pääluokan (Game1.cs) kautta (kuvio 4). Luokat voidaan jakaa kolmeen eri kategoriaan (taulukko 1).



Kuvio 4. Parade -sovelluksen luokkarakenne (Kuvio: Joonas Niinistö).

Taulukko 1. Esimerkki sovellusten luokkien esittely.

Toiminnalliset luokat	Luokat, joihin kuuluu piirto funktionaalisuuden lisäksi muita toiminnallisuuksia kuten näppäimistön käyttöä tai törmäystunnistusta. Suurin osa luokkien sisällöstä toimii sellaisenaan luokan sisällä, mutta tilanteissa, joissa luokkien täytyy kommunikoida keskenään, toteutetaan pääluokan kautta. Yleisin kommunikointitilanne on objektien, kuten pelaajan ja esineiden välinen törmäys ja sen vaikutus peliin. Ryhmään kuuluu luokat ”ParadeObject.cs” sekä ”Player.cs”.
Piirtoluokat	Luokat, jotka kommunikoivat vain pääluokan kanssa. Luokkien pääasiallinen tehtävä on suorittaa erinäisiä piirtokomentoja, joiden näkyvyys määritellään pääluokan

	kautta. Ryhmään kuuluu ”TitleScreen.cs”, ”HUDscore.cs” sekä ”ScrollingTexture.cs”.
Game1.cs/ Main	Sovelluksen ydin, joka suorittaa sovellusta tekemällä muihin luokkiin kutsuja sekä välittää niiden tarvitsemaa tietoa, kuten pisteet tai ruudun koko. Viitataan termeillä pääluokka tai pääsilmukka.

4.1.3 Pääsilmukan esittely

Aloittaessasi luodaan valitsemasi sovelluspohjasta uusi perusta projektille, joka on jaettu kahteen eri luokkaan, ”Program.cs” (alla) ja ”Game1.cs”.

```
using System;

namespace ParaatiOpen
{
#if WINDOWS || LINUX
    public static class Program
    {
        [STAThread]
        static void Main()
        {
         using (var game = new Game1())
            game.Run();
        }
    }
#endif}
```

Program-luokka toimii sovelluksen ”tulopisteenä” (entry point), jonka avulla varsinainen käynnistys tapahtuu (”game.Run()”). Luokkaan ei tarvitse tehdä muutoksia ja sitä toistetaan, kunnes sovellus suljetaan (”game.Exit()”). Se kutsuu Monogameen määriteltyä ”Game”-luokkaa, josta luodaan uusi Monogamen ydinluokan (Game) esiintymä/ instanssi, ”Game1”. Uusi esiintymä sisältää funktioita, joiden pohjalle lopullinen sovellus toteutetaan (taulukko 2).

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace ParaatiOpen
{
    public class Game1 : Game
    {
```

```
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        protected override void Initialize()
        {
            base.Initialize();
        }

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
            ButtonState.Pressed || Keyboard.GetState().
            IsKeyDown(Keys.Escape))
            Exit();
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            base.Draw(gameTime);
        }
    }
}
```

Taulukko 2. Pääsilmukan oletus funktiot.

Initialize()	Suoritetaan sovellukseen liittyvä alustus, jonka aikana voidaan ladata ja määritellä ei graafista sisältöä. Parade-pelissä alustusta käytetään määrittelemään aliluokille oikeudet käyttää Content Manageria tiedostojen hallintaan sekä pisteiden lataamiseen. Alustus suoritetaan pelin ennen pääsilmukan toistoa.
LoadContent()	Funktiossa alustetaan määritellyt objektit sekä ladataan pelin sisältö (kuvat, äänitiedostot, 3D-mallit), jotka voivat

	olla Content Managerissa tai muutoin määriteltyjä tiedostoja. Vaikka funktiota kutsutaan kerran pelin alussa, tiedostoja voi ladata myöhemmin omalla määrittelyllä.
UnloadContent()	Mikäli peliin on ladattu tiedostoja muualta kuin Content Managerin kautta, ne voidaan poistaa tässä. Kutsutaan vain kerran pelin aikana. Pienissä sovelluksissa funktion käyttö ei ole tarpeellista.
Update()	Varsinainen pääsilmukka, jonka tehtävänä on päivittää peliä toistuvasti, kuten objektien liikettä, ääntä tai tarkistamalla käyttäjän antaman syötteen. Oletuksena päivitys tehdään 60 kertaa sekunnissa (60 fps).
Draw()	Vastaa sovelluksen graafisen sisällön piirtämisestä ja päivittämisestä. Esimerkkisovelluksessa funktio käyttää apunaan ”SpriteBatch”-objektia, joka vastaa 2D-grafiikan piirtämisestä. Päivitys tehdään myös 60 kertaa sekunnissa.

Sovelluksen toiminta perustuu näiden funktioiden käyttämiseen. Käyttäjä voi laajentaa pääsilmukkaa omilla funktioillaan sekä erillisillä luokilla. Näiden laajennusten tulee kuitenkin toimia edellä mainittuja funktioita käyttäen.

4.2 Välikerrokset (Wrapper)

Saavuttaakseen avoimuutensa sekä yhteensopivuutensa dotNET ympäristön kanssa, Monogame käyttää apunaan erilaisia ”välikerroksia”, joiden tehtävänä on kääntää alustakohtaisia rajapintoja sopivaksi C#-kielelle, jotta se pysyisi mahdollisimman yhteensopivana alustalla. Monogameen valikoidut välikerrokset ovat alustakohtaisia, sillä samoja välikerroksia ei voi käyttää kaikkiin alustoihin.

Välikerrosten toiminnallisuus on yhdistetty XNA:n nimiavaruuksiin. Tästä syystä samoja nimiavaruuksia voi käyttää alustasta riippumatta, eikä käyttäjän välttämättä tarvitse tietää välikerrosten toiminnasta tai olemassaolosta mitään. Niihin liittyy kuitenkin muutamia rajoitteita.

4.2.1 SharpDX – DirectX & UWP

SharpDX on avoimen lähdekoodin ja matalan tason välikerros, jonka avulla käyttäjä voi suorittaa DirectX:n toimintoja käyttäen, oletuksellisesti yhteensopimatonta C#-ohjelmointikieltä (Falk 2015). Monogame käyttää tätä Windows ympäristöissä, suorittaakseen toimintoja kääntämällä C#-koodin SharpDX:lle, joka taas välittää tiedon grafiikkarajapintaan.

4.2.2 Simple DirectMedia Layer (libSDL) – OpenGL

OpenGL-toteutukset käyttävät apunaan ”Simple DirectMedia Layer” nimistä välikerrosta, joka toimii natiivisti ”C” ja ”C++”-ohjelmointikielillä. Siihen löytyy tuki myös C#:lle, vaikkei kaikkia SDL:n ominaisuuksia ole pystytty kääntämään yhteensopivaksi kielen rajoitteiden vuoksi. (Lee 2019.) Käyttö ei ole rajoitettu vain Linux ympäristöön, vaan sitä voi käyttää Windows, iOS, sekä Android alustoilla (SDL Project 2019). Koska SDL ei ole DirectX:ään keskittyvä ohjelmisto, se tarjoaa SharpDX:n nähden muutamia vapautuksia, kuten mahdollisuuden käyttää kolmannen osapuolen peliohjaimia.

4.3 XNA:n nimiavaruudet

C#:ssa törmää termiin ”namespace” (nimiavaruus), jonka tehtävänä on organisoida suuria määriä ohjelmistokoodia, kuten luokkia ja funktioita ohjelmoinnin selkeyttämiseksi sekä päällekkäisyyksien välttämiseksi. Tämä sallii samojen toimintojen käyttämisen eri puolilla ohjelmistokoodia luomatta

virhetilannetta. Nimiavaruuksia voidaan pitää myös jonkin sortin arkistoina, jotka sisältävät saman ”aihepiirin” toimintoja. Nimiavaruuksien sisältö jaetaan ”jäseniin” (members), jotka sisältävät omat funktionaalisuutensa (taulukko 3).

Taulukko 3. Nimiavaruuden jäsenet (Microsoft 2015b)

Luokat (classes)	Sisältää perustoiminnot ja funktionaalisuuden.
Rakenteet (structures)	Sisältää joukon erilaisia muuttujatyyppejä, joita ohjelmoija voi ottaa käyttöönsä projektin aikana. Muuttujatyyppien käyttötarkoitus vaihtelee nimiavaruuden mukaan.
Luettelointi (Enumerations)	Tyyppi, jossa kokonaisluvut (int) määritellään omiin luetteloihin. Luettelon luvut saavat myös omat nimensä, joita voidaan käyttää ohjelmakoodissa nimeä vastaavan kokonaisluvun sijaan.
Liitännät (interface)	Määrityksiä, mitä nimiavaruuden luokat sisältävät, kuten parametrejä ja funktionaalisuutta. Luokissa funktionaalisuus voidaan määrittää vapaasti, kunhan ne noudattavat liitäntään annettuja määrityksiä. Esimerkkinä pääsilmukan ”Update”-funktio kuuluu ”Microsoft.Xna.Framework” nimiavaruuden rajapintaan ”IUpdateable”, joka vaatii palautettavaksi ”GameTime”-parametrin. Ohjelmoija voi sittemmin määritellä funktion sisällön haluamallaan tavalla, kunhan haluttu parametri palautetaan liitäntään.
Delegaatit (Delegates)	Objekteja, joihin voidaan tallentaa muualta ohjelmakoodista saatua viitetietoa. Tieto voidaan välittää objektin kautta muualle ohjelmakoodiin delagaatissa määritellyssä muodossa.
Sisäkkäinen nimiavaruus (Nested NameSpace)	Toisen (ulkoisen) nimiavaruuden sisällä. Se pystyy hyödyntämään ulkoiseen nimiavaruuteen tehtyjä toimintoja sekä määrittelemään omiaan. Pääasiallisesti sisäkkäisiä nimiavaruuksia käytetään ohjelmakoodin organisoimiseen,

	kuten määrittelemään mistä nimiavaruus on peräisin tai millaista tekniikkaa se käyttää.

Monogamen käyttämät XNA-nimiavaruudet noudattavat Microsoftin määrittelemää nimiavaruus nimeämismenetelmää:

```
<Company>.(<Product>|<Technology>)[.<Feature>][.<Nested namespace>]
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
```

(Microsoft 2008).

System nimiavaruus on kuitenkin poikkeus. Syynä tähän voi olla mahdollinen duplikaattien tai ”nimitörmäyksen” estäminen tai standardi, joka on määritelty Common Language insfrastruktuuriin (CLI), johon System nimiavaruus kuuluu ”The Base Class Library” kirjaston kautta.

```
using System;
using System.IO;
using.System.Xml;
using System.Collections.Generic;
```

4.3.1 Microsoft.Xna.Framework

Monogame käyttää apunaan XNA:sta perittyjä nimiavaruuksia, jotka sisältävät peliohjelmointiin liittyviä toimintoja. Olen valinnut muutamia aihealueita, jotka esiintyvät Parade-pelin ohjelmakoodista tai ovat muuten hyödyllistä tuoda työssä esille. Ensimmäisenä läpikäytävä nimiavaruus sisältää Monogame sovelluksen toimintaan liittyviä perustoimenpiteitä, joita ajetaan yleensä taustalla ilman käyttäjän vaatimia toimintoja.

Game (Class)

Sovelluksen selkärankana toimiva luokka, joka kattaa kaikki sovelluksen perustoiminnot, kuten sovelluksen käynnistys, alustus, ruudun päivitys ja piirto.

Aiemmin käsitellyt funktiot, ”Initialize”, ”Update” ja ”Draw” peritään tästä luokasta ja noudattavat luokkaan määriteltyjä ehtoja (taulukko 4).

Taulukko 4. Game-luokan jäseniä

IsActive	Tarkistaa onko sovellus aktiivisena. Jos käyttäjä pienentää sovellusikkunan tai valitsee toisen sovelluksen aktiiviseksi, funktio pysäyttää sovelluksen toiminnan, kunnes se on valittu taas aktiiviseksi (Microsoft 2011b).
IsFixedTimeStep & TargetElapsedTime	Monogame tukee kahta tapaa päivittää sovelluksen ruudun piirtoa, reaaliaikaista (variable-step) tai ”kiinteää aikaa” (fixed time). Molemmat toiminnot käyttävät päivityksessä GameTime parametria. Reaaliajassa päivitykset toteutetaan heti edellisen päätyttyä. Tämän seurauksena ruudun päivityksen toisto saattaa vaihdella, vaikuttaen sovelluksen nopeuteen.

Ruudun päivitys

Käyttäessäsi kiinteää aikaa, sovellus päivittää ruutua ”TargetElapsedTime” parametriin määritellyn ajan tahtiin, oletuksena 60 päivitystä sekuntiin (60 fps).

```
private TimeSpan _targetElapsedTime = TimeSpan.FromTicks(166667); // 60 fps
```

Mikäli ruudun päivitys tapahtuu oletettua hitaammin, luokkaan määritelty toiminto kutsuu Update-funktiota tiheämpään tahtiin saavuttaakseen takaisin oletustilansa. Tällöin ruudun päivitystä kuitenkin vähennetään hidastuksen minimoimiseksi. Oletuksena Monogame käyttää kiinteää aikaa.

```
private bool _isFixedTimeStep = true;
```

(Ohjelmakoodin pätkät Monogamen lähdekoodista, luokasta ”Game1.cs”) (Monogame Team 2019h).

Vector2 & Rectangle (structure)

Vector2 on kaksiulotteiseen ympäristöön tarkoitettu rakenne, jonka avulla määritellään kaksi pistettä, yleensä objektin tai kuvan sijainti. Vastaavasti Vector3 on tarkoitettu kolmelle pisteelle (X, Y, Z) ja Vector4 neljälle (X, Y, Z, W). Tällaisia vektoreita hyödynnetään etenkin 3D-ohjelmoinnissa. Suorakulmio (Rectangle) on rakenne, johon syötetään tiedot objektin pituudesta, leveydestä sekä X- & Y-sijainnit, joita voidaan täydentää vektorin avulla. Esimerkkitilanteena Paraden ”vihollisobjektit”, paraatilaiset. Objektin määrittely alkaa hakemalla pääsilmukassa valmiiksi ladattu kuva (Texture2D image). Kuvan koon perusteella määritellään vector2 muuttujat ”TextureWidth” sekä ”Height”. Kun muuttujat on määritelty, annetaan ne suorakulmioon nimeltä ”personSize”, joka asettaa objektin lopullisen koon, joka pelissä näytetään.

```
public ParadeObject(Texture2D image,int newSpeed,int newPoints,int maxAnim)
{
    paradePerson = image;
    TextureWidth = (image.Width / maxAnim);
    TextureHeight = image.Height;
    personSize = new Rectangle(-100,0,(TextureWidth/3),(TextureHeight/ 3));
```

Animaation näyttämisestä vastaa suorakulmiomuuttuja ”PersonAnimate”.

```
public virtual void Animation(int Speed) // Default animation
{
    PersonAnimate = new Rectangle(TextureWidth * CurrentFrame, 0,
    TextureWidth, TextureHeight);
    frameCycle++;

    if (frameCycle >= Framedalay)
    {
       CurrentFrame = (CurrentFrame + 1) % (MaxFrames - LastFrame);
       frameCycle = 0;
    }
}
```

Lopuksi objekti piirretään, käyttämällä aiemmin määriteltyjä muuttujia:

```
spriteBatch.Draw(paradePerson, personSize, PersonAnimate, Color.White);
```

GameTime (Class)

Update ja Draw-funktiot palauttavat tästä luokasta perityn parametrin nimeltä ”GameTime”. Se sisältää aika-arvon, jonka perusteella sovellus tekee päätöksen, kuinka nopeasti peliä päivitetään (taulukko 5).

Taulukko 5. GameTime-luokan jäseniä

IsRunningSlowly	Palauttaa boolean totuusarvon sovelluksen suoritusnopeuden perusteella. Mikäli sovellus on oletettua hitaampi, arvo on tosi. (Monogame Team 2019h.)
TotalGameTime	Määrittää ajan, kuinka kauan sovellus on ollut yhtäjaksoisesti käynnissä.
ElapsedGameTime	Palauttaa aika-arvion kuinka kauan sovelluksella kesti toistaa Update()-funktio.

TotalGameTime ja ElapsedGameTime ovat osa TimeSpan kirjastoa, joka kuuluu C#:n Base Class kirjastoon. TimeSpan tulee viittä aikamuuttujaa, Days, Hours, Minutes, Seconds ja Milliseconds. Muuttujista on myös variaatit, ”Total”-etuliitteellä. Total alkuiset muuttujat ovat tarkoitettu sovelluksen sisäisiin laskutoimituksiin, kuten animaation vaihtoon liittyvä suurempi tai yhtä suuri vertailu. Niiden avulla mitataan yhtä aika-arviota, jonka voi esittää haluamassaan muodossa. Muuttujat ilman Total-etuliitettä ovat suunniteltu käytettävän yhdessä näyttämään ajan kokonaisuutena. (Microsoft 2010b.)

GraphicsDeviceManager (Class)

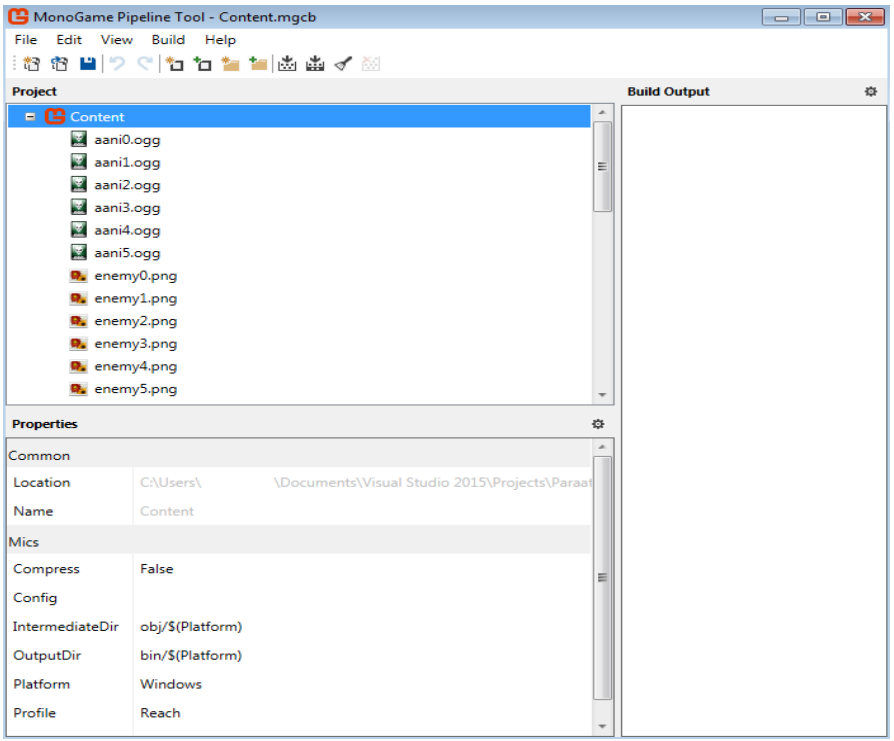
Luokan tehtävänä on hallinnoida sovelluksen käyttämää grafiikkalaitetta (Graphics Device). Hallinnointi määrityksiin kuuluu sovelluksen resoluutio (PreferredBackBufferWidth & Height), ruututila (IsFullScreen) ja sen vaihto (ToggleFullScreen). Aiheesta lisää kappaleessa 5.3.3.

4.3.2 Microsoft.XNA.Framework.Content

Tiedostojenhallintaan on tarjoilla valmiita toimenpiteitä, jotka saadaan käyttöön tällä nimiavaruudella. Parade-pelissä tätä nimiavaruutta käytetään visuaalisten ominaisuuksien, kuten grafiikan ja äänien määrittelyyn.

Content Pipeline ja Pipeline Tool – Tiedostojen hallinta

Kehittäessään sovellusta, käyttäjä voi hallinnoida tiedostojaan kahdella tavalla. Ensimmäinen vaihtoehto on “raaka data”-lähestymistapa, jossa tiedostojen käsittelyn käyttäjä hoitaa manuaalisesti. Tällöin käyttäjällä on kaikki valta käsittelyn toteuttamiseen. Toinen tapa on käyttää **Content Pipelineä**. Kyseessä on sisäkkäinen nimiavaruus Content-nimiavaruudelle, joka sisältää toimintoja erilaisten tiedostojen käsittelyyn, kuten niiden lukuun, pakkaamiseen ja purkamiseen huomioimalla samalla käyttäjän määrittelemä kohdealusta. Content Pipelinen yhteydessä Monogame tarjoaa **”Pipeline Tool”**-työkalun, jolla hallinnoida projektin tiedostoja ja niiden teknisiä toimenpiteitä (kuva 1).



Kuva 1. Pipeline Tool -ikkuna (Kuva: Joonas Niinistö).

Työkalulla tehtyjen määritysten perusteella luodaan erillinen ”mgbc”-tiedosto (Monogame Build Content), johon sisältyy kaikkien projektiin määriteltyjen tiedostojen tekniset määritykset sekä sijainti. Tiedostoa käytetään apuna sovelluksen sisällön prosessointiin, kun sovellusta käännetään (build) suoritettavaan muotoon. Tällöin tiedostot pakataan määriteltyyn tiedostomuotoon (oletuksena xnb) ja siirrettään automaattisesti yhteiseen kansioon, josta ne voidaan ladata Content Managerin avulla.

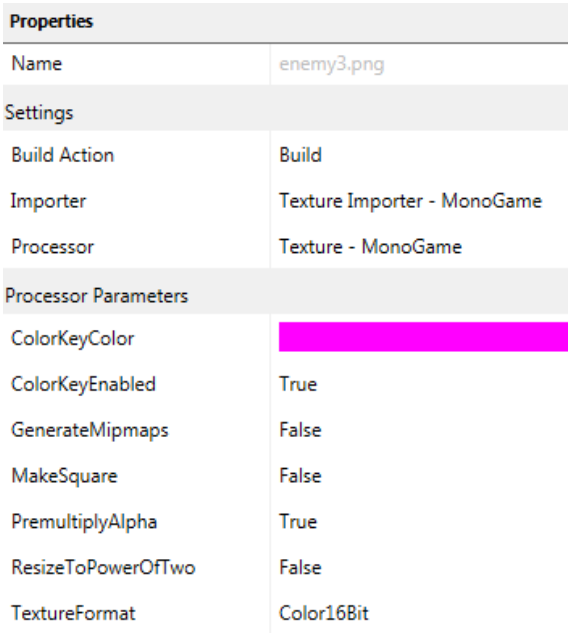
Työkalu on ideaalinen projekteissa, jossa tiedostoja päivitetään tiheään tahtiin. Esimerkiksi tekstuuria päivittävä graafikko voi työskennellä päivityksensä parissa ilman ohjelmoijan tai muun kehittäjän hyväksyntää (sovelluksen rakenteen toimivuuden kannalta). Ohjelmoijan taas ei tarvitse tehdä erityisiä päivityksiä ohjelmakoodiin tai muihin Monogamen valmiisiin toiminnallisuuksiin. (Jackson 2016.) Content Pipeline tukee useita tiedostomuotoja, jotka voivat olla tekstuureja, 3D-malleja, mediatiedostoja tai efektejä (taulukko 6).

Taulukko 6. Monogamen tukemat tiedostomuodot.

Tekstuuri	.png, .jpg, .bmp, .dds, .dib, .hdr, .pfm, .ppm, .tga
3D-mallit	.x, .fx (efektit), .fbx, .obj
Ääni	.mp3, .wav, .wma, .ogg
Video*	.mp4(H264), .wmv
Muuta	.spritefont (XML)

Processor Parameters

Prosessointi parametrit määrittävät, kuinka sovelluksen kokoonpanossa tiedostot käsitellään. Eri tiedostotyypeille on eri vaihtoehtoja (kuva 2). Esimerkiksi ”ColorKeyColor” parametri ottaa kuvasta käyttäjän valitseman taustavärin ja poistaa sen. Tiedostokokojen kasvamisen rajoittamiseksi on myös suositeltavaa asettaa parametri ”Compress” todeksi (kuva 1). Kun ohjelmakoodi käännetään suoritettavaksi sovellukseksi, mgcb-tiedostoon tallennetaan jokaiseen Pipelinen kautta ladattuun tiedostoon tehdyt prosessointi määritelmät. Prosessointiin liittyvistä määrityksistä ja niihin liittyvistä parametreistä enemmän Monogamen dokumentaatiossa. (Monogame Team 2019.)



Properties	
Name	enemy3.png
Settings	
Build Action	Build
Importer	Texture Importer - MonoGame
Processor	Texture - MonoGame
Processor Parameters	
ColorKeyColor	
ColorKeyEnabled	True
GenerateMipmaps	False
MakeSquare	False
PremultiplyAlpha	True
ResizeToPowerOfTwo	False
TextureFormat	Color16Bit

Kuva 2. Pipeline Tool-työkalun määrityksiä png-kuvatiedostolle (Kuva: Joonas Niinistö).

Määritykset SpriteFont-tiedostolle, sekä ”introbg.png”-tekstuurille:

```
#------------- Global Properties -------#

/outputDir:bin/$(Platform)
/intermediateDir:obj/$(Platform)
/platform:Windows
/config:
/profile:Reach
/compress:False

#------------- References --------------#

#------------- Content -----------------#

#begin fontti.spritefont
/importer:FontDescriptionImporter
/processor:FontDescriptionProcessor
/processorParam:PremultiplyAlpha=True
/processorParam:TextureFormat=Compressed
/build:fontti.spritefont

#begin introbg.png
/importer:TextureImporter
/processor:TextureProcessor
/processorParam:ColorKeyColor=255,0,255,255
/processorParam:ColorKeyEnabled=True
/processorParam:GenerateMipmaps=False
/processorParam:PremultiplyAlpha=True
/processorParam:ResizeToPowerOfTwo=False
/processorParam:MakeSquare=False
/processorParam:TextureFormat=Color
/build:introbg.png
```

ContentManager (Class)

Content Manager on ajo-aikainen komponentti, jonka tehtävänä on hallinnoida sovelluksen dataa lataamalla ja poistamalla sitä. Lataaminen määritellään ohjelmakoodiin hyödyntämällä ”Load()”-funktiota. Ennen lataamista, halutut tiedostot tulee olla käsiteltynä Content Pipelinessä (Content.mgcb), josta Content Manager hakee ne. Alla esimerkki tyypillisestä datan lataustilanteesta:

```
Texture2D MalliKuva = ContentManager.Load<Texture2D>(”image1”);
```

Ensin valitaan, mihin muuttujaan data ladataan (Texture2D muuttuja nimeltä MalliKuva). Lopuksi määritellään haettava data, joka on tallennettua Content Pipelineen nimellä ”image1”. Tiedostomuotoa ei tarvitse ilmoittaa. Tiedostot haetaan ”RootDirectory”-parametriin määritellystä kansiosta, joka käyttäjän tulee antaa tekstimuodossa (string). Oletuksena RootDirectory on määritelty jokaisen Monogame projektin yhteydessä kansioon ”Content”.

```
Content.RootDirectory = "Content";
```

Content Managerin kautta ladatut tiedostot voi myös poistaa käyttämällä luokan tarjoamia vaihtoehtoja. Mikäli käyttäjä haluaa poistaa tietyn tiedoston, tulee käyttää ”Dispose()”-funktiota tai ”Unload()”, kun hän haluaa tyhjentää Content Managerin sisällön kokonaan.

```
Ladattutiedosto.Dispose();
```

```
ContentManager.Unload();
```

Miksi käyttää Content Manageria ”FileStream” luokan (System.IO) sijaan?

Käyttäjän kannattaa pohtia millainen ratkaisu sopii parhaiten hänelle ja projektilleen. Content Manager huolehtii käyttäjän puolesta erilaisista toiminnoista, kuten tiedostojen hallinta ja purkaminen (unload). Tämä helpottaa käyttäjän työtä, säästää aikaa sekä vähentää riskiä liittyen mahdolliseen muistin ylivuotoon. Pipeline Tool puolestaan pystyy käsittelemään useita erilaisia tiedostomuotoja, joiden asetuksia käyttäjä voi määritellä erillisen käyttöliittymän

kautta. Edellä mainituista syistä Content Manager ja Pipeline käyttävät jonkin verran sovelluksen lopullista muistin käyttöä, verrattaessa FileStreamiin.

FileStream tukee yksinkertaista tiedostojen hallintaa, kuten grafiikan lataamista. Äänitiedostoja varten on olemassa erilliset ”FromStream” sekä ”FromUri”-luokat ja näiden käyttäminen olisi riittänyt sellaisenaan Parade-pelin sisällön lataamiseen. Luokkien käyttö edellyttää käyttäjältä enemmän toimintoja, kuten tiedostojen manuaalinen hallinta. Tämä kuitenkin tarjoaa käyttäjälle mahdollisuuden lisätä ja tehdä omia muokkauksia tiedostojen hallintaan ja käyttöön. FileStream-luokan nimiavaruutta, ”System.IO” voi käyttää myös Monogame pohjaisten projektien ulkopuolella tarjoten vastaavanlaisen tiedoston hallinnan kaikkiin dotNet-pohjaisiin projekteihin. Loppujen lopuksi, FileStream menetelmän käyttäminen on kuitenkin yksi tapa tuoda dataa projektiin.

4.3.3 Microsoft.XNA.Framework.Graphics

Nimiavaruuteen on koottu XNA:n ja Monogamen graafisia toimintoja järjestelmätason tehtävistä 2D-grafiikanpiirtoon ja siitä menetelmiin, jotka hyödyntävät saatavilla olevan laitteiston kiihtyvyysominaisuuksia kolmiulotteisten objektien näyttämiseen.

GraphicsDevice

GraphicsDevice luokka edustaa grafiikanhallintaan liittyvää ”väylää” käyttämäsi järjestelmän ja sovelluksen välillä. Sen tehtävänä on tarkistaa, millaisia grafiikkalaitteita on saatavilla ja mahdollistaa niiden käyttö. Järjestelmätasolla se vastaa primitiivipohjaisesta renderoinnista, käsittelee järjestelmätason muuttujia, gammatasoja sekä luo erilaisia varjostimia. (Microsoft 2012.) Pelin ohjelmakoodissa GraphicsDevice-luokkaan tehdään vain yksi suoranainen muutos, joka vaihtaa pelin oletus taustaväriä:

```
GraphicsDevice.Clear(Color.SlateGray);
```

Aloittaessasi uutta projektia käyttäjän ei tarvitse luoda erillistä GraphicsDevice-objektia. Kaikki oleellinen siihen liittyen määritellään GraphicsDeviceManager-objektin yhteydessä. (Microsoft 2010a.)

```
graphics = new GraphicsDeviceManager(this);
```

GraphicsDeviceManager objekti vastaakin lähes kaikista sovelluksen grafiikan piirtoon liittyvistä perusteista.

GraphicsDeviceManager ja ruudun piirto

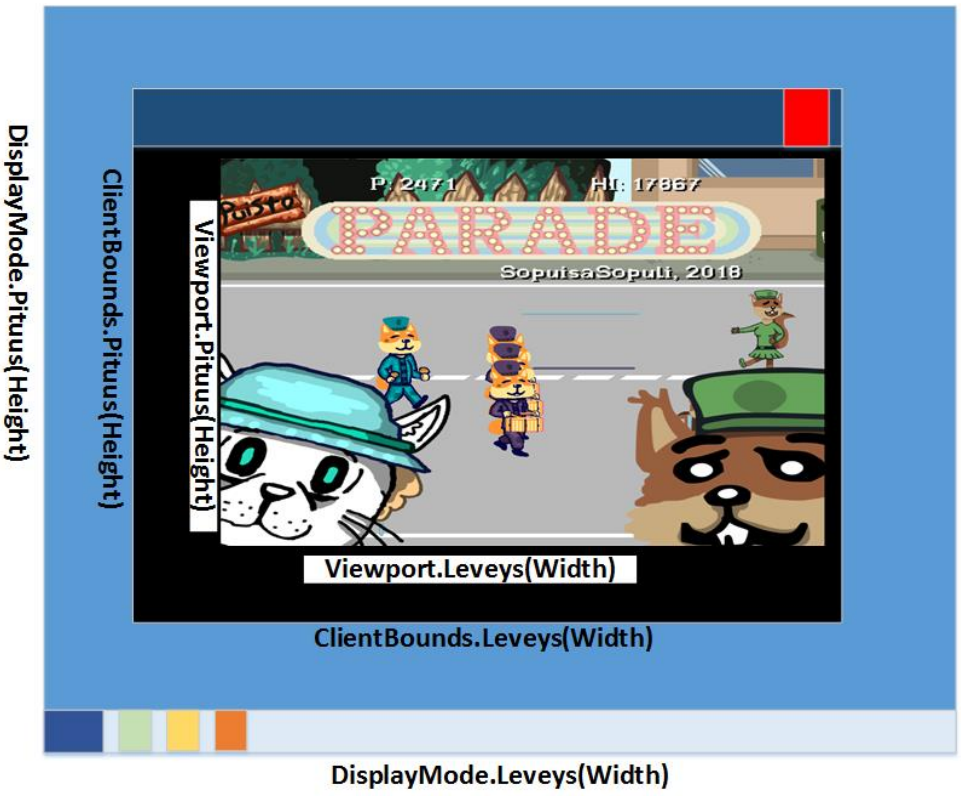
Monogamessa ruudun piirtoon vaikuttaa useampi luokka, jotka ovat sijoiteltu useamman nimiavaruuden alaisuuteen. GraphicsDeviceManager luokan avulla käyttäjä voi määritellä helposti haluamansa resoluution sovellukselleen. Ruudun piirtoon liittyy kuitenkin paljon tekijöitä, jotka on hyvä ottaa huomioon.

Ensimmäisenä on ”DisplayMode”, joka hakee järjestelmän käyttämän resoluution, väritarkkuuden sekä päivitystaajuuden. Luokkaa käytetään apuna, kun sovelluksen ruutua vaihdetaan ikkuna ja kokoruudun väliltä, jolloin sovelluksen resoluutio pyritään sovittamaan järjestelmän resoluutioon.

”ClientBounds” ominaisuus kuuluu ydin nimiavaruuden ”GameWindow”-luokkaan, joka määrittelee sovelluksen käyttämän ruudun/ikkunan koon (ClientBounds.Width ja Height) sekä sijainnin (ClientBounds.X ja Y).

Viimeisenä on ”Viewport” suorakulmio, johon sovelluksen varsinainen kuvasisältö määritellään. Oletuksena suorakulmion koko määritellään ”PreferredBackBuffer” arvojen mukaisesti. Luokkaan sisältyy pituus ja leveys asteiden lisäksi syvyys asteesta 3D-ohjelmointia varten. Se tulee määritellä ennen piirtokomentoja, jotta piirtäminen tapahtuisi oikein. 3D-ohjelmointia varten luokkaa kannattaa käyttää yhteistyössä määritellessä sovelluksen kameran sijaintia. Luokan avulla ruudusta voidaan piirtää vain haluttu osa tai monistaa se, mahdollistaen erilaisia toimintoja, esimerkkinä moninpelissä kuvan jakaminen pelaajille. Oletuksena Monogame käyttää vain yhtä viewport-suorakulmiota.

Takapuskuri (BackBuffer) sekä etupuskuri (FrontBuffer) vastaavat lopullisesta piirrosta. Takapuskuri piirtää seuraavan kuvan, jonka valmistuessa se lähetään etupuskuriin näytettäväksi ja prosessi alkaa alusta. Puskurien koko määritellään sovelluksen oletus resoluution mukaan (PreferredBackBuffer). (Microsoft 2010c.) Esimerkki (kuva 3), kuinka Monogame lukee käytössä olevia resoluutioita. Kuvassa Viewport on määritelty ikkunakokoa (ClientBounds) pienemmäksi.



Kuva 3. Havainnollistava esimerkki kuvapuskurien toiminnasta Monogamessa (Kuva: Joonas Niinistö).

Monogamessa takapuskurin voi määritellä seuraavasti:

Takapuskurin kooksi määritellään 800x800.

```
graphics.PreferredBackBufferWidth = 800;
graphics.PreferredBackBufferHeight = 800;
```

Takapuskuri määritellään yhtä suureksi järjestelmän resoluution kanssa:

```
graphics.PreferredBackBufferWidth = GraphicsDevice.DisplayMode.Width;
graphics.PreferredBackBufferHeight = GraphicsDevice.DisplayMode.Height;
```

SpriteBatch

Spritebatch on luokka 2D-grafiikan piirtämistä varten ja vastaa Parade-pelin graafisen ilmeen tulostamisesta ruudulle. Vaikka luokka onkin tarkoitettu kaksi ulotteisen grafiikan näyttöön, tapahtuu grafiikan renderöinti 3D-tilassa, ortografista kameratilaa käyttäen (Orthographic Camera). Tilalla tarkoitetaan piirtoa, jossa ei ole syvyyttää (Z-akseli), jolloin ladatut kuvat asetetaan kaksiulotteiseen neliöön ja piirretään ruudulle samansuuntaisesti kameran kanssa. Luokan tavoitteena on myös minimoida piirtokomentojen aiheuttama, sovelluksen suorituskykyyn vaikuttava kuormitus. Spritebatchin toiminta tapahtuu kutsumalla ”Start()”-funktiota, jonka jälkeen kutsutaan kaikki piirrettävät kuvat, ”Draw()”-funktiota käyttäen. Nimensä mukaisesti luokan tehtävänä on ”niputtaa” (batch) kaikki piirtokomennot talteen ja piirtää ne vasta kun ”End()”-funktio on määritelty. Esimerkkinä pelin logon taustakuvan piirto tapahtuu seuraavasti:

```
spriteBatch.Begin();
      spriteBatch.Draw(logoBG, introBGSize, introBGAnim, Color.White);
spriteBatch.End();
```

```
[spritebatch]. Draw ([Kuva], [Kohde -suorakulmio], [Lähde -suorakulmio],
[väri]);
```

Niputukseen voi asettaa myös erilaisia toimenpiteitä:

```
SBParallax.Begin(SpriteSortMode.Deferred, null, SamplerState.LinearWrap,
null, null);
      Scroll.Draw(SBParallax);
SBParallax.End();
```

```
[spritebatch]. Begin (SpriteSortMode, BlendState, SamplerState,
DepthStencilState, RasterizerState);
```

Esimerkissä SpriteBatch objekti ”SBParallax”:lle on määritelty muutamia toimenpiteitä, jotka vaikuttavat luokan ”ScrollingTexture” grafiikan piirtämiseen. Kyseinen luokka sisältää pelin taustakuvat, jotka liikkuvat jatkuvasti kohti ruudun oikeaa reunaa. SBParallaxin tehtävänä on mahdollistaa tietyn taustakuva elementin toistuva piirtäminen. Tämän saavuttamiseksi objektiin on määritelty parametri ”SamplerState” (taulukko 7), jonka tilana on määritys ”LinearWrap”.

Taulukko 7. SpriteBatchin parametrit, joita käyttää grafiikan piirtoon.

SpriteSortMode	Määrittelee kuvien piirtojärjestyksen. Oletuksena piirto tapahtuu niiden kutsumisjärjestyksessä. Aiemmin piirretyt kuvat jäävät kuitenkin aina uusien alle.
BlendState	Värien hallintaan ja sekoittamisesta vastaava parametri, jonka avulla voidaan määritellä toimintoja kuten kuvan häivyttäminen (Power 2013).
SamplerState	Lineaarinen suodatustila piirretyille tekstuureille. Parametrin avulla voidaan määrittää piirrettävälle grafiikalle eri toimintoja, kuten ”LinearWrapin” tarjoama toistuvuus.
DepthStencilState	Parametri sabloni/ sapluunakuvan piirtoon ja hallintaan (Franks 2014).
RasterizerState	3D-mallien hallintaa liittyvä parametri, jonka avulla käyttäjä voi määritellä kuinka vektoridata käännetään rasterimuotoon (pikseleiksi) (Microsoft 2011a).
Effect	Käyttäjä voi määritellä oman efektiluokkansa, joka ylikirjoittaa Monogamen käyttämän oletusluokan. Ylikirjoittaminen estää oletusluokan kuuluvien aliluokkien, kuten ”SpriteEffects” käytön. (Monogame 2019.)
Matrix	Matriisi, jota voi vapaavalintaisesti käyttää grafiikan piirron geometrian muuttamiseen

Parametrien eri tiloista ja niiden toiminnallisuudesta voi lukea lisää Monogamen virallisesta dokumentaatiosta. (Monogame Team 2019f.)

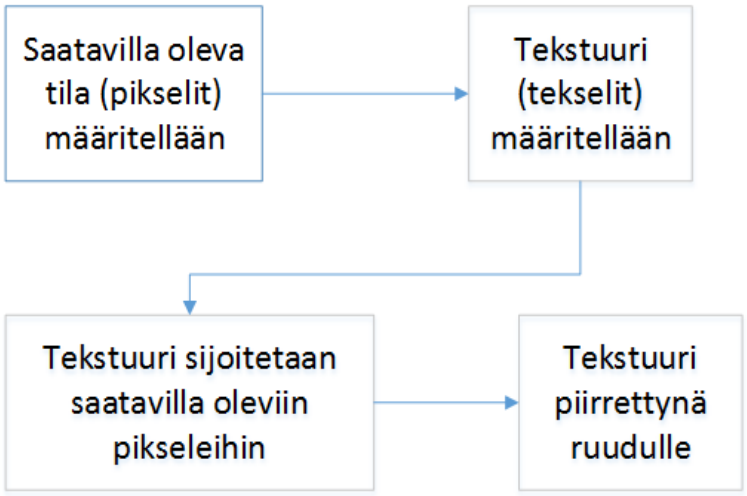
Texture2D & Texture3D

Luokan tehtävänä on esittää tekstuureja, jotka määritellään tekseleistä (texel) koostuvaan, 2- tai 3-ulotteiseen ruudukkoon (grid). Tekselistä voidaan käyttää myös nimitystä ”tekstuuri elementti” tai ”tekstuuri pikseli” ja se on tekstuurin tilan pienin yksikkö sekä perusyksikkö, jota voidaan kirjoittaa ja lukea grafiikkaprosessoriin. Luokkien jäseniin kuuluu ominaisuudet pituus (Height) ja leveys (Width), joiden avulla haluttua tekstuuria voidaan mitata pikseleissä. Monogamessa 2D-tekstuureja piirretään vektorien U ja V avulla, jotka ovat samansuuntaiset, mutta niiden koko voi vaihdella. Tämän lisäksi 3D–tekstuurit käyttävät W -vektoria. Ruudukko, johon tekstuurit piirretään voi sisältää mipmap-tasoja. Yksinkertaistettuna MipMap-tasoilla tarkoitetaan kuvan resoluution eri versioita. Ylin taso on kuvan oletus resoluutio, sitä alempi hieman pienennetty

resoluutio ja niin edelleen. Toiminnan avulla pyritään säästämään sovelluksen suorituskykyä. Kun tekstuurin siirretään kauemmaksi, valitaan sopiva mipmap-taso ja piirretään se. (Payne 2013.)

Miten tekseli eroaa pikselistä?

Pikseli on kuvan tilan pienin yksikkö ja perusyksikkö, joista sovelluksen, tietokoneen käyttöjärjestelmän tai puhelimen kuvaruutu koostuu. Pikselin tehtävänä on toistaa tekseleitä, joita kuvan renderöijä sijoittelee sopiviin paikkoihin (kuvio 5). Pikseli voi sisällyttää useamman tekselin, riippuen tekstuurin sijainnista katsojaan. Kun tekstuuri on suurennettuna katsojaan nähden, tekseli piirtoon tarvitaan useampi pikseli. Päinvastaisesti tekstuuria pienentäessä tekselit saattavat olla pikseliä pienempiä. Tällaisessa tilanteessa tekstuurin piirto voi kadota kokonaan katsojan havainnosta. (Microsoft 2018i.)



Kuvio 5. Tekstuurin piirtoprosessi (Kuvio: Joonas Niinistö)

SourceRectangle (Lähde suorakulmio)

Lähde suorakulmion tehtävänä on määrittää piirrettävän kuvan alustava koko ja sijainti, jota kohde suorakulmio käyttää pohjanaan. Yleensä lähdesuorakulmiota käytetään apuna suorakulmion sisällön animointiin. Jättäessäsi suorakulmion tyhjäksi (null), Monogame piirtää kuvasarjan kokonaisuudessaan.

DestinationRectangle (Kohde suorakulmio)

Kohde suorakulmiossa määritellään suorakulmion lopullinen koko ja sijainti, jossa se piirretään kuvaruudulle. Mikäli lähde suorakulmio on määritelty, kohde noudattaa sen sisältöä. Lopuksi piirtokomentoon saa valita värisävyn. Mikäli sävy ei ole tarpeen, väriarvo tulee syöttää valkoiseksi.

```
Color.White
[spritebatch]. DrawString ([fontti], [”teksti”], [sijaintivektorin X ja Y paikat], [väri]); // Tekstin piirtofunktio parametreineen
```

SpriteEffects

SpriteEffects luetteloinnin avulla piirrettävää tekstuuria voidaan peilata. (Oletuksena ”None” = ei peilausta).

```
SpriteEffects.None;
SpriteEffects.FlipHorizontally;
SpriteEffects.FlipVertically;
```

SpriteEffects muuttujaan ”mirror” säilytetään peilauksen nykyinen tila.

```
SpriteEffects mirror;
```

SpriteEffectsiä käytetään pelihahmon peilaamiseen pelaajaan ohjatessa oikealle.

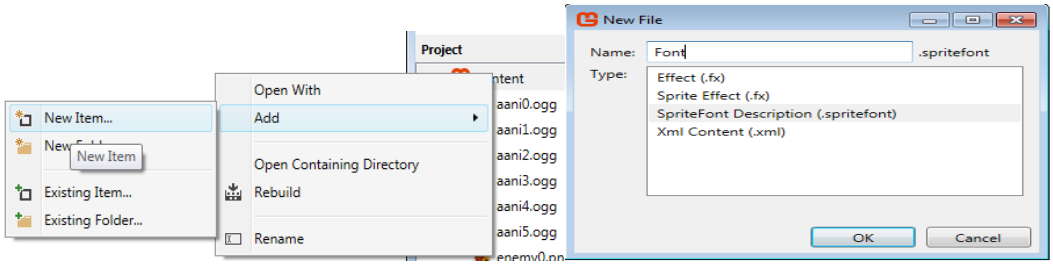
```
if (IfKeyDown(state, Keys.Right, Keys.D)) // Right
{
    PlayerSize.X += addSpeed;
    mirror = SpriteEffects.FlipHorizontally;
}
else if (IfKeyDown(state, Keys.Left, Keys.A)) // Left
{
    PlayerSize.X -= addSpeed;
    mirror = SpriteEffects.None;
}
```

Mirror-muuttuja sisällytetään pelihahmon piirtofunktioon.

```
public void Draw(SpriteBatch spriteBatch)
{
    if (invisibleCount % 2 == 0)
    {
        spriteBatch.Draw(PlayerTexture[CurrentTexture], PlayerSize,
        PlayerAnimation, Color.White, 0.0f, Vector2.Zero, mirror, 0.0f);
    }
}
```

SpriteFont

SpriteFont ei ole suoranainen fontti, vaan XML-tiedosto, joka sisältää fontin määritykseen liittyvää tietoa. Määritys tehdään Pipeline Tool-työkalun kautta seuraavasti: Content -> Add -> New Item (kuva 5).



Kuva 5. SpriteFontin määrittely (Kuva: Joonas Niinistö).

Toiminto luo sovelluksen tiedostot (Content) kansioon XML-tiedoston, johon määritellään fontin perusominaisuudet kuten tekstuurin, koon, rivivälin, asettelun sekä tyylittelyn (Regular, Bold ja Italic vaihtoehdoista). Oletuksena SpriteFont käyttää ”Arial”-fonttia.

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent
xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
 <Asset Type="Graphics:FontDescription">
    <FontName>Abergerfont</FontName>
    <Size>25</Size>
    <Spacing>0</Spacing>
    <UseKerning>true</UseKerning>
    <Style>Regular</Style>
    <!-- <DefaultCharacter>*</DefaultCharacter> -->
    <CharacterRegions>
       <CharacterRegion>
         <Start>&#32;</Start>
         <End>&#126;</End>
       </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

Mikäli haluat käyttää kustomoitua fonttia, aseta se XML-tiedoston sisältämään kansioon. Fonttia ei tarvitse ladata erikseen, vaan se otetaan käyttöön Content Managerin lukiessa XML-tiedostoon tehdyt määritykset. Määriteltyä fonttia voi nyt käyttää SpriteBatch-luokan ”DrawString()”-funktiossa.

```
spriteBatch.DrawString(Font, "Hit enter to start!", new
Vector2(CommandText.X+3, CommandText.Y+3), Color.Black);
```

Huomautus: Tarkista fontin käyttöoikeudet ennen käyttöönottoa.

4.3.4 Microsoft.XNA.Framework.Input

Syötetietoja vastaanottavia ja käsitteleviä luokkia sisältävä nimiavaruus, jonka käsittelyyn kuuluu näppäimistö, hiiri sekä peliohjain (gamepad). Peliohjaimen käyttö on rajoitettu vain Xbox360 ohjaimeen, mikäli projektia suoritetaan DirectX rajapintaan.

Taulukko 8. Monogamessa ja sovelluksessa käytettyjä näppäinfunktioita

KeyboardState	Tarkistaa näppäimistön tilan (painettu/ ei painettu)
Keys	Luettelointi, joka tunnistaa halutun näppäimen. Sen jäseniksi on pyritty keräämään mahdollisimman paljon eri näppäimiä, kuten japanilaisesta näppäimistöstä Kana ja Kanji-merkit. Monogamen dokumentaation mukaan skandinaaviset kirjaimet eivät ole kuitenkaan sisällytetty luettelointiin.
KeyState	Tarkistaa, onko näppäin painettu pohjaan vai ei. Luetteloinnin jäsenet ovat ”KeyState.Up” ja ”KeyState.Down”.

Hiiren ja peliohjaimen käyttäminen toimii samanlaisia toimintoja käyttäen. Peliohjaimen luettelointi (buttons) sisältää listan painikkeista, joita nykyaikaiset ohjaimet käyttävät.

Näppäinten ja ohjaimien toiminta

Syöteitä käsitellään kahdella tapaa, digitaalisesti ja analogisesti. Digitaalinen syöte sisältää joko tosi tai epätosi tiedon näppäinsyötteestä, esimerkiksi painaessasi näppäimistöstä tai ohjaimesta haluamaasi painiketta.

Analogisella syötteellä on tietty arvoalue, josta palauttaa tietoa, esimerkiksi peliohjaimen ”tattiohjain” tai hiiren liike. Näitä syötteitä voidaan palauttaa eri tavoin, esimerkiksi desimaaliluvulla tai pikseleinä sovelluksen kuvaruudusta. Syötetieto vastaanotetaan kyselyn (polling) avulla, joka toteutetaan pelin kuvan (frame) päivittämisen yhteydessä. Päivittäessä funktio tarkistaa halutun syöttölaitteen tilan (state), palauttaa sen ja vertaa saatua tilaa aiempaan.

Vertailun lopuksi näppäinsyöte päivitetään, mikäli muuttaminen on tarpeen. Peliohjaimien lukuun tarvitaan välikerroksia, jotka lukevat ja kääntävät ohjaimia käsittelevän rajapinnan C#-kielelle sopivaan muotoon. Syötetietojen lukuun Monogame käyttää apunaan välikerroksia (taulukko 9), jotka sisältävät pieniä eroavaisuuksia.

Taulukko 9: Välikerrokset ja peliohjaimet

SharpDX/ Xinput (DirectX, UWP)	XInput on rajapinta Microsoftin Xbox 360-pelikonsolin ohjaimen määrittelyyn peliprojektiin ja on rajoitettu vain Xbox360:n käytössä oleviin ohjaimiin, joita yhtaikaa voi olla neljä kappaletta. Rajapintaan ei sisälly näppäimistö tai hiiri funktionaalisuutta. (Microsoft 2018c.)
SDL (OpenGL)	Monogamen OpenGL-ympäristö käyttää SDL-kirjastoa, joka mahdollistaa matalan tason käsittelyn alustan eri toimenpiteisiin, kuten myös usb-peliohjaimiin. Koska kyseessä on avoimempi välikerros toteutus, se sallii muiden USB-ohjainten käyttämisen. Xinputin tavoin se on kuitenkin rajoitettu neljään ohjaimeen. (Monogamen Team 2019b.)

Xinputin edeltäjää nimeltä ”DirectInput” on myös mahdollista käyttää, mutta tätä ei suositella sen lakkautetun kehityksen vuoksi.

4.3.5 Microsoft.XNA.Framework.Audio

Nimiavaruus ääniefektien toistoon ja hallintaan. Monogamessa yleisimmät audio formaatit, kuten Windows Media Audio (wma), MP3, WAVE ja Vorbis (ogg) ovat tuettuja. Formaattien käytössä kannattaa miettiä, mille alustalle sovellusta ollaan kehittämässä, sillä eri alustoilla on tietyt vaatimukset ja tukensa eri formaateille. Esimerkiksi iOS järjestelmät eivät tue Windows Media Audiota tai Vorbista (Apple Inc 2017). Äänentoistossa käyttäjä voi käyttää XNA:n ”Cue”-luokkaa tai Monogamessa paranneltua versiota ”SoundEffect”-luokkaa. Niiden avulla

kehittäjä määrittää eri toimintoja, kuten halutun ääniefektin nykyisen tilan tarkistus tai määrittää missä kohti ääniefektiä soitetaan. SoundEffect luokkaa käyttävät tiedostot ladataan suoraan sovelluksen muistiin. (KojanuGames 2017.)

Parade käyttää SoundEffect-luokkaa ääniefektien toistamiseen. Alun perin luokkaa käytettiin myös taustamusiikin soittamisessa, mutta tähän liittyi omat ongelmansa. Kääntäjän luodessa suoritettavaa sovellusta taustamusiikin tiedostokoko (.xnb) kasvoi moninkertaisesti alkuperäisen tiedoston (.ogg) kokoon nähden, suuremmaksi kuin muut sovelluksen tiedostot yhteensä. Tästä syystä taustamusiikki määriteltiin ”Media”-nimiavaruuden avulla.

Ääniefektin toistoon riittää ”Play()”-funktio, joka toistaa äänen kerran. Toistoon liittyviä asetuksia voi kuitenkin vaihtaa luomalla erillisen instanssin. Kuten peliohjaimen kanssa, Monogame projekti käyttää alustan mukaan eri välikerroksia äänen toistamiseen ja hallintaan (taulukko 10).

Taulukko 10. Välikerrokset audion toistossa.

SharpDX / XAudio2	XAudio2 on matalan tason äänentoisto rajapinta, joka on tarkoitettu pelien audio puolen kehittämiseen. Ensisijaisesti se vastaa sovelluksen äänentoistosta. Se sisältää myös toimintoja, joilla kehittää ja tehostaa äänentoistoa, kuten etäisyyden määrittely DSP-tehosteilla. (Microsoft 2018a.)
OpenAL (libopenal)	Audiokirjasto OpenGL rajapintaan. XAudion tavoin se sisältää samanlaisia toimintoja 3D-ympäristössä, kuten äänentoisto tietyllä voimakkuudella riippuen pelaajan ja ääntä pitävän objektin etäisyydestä.

4.3.6 Microsoft.Xna.Framework.Media

Nimiavaruus käyttää Media Foundation nimistä multimedia-alustaa, jonka toteuttamisessa on pyritty huomioimaan niin audion kuin videon käsittelyyn ja toistoon liittyviä toimintoja. (Microsoft 2018b). Toisin kuin audio-nimiavaruus, media on tarkoitettu laajempaa käyttöön, kuten soittolistojen määrittämiseen

erilaisten luokkien avulla (Album, Artist, Genre). Nimiavaruuden kautta pystyy myös määrittelemään videon toistoon liittyviä toimintoja, mutta tämä on rajattu vain DirectX ympäristöön (”VideoPlayer”-luokka). (Monogame Team 2019i.)

Parade käyttää nimiavaruutta taustamusiikin soittamiseen ja hallintaan. Musiikin toisto tapahtuu ”Song”-luokan kautta, johon voi asettaa kappaleen nimen, artistin ja albumitiedot. Halutun kappaleen toistamiseen hyödynnetään erillistä ”MediaPlayer”–luokkaa (taulukko 11).

Taulukko 11. MediaPlayer luokan funktionaalisuutta.

`MediaPlayer.Play`
`MediaPlayer.Pause`
`MediaPlayer.Stop`
`MediaPlayer.Resume`
	Median toistaminen

`MediaPlayer.Volume`	Median äänenvoimakkuus
`MediaPlayer.IsRepeating`	Toistetaanko media uudestaan päätöksen jälkeen (kyllä =tosi).

Cross Platform Audio Creation Tool (XACT)

Audio-nimiavaruus tukee myös XACT-työkalun käyttöä, jota XNA alun perin käytti audion toistoon ja hallintaan. Toteutuksen ideana on erottaa projektin audio vastaavan ja ohjelmoijan tehtävät toisistaan. Työkalu sisältää erilaisia ominaisuuksia kuten 3D–äänen toisto esimerkiksi ääni on kovempi lähellä ja hiljaisempi kaukana. (Whitaker 2019.) Työkalulle ei kuitenkaan ole enää aktiivista tukea. Nykyisin audiokirjastoja/työkaluja, kuten FMOD suositellaan käyttämään, mikäli vastaavanlaista työkalua haluaa käyttää Monogamessa.

MP3–formaatin patentti

Monogamen tukemista audio-formaateista MP3 on tarvinnut lisenssin oston, mikäli sovelluksella on yli 5000 käyttäjää sen kaupallisuudesta huolimatta (Gullen 2011). 23.4.2017 tilanne muuttui formaatin kehityksen rahoittajan, Fraunhofer instituution ilmoittaessa lopettavansa MP3-formaatin lisensoinnin. Syynä tähän kehittyneempien formaattien olemassaolo, jolloin MP3:n kehitys ei ollut enää kannattavaa. (Fraunhofer IIS 2017.) Tämän myötä MP3-käyttö ja kehittäminen ovat muuttuneet vapaiksi.

4.4 DotNET – Monogamen ydin

Monogamen toiminnan pohjalla on Microsoftin kehittämä ohjelmistokomponenttikirjasto, dotNET, johon sisältyy runsaasti erilaisia luokkakirjastoja, mukaan lukien XNA. Tästä syystä kaikki Monogame sovellukset ovat riippuvaisia dotNETin käyttämisestä. Vaikka Monogame suosii pääasiallisesti C#-ohjelmointikieltä, DotNET tukee myös kieliä kuten ”Visual Basic”, ”Visual C++”, ja ”F#”. Kieliä voi käyttää myös ristiin, mikä vaatii kääntäjältä kykyä tehdä niistä yhteensopivia aiheuttamatta virhetilanteita. DotNET:n arkkitehtuuri onkin kehitetty vastaamaan näitä vaatimuksia.

4.4.1 Esikäsittely - Common Intermediate Language (CIL)

Kun projektin sovelluskoodi halutaan suorittaa, se tulee kääntää tietokoneen ymmärtämään muotoon. DotNET ympäristössä ohjelmakoodi välitetään kääntäjään (compiler), joka taas välittää sen ”esikäsittelyyn” (Common Intermediate Language, CIL). Esikäsittelyn aikana koodi muunnetaan luettavaan muotoon ”ajonaikaiseen ympäristöön” (CLR), jossa sovelluskoodia käsitellään tarkemmin. Esikäsittely tunnetaan myös nimillä “Microsoft Intermediate Language” (MSIL) tai “Intermediate Language” (IL). (Scriptol 2014).

4.4.2 Ajonaikainen ympäristö - Common Language Runtime (CLR)

Esikäsittelyn jälkeen ohjelmakoodi viedään ajonaikaiseen ympäristöön (CLR), jossa koodista pyritään poistamaan dotNET ohjelmointikielien välisiä ristiriitoja, mikäli sovellus käyttää enemmän kuin yhtä ohjelmointikieltä. Tämän jälkeen koodi käännetään järjestelmän ymmärtämään binäärimuotoon. Kääntämiseen on kaksi menetelmää; ”Hallittu koodi” (Managed Code), jonka hallintaominaisuuksia

ovat mm. koodin turvallisuutta tarkistava ”Expection handling”, käyttöoikeudet ja rajoitteet määrittelevä ”Code Access Security” sekä Automaattinen muistinhallinta tai ”roskien kerääjä” (garbage collector), joka estää RAM-muistin ylivuotoa. (Microsoft 2018g.) ”Hallitsematon koodi” (Unmanaged Code) on dotNET ympäristön ulkopuolella toteutettua ohjelmistokoodia. Käyttöjärjestelmästä riippumatta Monogame käyttää aina dotNET ympäristöä, näin ollen käsitellen ohjelmistokoodin aina hallitusti.

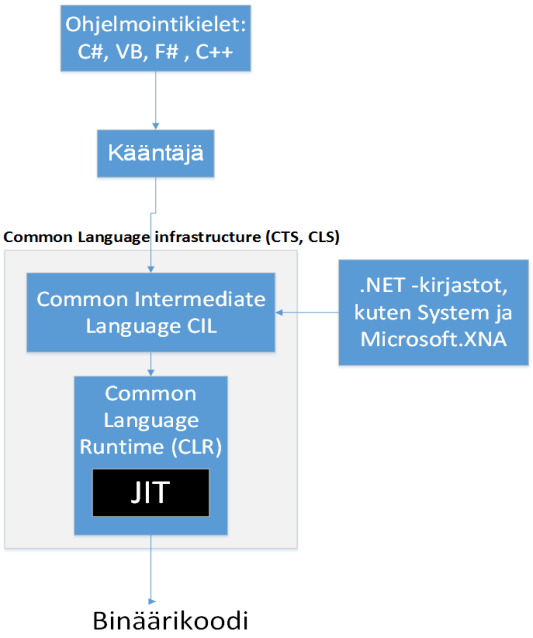
4.4.3 Common Type System (CTS) & Language Specification (CLS)

Common Type System on dotNET ympäristön standardi, joka määrittelee, kuinka tietyt muuttujatyypit määritellään tietokoneen muistiin. Määrittelyyn lukeutuu myös eri dotNET yhteensopivien ohjelmointikielinen käyttämien tyyppien ja muuttujien yhteensopivuuden säilyttäminen. CTS:ssä muuttujat on jaettu kahteen erilaiseen tyyppi kategoriaan; ”Value Types” (arvotyypit), johon kuuluu yleisiä muuttujia, kuten ”int”, ”float” tai ”bool” (TutorialPoint 2019). ”Reference Types” (viitetyypit) ovat tiettyyn muistipaikkaan viittaavaa tietoa (Microsoft 2019a). Viitetyyppien määrää on laajennettu C#-kielessä merkkijono(string), objekti ja dynaamisilla-tyypeillä (Microsoft 2016b). Common Language Specification on CTS:n osajoukko, joka noudattaa standardin mukaisia sääntöjä. Spesifikaatioon voi kuitenkin tehdä omia määrityksiään tai kiristää CTS:än sääntöjä. Sääntöjen tavoitteena on kuitenkin sallia dotNET kielien välinen yhteensopivuus sekä integrointi. (Microsoft 2016b.)

4.4.4 Kääntäjä - Just In Time (JIT)

Termillä ”Just in time” tarkoitetaan myös teollisuudessa menetelmää, jonka avulla vähennetään ja vältetään tarpeettomia kustannuksia, resursseja ja aikaa. (Banton 2019). Samannimisen kääntäjän tehtävänä on muuttaa esikäsitelty koodi alustan ymmärtämään binäärimuotoon käyttämällä termin periaatteita (kuvio 6).

Kääntäjästä on kolme versiota; Pre-JIT, Normal JIT sekä Econo JIT (Todorov 2013).



Kuvio 6. Yhteenveto dotNET:n toimintaperiaatteesta (Kuvio: Joonas Niinistö).

5 Alustojen ja rajapintojen vertailua

Nyt kun tyypillisen Monogame projektin rakenne on käyty pääpiirteittään läpi, on aika kokeilla kuinka hyvin dotNET pohjainen sovellus kääntyy erilaisille alustoille. DirectX ympäristön lisäksi minulla ei ole paljoa aiempaa kokemusta muista kokeiluun otetuista alustoista, joten dokumentointini perustuu pääasiallisesti kohtaamieni esteiden ja eroavaisuuksien tulkitsemisesta. Pyrin myös analysoimaan ja kertomaan kustakin alustasta oleellisimmat erot.

5.1 DirectX ja Universal Windows Platform (UWP)

DirectX on Microsoftin kehittämä multimedian toteuttamiseen ja pelikehitykseen tarkoitettu ohjelmistorajapinta kokoelma. Se sisältää erilaisia komponentteja ja

työkaluja useisiin kehitystarkoituksiin, kuten 3D-grafiikan, äänen toistamiseen, verkkoyhteyden luomiseen ja ohjaamaan matalan tason toimintoja käyttäjän puolesta, kuten laitteiston grafiikkaresurssien lukuun ja käyttöön. Ollessaan Microsoftin kehitysalusta, sitä ei voi käyttää muissa kuin Microsoftin käyttöjärjestelmissä sekä Xbox-pelikonsolissa. (Tiedettävästi sitä voi käyttää ”Wine”-ohjelmiston alaisuudessa, joka suorittaa Windows pohjaisia sovelluksia muissa käyttöjärjestelmissä vaihtelevalla menestyksellä). Nykyisimmät versiot ovat DirectX 11 ja 12. Jälkimmäinen julkistettiin ensimmäisen kerran 20.3.2014 (Lowe 2016) ja se on saatavilla vain Windows 10 käyttöjärjestelmille (Tamminen 2018).

Universal Windows Platform (UWP) on vain Windows 10-pohjaisille laitteille tarkoitettu kehitysalusta, joka kattaa Microsoftin nykyiset (2019) laitteistot, kuten XboxOne, Windows Phone ja HoloLens. Tuki löytyy myös IOT-laitteille, kuten Raspberry Pi:lle, johon käyttäjä voi asentaa Microsoftin Windows 10 käyttöjärjestelmästä toteutetun IOT-version. (Microsoft 2018e.) UWP sovellusten toteuttamiseen voi käyttää Microsoftin kehittämiä rajapintoja, kuten DirectX 12, dotNET ja siihen perustuvia ohjelmointikieliä, kuten C# ja Visual Basic (Microsoft 2018d). Kehittäessään sovellusta UWP:lla, käyttäjällä on mahdollisuus testata sitä Microsoftin laitteilla ilman virallista hyväksyntää. Tämä kuitenkin rajoittaa laitteen tehojen käyttöä verrattuna ”natiivisti” kehitettyihin sovelluksiin. Monogame tukee molempia kehitysmuotoja. (Acevedo 2016b.) XboxOne konsolilla testaaminen vaatii kuitenkin aktivoinnin konsolin kehittäjä tilaan (Developer mode). Microsoft kumppaneille (Managed partners) on tarjoilla erillinen kehityslaitteisto, ”Xbox Development Kit” (XDK), joka mahdollistaa sovelluksen laajemman testaamisen, verrattuna kehittäjätilaan. Kehityslaitteistolla kehitettyjä sovelluksia ei voi testata erikseen tavallisella XboxOne konsolilla. (Microsoft 2018f.)

5.1.1 UWP toteutus

Toteutuksessa tutustutaan kuinka hyvin Parade–peli kääntyy Windows 10 käyttöjärjestelmällä, käyttäen Visual Studio 2019 IDE:ä.

Monogamen tarjoamat sovelluspohjat

DirectX–sovellus toteutettiin käyttämällä Visual Studio 2015 IDE:ä, johon Monogame pystyi asentamaan suoraan eri alustoille tarkoitettuja sovelluspohjia. VS 2019 versiossa tätä tukea kirjoitushetkellä ollut, joten sovelluspohjat tuli hakea erikseen Monogamen Github sivuilta ja sijoittamaan oikeaan polkuun.

```
C:\Users\KayttajaWin10\Documents\Visual Studio
2019\Templates\ProjectTemplates
```

Lisäksi Universal Windows Platformin käyttäminen edellyttää ”Windows 10 SDK”-asentamisen, Visual Studio 2019 asennussovelluksesta.

Ongelmatilanteita

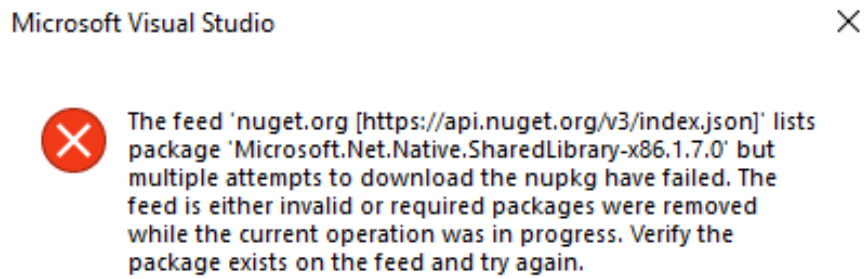
Visual Studio (VS) 2019 ympäristön määrittelyyn liittyi muutamia ongelmia, pääasiallisesti Monogamen yhteensopivuuteen. Ensimmäinen kohtaamani ongelma tuli Monogamen tekstuureja käsittelevän "TextureImporter”-komponentin kanssa, joka ei suostunut kääntämään sovellukseen kuuluvia kuvatiedostoja. Tämän korjaamiseksi riitti, kun asensi Microsoft Visual C++ 2013 Redistributable (x64)-12.0.40664-paketin, joka tuli hakea erikseen Microsoftin sivuilta. Sitä pystynyt asentamaan Visual Studion asennussovelluksella (ainakaan community 2019 versioon). Seuraavaksi Visual Studio väitti, ettei se pysty löytämään "MonoGame.Framework.xr"-tiedostoa, joka oli kylläkin olemassa ja sijaitsi seuraavassa polussa:

```
C:\Users\<kayttaja>\.nuget\packages\monogame.framework.windowsuap\3.4.0.459
\lib\netcore
```

Ongelman ratkaisemiseksi kopioin tiedoston seuraavaan alikansioon:

```
C:\Users\<kayttaja>\.nuget\packages\monogame.framework.windowsuap\3.4.0.459
\lib\netcore\MonoGame.Framework
```

Sitten Visual Studio ilmoitti, ettei pakentinhallintaongelma Nuget kyennyt löytämään ja lataamaan tarvittavaa pakettia (kuva 6), jonka seurauksena kääntäjä ei pystynyt suorittamaan sovellusta.



Kuva 6. Nuget ongelma (Kuva: Joonas Niinistö).

Pienen haun jälkeen ratkaisuksi löytyi asentaa Universal Windows alusta uudelleen. Tein tämän käyttämällä Nugetin komentoriviä:

```
Install-Package Microsoft.NETCore.UniversalWindowsPlatform -Version 6.2.9
```

Komentorivin saa aktivoitua valikosta seuraavaa polkua pitkin:

```
Tools > Nuget Package Manager > Package Manager Console
```

On kuitenkin huomioitavaa, että ongelma korjaantui vain käsittelyssä olleeseen projektiin ja uuden projektin luonnissa ongelma uusiutui. Lopuksi VS ilmoitti, ettei projektille luotua sertifikaattiavainta pystytty avaamaan, jonka korjaamiseen riitti poistaa kyseinen, automaattisesti määritelty avain. Sertifikaatti ongelmaan ja poistamiseen liittyi todennäköisesti Microsoftin tekemä määritellä, jossa projektin yhteydessä määritellään sertifikaatti sekä erillinen "sormenjälki"-tunniste. Ilmeisesti projektistani puuttui kyseinen tunniste, mahdollisesti seurauksena aiemmista ongelmista. Sertifikaatin poistaminen asettaa projektin käyttämään Microsoftin aiempaa määritystä, jolloin virheen pystyi ohittamaan. (Microsoft 2019e.)

Sovellus UWP -ympäristössä

Yleensä Monogamella on tarjoilla yksi sovelluspohja kutakin alustaa kohden, mutta UWP–sovelluksille niitä on kolme (taulukko 12), joista valita sopivin Parade-pelille.

Taulukko 12. UWP-pohjan eri vaihtoehdot.

Universal Project	Tavallinen pohja, joka käyttää Visual Basic ohjelmointikieltä.
Universal (XAML)	Tavallinen pohja, joka käyttää C#-ohjelmointikieltä sekä XAML:ia. XAML on XML–kieleen perustuva ”deklaratiivinen” merkintäkieli ja on tarkoitettu pääasiallisesti erilaisten käyttöliittymien suunnitteluun ja toteuttamiseen. Syntaksiltaan se muistuttaa paljon tyypillistä XML tai HTML-syntaksia.
Universal (Core Application) Project	Käyttää apunaan ”*Core Application*”-ohjelmistokehystä. Ohjelmointikielenä on C#. XAML on poistettu käytöstä. Mikäli ”Core Application”-sovelluspohja on asetettuna muiden pohjien kanssa samaan kansioon, se yli kirjoittaa XAML-pohjan estäen sen valitsemisen Visual Studion valikosta. Mikäli XAML-pohjaa haluaa käyttää, tulee ”Core Application”-pohja siirtää pois kansiosta.

Koska sovellus on kirjoitettu C#-kielellä eikä tarvitse XAML:n tuomia etuja käyttöliittymän toteuttamiseen, valitsin ”Core Application” pohjan.

Seuraavaksi projektille sai päättää nimen ja tallennussijainnin. Jätin tallennuspaikan oletukseen, mutta projektin nimeksi vaihdoin DirectX versiossa käyttämääni nimeä ”ParaatiOpen”. Tämän avulla uuteen projektiin ei tarvinnut tehdä muuta kuin kopioida DirectX-version ohjelmakoodi, joka toimi yksi yhteen UWP-ympäristössä, pois lukien XML-tiedoston muokkaus ja tallennus funktionaalisuudet. XML-lukua on muutettu ympäristöä varten, eikä käyttäjä voi käyttää aiemmin käytettyä dotNET menetelmää. Tämän korjaamiseksi tarvitaan erillisiä nimiavaruuksia. Myös XML-muuttuja muutetaan Xdocumentiksi.

```
using System.Xml.Linq;
using System.IO;
using Windows.Storage;
```

```
XDocument SaveFile = new XDocument();
```

Pisteiden lataus sovelluksen alussa tapahtuu ilman suurempia muutoksia.

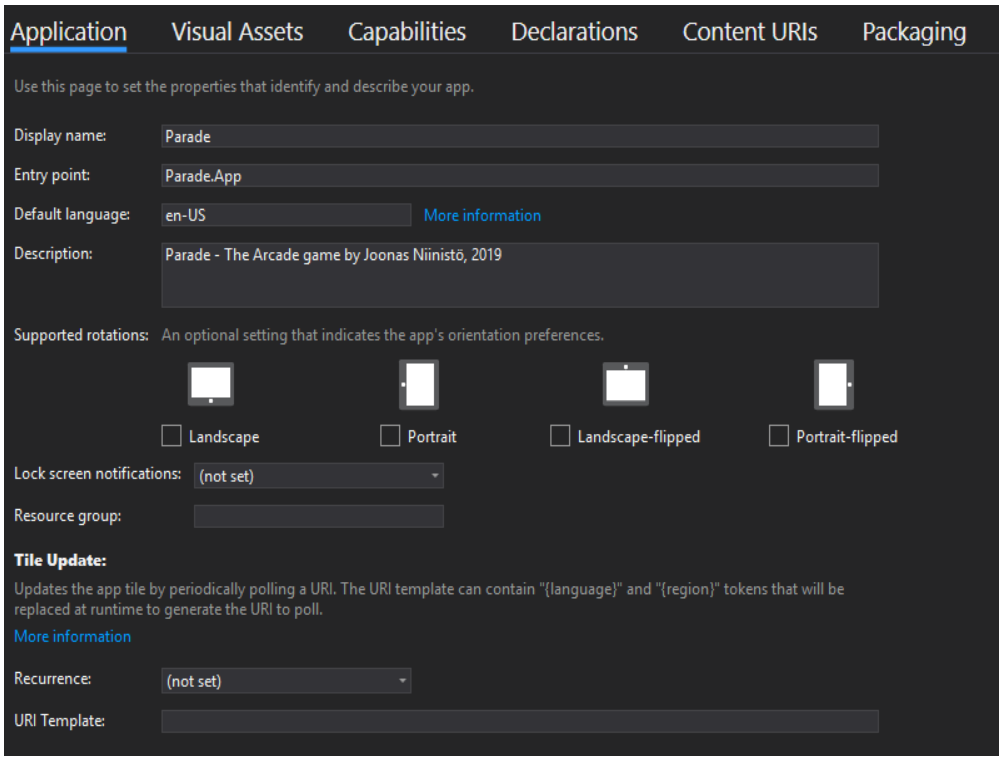
```
if (File.Exists("SharpDXV.xml"))
{
    XDocument SaveFile = XDocument.Load("SharpDXV.xml");
    Score = Convert.ToInt32(SaveFile.Root.Element("Nykyinen").Value);
    HiScore = Convert.ToInt32(SaveFile.Root.Element("Huipputulos").Value);
}
```

Sen sijaan tiedoston muokkaaminen, tallentaminen ja luominen eivät puolestaan suostuneet toimimaan oikeastaan mitenkään. Käytössäni oli nimiavaruuden ”Windows.Storage” toiminnallisuuksia, joiden pitäisi auttaa tiedoston tallentamisessa. Nykyisessä muodossaan koodi antaa virheilmoituksen, joka väittää XML tiedoston olevan virheellinen puuttuvan ”juuri elementin” takia. Tämä tarkoittaa sitä, että XML-tiedoston rakenne olisi virheellinen. Rakenne oli kuitenkin pysynyt täysin samana kuin ennen, enkä löytänyt ohjeistusta, joka kehottaisi määrittelemään erillisiä muutoksia UWP–ympäristöön. Olenko jättänyt jotain huomioimatta, vai onko tämä seurausta aiemmista virhetilanteista? Todennäköisesti molempia.

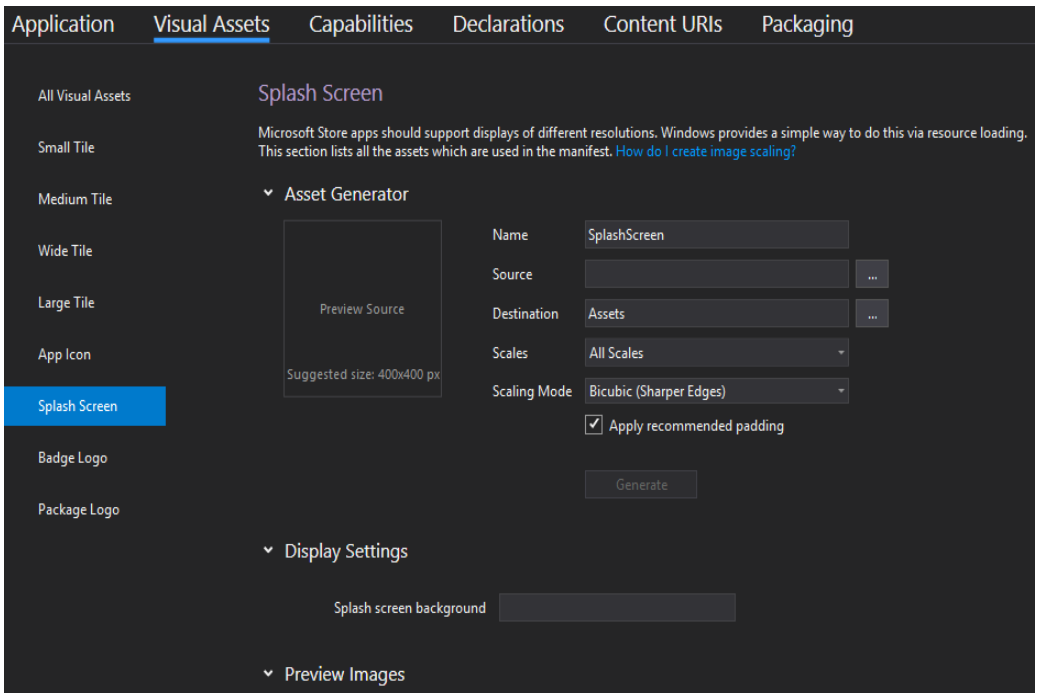
```
if (File.Exists("SharpDXV.xml")) // If file "SharpDXV.xml" is found, modify it
{
   StorageFolder storageFolder =
Windows.Storage.ApplicationData.Current.LocalFolder;
   StorageFile file = await storageFolder.CreateFileAsync("SharpDXV.xml",
CreationCollisionOption.ReplaceExisting);

   using (Stream fileStream = await file.OpenStreamForWriteAsync())
   {
        SaveFile = XDocument.Load(fileStream);
        SaveFile.Root.Element("Nykyinen").Value = Score.ToString();
        SaveFile.Root.Element("Huipputulos").Value = HiScore.ToString();
        SaveFile.Save(fileStream);
        fileStream.Flush();
   }
}
```

Visual Studio kuitenkin tarjoaa erilaisia, mobiililaitteille ominaisia asetuksia, joita käyttäjä voi määritellä haluamallaan tavalla. Esimerkiksi projektiin lisätty kansio nimeltä ”assets”. Se sisältää eri kokoisia ”mallikuvia”, joita käytetään sovelluksessa eri yhteyksissä, kuten sovelluksen alussa, Microsoftin kauppapaikalla tai mobiilialustalla logona. Kuva, joka näytetään ennen varsinaisen sovelluksen alkua, on nimeltään ”SplashScreen.scale-200.png”. Käyttäjä voi muokata kuvaa, joko käyttämällä erillistä kuvankäsittely ohjelmaa, tai Visual Studiossa muokkaamalla ”Package.appxManifest” tiedostoa (kuva 7). Kyseiseen tiedostoon voi tehdä muitakin UWP-sovellukseen ominaisia toimintoja,

kuten määritellä hyödyntääkö sovellus mobiililaitteen ”automaattinen käännös”-ominaisuutta (kuva 8).



Kuva 7. Package.appxManifest–ikkuna, Application (Kuva: Joonas Niinistö).



Kuva 8. Package.appxManifest–ikkuna, Visual Assets (Kuva: Joonas Niinistö).

5.1.2 Yhteenveto UWP -alustasta

Kokeilu antoi ensivaikutelmaa niin Universal Windows Platformista ja siitä, millaisiin ongelmiin ja ristiriitoihin kannattaa seuraavissa kokeiluissa varautua, niin pelin ohjelmakoodin kuin käytettävien sovellusten ja laitteiston osalta.

DirectX ympäristöön nähden UWP kuitenkin toimi hyvin samoilla periaatteilla. Molemmat käyttivät SharpDX-välikerrosta ja sen kautta luettavia rajapintoja, kuten XAudio2 tai XInput eikä ohjelmakoodiin tarvinnut tehdä suuremmin muutoksia. Lisäksi kaikki tiedostot tallennetaan yhä oletuksena xnb-muodossa.

UWP-alusta itsessään tuntui olevan hyvin mobiilikeskeinen, tarjoten erilaisia vaihtoehtoja Windows Phone sovelluksen toteuttamiseen. Tämä saattoi olla tietoinen valinta, ottaen huomioon todennäköisimmät alustat joihin kehittäjät tulevat sovelluksia kehittämään. Laitteistojen puutteen vuoksi en päässyt kuitenkaan kokeilemaan sovelluksen suorittamista Windows Phone mobiililaitteella.

5.1.3 Microsoft Store ja kritiikki

UWP-sovelluksien käyttöön liittyy muutamia rajoitteita, joista suurin vaikuttava tekijä on Microsoft Store, johon UWP sovellusten myynti ja jako on rajoitettu. Malliesimerkkinä Square Enixin julkaisema “Rise of the Tomb Raider” peli, jonka Windows Store julkaisussa asetuksia, kuten kuvaruudun päivitys (v-sync) ei pystynyt poistamaan käytöstä. Peli oli myös lukittu ”koko näytön ikkunatilaan” (Always Borderless Fullscreen Mode), joka rajoittaa laitteiston graafisen suorituskyvyn käyttöä. Verrattaessa pelin Steam julkaisuun näitä rajoitteita ei ollut havaittavissa. (Hoffman 2016.)

Esimerkkiin vaikuttaa kuitenkin muutama tekijä, kuten kuinka paljon Microsoft Store julkaisuun kehittäjät ovat käyttäneet resursseja verrattaessa Steam julkaisuun. Pelaajat ovat kokemustensa perusteella verranneet Microsoft Store PC versiota pelin XboxONE versioon (Steam Community 2017). Alun perin “Rise of the Tomb Raider” julkaistiin XboxONE- ja 360-konsoleille, joista se

myöhemmin käännettiin ja julkaistiin PC-alustalle. Tämä taas saattaa viitata siihen, että Windows Store versio on vain pieni muunnelma UWP-pohjaisesta XboxONE versiosta, jolloin kohdealustan (PC) täydellistä potentiaalia ei hyödynnetty oikein. Sen sijaan aiemmin mainittu kuvaruudun päivitys (v-sync) asetus ja hallinnan puute on johtunut UWP-alustan rajoitteesta (Nixxes, edustushenkilö Steam foorumeilla 2016). UWP pohjaisten sovellusten käyttöä on muutenkin pidettynä rajoitetumpana. Käyttäjät eivät ole voineet käyttää kolmannen osapuolen sovelluksia, kuten hiiri ohjattavuuteen vaikuttavia makroja, videokuvaa kaappaavia tai suorituskykyä ylläpitäviä ”sovelluspäällyksiä” (overlay). Kehitysalustan luonteen vuoksi sovelluksia voi käyttää vain ja ainoastaan Windows 10 pohjaisissa järjestelmissä.

Rajoitteet ovat poikineet paljon kritiikkiä, kuten Epic Gamesin toimitusjohtajalta Tim Sweeneyniltä. Hänen mukaansa Microsoft pyrkii ajamaan PC:tä alustana Microsoftin hallinnassa olevaa suljetuksi ympäristöksi, rajoittaen aiemmin avoimena tunnettua ympäristöä, jossa kehittäjillä on vapaus jakaa ja myydä sovelluksiaan haluamillaan vaihtoehdoilla. (Walton 2016.) Epic Games on kuitenkin harjoittanut osittain samanlaista rajoittamista tietyissä tilanteissa, kuten kieltäytymällä julkaisemasta Unfold Gamesin toteuttaa indie peliä ”DARQ” Epic Game Storessa pelin Steam ja Good Old Games julkaisujen ohella (Unfold Games 2019). Kritiikin myötä UWP ympäristöstä on luvattu tehdä avoimempaa (Spencer 2016). Nykyisin alustan tilanne ja tulevaisuus vaikuttavat hieman epävakailta, johtuen Microsoftin laitteistojen, kuten Windows Phonen huonon menestyksen ja sovellustuen puutteen vuoksi. Jälkimmäiseen on vaikuttanut kehittäjille vakiintunut, Win32-kehitysrajapinta, joka on paljon vapaampi ja monimuotoisempi tapa kehittää sovelluksia. Sittemmin Win32-pohjaisille sovelluksille on annettu julkaisumahdollisuus myös Microsoft Storeen (Spencer 2019).

5.2 OpenGL ja Mono -projekti

OpenGL (Open Graphics Library) on Silicon Graphicsin määrittelemä avoin ”spesifikaatio”, joka sai alkunsa 90-luvun alussa. Tavoitteena oli toteuttaa

spesifikaatiosta mahdollisimman ”siirrettävä” (portability) ja yhteensopiva (compatible), jolloin uuden laitteiston ilmestyessä sitä pystyi helposti muokkaamaan (Graham 2016).

Spesifikaation voi ladata kuka tahansa ilmaiseksi ja kehittää sen pohjalta oman OpenGL versionsa. Tästä syystä OpenGL:stä itsestään ei ole olemassa lähdekoodia. (Segal & Akeley 2004.) Se tehtävänä on ainoastaan renderöidä grafiikkaa ja se sisältää useita satoja funktioita ja menettelytapoja, joita kehittäjät voivat hyödyntää sovelluskehityksessä. Se on laitteisto- ja alustariippumaton, koska sen määrittelemät renderöintirutiinit pysyvät samana alustasta riippumatta.

OpenGL on jaettu kahteen erilaiseen profiiliin. Modernimpi ”core” profiili sisältää vain oleellisimmat ja tuoreimmat spesifikaatioon tehdyt määritykset. ”Compatibility” profiili puolestaan sisältää yhteensopivuuden OpenGL:n kaikkiin aiempiin versioihin ja niiden ominaisuuksiin (legacy feature). Profiilien väliltä ”core” profiilin käyttö on suositeltavaa, sillä viimeisimmät määritykset takaavat parhaimman suorituskyvyn, sekä alhaisemman riskin yhteensopivuusongelmille. OpenGL:n suurimpiin vahvuuksiin lukeutuu jatkokehitys, jossa core profiiliin tehdään erilaisia lisäyksiä ja muutoksia (Graham 2016).

Mono on Xamarinin (yritys) kehittämä, avoimen lähdekoodin toteutus Microsoftin dotNET-rajapinnasta, jonka toteutuksessa on nojauduttu ECMA standarteihin, sekä rajapinnan hyödyntämään ajonaikaiseen ympäristöön, Common Language Runtimeen. Projektin tarkoituksena on ollut mahdollistaa dotNET ympäristön käyttö muilla kuin Windows alustoilla. Sen kehitysympäristöön kuuluu C#-kääntäjä ja dotNET kirjaston ohella oma Mono-kirjasto, joka tulee useita, erityisesti Linux yhteensopivia kirjastoja, kuten OpenGL. Mono tukee myös useita ohjelmointikieliä kuten ”Java”, ”Python” tai ”PHP” ja niitä voi käyttää keskenään ajonaikaisen ympäristön (CLR) kautta. (Mono-project 2018.) Monoa voi käyttää kaikissa yleisimmissä käyttöjärjestelmissä, mutta Monogamessa sitä hyödynnetään ensisijaisesti Linux- ja Mac-ympäristöissä. Monogame projektissa Monon ja OpenGL:n välinen toiminta ja yhteensopivuus saavutetaan ”Simple DirectMedia Layer” (SDL)-välikerroksen avulla. SDL käyttää omia versioitaan Core ja Compatibility profiileista. (Simple DirectMedia Layer Wiki 2019.)

5.2.1 OpenGL toteutus Linux ympäristössä

Kuten UWP-alustalla, minulla ei ole ollut aiempaa kokemusta sovelluskehittämisestä Linux pohjaisella käyttöjärjestelmällä, pois lukien muutamia C#-kokeiluja ja mitä aiheeseen on tullut muuten tutustuttua. Valitsin käyttööni Linux Mint käyttöjärjestelmän, koska siitä minulla oli aiempaa kokemusta. Ohjelmakoodin käsittelyyn käytän ”Monodevelop” nimistä IDE:ä. Visual Studio Code olisi ollut saatavilla Linuxille, mutta päätin käyttää jotain muuta vaihtelun vuoksi. Lisäksi uskoisin Monodevelopin olevan Linux ympäristössä suositumpi vaihtoehto verrattaessa Microsoft-palveluihin.

Määrittely

Monodevelop on riippuvainen Mono-projektista, jonka asentaminen tapahtuu nopeinten käyttämällä käyttöjärjestelmän tarjoamaa komentopäätettä/ terminaalia. Monon verkkosivuilta löytyvistä asennusohjeista on tarjoilla komentosarjoja eri Ubuntu Linux pohjaisille käyttöjärjestelmille. Koska käytin kirjoitushetkellä uusinta versiota Linux Mint-käyttöjärjestelmästä (19.2 Tina), hyödynsin kohtaan ”Ubuntu 18.4” merkittyjä päätekomentoja. Mint pohjautuu Ubuntuun, joten asentamisen suhteen ei pitäisi tulla yhteensopivuus ongelmia. Ensimmäinen komentosarja luo järjestelmään erillisen ”säilöntäpaikan” (repository) Monolle, joka sisältää tarvittavat paketit.

```
sudo apt install gnupg ca-certificates

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF

echo "deb https://download.mono-project.com/repo/ubuntu stable-bionic
main" | sudo tee /etc/apt/sources.list.d/mono-official-stable.list

sudo apt update
```

Jotta sovellus saadaan käännettyä, tarvitaan siihen erillinen paketti.

```
sudo apt install mono-devel
```

Asennusohje suosittelee myös paketin ”Mono-complete” asentamista, pääasiallisesti täydentämään edellä mainittuja paketteja (Mono-project 2019).

```
sudo apt-get install mono-complete
```

Seuraavaksi asennetaan Monodevelop.

```
sudo apt install apt-transport-https dirmngr

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF

echo "deb https://download.mono-project.com/repo/ubuntu vs-bionic main" |
sudo tee /etc/apt/sources.list.d/mono-official-vs.list

sudo apt update
```

```
sudo apt-get install monodevelop
```

Komennot: (MonoDevelop Project 2019).

Muuta huomioitavaa

Monon asennuspaketti on saatavilla myös Linux Mintin ”ohjelmistohallinnasta” nimellä ”Mono-complete”. Paketin versio kannattaa kuitenkin tarkistaa ennen asentamista. Mikäli asennuksen tai Monodevelopin käynnistämisessä on ongelmia, kannattaa kokeilla myös seuraavia komentoja.

```
sudo apt-get install gtk-sharp2
sudo apt-get install gtk-sharp3
```

GtkSharp on Mono- ja dotNET-kehitystä varten toteutettu käyttöliittymätyökalu, jonka asentaminen on tiedettävästi auttanut korjaamaan Monodevelopin yhteensopivuus ongelmia (Stack Overflow 2017).

Monogamen asentaminen

Monogamen asentamisen voi suorittaa täysin käyttämällä erinäisiä komentoja, mutta tätä asennusta varten latasin Monogamen asennustiedoston sen virallisilta sivuilta, kun sellainen oli saatavilla. Asennuksen jatkamiseen käytin kuitenkin päätekomentoja. Komennon ”cd” käyttö riippuu siitä mihin Monogamen asennustiedosto on tallennettuna. Minulla se löytyy kansiosta ”Lataukset” (kansion nimi määräytyy käyttöjärjestelmälle määritellyn kielen mukaan). Tämän

jälkeen asennustiedostolle voidaan määritellä erilaisia toimintoa. Komennolla ”chmod” määritetään tiedosto suoritettavaa muotoon (+ = lisää, x =suoritus). Jälkimmäinen komento toteuttaa varsinaisen suorittamisen, joka käynnistää Monogamen asennuksen.

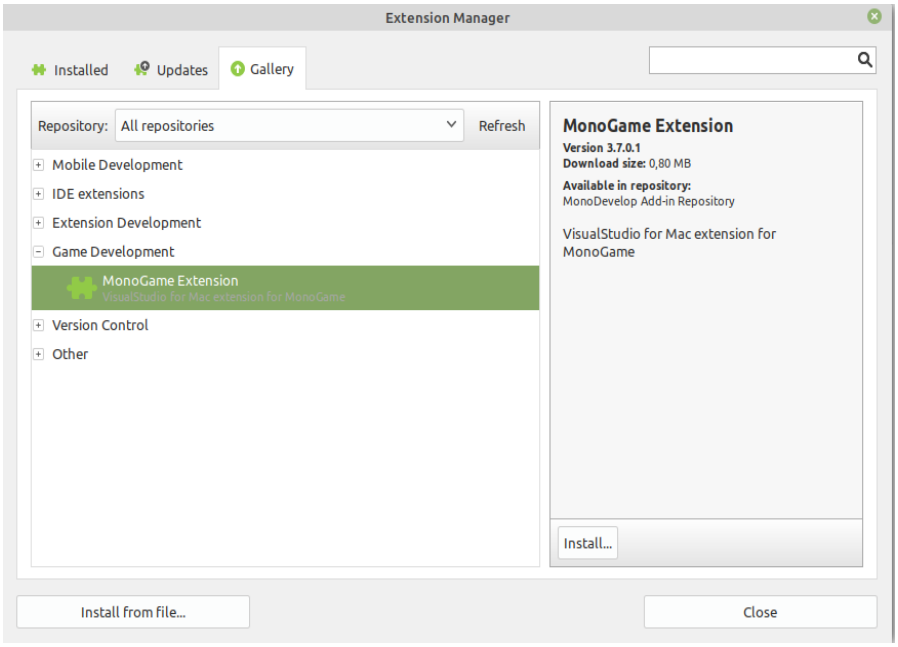
```
cd Lataukset
chmod +x monogame-sdk.run
sudo ./monogame-sdk.run
```

Asennus suoritetaan automaattisesti ja käyttäjän tarvitsee vastata vain yhteen asennuksen jatkamiseen liittyvään kysymykseen. Asennuksen lopussa käyttäjälle annetaan erilaisia ohjeita, jotka eivät vaadi erityistoimenpiteitä.

```
 - To uninstall MonoGame SDK you can run 'monogame-uninstall' from terminal.
 - To install templates for MonoDevelop, go Tools > Extensions > Gallery > Game
Development > MonoGame Extensions, and install it.
 - To install templates for Rider simply run 'dotnet new --install
MonoGame.Templates.CSharp'
```

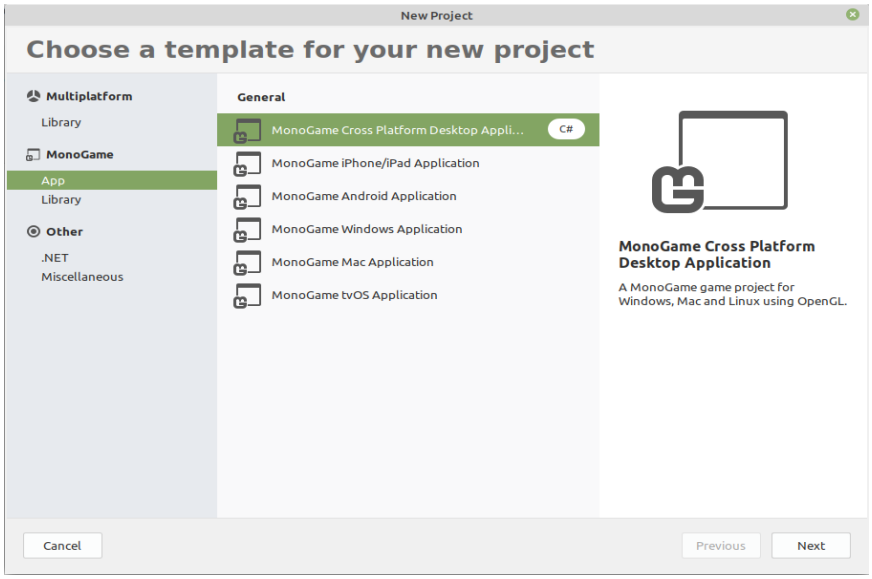
Seuraavaksi Monogamen ”alustapohjien” määrittely MonoDevelopiin, jonka voi suorittaa seuraavan ohjeen mukaisesti navigoimalla valikosta (kuva 9).

```
To install templates for MonoDevelop, go Tools > Extensions > Gallery >
Game Development > MonoGame Extensions, and install it.
```



Kuva 9: MonoDevelop Extension Manager (Joonas Niinistö)

Nyt Monogamen alustapohjat ovat asennettu ja varsinainen kokeilu voidaan aloittaa. Kuten aiemmin, voit nyt valita haluamasi alustapohjan, joka määritellään käyttöösi. Koska testissä on Linux ympäristö, valitsen ”MonoGame Cross Platform Desktop Application”-pohjan (kuva 10).

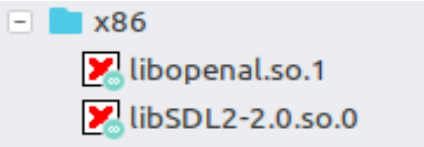


Kuva 10: Monogamen alustapohjat MonoDevelopissa (Kuva: Joonas Niinistö)

Aloittaessani projektin kopioin tarvittavat luokat uuteen projektiin. Käytin tuttua nimiavaruutta ”ParadeOpen”, jolloin luokkiin ei tarvinnut tehdä muita muutoksia, kuin muuttaa nimiavaruus muotoon ”ParadeOpen.Desktop”. Content Pipeline toimi myös kuten aiemmin, jota käyttäen siirsin kaikki pelin tarvitsemat tiedostot uuteen projektiin.

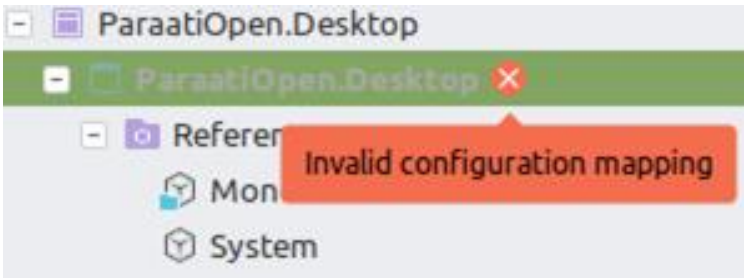
Erikoinen ratkaisu sovelluksen käynnistämiseen

Kun tiedostot ja nimiavaruudet olivat kunnossa, ei ollut oikeastaan muuta jäljellä kuin kokeilla kääntää sovellus suoritettavaan muotoon. Vaihtoehtoina oli ”Debug” sekä ”Release”, jotka johtivat virheilmoitukseen. Ilmoituksen mukaan projektista puuttui ”exe” tiedosto, joka käynnistäisi sovelluksen. Ensin oletin kirjastojen ”libopenal.so.1” sekä ”libSDL2-2.0.so.0” puuttuvan tai olevan viallisia, sillä Monodevelopin käyttöliittymässä kirjastoille oli määritelty punainen ruksi (kuva 11).



Kuva 11: ”Kirjastot” (Kuva: Joonas Niinistö)

Ruksi päätelmä ei kuitenkaan pitänyt paikkaansa, sillä kirjastot (sekä muut samaisella ruksilla merkityt tiedostot) löytyivät projektikansiosta. Ruksien todellinen merkitys mitä todennäköisimmin pohjautuu Monodevelopin kykyyn tunnistaa tiedostoja. Seuraavaksi tulkitsin virheen johtuvan Monodevelop yrittävän kääntää projektia väärälle alustalle. Exe-tiedostot kuitenkin tunnetaan Windows-ympäristössä eikä varsinaisesti Linuxilla (paitsi Winellä). Aloinkin tutkimaan projektitiedostoja, kuten ”app.manifest”, ”AssemblyInfo.cs” sekä ”program.cs”. Tämän aikana kokeilin satunnaisesti kääntää sovellusta, joka loppujen lopuksi kääntyi onnistuneesti ja käynnistyi. En ole ihan varma, miten tämä tarkalleen tapahtui. Arvelisin projektin kääntämisvaiheessa olleen jotain, jonka kääntäjä pystyi automaattisesti korjaamaan tai sitten tein jonkin pienen, mutta oleellisen muokkauksen, jota en huomioinut. Mahdollinen syy saattaa löytyä virheilmoituksen Invalid Configuration Mapping takaa (kuva 12), johon perehdyin vasta onnistuneen kääntämisen jälkeen.



Kuva 12: ”Invalid Configuration Mapping” (Kuva: Joonas Niinistö)

Aloittaessasi uuden projektin IDE:n sivupalkkiin saattaa ilmestyä virheilmoitukseen viittaava ruksi kuvake. Hiiren vieminen sen päälle tulostaa tekstin ”Invalid Configuration Mapping”. Ongelma johtuu projektin kansiorakenteesta, tarkalleen ottaen kansiosta ”Any CPU”, joka sijaitsee seuraavassa polussa.

```
/home/[käyttäjä]/Projects/ParaatiOpen/ParaatiOpen/bin/DesktopGL
```

Linux ei hyväksy kansioiden nimissä olevia välilyöntejä ja ongelman korjaamiseksi ne on hyvä poistaa. Tämän voi tehdä seuraavasti.

- Avaa tiedosto ”[Projektin nimi].Desktop.csproj” tai vaihtoehtoisesti hiiren oikea klikkaus virheilmoitusta antavaan tiedostoon (ParadeOpen.Desktop).
- Tools > Edit file
- Muuta kaikki ”Any CPU” tekstit muotoon ”AnyCPU

Tämä määrittelee automaattisesti loput muutokset, mukaan lukien uuden projektikansion, johon tiedostot tallennetaan. (Monogame Team 2019g.)

5.2.2 Mono Runtime – Sovelluksen suorittaminen ja yhteenveto

Varsinaisen sovelluksen toimintaan nähden ei ole suuremmin kommentoitavaa, sillä se toimi normaalisti XML-tiedostoa myöten. Taustalla tapahtuviin toimenpiteisiin kannattaa kuitenkin kiinnittää enemmän huomiota. Aiemmin mainittu exe-tiedosto ei suinkaan ollut virheellinen. Alustasta huolimatta kaikki dotNET/ Monogame pohjaiset sovellukset käännetään Microsoftin määrittelemiä ehtoja käyttäen. Kaikki Monogamella toteutetut OpenGL sovellukset suoritetaan käyttämällä ”Mono Runtimea”, joka on implementaatio aiemmin käsitellystä ”Common Language Infrastruktuurista” (CLI). Se mahdollistaa dotNET pohjaisten sovellusten kääntämisen sekä suorittamisen. DotNET pohjaisen Parade-pelin yksinkertainen kääntyminen Linuxille on pääasiallisesti Mono-projektin ansiosta. Olin kuitenkin positiivisesti yllättynyt kuinka helposti sovellus loppujen lopuksi oli suoritettavissa ilman erillisiä muutoksia ohjelmakoodiin.

5.3 Android ja Xamarin

Monogamen Android (ja IOS) sovellukset käyttävät apunaan erillistä kehitysympäristöä, nimeltä ”Xamarin”. Se on Mono-projektiin pohjautuva ja alustariippumattomaan mobiilikehitykseen tarkoitettu kehitysympäristö, joka käyttää dotNET rajapintaa sekä C#-kieltä. Ympäristöön kuuluu Xamarin.Forms

on kirjasto, jonka avulla voi kehittää yhteensopivaa koodia muille Xamarinin ympäristöille. Näihin kuuluu Monon päälle kehitetyt Xamarin.iOS ja Xamarin.Android. Tuki löytyy myös Microsoftin Universal Windows Platformille (Windows Phone). Form-ympäristöä ei kuitenkaan ole pakko käyttää, vaan kehitystä voi toteuttaa suoraan halutulle alustalle. Nykyinen toimintamalli on saavutettu kääntämällä mobiilialustojen kehitystyökalut (SDK) C#:lle luettavaan muotoon. Tästä syystä käyttäjän tulee varmistaa mobiilialustan kehitystyökalujen yhteensopivuus Xamarinin kanssa, ennen käyttöönottoa. (Microsoft 2017.)

Xamarin.Androidilla toteutetut sovellukset käyttävät Monoa, sekä Androidin ajonaikaista ympäristöä (Android Runtime, ART) samanaikaisesti Linux Kernelin päällä. Tämä on mahdollistettu hyödyntämällä ”Java Native interface” (JNI)–rajapinnan sisältämiä kutsuja (calls). Kokonaisuus tarjoaa käyttäjälle mahdollisuuden hyödyntää Monon ja ART:n funktionaalisuuksia yhtenäisesti.

5.3.1 Android toteutus

Viimeisenä testattava alustana on Android, jossa hyödynnän käytössäni olevaa One Plus 3-puhelinta. Toisin kuin aiemmissa kokeiluissa, tällä kertaa ohjelmakoodiin tulee tehdä muutamia muutoksia, pääasiassa muokkaamalla pelin kontrolleja sopivaksi kohdealustaa varten. Tämän lisäksi kaikki pelissä esiintyvät näppäimistökomentoihin viittaavat tekstit tulee muuttaa tai poistaa kokonaan.

Päätin aluksi jatkaa projektin työstämistä Linuxilla ja katsoa kuinka pitkälle sen kanssa pääsee. Vaikka projektin määrittely onnistuikin ilman mainittavia ongelmia, Monodevelop ilmoitti heti, ettei projekti ole yhteensopiva IDE:n Linux versiossa. Tutkittuani Monogamen foorumeita hetken löysin aiheeseen liittyvän keskustelun, jossa kävi ilmi, ettei Linux pohjainen Android kehitys suoranaisesti ole tuettu, vaikkakin mahdollista. Projektin jatkaminen edellyttäisi projektin kääntämisen manuaalisesti sekä muokkaamaan Monodevelopiin saatavia plugineja yhteensopivammaksi. (Monogame Team 2019e.) Vaikkakin pluginien muokkaaminen ja manuaalinen kääntäminen olisivatkin oiva lisä työhöni, päätin

suosia yleisempää Windows menetelmää. Tämä on kuitenkin ensimmäinen yritykseni toteuttaa Android sovellusta Monogamella.
Monogamen sivuilla on annettu erilliset ohjeet Android ympäristön määrittelemiseen, jossa pääasiallisesti halutaan huomioida vaadittu Android SDK:n minimiversio 4.2 (API level 17, Jelly Bean) (Monogame Team 2019d). SDK:n lataamiseen on muutama vaihtoehto, kuten Android Studio, joka keskittyy Android sovellusten kehittämiseen. Mikäli kyseiselle sovellukselle ei ole muuta käyttöä, Androidin sivuilta saa ladattua erillisen työkalun ”SDK-tools”, jolla eri versioita voi myös ladata ja asentaa. (Google 2019.) SDK-paketteja on saatavilla myös kolmannen osapuolen sivuilta, mutta tieto- ja sovellusturvaan liittyvien riskien vuoksi niiden käyttö ei ole suositeltavaa.

Ympäristön määritys

Käytössäni on DirectX toteutuksessa käyttämäni Visual Studio 2015 sekä Xamarin viimeisine päivityksineen. Lisäksi mobiilikehitykseen tarvitaan Java-alustaa. Android projektia varten on oma pohjansa, jota käytin projektin määrittelyyn. Seuraavaksi siirsin DirectX version ohjelmakoodin sekä tiedostot uuteen projektiin ja yritin kääntää projektia. Sovelluksen kääntäminen ei suorita sovellusta, mikäli käytössä ei ole emulaattoria, mutta sitä voi yhä käyttää virhe- ja varoitusilmoitusten kartoittamiseen. (Emulaattori on saatavilla Android Studion kautta). Ohessa muutamia kohtaamiani virhetilanteita.

Xamarin.Live

Mobiilisovellus, joka toimii välikappaleena projektin testaamisessa suoraan puhelimeen. Sovellusta ei ole enää edes saatavilla ja näin ollen sen voi jättää huomioimatta.

Activity1.cs

“ScreenOrientation' does not contain a definition for 'FullUser'”. Ilmoitus tiedostosta ”Activity1.cs”, joka sisältää Android sovellukselle tyypillistä perustietoa, kuten sovellusikonin sekä aloituskuvan sijainnit. ”ScreenOrientation”

parametri tarkoittaa näytön suuntaa ja kuinka paljon sovellus voi kääntää kuvaansa. Tässä tapauksessa parametri ei löytänyt määritelmää ”Full User”, jolla tarkoitetaan automaattisesti kääntyvää näyttöä riippuen missä asennossa käyttäjä pitelee laitetta. Sille ei ole hyötyä projektissa, jonka suunnittelin käyttämään ”landscape” näyttötilaa, joka lukitsee näytön vaakasuoraan tilaan, mikä sopiikin Parade-pelin tyyliin.

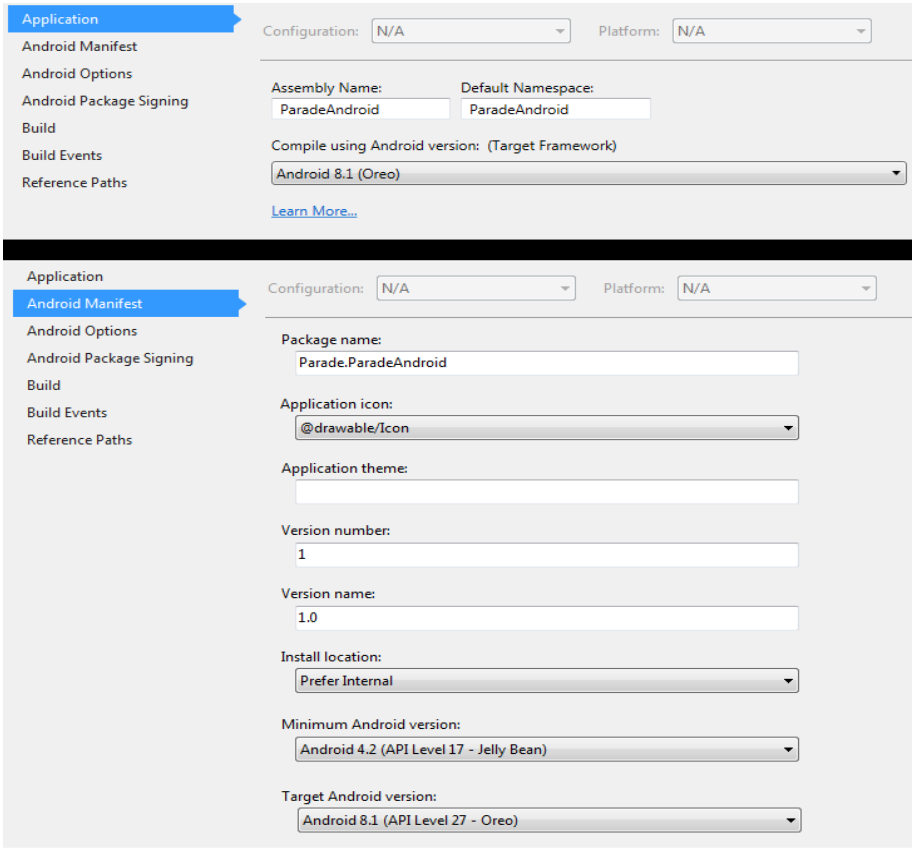
`ScreenOrientation = ScreenOrientation.FullUser`	`ScreenOrientation = ScreenOrientation.Landscape`

Java Max Heap Size

Mikäli kääntäjä antaa projektin tilaan liittyviä virheilmoituksia, voi Javan tilarajaa nostaa kohdasta ”Properties -> Android Options -> Advanced”. Tilarajan maksimimäärä on ”1024m” (megatavua) ilman lainausmerkkejä.

Android SDK:n määrittely

Projektiin tarvittavat SDK-paketit tulee olla valmiiksi ladattuna ja asennettuna ennen määrittelyä. Toimenpiteet suoritin Android Studion ”SDK Managerin” kautta. Puhelimeni tämänhetkinen versio Androidista on 9 (Pie), joten Android 8 käyttäminen sovelluksen kääntämiseen on riittävä. Minimiversioksi on asetettu Monogamen ohjeistama Android 4.2 (Level 17-Jelly Bean). SDK-määrityksiä pääset muokkaamaan navigoimalla seuraavasti. ”Project -> [projektin nimi].Properties” (kuva 13).



Kuva 13. Android SDK määritys (Kuva: Joonas Niinistö).

Keystore ja APK -paketti

Kun SDK on määritelty oikein, voi sovelluksen kääntää asennettavaksi Android sovellukseksi, APK-paketiksi. Toimiakseen oikein paketti tarvitsee erillisen ”avainsäiliön” (Keystore), joka sisältää tietoja sovelluksen kehittäjästä sekä salasanoja sovelluksen suojaamiseksi. Avainsäilön luontiin on monta tapaa, kuten Android Studio tai Xamarinin ”Distribute” vaihtoehto, jota kokeilin tuloksetta. Mikäli edellä mainittuja sovelluksia ei halua käyttää, voi avainsäiliön luoda komentoriviltä Javan työkalua ”keytool” käyttäen. Alla esimerkki keytoolilla määritellystä avainsäilö komennosta sekä parametreista (taulukko 13).

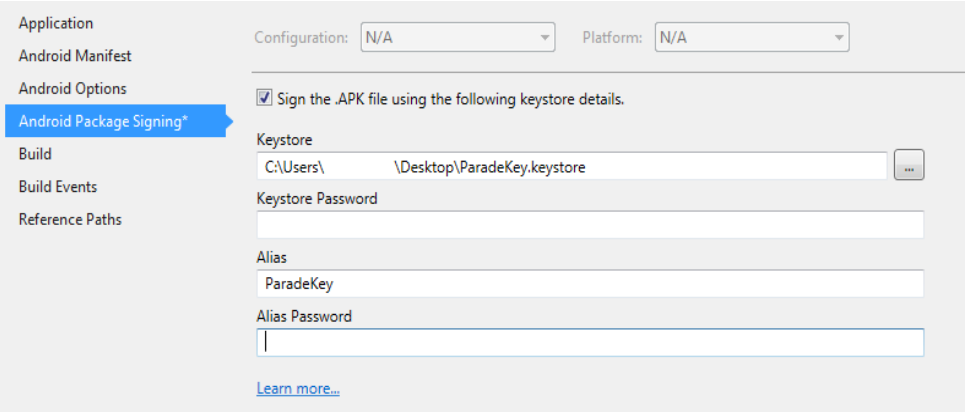
```
keytool -genkeypair -v -alias ParadeKey -keyalg RSA -keystore
C:\Users\[kayttaja]\Desktop\ParadeKey.jks -keysize 2048 -validity 10000
```

Taulukko 13. Käytetyn Keytool komennon parametrit (Oracle 2019).

-genkey/ -genkeypair	Komento, joka määrittää varsinaisen avaimen luonnin.

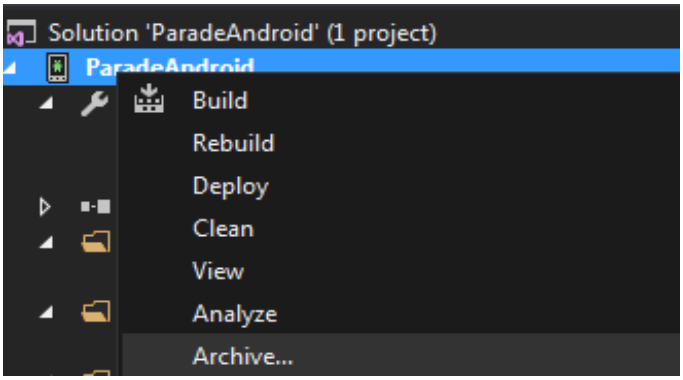
-v	Aktivoi monitahoisen tilan (Verbose mode), jolloin keytool tulostaa lisätietoa luettavassa muodossa komentoriville.
-alias	Avaimen nimi
-keyalg	Määritelee algoritmin avainsäilön suojaamiseksi. Käytössä on RSA -suojaus.
-keystore	Säilön sijainti. Esimerkissä asetettu työpöydälle
-keysize	Määrittelee avainsäilön koon. ”RSA” algoritmin vuoksi kooksi tulee määritellä 2048 bittiä.
-validity	Voimassaolo aika päivinä. Esimerkissä 10000 päivää, joka on karkeasti 25-vuotta.

Komennon jälkeen kysytään muutamia sovelluksen toteutukseen liittyviä tietoja, kuten kehittäjän etunimi ja sukunimi, yritys- ja sijaintitietoja, joihin käyttäjä voi vastata parhaiten katsomallaan tavalla. Tämän lisäksi avainsäilöön tulee antaa salasanat. Kun avainsäilö on määritelty, lisätään se projektiin (kuva 14).



Kuva 14. Android paketin avainmääritys (Kuva: Joonas Niinistö)

Projektin voi nyt viedä asennettavaan muotoon Xamarinin Archive toiminnolla (kuva 15).



Kuva 15. Archive valinta (Kuva: Joonas Niinistö)

5.3.2 Microsoft.Xna.Framework.Input.Touch

Kun sovelluksen toteuttamiseen tarvittavat määritykset ovat valmiina, on aika muokata varsinaista ohjelmakoodia. Mikäli mobiilisovelluksia on toteuttanut aiemmin, tietokonehiirtä käyttävien funktionaalisuuksien voisi olettaa toimivan myös kosketusnäytöllä. Monogamen tapauksessa näin ei kuitenkaan ole vaan se käyttää ”Touch”-nimiavaruutta ja sen funktionaalisuutta (taulukko 14).

Taulukko 14. TouchlocationState vaihtoehdot ja vaikutukset (Martin 2017).

Pressed	Tarkistaa onko kosketus tehty. Toisin kuin toiminnon nimi antaa ymmärtää, vaikutus ei ole pysyvä. Käytetään tarkistaessa, onko painikekuvia painettu.
Moved	Määrittää onko kosketus vaihtanut paikkaa. Koska vaikutus on pysyvä, sitä käytetään pelihahmon liikuttamiseen.
Released	Aktivoituu, kun kosketusta ei enää tapahdu
Invalid	Mikäli kosketus on vääränlainen,

Kosketuskontrolleja varten pelihahmon ohjattavuus on pistetty uusiksi, jossa. pelihahmoa voi liikuttaa koskettamalla näyttöä, hahmon alkaen seuraamaan kosketuksen sijaintia. Kontrolleihin on myös määritelty ”näpäytys ominaisuus”, jossa pelaajan tehdessä kaksi nopeaa näpäytystä aktivoi esineen.

```
public void UpdateMovementAndroid(TouchCollection Mstate, int SpeedBonus,
GraphicsDevice graphics, GameTime gameTime)
{
    int addSpeed = (playerSpeed + SpeedBonus);
    GetTouchLocation = Mstate;
    PlayerCollision.X = (PlayerSize.X + ((PlayerSize.Width / 3) - 10));
    PlayerCollision.Y = (PlayerSize.Y + (PlayerSize.Height / 3) - 20);

    foreach (TouchLocation location in GetTouchLocation)
    {
       if (location.State == TouchLocationState.Moved &&
       GetTouchLocation.Count > 0)
       {
          if (!TappingState) // Tapped once
          {
              ItemTap++;
              TappingState = true;
          }
          TargetPosition = new Vector2((int)location.Position.X,
          (int)location.Position.Y);
          targetX = (int)TargetPosition.X - PlayerSize.X;
          targetY = (int)TargetPosition.Y - PlayerSize.Y;
          Distance = Math.Sqrt(targetX * targetX + targetY * targetY);
          velo1 = (targetX / Distance) * addSpeed;
```

```
            velo2 = (targetY / Distance) * addSpeed;

            if (Distance > PlayerSize.X / 10)
            {
               PlayerSize.X += (int)velo1;
               PlayerSize.Y += (int)velo2;
               IsMoving = true;
            }
         }
         else
         {
            IsMoving = false;
            TappingState = false;
         }

         if (IsMoving)
         {
             if (PlayerSize.X < TargetPosition.X) // Moving Right
             {
                 mirror = SpriteEffects.FlipHorizontally;
             }
             else // Moving left
             {
                 mirror = SpriteEffects.None;
             }
          }
          else
          {
             PlayerSize.X += SpeedBonus;
          }

          if (ItemTap > 0) // Use items by tapping screen
          {
             ItemTapTimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
          }
          if (ItemTapTimer >= 400f)
          {
              ItemTap = 0;
              ItemTapTimer = 0;
          }


          if (ItemTap >= 2 && CurrentHolding != 0)
          {
              ReleaseItem();
              ItemTap = 0;
              ItemTapTimer = 0;
          }
          AllowScore = !IsMoving;
          CorrectAnimationAndroid(graphics);
          NonKeyActions(graphics);
      }
   }
```

Kontrollien lisäksi Android versio saa myös pieniä graafisia lisäyksiä (kuva 16).

Kuva 16: Android version painikkeet (Kuva: Joonas Niinistö).

Painikkeet korvaavat pelin asetuksia määritteleviä toimenpiteitä, jotka aktivoituvat käyttäjän näpäyttäessä niitä. Toimintoja varten on luotu luokka ”TouchInterface.cs”.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input.Touch;

namespace ParadeAndroid
{
    class TouchInterface
    {
        protected Texture2D TouchScreenHUD;
        protected TouchCollection GetTouchLocation;
        protected Rectangle TouchRadius, ButtonRectangle;
        protected Color ButtonColor;
        public bool ButtonPressed;

        public TouchInterface(Texture2D Image, int X, int Y) // Setup
        {
            TouchScreenHUD = Image;
            ButtonRectangle = new Rectangle(X, Y, Image.Width, Image.Height);
        }
        public void Update(TouchCollection IsTouching)
        {
            GetTouchLocation = IsTouching;
            foreach (TouchLocation location in GetTouchLocation)
            {
                TouchRadius = new Rectangle((int)location.Position.X,
                (int)location.Position.Y, 5, 5);


                if (TouchRadius.Intersects(ButtonRectangle))
                {
                    if (location.State == TouchLocationState.Pressed &&
                    GetTouchLocation.Count > 0)  //ANDROID
                    {
                        ButtonPressed = !ButtonPressed;
                    }
                }
            }
        }
        public void Draw(SpriteBatch spriteBatch)
        {
            if (ButtonPressed)
            {
                ButtonColor = Color.Red;
            }
            else
            {
                ButtonColor = Color.White;
            }
            spriteBatch.Draw(TouchScreenHUD, ButtonRectangle, ButtonColor);
        }
    }
}
```

5.3.3 Yhteenveto Android versiosta ja loppumaininnat

Edellä mainittujen muutosten lisäksi sovelluksen pääsilmukkaan tehtiin muutamia kosketusnäyttötoimintoja. Muokkauksen jälkeen Android versio oli pelattavassa tilassa, vaikka se kärsi yhä muutamasta graafisesta virheestä, sekä XML-tallennuksen yhteensopimattomuudesta. Joka tapauksessa, erilaisuutensa vuoksi tämä kokeilu oli mielestäni kaikkein mielenkiintoisin ja toi hieman suuntaa mahdollisille mobiiliprojekteille. Mitä mieltä olen alustavertailusta kokonaisuutena?

Monogamen tuntuman saamiseen Parade-peli oli mielestäni sopiva. Se tarvitsisi paljon hiomista, mutta tarkoitusperänsä ja alustavertailun vuoksi se ei ole välttämättä tarpeen. Pelin kääntäminen eri alustoille itsessään oli hyvin yksinkertainen prosessi, eikä vaatinut suuremmin mitään muutoksia varsinaiseen ohjelmakoodiin. Suurimmat ristiriidat tulivat yleensä käytettävien työkalujen kanssa sekä pelin tallentamisen (XML) yhteydessä. Ongelmat olivat kuitenkin yleensä korjata pienen taustatutkimuksen avulla. Alustavertailun tärkein ominaisuus oli kuitenkin tutusta käytössä oleviin alustoihin sekä alustakohtaisiin työkaluihin ja eroavaisuuksiin. Vaikka Monogame projektin lähdekoodi pysyisikin identtisenä, alustan mukaan, taustalla tapahtuvat toimenpiteet vaihtelevat suuresti välikerroksien ja rajapintojen kautta. Mikäli kirjoituksesta pois jätettyjä alustoja mielii kokeilla, Parade-peli on vapaasti muokattavaa, avointa lähdekoodia ja löytyy Gitlab profiilistani.

6 Monogamen tulevaisuus

On selvää, että kilpailuasemassa Monogame jää monipuolisempien ja kaupallisen pelikehitykseen tarkoitettujen työkalujen ja moottorien varjoon. Parantaakseen asemaansa Monogame tarvitsisi lisää verkostovaikutusta, jonka avulla sen dokumentaatio ja käyttäjämäärä kasvaisivat. Tekniikan puolesta sen nykyinen käyttöikä riippuu siitä, kuinka pitkään XNA-yhteensopivuus halutaan säilyttää. Monogamen pääkehittäjä Tom Spilman (2016) on maininnut

päivittävänsä ja parantavansa ”kaikki huonoiksi todetut osat”, mutta toistaiseksi yhteensopivuus XNA:han on haluttu säilyttää. Kun XNA:sta luopuminen tulee väistämättömäksi, Monogamen nykyistä rakennetta joudutaan muokkaamaan laajalti. Olemalla vapaan lähdekoodin ohjelmisto Monogame tarjoaa erilaisia mahdollisuuksia. Lisensointiin liittyviä rajoitteita tai riippuvuutta Monogamen nykyiseen pääasialliseen kehitystiimiin ei ole (Monogame Team 2019a). Projektin kehitykseen voi osallistua niin kauan, kun siihen on kiinnostusta tai kuten Monogamen historiassa on toimittu useaan otteeseen, aloitetaan kehittää jotain täysin uutta sovellusta, johon hyödynnetään Monogamen ominaisuuksia.

C#-kielen sekä dotNET ympäristön tulevaisuus näyttää lupaavalta. Richard Landenin mukaan ympäristöä pyritään päivittämään vuosittain marraskuussa. Julkaisuja on suunniteltu alustavasti vuoteen 2023 saakka, joista seuraavat ovat dotNET Core 3.0 version julkaisu syyskuussa sekä dotNET 5.0 marraskuussa 2020. (Larden 2019.)

Sitten on grafiikkarajapinnat, joita täytyy päivittää ja kehittää vastaamaan aikansa vaatimuksia. Sen myötä jotkin rajapinnat saattavat vanhentua kokonaan ja tilalle tulee modernimpi vaihtoehto. Koska Monogamen toiminta riippuu hyvin paljon erilaisista grafiikkarajapinnoista, tulee sen myös päivittyä pysyäkseen ajan tasalla. Rajapintojen kehityksestä kuitenkin vastaa muut tahot ja Monogamen kehittäjien tulee vain soveltaa niihin yhteensopivuus.

6.1 DirectX 12 - Microsoft

Uusimman DirectX-version tavoitteena on antaa kehittäjälle mahdollisuus toteuttaa kehittyneempää matalan tason ohjelmoinnin ratkaisuja, joilla kehittäjät pystyvät luomaan nopeampia sovelluksia rasittamatta käytössä olevaa laitteistoa.

Uudistuksiin lukeutuu ”Eksplisiivinen monisovitin” (Explicit Multi-Adapter), joka pystyy käyttämään monigrafiikkaprosessorien yhdistelmiä eri valmistajien ja vaihtelevien nopeuksien väliltä. Sen toimivuus kattaa myös nykyaikaiset suorittimet. Käyttäjät pystyvät myös määrittelemään omia ”komentolistojaan”, joilla määrittää haluamiaan renderöintikutsuja grafiikka prosessoriin. Rajapinnan

suorituskykyä on pyritty parantamaan vähentämällä piirtokutsuun liittyviä ongelmia. Toiminto on perustunut yhteistyöhön, jossa suoritin on lähettänyt tiedon grafiikkaprosessorille siitä mitä ruudulle piirretään. Tämä on rajoittanut etenkin peleissä objektien lukumäärää ruudulla, suorittimen hoitaessa samalla muita tehtäviä, kuten tekoälyä. (Chester 2015.) Microsoft on lupaillut rajapinnan saatavuutta vain Windows 10 laitteille. Sitä aiemmat käyttöjärjestelmät ovat saaneet tukea vain tiettyjen sovellusten kohdalla, kuten World of Warcraft. (Lu 2019). Monogame ei vielä tue DirectX 12 rajapintaa, mutta sen käyttämässä SharpDX wrapperista löytyy tarvittava tuki.

6.2 Metal – Apple

Applen kehittämä matalan tason grafiikkarajapinta, jossa kehittäjällä on suora pääsy macOS, iOS sekä tvOS-alustojen grafiikkaprosessoriin. Se on Applen korvike OpenGL-rajapinnalle, jonka Apple poisti käytöstään iOS12-versiosta. Rajapinta tarjoaa käyttäjilleen vapaamman pääsyn laittein rautaan, joka mahdollistaa tehokkaampien ja näyttävämpien sovellusten kehittämisen. Tämän lisäksi rajapinnan toimintaa on pyritty kehittämään OpenGL:stä monin tavoin, kuten suorittimen ja grafiikkaprosessorin välinen yhteistyö muistin jakamisen ja synkronoinnin kanssa. OpenGL:n käytöstä poisto ei kuitenkaan toistaiseksi estä kehittäjiä käyttämästä rajapintaa, mutta tulevaisuudessa Metal tulee korvaamaan sen lopullisesti. OpenGL kehittäjät voivat myös hyödyntää maksullista MoltenGL-ohjelmistokirjastoa, joka suorittaa OpenGL sovelluksia Metal-rajapinnan päällä (MoltenGL kehitystiimi 2019).

Monogamelle ei ole vielä tukea rajapintaan, mutta Xamariniin on julkaistu jo käyttöönotettava esimerkkipohja ”MetalBasic3D”. Sen avulla käyttäjä voi hyödyntää muun muassa Metal rajapinnalle ominaisia luokkia ja tiedostomuotoja sekä määritellä varjostimia. (Demchenko 2019.)

6.3 Vulkan – Khronos Group

Aiemmin OpenGL rajapintaa kehittänyt Khronos Groupin kehittämä ja julkaisema matalan tason grafiikkarajapinta, joka julkaistiin alun perin 16.2.2016 (Khronos Group 2016). Aiemmin projekti tunnettiin nimiltä ”glNext” sekä ”Next Generation OpenGL Initiative” (Smith 2014). Rajapinnan kehitys on alkujaan johdettu prosessoreja ja muita elektroniikka komponentteja valmistavan AMD:n ”Mantle” grafiikkarajapinnasta (Hallock 2015) jonka kehitys ilmoitettiin päättyvän vuonna 2019 (AMD 2019). Verrattuna OpenGL:ään, Vulkan pyrkii antamaan, kehittäjälleen aiempaa suuremman hallinnan näytönohjaimeen, paremman tuen monisäikeisille suorittimille, sekä vähentämään niihin kohdistuvaa kuormitusta. Ollessaan matalan tason rajapinta, siihen kehitettävät ajurit tulevat olemaan kevyempiä. (Hajdarbegovic 2016.) Vulkanin kehitys ei automaattisesti syrjäytä OpenGL:ää, mutta sen edistyneemmän teknologian myötä, sitä todennäköisesti aletaan suosimaan enemmän tulevaisuudessa. Vulkan tukee useita alustoja, kuten Windows, Linux, Android, Nintendo Switch sekä Apple-ympäristöt MoltenVK:n avustamana (Khronos Group 2019).

6.4 MoltenVK – The Brenwill Workshop

MoltenVK on ”The Brenwill Workshopin” kehittämä ohjelmistokirjasto, joka mahdollistaa Vulkanilla kehitettyjen sovellusten toiminnan Applen Metal-rajapinnassa. Alun perin julkaistu 27.7.2016 kaupallisena tuotteena (MoltenVK Team 2016). Sittemmin se siirrettiin Vulkan Portable Initiative-projektiin Khronos Groupin, LunarG:n, Valven ja The Brenwill Workshopin yhteistyön tuloksena (Khronos Group 2018). Sopimuksen yhteydessä MoltenVK julkaistiin vapaan lähdekoodin ohjelmistona Apache lisenssillä (Larabel 2018). Valven kehittämässä Dota2 pelin macOS-versiossa voi vaihtoehtoisesti käyttää MoltenVK:n avulla toimivaa Vulkan rajapintaa. Valve on kehoittanutkin kehittäjiä testaamaan MoltenVK:ta. (Valve 2018.) Alkuperäiseen Vulkan-toteutukseen nähden MoltenVK:lla on muutamia rajoitteita liittyen Vulkanin yhteensopivuuteen (Khronos Group 2019).

7 Pohdinta

Opinnäytetyö ei ollut vain lyhyt tutustuminen Monogamen tarjontaan, vaan jonkinlainen dokumentaatio sen eri osa-alueista. Työssä tutustuttiin ohjelmistokehykseen, sen pitkään historiaan ja nykyhetken vaikuttajiin sekä mahdollisiin tulevaisuuden näkymiin. Rajapinnoista esiteltiin perusteita, kuten niiden rakennetta ja vaikutusta sovelluksen toteuttamiseen. Tämän lisäksi työssä käsiteltiin Monogamella toteutettua sovellusta, sekä testattiin sitä eri alustoilla. Tutustumisen aikana käytiin läpi toteutukseen tarvittavia nimiavaruuksia sekä niiden tarjoamia ominaisuuksia.

Monogamen käyttö ja siihen tutustuminen on ollut minulle hyvin avartava, yllättävä sekä hieman sekava kokemus. Kerättävää tietoa on ollut paljon ja sen koostaminen oli välillä aika vaikeaa. Esimerkiksi XNA nimiavaruuksista kirjoittaminen tai historia olisivat riittäneet aiheina sellaisinaan. Pitääkseni jonkin rajauksen aiheiden suhteen päätin kirjoittaa vain ominaisuuksista, joihin käyttäjä tulee ensisijaisesti törmäämään ja joiden pohjalta on helpompi syventyä aiheeseen. Jos puhutaan työn puutteista, se on 3D-esimerkkisovellus, joka olisi nostanut työtä paremmin esille ja samalla antanut Monogamen 3D-kehittämisestä tuntumaa, joka toistaiseksi on melko vähäistä. Tästä jouduttiin kuitenkin muutaman muun aihealueen lisäksi luopumaan ajanpuutteen vuoksi. Mikäli työtä haluaa joskus jatkaa, tutustuisin paremmin 3D-ohjelmointiin, Monogamen lähdekoodiin, työn ulkopuolelle jätettyihin alustoihin sekä uusien grafiikkarajapintojen käyttöönottoon. Näiden ohella tutustuisin paremmin jo olemassa oleviin työkaluihin, jotka parantavat tai helpottavat Monogamen käyttämistä. Kaikesta huolimatta, työn aikana opin enemmän sovellusten toiminnasta, historiasta ja kehityksestä sekä niiden käyttämisestä. Uuden tiedon ansiosta uskoisin pystyväni toimimaan paremmin erilaisten työkalujen ja rajapintojen kanssa sekä hyödyntämään Monogamea paremmin.

Lähteet

Mikäli ollut mahdollista, lähde on ”arkistoitu” katoamistilanteen välttämiseksi käyttämällä ”The Internet Archive Wayback Machine”-verkkopalvelua. Arkistoitu lähde on suora kopio alkuperäisestä.

Acevedo, P. 2016a. Axiom Verge bringing classic looks, gameplay, and MonoGame to Xbox One this year. Windows Central. https://web.archive.org/web/20191021110254/https://www.windowscentral.com/axiom-verge-pax-east-interview. 15.6.2019.

Acevedo, P. 2016b. How MonoGame is bringing more indie developers to XboxOne. Windows Central. https://web.archive.org/web/20191021135711/https://www.windowscentral.com/monogame-developer-interview. 2.10.2019.

Advanced Micro Devices, Inc. 2019. “Radeon Software Adrenalin 2019 Edition 19.5.1 Highlights”. Advanced Micro Devices, Inc.https://web.archive.org/web/20191021123438/https://www.amd.com/en/support/kb/release-notes/rn-rad-win-19-5-1. 21.7.2019.

Apple Inc. 2017. Supported Audio File and Data Formats in OS X. Apple Inc. https://web.archive.org/web/20191021134557/https://developer.apple.com/library/archive/documentation/MusicAudio/Conceptual/CoreAudioOverview/SupportedAudioFormatsMacOSX/SupportedAudioFormatsMacOSX.html. 21.9.2019.

Banton, C. 2019. ”What Is Just in Time (JIT)?”. Investopedia. https://web.archive.org/web/20191021132931/https://www.investopedia.com/terms/j/jit.asp. 4.8.2019.

Ballard, E & Gallivan, R. 2014. Micro Focus to Buy Attachmate in $1.2 Billion Deal. The Wall Street Journal. https://www.wsj.com/articles/micro-focus-to-buy-attachmate-in-1-2-billion-deal-1410780066. 6.11.2019.

Barone, E. 2017. Stardew Valley Coming Soon to Nintendo Switch. Developer Blog. 2.10.2017. https://web.archive.org/web/20190828134431/https://www.stardewvalley.net/stardew-valley-coming-soon-to-nintendo-switch/. 15.8.2019.

Barone, E. 2019. Android Version Coming March 14th. Developer Blog. https://web.archive.org/web/20190828134558/https://www.stardewvalley.net/android-version-coming-soon/. 14.2.2019.

BlitWorks. 2019. FEZ. https://web.archive.org/web/20191021115814/https://www.blitworks.com/fez/. 16.6.2019.

Celeste -Development Team. 2019. ”Tools”. https://web.archive.org/web/20190828123809/https://celestegame.tumblr.com/tools. 22.8.2019.

Charla, C & Dunn, A. 2015. ”XBLIG Program Announcement”. https://web.archive.org/web/20190828120932/https://blogs.msdn.microsoft.com/xna/2015/09/09/xblig-program-announcement/. 23.8.2019.

Chaudhary, V. 2016. Welcome to the UWP World. DotNetForAll. https://web.archive.org/save/https://www.dotnetforall.com/welcome-uwp-world/.17.1.2018

Chester, E. 2015. DirectX 12 vs DirectX 11 – How DX12 will transform PC gaming on Windows 10. Trusted Reviews. https://web.archive.org/web/20191023104338/https://www.trustedreviews.com/opinion/directx-12-vs-directx-11-what-s-new-2922591.4.6.2019

CodePlex Archive. 2019. XNA 3D Level Editor. Microsoft. https://web.archive.org/web/20191021113015/https://archive.codeplex.com/?p=xna3dleveleditor. 20.6.2019.

Computer Hope. 2019. Framework. https://web.archive.org/web/20191021125430/https://www.computerhope.com/jargon/f/framework.htm. 5.5.2019

ConcernedApe LLC. 2019. FAQ- “Who made it?”. https://web.archive.org/web/20190828134136/https://www.stardewvalley.net/faq/ .15.8.2019

Davis, S. 2015. One of Mantle's Futures: Vulkan. Advanced Micro Devices, Inc. https://web.archive.org/web/20191021122604/https://community.amd.com/community/gaming/blog/2015/05/12/one-of-mantles-futures-vulkan. 21.7.2019.

”DevToolsGuy”. 2013. “What is MonoTouch?”. Infragistics. https://web.archive.org/web/20191021115413/https://www.infragistics.com/community/blogs/b/marketing/posts/what-is-monotouch. 16.6.2019

De Vriez,J. 2017. Framebuffers. learnopengl.com. https://web.archive.org/save/https://learnopengl.com/Advanced-OpenGL/Framebuffers. 17.1.2018.

De Icaza, M. 2014. Microsoft Open Sources .NET and Mono. Tirania. https://web.archive.org/save/https://tirania.org/blog/archive/2014/Nov-12.html. 18.1.2018

Demchenko, O. 2019. Xamarin.iOS - MetalBasic3D. Microsoft. https://web.archive.org/web/20191021133152/https://docs.microsoft.com/fi-fi/samples/xamarin/ios-samples/ios8-metalbasic3d/. 14.8.2019.

Etcheverry, I. 2017. Introducing C# in Godot. Godot. https://web.archive.org/web/20191021132810/https://godotengine.org/article/introducing-csharp-godot. 7.8.2019

Falk, J. 2015 SharpDX Beginners Tutorial Part 1: Setting up a SharpDX project in Visual Studio 2013. “johanfalk.eu”.

https://web.archive.org/web/20190123130706/http://www.johanfalk.eu/blog/tag/sharpdx-beginners-tutorial.19.12.2018

Fáyer. 2014. " @renaudbedard what did you use for Fez?". Twiitti. 22.8.2019. Renaud Bédard. 2014. " @fire_tony MonoGame on PC/Mac/Linux". Twiitti. 22.8.2019.https://web.archive.org/web/20190828124717/https://twitter.com/renaudbedard/status/532564905141215233

Finley, K. 2016. Microsoft Says It's in Love With Linux. Now It's Finally Proving It. Wired. https://web.archive.org/web/20191028140017/https://www.wired.com/2016/06/microsofts-open-source-love-affair-reaches-new-heights/.28.10.2019.

Franks,S. 2014. Mask 2D -sprites in XNA. Medium. https://web.archive.org/web/20191022110023/https://medium.com/@sfranks/i-originally-wrote-this-for-the-best-way-to-mask-2d-sprites-in-xna-game-development-stack-949cf7bd7421. 5.9.2019.

Ferbrache, A. 2019. Tooth And Tail Credits. Blue Flame Labs. https://web.archive.org/web/20191021111737/https://www.mobygames.com/game/windows/tooth-and-tail/credits. 15.6.2019.

Fraunhofer Institute. 2017a. Mp3- Overview. https://web.archive.org/web/20191022103849/https://www.iis.fraunhofer.de/en/ff/amm/consumer-electronics/mp3.html. 4.9.2019

Fraunhofer Institute. 2017b. Alive and Kicking – mp3 Software, Patents and Licenses.https://web.archive.org/web/20191022103943/https://www.audioblog.iis.fraunhofer.com/mp3-software-patents-licenses. 4.9.2019

Gibson, E. 2006. “Microsoft releases XNA Game Studio Express”. Gameindustry.biz.https://web.archive.org/web/20190828084847/https://www.gamesindustry.biz/articles/microsoft-releases-xna-game-studio-express .26.8.2019.

GnFur. 2019. MonoFoxe. GitHub. https://web.archive.org/web/20191021123812/https://github.com/gnFur/Monofoxe/. 5.7.2019.

Google. 2019. Android Studio - Command line tools only. https://web.archive.org/web/20191030103959/https://developer.android.com/studio/index.html .29.10.2019.

Google Code Archive. 2019. “MonoXNA”. https://web.archive.org/web/20190828110921/https://code.google.com/archive/p/monoxna/. 17.7.2019

Gullen, A. 2011. Even more about audio licenses on the web. Scirra. https://web.archive.org/web/20191021134710/https://www.construct.net/en/blogs/construct-official-blog-1/even-audio-licenses-web-750. 22.9.2019.

Hajdarbegovic, N. 2015. A Brief Overview Of Vulkan API. Toptotal. https://www.toptal.com/api-developers/a-brief-overview-of-vulkan-api. 9.7.2019.

”Harkiran78”. 2019. What is Just-In-Time (JIT) Compiler in NET. GeeksForGeeks.https://web.archive.org/web/20191021131157/https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net/. 3.8.2019.

Happ, T. 2016. “Achievement Unlocked!”. Thomas Happ Games LLC. https://web.archive.org/web/20191021110557/http://www.axiomverge.com/blog/achievement-unlocked. 15.6.2019.

Hoffman, C. 2016. Why You Shouldn’t Buy Rise of the Tomb Raider (and Other PC Games) from the Windows Store. How to Geek, LLC. https://web.archive.org/web/20191022125634/https://www.howtogeek.com/243012/why-you-shouldnt-buy-rise-of-the-tomb-raider-and-other-pc-games-from-the-windows-store/. 2.6.2019.

Independent Games Festival. 2014. 2012 Independent Games Festival Winners. UBM Tech.https://web.archive.org/web/20140327124703/http://www.igf.com/2012finalistswinners.html. 16.6.2019.

Infinite Flight LLC. 2011. Announcing Infinite Flight for iOS! (iPad/iPhone). Blogspot.https://web.archive.org/web/20191021121320/https://flyingdevstudio.blogspot.com/2011/11/announcing-infinite-flight-for-ios.html. 16.6.2019.

Infinite Flight LLC. 2019. Infinite Flight - Flight Simulator. Google. https://web.archive.org/web/20191021120026/https://play.google.com/store/apps/details?id=com.fds.infiniteflight. 16.6.2019

itameiosarabic. 2016. " @ConcernedApe was wondering what engine/framework you used to make #StardewValley?". Twiitti. 15.8.2019.

ConcernedApe (Eric Barone). 2016. "custom engine, C# using the XNA framework". Twiitti. 15.8.2019. https://web.archive.org/web/20190828133316/https:/twitter.com/concernedape/status/750858584603242496?lang=fi

ISO, International Electrotechnical Commission. 2012. Information technology — Common Language Infrastructure (CLI). https://web.archive.org/web/20191021131816/https://www.iso.org/standard/58046.html. 4.8.2019

Jackson, S. 2016. Getting the most out of your assets - The MonoGame Content Pipeline. Gamasutra https://web.archive.org/web/20191021134329/https://www.gamasutra.com/blogs/SimonJackson/20161027/284243/Getting_the_most_out_of_your_assets__The_MonoGame_Content_Pipeline.php. 20.9.2019.

”Jjagg”. 2019. Ps4 dualshock4 controllers on PC. Monogame Team. http://community.monogame.net/t/ps4-dualshock4-controllers-on-pc/9458/2. 8.9.2019.

Järvinen, P. 2000. IT- tietosanakirja. Jyväskylä: Docendo.

Khronos Group. 2019. MoltenVK Runtime User Guide. GitHub. https://web.archive.org/web/20191021124718/https://github.com/Khronos

Group/MoltenVK/blob/master/Docs/MoltenVK_Runtime_UserGuide.md. 21.7.2019.

Khronos. 2019. Portability Initiative Overview. https://web.archive.org/web/20191022114309/https://www.khronos.org/vulkan/portability-initiative. 9.7.2019.

Khronos. 2019. Vulkan. https://web.archive.org/web/20191022113707/https://www.khronos.org/vulkan/.9.7.2019.

Khronos. 2017. Khronos Announces the Vulkan Portability Initiative. https://web.archive.org/web/20191022113818/https://www.khronos.org/blog/khronos-announces-the-vulkan-portability-initiative.9.7.2019

Khronos. 2019. Khronos Releases Vulkan 1.0 Specification. https://web.archive.org/web/20191022114006/https://www.khronos.org/news/press/khronos-releases-vulkan-1-0-specification. 9.7.2019.

”KojanuGames”. 2019. SoundEffectInstance - Memory consumption. Monogame Team. http://community.monogame.net/t/soundeffectinstance-memory-consumption/9343/3. 3.10.2019.

Krill, P. 2014. Microsoft's open source .Net still can't match open source Java. Info World. https://web.archive.org/web/20191028140600/https://www.infoworld.com/article/2850050/microsoft-open-source-net-cant-match-open-source-java.html. 28.10.2019.

Lander, R. 2019. ”Introducing .NET 5”. NET Blog. https://web.archive.org/save/https://devblogs.microsoft.com/dotnet/introducing-net-5/. 9.7.2019.

Larabel, M. 2018. Vulkan Is Now Available On macOS/iOS By MoltenVK Being Open-Sourced, Vulkan SDK for Mac. Phoronix. https://web.archive.org/web/20191021124123/https://www.phoronix.com/scan.php?page=article&item=vulkan-on-mac&num=1. 21.7.2019.

Lee, E. 2019. “SDL2# - C# Wrapper for SDL2”. GitHub. https://web.archive.org/web/20191108103230/https://github.com/flibitijibibo/SDL2-CS .7.11.2019.

Lowe, S. 2016. GDC: Microsoft to Debut DirectX 12 on March 20. Imagine Games Network. https://web.archive.org/web/20191022130622/https://www.ign.com/articles/2014/03/06/gdc-microsoft-to-debut-directx-12. 4.6.2019.

Lu, J. 2019. ” World of Warcraft uses DirectX 12 running on Windows 7”. DirectX Developer Blog. https://web.archive.org/web/20190828125709/https://devblogs.microsoft.com/directx/world-of-warcraft-uses-directx-12-running-on-windows-7/. 9.7.2019.

Martin, G. 2017. Monogame – Working with Touch. WordPress. https://web.archive.org/web/20191030103828/https://gregfmartin.com/2017/12/27/monogame-working-with-touch/. 29.10.2019.

Metz, C. 2014. Microsoft Open Sources .NET, Saying It Will Run on Linux and Mac. Wired. https://web.archive.org/web/20191028140339/https://www.wired.com/2014/11/microsoft-open-sources-net-says-will-run-linux-mac/ .28.10.2019.

Microsoft. 2004. ”Next Generation of Games Starts With XNA”. https://web.archive.org/web/20190828120654/https://news.microsoft.com/2004/03/24/microsoft-next-generation-of-games-starts-with-xna/ .23.8.2019.

Microsoft. 2006a. ”Microsoft Invites the World to Create Its Own Xbox 360 Console Games for the First Time”. https://web.archive.org/web/20060820071806/http:/www.microsoft.com/presspass/press/2006/aug06/08-13XNAGameStudioPR.mspx .23.8.2019.

Microsoft. 2006b. “Microsoft Releases XNA Game Studio Express and Launches XNA Creators Club, Enabling the World to Play Their Creations on Xbox 360”. https://web.archive.org/web/20190828110314/https://news.microsoft.com/2006/12/11/microsoft-releases-xna-game-studio-express-and-launches-xna-creators-club-enabling-the-world-to-play-their-creations-on-xbox-360/. 23.8.2019.

Microsoft. 2007. “Microsoft Unlocks Game Development and Encourages Everyone to “Dream-Build-Play”. https://web.archive.org/web/20190828110117/https://news.microsoft.com/2007/03/04/microsoft-unlocks-game-development-and-encourages-everyone-to-dream-build-play/. 23.8.2019.

Microsoft. 2010a. GraphicsDevice Constructor. https://web.archive.org/web/20191022111145/https://docs.microsoft.com/en-us/previous-versions/windows/xna/bb195941(v=xnagamestudio.10). 7.9.2019.

Microsoft. 2010b. TimeSpan.Days Property. https://web.archive.org/web/20191022111307/https://docs.microsoft.com/en-us/dotnet/api/system.timespan.days?view=netframework-4.8. 7.9.2019.

Microsoft. 2010c. Displays, Client Bounds, Viewports, and Back Buffers. https://web.archive.org/web/20191108104118/https://docs.microsoft.com/en-us/previous-versions/windows/xna/bb203889(v=xnagamestudio.10). 7.11.2019

Microsoft. 2011a. RasterizerState Class. https://web.archive.org/web/20191022110258/https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/ff434386(v=xnagamestudio.35). 5.9.2019.

Microsoft. 2011b. ”Game.IsActive Property”. https://web.archive.org/web/20191108103906/https://docs.microsoft.com/en-us/previous-versions/windows/xna/bb197664(v=xnagamestudio.41). 7.11.2019.

Microsoft. 2015a. NET Framework Class Library. https://web.archive.org/web/20190828131839/https://docs.microsoft.com/en-us/previous-versions/gg145045(v=vs.110).12.8.2019.

Microsoft. 2015b. Namespace (C# Reference). https://web.archive.org/web/20191108104927/https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/namespace. 7.11.2019.

Microsoft. 2016a. “What is managed code?”. https://web.archive.org/web/20190828131226/https://docs.microsoft.com/en-us/dotnet/standard/managed-code.12.8.2019.

Microsoft. 2016b. Common Type System & Common Language Specification. https://web.archive.org/web/20190828131406/https://docs.microsoft.com/en-us/dotnet/standard/common-type-system.12.8.2019.

Microsoft. 2017. Secure Coding Guidelines for Unmanaged Code. https://web.archive.org/web/20190828132115/https://docs.microsoft.com/en-us/dotnet/framework/security/secure-coding-guidelines-for-unmanaged-code .12.8.2019.

Microsoft. 2018a. XAudio2 Introduction. https://web.archive.org/web/20191021133311/https://docs.microsoft.com/fi-fi/windows/win32/xaudio2/xaudio2-introduction. 14.8.2019.

Microsoft. 2018b. About Media Foundation. https://web.archive.org/web/20191021134105/https://docs.microsoft.com/en-us/windows/win32/medfound/about-the-media-foundation-sdk.15.8.2019.

Microsoft. 2018c. Getting Started With XInput. https://web.archive.org/web/20191022115837/https://docs.microsoft.com/en-us/windows/win32/xinput/getting-started-with-xinput. 22.8.2019.

Microsoft. 2018d. ”What's a Universal Windows Platform (UWP) app?”. https://web.archive.org/web/20191022115933/https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide. 22.8.2019.

Microsoft. 2018e. An overview of Windows 10 IoT. https://web.archive.org/web/20191022120031/https://docs.microsoft.com/fi-fi/windows/iot-core/windows-iot. 22.8.2019.

Microsoft. 2018f. Testing on the Xbox One console. https://web.archive.org/web/20191022120123/https://docs.microsoft.com/en-us/gaming/xbox-live/testing-on-console .22.8.2019.

Microsoft. 2018g. Fundamentals of garbage collection. https://web.archive.org/web/20190828131015/https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals. 12.8.2019

Microsoft. 2018h. Compiling and building in Visual Studio for Mac. https://web.archive.org/web/20190123141726/https://docs.microsoft.com/en-us/visualstudio/mac/compiling-and-building.14.9.2019

Microsoft. 2018i. Introduction to Textures in Direct3D 11. https://web.archive.org/web/20191108104332/https://docs.microsoft.com/en-us/windows/win32/direct3d11/overviews-direct3d-11-resources-textures-intro. 7.11.2019.

Microsoft. 2019a. Built-in reference types (C# reference). https://web.archive.org/web/20191022120332/https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types. 22.8.2019.

Microsoft. 2019b. ”Silverlightin tuen päättyminen”. https://support.microsoft.com/fi-fi/help/4511036/silverlight-end-of-support.17.7.2019.

Microsoft. 2019c. “What is ASP.NET?”. https://web.archive.org/web/20190828131629/https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet.12.8.2019

Microsoft. 2019d. "Certificate could not be opened: ***.pfx. The specified network password is not correct. The potential bug of VS2019 V16.2.2". https://web.archive.org/web/20191030104503/https://developercommunity.visualstudio.com/content/problem/695055/certificate-could-not-be-opened-pfx-the-specified.html .29.10.2019.

Moghadampour, G. 2011. C# Windows- ja tietokantaohjelmointi. Jyväskylä: Docendo.

MonoDevelop Project. 2019. Download. http://web.archive.org/web/20191106134701/https://www.monodevelop.com/download/. 6.11.2019.

Monogame Team. 2019a. Microsoft Public License (Ms-PL). https://web.archive.org/web/20191021122116/https://github.com/MonoGame/MonoGame/blob/develop/LICENSE.txt.2.7.2019.

Monogame Team. 2019b. ”Gamepad.SDL.cs”. https://web.archive.org/web/20191022104555/https://github.com/labnation/MonoGame/blob/master/MonoGame.Framework/Input/GamePad.SDL.cs. 4.10.2019.

Monogame Team. 2019c. Monogame – About. https://web.archive.org/web/20190123124217/http://www.monogame.net/about.17.10.2018.

Monogame Team. 2019d. Android. https://web.archive.org/web/20191030103111/http://www.monogame.net/documentation/?page=Android. 29.10.2019.

Monogame Team. 2019e. Android apps on Linux. http://community.monogame.net/t/android-apps-on-linux/7689 . 29.10.2019.

Monogame Team. 2019f. “SpriteBatch.Begin Method”. https://web.archive.org/web/20191030103408/http://www.monogame.net/documentation/?page=M_Microsoft_Xna_Framework_Graphics_SpriteBatch_Begin. 29.10.2019.

Monogame Team. 2019g. Building Directory contains space. http://community.monogame.net/t/building-directory-contains-space/11230. 6.11.2019.

Monogame Team. 2019h. ”Game.cs”. GitHub. https://web.archive.org/web/20191108103603/https://github.com/MonoGame/MonoGame/blob/develop/MonoGame.Framework/Game.cs. 7.11.2019.

Monogame Team. 2019i. VideoPlayer Class. https://web.archive.org/web/20191108103714/http://www.monogame.net/documentation/?page=T_Microsoft_Xna_Framework_Media_VideoPlayer. 7.11.2019.

MoltenGL. 2019. ”Protect your investment in OpenGL ES on iOS and macOS!”. https://web.archive.org/web/20191021125739/https://moltengl.com/moltengl/. 21.7.2019.

MoltenGL. 2016. MoltenVK brings Vulkan to iOS and macOS. https://web.archive.org/web/20191021125933/https://moltengl.com/press/moltenvk-brings-vulkan-to-ios-and-macos/. 21.7.2019.

Mono -project. 2018b. NET Source Code Integration. Mono-project.com. https://web.archive.org/web/20190123135825/https://www.mono-project.com/docs/about-mono/dotnet-integration/.18.1.2018

Mono -project. 2019. Download. http://web.archive.org/web/20191106134418/https://www.mono-project.com/download/stable/. 6.11.2019.

MonoXNA. 2009. ”Games using MonoXNA”. https://web.archive.org/web/20121029164308/http://www.monoxna.org/GamesUsingMonoxna.17.7.2019 *Sivuston ”MonoXNA” alkuperäinen toiminta on päättynyt vuonna 2013, tämän jälkeen domain on ollut joko tyhjillään tai toisen tahon omistuksessa. Vuosiluku määritelty sivulla mainitun Youtube -videon latauspäivän mukaan.

Net Foundation.2019. Roslyn. Microsoft & Gitlab https://web.archive.org/web/20190123130237/https://github.com/dotnet/roslyn. 20.12.2018

”Nixxes_Official”. 2016. Rise of the Tomb Raider – “When Is Windows Store Version Gonna Get Patched?”. Valve. https://web.archive.org/web/20191022132029/https://steamcommunity.com/app/391220/discussions/0/405690850608756638/. 8.7.2019.

Oracle. 2019. keytool - Key and Certificate Management Tool. https://web.archive.org/web/20191030104247/https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html. 29.10.2019.

Orland, K. 2010. “Microsoft Rebrands XNA Creators Club as 'App Hub,' Adds Windows Phone 7 Support”. Gamasutra. https://web.archive.org/save/https://www.gamasutra.com/view/news/121660/Microsoft_Rebrands_XNA_Creators_Club_as_App_Hub_Adds_Windows_Phone_7_Support.php. 22.8.2019

Parsons, A. 2017. “Calling all game devs: The Dream. Build. Play 2017 Challenge is Here!”. Microsoft Blogs. 12.7.2017 https://web.archive.org/web/20190828084215/https://blogs.windows.com/windowsdeveloper/2017/07/12/calling-game-devs-dream-build-play-2017-challenge/ .26.8.2019.

Paoli, J. 2015. Open source at Microsoft: The next chapter for Microsoft Open Technologies. Microsoft Open Technologies. https://web.archive.org/web/20160327155723/https://msopentech.com/blog/2015/04/17/nextchapter/. 28.10.2019

Payne, P. 2019. Texture MipMaps. Tafe NSW. https://web.archive.org/web/20191108104518/https://relativity.net.au/gaming/java/TextureMipmap.html. 7.11.2019.

”Pelaaja Lehti”. 2006. “Sinustako pelintekijä?”. Pelaaja -lehti. https://web.archive.org/save/https://pelaaja.fi/uutiset/sinustako-pelintekija?page=. 22.8.2019.

Phil Spencer (XboxP3). 2016. "@BigMouthGamer We know lists like this include features PC gamers want to see from us, we appreciate the feedback and have plans to improve". Twiitti. 8.7.2017. https://web.archive.org/web/20191022132851/https:/twitter.com/XboxP3/status/703321279999275008

Pocket Watch Games, LLC. 2017. Tooth And Tail - F.A.Q. Pocket Watch Games, LLC /Valve. https://web.archive.org/web/20191021112657/https://steamcommunity.com/app/286000/discussions/0/1458455461478916710/. 15.6.2019.

Power, K. 2013. Blending in XNA. Carlow - Institute of Technology. https://web.archive.org/web/20191022105801/http://glasnost.itcarlow.ie/~powerk/technology/xna/blending/blending.html. 5.9.2019.

Promit, R. 2013. ”DirectX/XNA Phase Out Continues”. ” Promit's Ventspace” https://web.archive.org/web/20190828123151/https://ventspace.wordpress.com/2013/01/30/directxxna-phase-out-continues/. 24.8.2019.

Protalinski, E. 2014. Microsoft starts to open source .NET and take it cross-platform to Mac, Linux. VentureBeat. https://web.archive.org/web/20191028141457/https://venturebeat.com/2014/11/12/microsoft-starts-to-open-source-net-and-take-it-cross-platform-to-mac-linux/ .28.10.2019

RedHat, Inc. 2019. ”What is middleware?”. https://web.archive.org/web/20191021125221/https://www.redhat.com/en/topics/middleware/what-is-middleware. 23.1.2019.

Reiss, B. 2019. ”SilverSprite”. Codeplex Archive. https://web.archive.org/web/20190828105039/https://archive.codeplex.com/?p=silversprite.17.7.2019.

Rouse M. 2015. Framework. WhatIs. https://web.archive.org/web/20190123124617/https://whatis.techtarget.com/definition/framework.17.10.2018

Sam, S. 2019. ”What is Type safe in C#?”. Tutorials Point. https://web.archive.org/web/20191021130957/https://www.tutorialspoint.com/What-is-Type-safe-in-Chash. 3.8.2019.

Scriptol. 2014. CIL - Common Intermediate Language.https://web.archive.org/web/20191021130109/https://www.scriptol.com/programming/cil.php. 2.8.2019.

SDL Project. 2019. About SDL. https://web.archive.org/web/20191108103415/https://www.libsdl.org/index.php. 7.11.2019.

Secret Police, The. 2018. “Stardew Valley Comes to Mobile!”. https://web.archive.org/web/20190828134851/https://www.thesecretpolice.org/single-post/2018/10/11/Stardew-Valley-Comes-to-Mobile .15.8.2019

Segal,M & Akeley,K, 2004. The OpenGL Graphics System: A Specification. Silicon Graphics, Inc. https://web.archive.org/web/20190123131732/https://www.khronos.org/registry/OpenGL/specs/gl/glspec20.pdf. 17.1.2018.

Sellers, G., Wright, R.S. & Haemel, N. 2016. OpenGL SuperBible: Comprehensive Tutorial and Reference, Seventh Edition. New Jersey: Addison -Wesley Educational Publishers Inc

Sickhead Games. 2018. ”What is BRUTE?”. https://web.archive.org/web/20191108114933/http://brute.rocks/. 8.11.2019.

Simple DirectMedia Layer Wiki. 2019. SDL_GLprofile. https://web.archive.org/web/20191031122503/https://wiki.libsdl.org/SDL_GLprofile.31.10.2019.

Spencer, P. 2019. Our Approach to PC Gaming. Microsoft. https://web.archive.org/web/20191021140505/https://news.xbox.com/en-us/2019/05/30/microsoft-approach-to-pc-gaming/. 3.10.2019.

Spilman, T. 2013. ”2 MonoGame titles coming to PS4”. Monogame. https://web.archive.org/web/20190828113435/http://www.monogame.net/2013/06/11/2-monogame-titles-coming-to-ps4/. 5.8.2019.

Spilman, T. 2019. ”Monogame for Nintendo Switch. Monogame Team. http://community.monogame.net/t/monogame-for-nintendo-switch/9841. 8.11.2019.

Stack Overflow. 2017. ”Monodevelop will not start in Mint 17”. http://web.archive.org/web/20191004094954/https://stackoverflow.com/questions/25791149/monodevelop-will-not-start-in-mint-17. 6.11.2019.

”Steam Community”. 2017. Rise of the Tomb Raider - Windows Store or Steam version. Valve. https://web.archive.org/web/20191021140607/https://steamcommunity.com/app/391220/discussions/0/1488866180599000231/. 3.10.2019.

Sydeek, R. 2017. ”What Is Middleware & How Does It Work?”. Feedough. https://web.archive.org/web/20191021122916/https://www.feedough.com/what-is-middleware-how-does-it-work/. 3.7.2019.

Tamminen, T. 2016. OpenGL:n korvaajan ja DirectX 12:n haastajan ensimmäinen versio julkaistiin. Mikrobitti. https://www.mikrobitti.fi/uutiset/opengln-korvaajan-ja-directx-12n-haastajan-ensimmainen-versio-julkaistiin. 9.9.2019.

Techopedia.2018. Application Framework. Techopedia.com https://web.archive.org/save/https://www.techopedia.com/definition/6005/application-framework. 17.10.18

Templeman, J & Olsen, A. 2003. Microsoft Visual C++ .NET Step by step. Washington: Microsoft Press

Todorov, T. 2013. Understanding .NET Just-In-Time Compilation. Telerik. https://web.archive.org/web/20191021131411/https://www.telerik.com/blogs/understanding-net-just-in-time-compilation. 4.8.2019.

Tom Spilman (tomSpilman). 2017. " MonoGame is official middleware for Nintendo Switch. You can either port yourself for free or consider contracting us @Sickhead to do it. ". Twiitti.1.10.2019. https://web.archive.org/web/20191021135110/https://twitter.com/tomspilman/status/909843718383001601

Tutorial Point. 2019. ”C# - Data Types”. https://web.archive.org/web/20191021130914/https://www.tutorialspoint.com/csharp/csharp_data_types. 3.8.2019.

Tom Spilman (Tomspilman). 2016 “Move away from XNA?”. Monogame Team. https://web.archive.org/web/20190123124401/https://github.com/MonoGame/MonoGame/issues/4984 .14.1.2019

Unity Technologies. 2019. Creating and Using Scripts. Unity Technologies.https://web.archive.org/web/20191021132611/https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. 7.8.2019.

Unfold Games. 2019. Why I turned down exclusivity deal from the Epic Store (developer of “DARQ”). Unfold Games LLC. https://web.archive.org/web/20191021135938/https://medium.com/@info_68117/why-i-turned-down-exclusivity-deal-from-the-epic-store-developer-of-darq-7ee834ed0ac7. 3.11.2019.

Valve. 2018*. Free and Open-Source Vulkan on macOS and iOS. https://web.archive.org/web/20191021124426/https://store.steampowered.com/news/37575/. 21.7.2019. *Uutisessa ei ole ilmoitettuna päiväystä, joten se on otettu uutisen sisältämän videon latauspäivästä.

Vella, C. 2019. ”tIDE: Tilemap Integrated Development Environment”. GitHub. https://web.archive.org/web/20190828114524/https://colinvella.github.io/tIDE/.26.8.2019

Walton, M. 2016. Epic CEO: “Universal Windows Platform can, should, must, and will die”. Ars Technica. https://web.archive.org/web/20191022130324/https://arstechnica.com/gaming/2016/03/tim-sweeney-to-microsoft-universal-windows-platform-can-should-must-and-will-die/. 8.7.2019.

Ward, J. 2008.What is game engine. https://web.archive.org/web/20190125002401/http://www.gamecareerguide.com/features/529/what_is_a_game_.php?page=1. 23.1.2019

WawRo, A. 2014. What exactly goes into porting a video game? BlitWorks explains. GamaSutra, InformaTech. https://web.archive.org/web/20191021114532/https://www.gamasutra.com/view/news/222363/What_exactly_goes_into_porting_a_video_game_BlitWorks_explains.php. 16.6.2019

Whitaker, R. 2019. Using XACT. RB Whitaker's Wiki. https://web.archive.org/web/20191022105423/http://rbwhitaker.wikidot.com/using-xact. 4.9.2019.

Parade -sovelluksen lähdekoodi (DirectX)

Ihanteellisempi versio: https://gitlab.com/Joonasn/parade--game

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Audio;
using System;
using System.Collections.Generic;
using System.IO;
using System.Xml;
using Microsoft.Xna.Framework.Media;

namespace ParaatiOpen
{
    public class Game1 : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch, SBParallax;
        SpriteFont Font;
        KeyboardState state, pauseState, AudioState;
        Song inGameTune;
        SoundEffect enemySFX1, enemySFX2;
        TitleScreen Title;
        ScrollingTexture Scroll;
        HUDScore HUD;
        Player Character;
        List<ParadeFolk> ParadeList = new List<ParadeFolk>();
        List<ParadeFolk> ParadeClones = new List<ParadeFolk>();
        List<Items> Items = new List<Items>();
        List<Items> ItemClones = new List<Items>();
        List<PlayersWand> ThrowWandClones = new List<PlayersWand>();
        Torpedo BikerGuy;
        PlayersWand WandTemplate;
        Texture2D[] ParadeImage = new Texture2D[8];
        Texture2D[] ItemImage = new Texture2D[5];
        XmlDocument SaveFile = new XmlDocument();
        Random EnemySelector = new Random();
        Random ItemSelector = new Random();

        const int ScreenWidth = 1200, ScreenHeight = 800; // Adjust resolution
        int GameSpeed=5, CurrentLevelMax, CurrentLevel=1, CurrentLevelReset=1,
        SpeedRise, SpeedRiseDelay = 15, Score = 0, HiScore = 7000,
        WalkingScoreDelay = 0, resetTimer = 0;

        bool gameover = false, pause = false, AcceptExit = false,
        GameMode = false, intro = false, ShowHighScore = false,
        audioSetup = true;

        float ReleaseTimer = 0, ReleaseDelay = 1000f;
        int[] EnemyWeight, WeightStack;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            graphics.PreferredBackBufferWidth = ScreenWidth;
            graphics.PreferredBackBufferHeight = ScreenHeight;
            Content.RootDirectory = "Content";
        }
```

```
        protected override void Initialize()
        {
            TitleScreen.Content = Content;
            ScrollingTexture.Content = Content;
            HUDScore.Content = Content;
            Player.Content = Content;
            MediaPlayer.IsRepeating = true;

            base.Initialize();

            if (File.Exists("SharpDXV.xml"))
            {
                SaveFile.Load("SharpDXV.xml");
                Score = Convert.ToInt32(SaveFile.SelectSingleNode
                ("Pisteet/Nykyinen").InnerText);
                HiScore = Convert.ToInt32(SaveFile.SelectSingleNode
                ("Pisteet/Huipputulos").InnerText);
            }
            else
            {
                Score = 0;
                HiScore = 7000;
            }
        }

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
            SBParallax = new SpriteBatch(GraphicsDevice);
            Font = Content.Load<SpriteFont>("fontti");
            Title = new TitleScreen(Font, ScreenWidth, ScreenHeight);
            HUD = new HUDScore(Font, ScreenWidth, ScreenHeight);
            Scroll = new ScrollingTexture(GraphicsDevice);
            inGameTune = Content.Load<Song>("musa");
            Character = new Player(GraphicsDevice);
            enemySFX1 = Content.Load<SoundEffect>("aani4");
            enemySFX2 = Content.Load<SoundEffect>("aani5");

            for (int i = 0; i < ParadeImage.Length; i++)
            {
                ParadeImage[i] = Content.Load<Texture2D>("enemy" + i);
            }

            for (int i = 0; i < ItemImage.Length; i++)
            {
                ItemImage[i] = Content.Load<Texture2D>("tavara" + i);
                Items.Add(new Items(ItemImage[i], 1, 0, 2));
            }
            // Normal enemy (0) Weight = 6
            ParadeList.Add(new ParadeFolk(ParadeImage[0], 3, 50, 6));
            // "Row" enemy (1) Weight = 5
            ParadeList.Add(new RowFolk(ParadeImage[1], 3, 50, 6));
            // Tuba enemy (2) Weight = 4
            ParadeList.Add(
            new TubaPlayer(ParadeImage[3], 1, 300, 7)
            {
                // Collectable Tuba instrument
                TubaInstru = new Items(ItemImage[4], 1, 0, 2),
            });
            ParadeList.Add(
            new Squirrel(ParadeImage[2], 4, 50, 10) // Squirrel (3) Weight = 3
            {
                SquirrelsWand = new Wand(ParadeImage[7], 4, 0, 5) // Wand
                {
                    // Collectable Wand
                    WandCollect = new Items(ItemImage[3], 1, 0, 2),
```

```
            }
        });
        // Floating Ghost (4) Weight = 2
        ParadeList.Add(new FloatyGhost(ParadeImage[6], 5, 50, 3));
        //Disappearing Ghost (5) Weight = 1
        ParadeList.Add(new HidingGhost(ParadeImage[6], 3, 150, 3));
        // Agent (6) Weight = 0

        ParadeList.Add(new Agent(ParadeImage[4], 2, 500, 3));
        /////// Torpedo
        BikerGuy = new Torpedo(ParadeImage[5], 5, 100, 3,
        enemySFX2, GraphicsDevice);
        // UsableWand
        WandTemplate = new PlayersWand(ParadeImage[7],
        10, 200, 5, enemySFX1);
        // Enemyweight = 7 (for "Weighted Random Selector")
        EnemyWeight = new int [ParadeList.Count];
        for (int i=0; i < ParadeList.Count; i++)
        {
            // Normal = 7.. Row = 6 .. Tuba = 5 and so on...

            EnemyWeight[i] = ParadeList.Count - i;
        }
        WeightStack = new int [ParadeList.Count]; // WeightStack = 6 (Array)
        WeightStack[0] = EnemyWeight[0];   // Setup; WeightStack[0] = 7
        // Start counting from column 1 (Row enemy)
        for (int i = 1; i < WeightStack.Length; i++)
        {
            // WeightStack[1] = 7 + 6 (12) <-- Stack for Row enemy
            WeightStack[i] = WeightStack[i - 1] + EnemyWeight[i];
        }
        // Since there's no column 7, maximun "CurrentLevel" value is 6
        CurrentLevelMax = (WeightStack.Length -1);
    }

    protected override void UnloadContent()
    {

    }

    protected void ResetValues() // Reset values when starting a new game
    {
        CurrentLevel = CurrentLevelReset;
        Score = 0;
        SpeedRise = 0;
        resetTimer = 0;
        gameover = false;
        pause = false;
        AcceptExit = false;
        ShowHighScore = false;
        pauseState = state;
        HUD.ResetHUD();
        Character.ResetPlayer(GraphicsDevice);
        ParadeClones.Clear();
        ItemClones.Clear();
    }

    protected void inGameReset() // If player uses clock, these values get a reset
    {
        CurrentLevel = CurrentLevelReset;
        SpeedRise = 0;
        resetTimer = 0;
        Character.ClockUsed = false;
    }
```

```
        // Save current "Score" and "HiScore" values to XML -file
        protected void SaveScores()
        {
            if (File.Exists("SharpDXV.xml")) // If file "SharpDXV.xml" is found, modify it
            {
                XmlNodeList piste1 = SaveFile.SelectNodes("//Pisteet/Nykyinen");
                XmlNodeList piste2 = SaveFile.SelectNodes("//Pisteet/Huipputulos");

                foreach (XmlNode CurrentScoreNmb in piste1)
                {
                    CurrentScoreNmb.InnerText = Score.ToString();
                }

                foreach (XmlNode HiScoreNmb in piste2)
                {
                    HiScoreNmb.InnerText = HiScore.ToString();
                }
                SaveFile.Save("SharpDXV.Xml");
            }
            else // If file "SharpDXV.xml" is missing create new one
            {
                XmlWriter xmlWriter = XmlWriter.Create("SharpDXV.xml");

                xmlWriter.WriteStartDocument();
                xmlWriter.WriteStartElement("Pisteet");

                xmlWriter.WriteStartElement("Nykyinen");
                xmlWriter.WriteString(Score.ToString());
                xmlWriter.WriteEndElement();

                xmlWriter.WriteStartElement("Huipputulos");
                xmlWriter.WriteString(HiScore.ToString());

                xmlWriter.WriteEndDocument();
                xmlWriter.Close();
            }
        }

        protected void HandleGameSpeed(GameTime gameTime) // Actions related to gamespeed
        {
            ReleaseDelay = (750f -(float)(CurrentLevel*10));
            GameSpeed = (4 + CurrentLevel);

            ReleaseTimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
            if (ReleaseTimer > ReleaseDelay) // Release enemies
            {
                SpeedRise++;
                if (SpeedRise == 10)
                {
                    UnlockItems();
                }
                ReleaseTimer = 0;
                UnlockParade();
            }

            if (SpeedRise > (SpeedRiseDelay + GameSpeed))
            {
                if (CurrentLevel < CurrentLevelMax)
                {
                    CurrentLevel++;
                }
                SpeedRise = 0;
            }
```

```
            if (Character.ClockUsed)
            {
                inGameReset();
            }

            if (CurrentLevel > CurrentLevelMax)
            {
                CurrentLevel = CurrentLevelMax;
                CurrentLevelReset = CurrentLevelMax;
            }
        }
        protected void HandleScore()
        {
            if (Score > HiScore)
            {
                if (!ShowHighScore)
                {
                    ShowHighScore = true;
                }
                HiScore = Score;
            }

            if (Score <= 9999999)
            {
                if (Character.AllowScore)
                {
                   WalkingScoreDelay++;
                }
                else
                {
                    WalkingScoreDelay = 0;
                }

                if (WalkingScoreDelay > (20 + GameSpeed))
                {
                    Score++;
                    WalkingScoreDelay = 0;
                }

                for (int i = 0; i < ParadeClones.Count; i++)
                {
                    if (ParadeClones[i].ReturnScore)
                    {
                        Score += ParadeClones[i].returnPoints();
                    }
                }

                for (int i = 0; i < ThrowWandClones.Count; i++)
                {
                   if (ThrowWandClones[i].ReturnScore)
                   {
                       Score += ThrowWandClones[i].returnPoints();
                   }
               }

               if (BikerGuy.ReturnScore)
               {
                   Score += BikerGuy.returnPoints();
               }
            }
            else
            {
                Score = 9999999;
            }
        }
```

```
protected void HandleSubClasses() // Actions for subclasses, mostly from player
{
    for (int i = 0; i < ItemClones.Count; i++)
    {
        // Collision with playable character IF CLOAK IS OFF
        if (Character.invisibleCount == 0 && Character.PlayerSize.
        Intersects(ItemClones[i].personSize))
        {
            Character.ItemCollision(ItemClones[i].ItemValue);
             ItemClones[i].AllowRemove = true;
         }
    }

    if (Character.ThrowWand)  // Character is preparing to throw a wand
    {
        Character.ThrowWand = false;
        WandTemplate.AddWandCopy(ThrowWandClones, WandTemplate,
        Character.PlayerCollision.Y, Character.PlayerCollision.X - 30);
    }

    if (Character.TubaActive) // Character is preparing to play tuba
    {
        // Onscreen enemies gets damaged
        for (int i = 0; i < ParadeClones.Count; i++)
        {
            ParadeClones[i].isDamageTaken = true;
        }
        Character.TubaActive = false;
    }

    for (int i = 0; i < ParadeClones.Count; i++) // Enemy Collision
    {
        // Player and Parade collision
        if (Character.invisibleCount == 0 && Character.PlayerCollision.
        Intersects(ParadeClones[i].personCollision))
        {
            if (!ParadeClones[i].isDamageTaken)
            {
                Character.EnemyCollision();
            }
        }
        // Parade and Wand Collision
        for (int j = 0; j < ThrowWandClones.Count; j++)
        {
            if (ParadeClones[i].personSize.
            Intersects(ThrowWandClones[j].personSize))
            {
                ParadeClones[i].isDamageTaken = true;
                ThrowWandClones[j].PlaySFX();
                ThrowWandClones[j].ReturnScore = true;
            }
        }
    }

    if (Character.invisibleCount == 0 && Character.PlayerCollision.Intersects
    (BikerGuy.personCollision)) // Collision on Biker character
    {
        if (!BikerGuy.isDamageTaken)
        {
            Character.EnemyCollision();
        }
    }
}
```

```
        protected void UnlockParade()
        {
            int WeightCheck = 0, NextEnemy=0;
            WeightCheck = EnemySelector.Next(0, WeightStack[CurrentLevel]) +1;
            NextEnemy = Array.BinarySearch(WeightStack, WeightCheck);

            if (NextEnemy > ParadeList.Count || NextEnemy < 0)
            {
                ParadeList[0].AddCopy(ParadeClones, ParadeList[0]);
            }
            else
            {
                ParadeList[NextEnemy].AddCopy(ParadeClones, ParadeList[NextEnemy]);
            }
        }

        protected void UnlockItems()
        {
            int NextItem = 0, SelectedItem = 0, nextPosition;
            NextItem = ItemSelector.Next(0, GameSpeed);

            if (NextItem % 2 == 0)
            {
                if (Score > 10000 && !Character.CoinCollected && CurrentLevel > 4)
                {
                    SelectedItem = 0; //Coin
                }
                else
                {
                    SelectedItem = 1; //Cloak
                }
            }
            else
            {
                if (CurrentLevel > 2 && Character.CurrentHolding == 1)
                {
                    SelectedItem = 2; //Clock
                }
                else
                {
                    SelectedItem = 1; //Cloak
                }
            }
            nextPosition = ItemSelector.Next(240, 650);

            if (Character.CurrentHolding != SelectedItem && SelectedItem <= Items.Count)
            {
                Items[SelectedItem].AddCopy(ItemClones, Items[SelectedItem], nextPosition,
               -100, SelectedItem);
            }
        }

        protected override void Update(GameTime gameTime)
        {
            state = Keyboard.GetState();
            if (state.IsKeyDown(Keys.Escape) && AcceptExit)
            {
                SaveScores();
                Exit();
            }
            // Update Scrolling if Pause, Gameover and Intro values are false
            if (!pause ^ gameover ^ intro)
            {
                Scroll.Update(GameSpeed);
            }
```

```
            //Allow Exit and Full Screen -toggle, if Gameover or Pause values are true
            if (gameover ^ pause | !GameMode)
            {
                AcceptExit = true;
                if (state.IsKeyDown(Keys.F))
                {
                    graphics.IsFullScreen = !graphics.IsFullScreen;
                    graphics.ApplyChanges();
                }
            }

            switch (GameMode)
            {
                case false: //Title screen
                    Title.Update(gameTime);
                    if (state.IsKeyDown(Keys.Enter))
                    {
                        ResetValues();
                        intro = true;
                        GameMode = true;
                    }
                    break;
                case true: // Game mode
                default:
                    MainLoop(gameTime);
                    break;
            }
            base.Update(gameTime);
        }

        protected void MainLoop (GameTime gameTime)
        {
            if (state.IsKeyDown(Keys.Enter) && pauseState.IsKeyUp(Keys.Enter) && !gameover)
            {
                pause = !pause; // Enter  = Pause toggle
            }
            pauseState = state;

            if (state.IsKeyDown(Keys.M) && AudioState.IsKeyUp(Keys.M))
            {
                audioSetup = !audioSetup;
            }
            AudioState = state;

            if (audioSetup)
            {
                MediaPlayer.Volume = 0.5f;
            }
            else
            {
                MediaPlayer.Volume = 0.0f;
            }

            if (!pause)
            {
                if (intro)
                {
                    if (resetTimer < 30)
                    {
                        resetTimer++;
                        Character.GameOverAnimation(true);
                    }
                    else
                    {
```

```
                if (Character.PlayerSize.X > (ScreenWidth/2))
                {
                    Character.IntroWalking(GameSpeed);
                }
                else
                {
                    MediaPlayer.Play(inGameTune);
                    intro = !intro;
                }
            }
        }
        else
        {
            for (int i = 0; i < ParadeClones.Count; i++) // Update enemy movement
            {
                ParadeClones[i].Update(GameSpeed, GraphicsDevice, ParadeClones,
                ItemClones, Character.PlayerCollision.X,
                Character.PlayerCollision.Y);
                if (ParadeClones[i].AllowRemove)
                {
                    ParadeClones.Remove(ParadeClones[i]);
                    i--;
                }
            }

            for (int i = 0; i < ThrowWandClones.Count; i++) // Update Wand movement
            {
                ThrowWandClones[i].Update(GameSpeed, GraphicsDevice);
                if (ThrowWandClones[i].AllowRemove)
                {
                    ThrowWandClones.Remove(ThrowWandClones[i]);
                    i--;
                }
            }
            // Update "Torpedo" movement
            BikerGuy.Update(GameSpeed, GraphicsDevice);

            if (!gameover)
            {
                AcceptExit = false;
                HandleGameSpeed(gameTime);
                HandleScore();
                HandleSubClasses();

                for (int i = 0; i < ItemClones.Count; i++) // Update item movement
                {
                    ItemClones[i].Update(GameSpeed, GraphicsDevice);
                    if (ItemClones[i].AllowRemove)
                    {
                        ItemClones.Remove(ItemClones[i]);
                        i--;
                    }
                }
                Character.UpdateMovement(state, GameSpeed, GraphicsDevice);
                BikerGuy.PlayerFound(Character.PlayerCollision.X,
                Character.PlayerCollision.Y);
                gameover = Character.OutFromScreen;
            }
            else // If Gameover
            {
                MediaPlayer.Stop();
                Character.GameOverAnimation(false);
                HUD.UpdateGameOver(gameTime);
```

```
                        if (state.IsKeyDown(Keys.Enter))
                        {
                            SaveScores();
                            ResetValues();
                            intro = true;
                        }
                    }
                }
            }
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.SlateGray); // Asphalt color
            spriteBatch.Begin(); // Spritebatch for drawing graphics

                SBParallax.Begin(SpriteSortMode.Deferred, null,
                SamplerState.LinearWrap, null, null);
                    // Seperate spritebatch for scrolling backgrounds
                    Scroll.Draw(SBParallax);
                SBParallax.End();

                HUD.DrawStats(spriteBatch, Score, HiScore, Character.
                HoldingText[Character.CurrentHolding]);

                switch (GameMode)
                {
                    case false:
                        Title.Draw(spriteBatch);
                        break;
                    case true:
                    default:
                        Character.Draw(spriteBatch);
                        BikerGuy.Draw(spriteBatch);

                        for (int i = 0; i < ItemClones.Count; i++)
                        {
                            ItemClones[i].Draw(spriteBatch);
                        }

                        for (int i = 0; i < ThrowWandClones.Count; i++)
                        {
                            ThrowWandClones[i].Draw(spriteBatch);
                        }

                        for (int i = 0; i < ParadeClones.Count; i++)
                        {
                            ParadeClones[i].Draw(spriteBatch);
                        }

                        if (pause && !gameover)
                        {
                            HUD.DrawPause(spriteBatch);
                        }

                        if (gameover)
                        {
                            HUD.DrawGameOver(spriteBatch, Score, HiScore, ShowHighScore);
                        }
                        break;
                }
            spriteBatch.End();
            base.Draw(gameTime);
        }
    }
}
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using System;

namespace ParaatiOpen
{
    class HUDScore
    {
        protected SpriteFont Font;
        protected Texture2D GameOverImage;
        protected Rectangle GameOverImageLocation, GameOverAnimation;
        protected Color LerpingColors;
        protected Color lerp1 = new Color(255, 255, 255);
        protected Color lerp2 = new Color(255, 255, 255);
        protected Random ColorChange = new Random();
        protected int red, green, blue, GameoverDelay = 0, GameOverFrame = 0,
                      FrameDivider = 2;
        protected float GameoverTimer = 0, TimerDelay = 200;
        protected string[] PauseText = new string[] { "-PAUSE-", "WASD/Arrow-keys = Walk",
        "Space = Use item", "F = Toggle screen size","", "M = Toggle Audio",
               "ESC = End game" };
        protected string[] ScoreText = new string[] { "", "", "HI-SCORE!!!",
               "Hit enter to try again!"};
        protected Vector2 AdjustedTextLocation;
        protected Vector2[,] AdjustedTextYX;
        protected Vector2 HUDLocation;

        private static ContentManager content;
        public static ContentManager Content
        {
            protected get { return content; }
            set { content = value; }
        }

        public HUDScore(SpriteFont font, int SW, int SH) // Setup
        {
            Font = font;
            GameOverImage = Content.Load<Texture2D>("gameover");
            GameOverImageLocation = new Rectangle(((SW - 600) / 2),
                                    ((SH - 120) / 2) - 240, 587, 118);
            GameOverAnimation = GameOverImageLocation;
            AdjustedTextYX = new Vector2[PauseText.Length, PauseText.Length];
            AdjustedTextLocation = new Vector2((SW / 2), (SH / 2) - 150);
            HUDLocation = new Vector2((SW / 2) -230, (SH / 2) - 390);
            AdjustText(PauseText, AdjustedTextYX);
        }
        // Center text based on text length
        protected void AdjustText(string[] Text, Vector2[,] Location)
        {
            for (int i = 0; i < Text.Length; i++)
            {
                Location[1, i] = Font.MeasureString(Text[i]);
                Location[0, i] = new Vector2(Font.MeasureString(Text[i]).X / 2,
                (Location[1, i].Y / 2) - 50 * i);
            }
        }

        public void ResetHUD() // Reset values and adjust text for pause screen
        {
            GameoverTimer = 0;
            GameoverDelay = 0;
            // Overwrites text location values of "Gameover screen".
            AdjustText(PauseText, AdjustedTextYX);
        }
```

```
protected void ColorLerpOn() //Color lerping for "HI-Score" and "Items" texts.
{
    red = ColorChange.Next(256);  // Select random color from values 0-255
    green = ColorChange.Next(256);
    blue = ColorChange.Next(256);
    lerp1 = new Color(red, green, blue); // Create color from random values
    LerpingColors = Color.Lerp(lerp1, lerp2, 0.5f); // Create color lerp
}

public void UpdateGameOver(GameTime gameTime) // Gameover, Draw stuff with delay
{
    ColorLerpOn();
    GameoverTimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;

    if (GameoverTimer >= TimerDelay)
    {
        // Change frames
        GameOverAnimation = new Rectangle(587 * GameOverFrame, 0, 587, 118);
        GameOverFrame = (GameOverFrame + 1) % FrameDivider;

        if (GameoverDelay <2)
        {
            GameoverDelay++;
        }
        GameoverTimer = 0;
    }
}

public void DrawStats(SpriteBatch spriteBatch, int Score, int HiScore,
        string HoldText) //Ingame "score -bar"
{
    ColorLerpOn();
    spriteBatch.DrawString(Font, "P: " + Score.ToString("0000000"),
          new Vector2(HUDLocation.X, HUDLocation.Y + 13), Color.Black);
    spriteBatch.DrawString(Font, "P: " + Score.ToString("0000000"),
          new Vector2(HUDLocation.X, HUDLocation.Y + 10), Color.White);
    spriteBatch.DrawString(Font, HoldText, new Vector2(HUDLocation.X +
          200, HUDLocation.Y + 13), Color.Black);
    spriteBatch.DrawString(Font, HoldText, new Vector2(HUDLocation.X +
          200, HUDLocation.Y + 10), LerpingColors);
    spriteBatch.DrawString(Font, "--------", new Vector2(HUDLocation.X +
          200, HUDLocation.Y + 33), Color.Black);
    spriteBatch.DrawString(Font, "--------", new Vector2(HUDLocation.X +
          200, HUDLocation.Y + 30), Color.White);
    spriteBatch.DrawString(Font, "H: " + HiScore.ToString("0000000"),
          new Vector2(HUDLocation.X + 330, HUDLocation.Y + 13), Color.Black);
    spriteBatch.DrawString(Font, "H: " + HiScore.ToString("0000000"),
          new Vector2(HUDLocation.X + 330, HUDLocation.Y + 10), Color.White);
}

public void DrawPause(SpriteBatch spriteBatch) // Pause text
{
    for (int i= 0; i < PauseText.Length; i++)
    {
        spriteBatch.DrawString(Font, PauseText[i], new
          Vector2((int)AdjustedTextLocation.X - AdjustedTextYX[0, i].X+3,
          (int)AdjustedTextLocation.Y - AdjustedTextYX[0, i].Y+3), Color.Black);
        spriteBatch.DrawString(Font, PauseText[i], new
          Vector2((int)AdjustedTextLocation.X - AdjustedTextYX[0, i].X,
          (int)AdjustedTextLocation.Y - AdjustedTextYX[0, i].Y), Color.White);
    }
}
```

```
        public void DrawGameOver(SpriteBatch spriteBatch, int Score, int HiScore,
        bool ShowHighScore) // Gameover text
        {
            if (GameoverDelay == 0)
            {
                ScoreText[1] = "Final score: " + Score;
                AdjustText(ScoreText, AdjustedTextYX);
            }

            if (GameoverDelay >= 1)
            {
                spriteBatch.Draw(GameOverImage, GameOverImageLocation,
                       GameOverAnimation, Color.White);
            }

            if (GameoverDelay >= 2)
            {
                for (int i = 0; i < ScoreText.Length; i++)
                {
                    if (i != 2)
                    {
                        spriteBatch.DrawString(Font, ScoreText[i], new
             Vector2((int)AdjustedTextLocation.X - AdjustedTextYX[0, i].X + 3,
             (int)AdjustedTextLocation.Y - AdjustedTextYX[0, i].Y + 3), Color.Black);
                        spriteBatch.DrawString(Font, ScoreText[i], new
             Vector2((int)AdjustedTextLocation.X - AdjustedTextYX[0, i].X,
             (int)AdjustedTextLocation.Y - AdjustedTextYX[0, i].Y), Color.White);
                    }
                    if (ShowHighScore)
                    {
                        spriteBatch.DrawString(Font, ScoreText[1], new
                            Vector2((int)AdjustedTextLocation.X - AdjustedTextYX[0, 1].X +
                            3, (int)AdjustedTextLocation.Y - AdjustedTextYX[0, 1].Y + 3),
                            Color.Black);
                        spriteBatch.DrawString(Font, ScoreText[1], new
                            Vector2((int)AdjustedTextLocation.X - AdjustedTextYX[0, 1].X,
                            (int)AdjustedTextLocation.Y - AdjustedTextYX[0, 1].Y),
                            LerpingColors);
                    }
                }
            }
        }
    }
}
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Linq;

namespace ParaatiOpen
{
    class ScrollingTexture
    {
        protected Texture2D Ground, BackgroundIMG, BackgroundIMG2,
                    Line1, Line2;
        protected Texture2D[] Background = new Texture2D[5];
        protected Vector2 Offset;
        protected Viewport Viewport;
        protected Rectangle Background1Size, Background2Size;
        protected int chosenBackground1, chosenBackground2,
                    addSpeed, ScreenCenter;
        protected Random SelectBackground = new Random();
        protected Color[] Lines;
```

```
        private static ContentManager content;
        public static ContentManager Content
        {
            protected get { return content; }
            set { content = value; }
        }

        public ScrollingTexture(GraphicsDevice device) //Setup
        {
            Viewport = device.Viewport;
            Ground = Content.Load<Texture2D>("parallax");
            Offset = Vector2.Zero;

            for (int i = 0; i < 5; i++)
            {
                Background[i] = Content.Load<Texture2D>("tausta" + i);
            }

            ScreenCenter = (Viewport.Height / 2) +50;
            Line1 = new Texture2D(device, 1, 10);
            Line2 = new Texture2D(device, 1, 10);

            BackgroundIMG = Background[0];
            BackgroundIMG2 = Background[1];
            Background1Size = new Rectangle(0, -50, Viewport.Width, 300);
            Background2Size = new Rectangle
            ((Background2Size.X-Viewport.Width) , -50, Viewport.Width, 300);

            Lines = new Color[1 * 10];
            // Create road surface marking lines
            for (int paint = 0; paint < Lines.Count(); paint++)
            {
                Lines[paint] = new Color(255,255,255);
            }
            Line1.SetData(Lines);
            Line2.SetData(Lines);
        }

        public void Update(int GameSpeed)
        {
            addSpeed = (GameSpeed);
            Background1Size.X += addSpeed;
            Background2Size.X += addSpeed;
            Offset.X += (-1 * addSpeed);

            if (Offset.X < -Ground.Width)
            {
                Offset = new Vector2(-4,0);
            }
            // If background is out of game screen, change it
            if (Background1Size.X > Viewport.Width)
            {
                chosenBackground1 = SelectBackground.Next(0, 3); //0,2
                BackgroundIMG = Background[chosenBackground1];
                Background1Size.X = (Background2Size.X - Viewport.Width);
            }
            if (Background2Size.X > Viewport.Width)
            {
                chosenBackground2 = SelectBackground.Next
                       ((chosenBackground1 += 1),5);
                BackgroundIMG2 = Background[chosenBackground2];
                Background2Size.X = (Background1Size.X - Viewport.Width);
            }
        }
```

```
        public void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(Ground, new Vector2(Viewport.X, ScreenCenter),
                 new Rectangle((int)Offset.X, (int)Offset.Y, Viewport.Width,
                    Ground.Height), Color.White);
            spriteBatch.Draw(BackgroundIMG, Background1Size, Color.White);
            spriteBatch.Draw(BackgroundIMG2, Background2Size, Color.White);
            spriteBatch.Draw(Line1, new Vector2(Viewport.X,
                    BackgroundIMG.Height + 40), new Rectangle((int)Offset.X,
                    (int)Offset.Y, Viewport.Width, Line1.Height), Color.White);
            spriteBatch.Draw(Line2, new Vector2(Viewport.X,
                   Viewport.Height - 30), new Rectangle((int)Offset.X,
                  (int)Offset.Y, Viewport.Width, Line2.Height), Color.White);
        }
    }
}
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace ParaatiOpen
{
    class TitleScreen
    {
        protected Texture2D[] logoText = new Texture2D[6];
        protected Texture2D logoBG;
        protected SpriteFont Font;
        protected Rectangle introBGAnim, introBGSize;
        protected int introBGAnimMax, introBGWidth;
        protected Rectangle[] introTextLocation = new Rectangle[6];
        protected Rectangle[] introTextAnimBegin = new Rectangle[6];
        protected int[] introTextframe = new int[6];
        protected float timer, delay = 200f;
        protected Vector2 CreatorText, CommandText;

        private static ContentManager content; // Get data from Main -class
        public static ContentManager Content
        {
            protected get { return content; }
            set { content = value; }
        }

        public TitleScreen (SpriteFont font, int SW, int SH) //Setup
        {
            Font = font;
            logoBG = Content.Load<Texture2D>("introbg");

            for (int i = 0; i < 6; i++)
            {
                logoText[i] = Content.Load<Texture2D>("logo" + i);
            }
            //Adjust positions based on screen width(SW) and height(SH)
            introBGSize = new Rectangle(((SW - 600) / 2),
                      ((SH - 120) / 2)-240, 600, 120);
            introBGAnim = introBGSize;
            introBGWidth = (logoBG.Width / 3);
            CreatorText = new Vector2(introBGSize.X + 170,
                         introBGSize.Y + 120);
            CommandText = new Vector2((SW / 2) - 120, ((SH / 2) - 10));

            for (int i = 0; i < logoText.Length; i++) // Adjust text
            {
                introTextLocation[i] = new Rectangle((introBGSize.X+30) +
                     (i * 90), (introBGSize.Y+10), 90, 100);
```

```
            }
        }

        public void Update (GameTime gametime) // Handle Animations
        {
            timer += (float)gametime.ElapsedGameTime.TotalMilliseconds;
            if (timer >= delay)
            {
                introBGAnim = new Rectangle(introBGWidth * introBGAnimMax, 0,
                     introBGWidth, logoBG.Height); // Change animation frame
                introBGAnimMax = (introBGAnimMax + 1) % 3;

                for (int i = 0; i < logoText.Length; i++)
                {
                    // Current image, frame 0/"light" -frame
                    introTextAnimBegin[i] = new Rectangle(45 *
                       introTextframe[i], 0, 44, 50);
                    introTextframe[i] = 0;

                    // Other images play frame number 1/ "Shadow" -frame
                    for (int j = 0; j < logoText.Length; j++)
                    {
                        if (introTextAnimBegin[j] != introTextAnimBegin[i])
                        {
                            introTextframe[j] = 1;
                        }
                    }
                }
                timer = 0; // Reset timer
            }
        }

        public void Draw (SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(logoBG, introBGSize, introBGAnim, Color.White);
            for (int i = 0; i < logoText.Length; i++)
            {
                spriteBatch.Draw(logoText[i], introTextLocation[i],
                    introTextAnimBegin[i], Color.White);
            }
            spriteBatch.DrawString(Font, "SopuisaSopuli, 2018-2019", new
                Vector2(CreatorText.X+3, CreatorText.Y+3), Color.Black);
            spriteBatch.DrawString(Font, "SopuisaSopuli, 2018-2019", new
                Vector2(CreatorText.X, CreatorText.Y), Color.White);
            spriteBatch.DrawString(Font, "Hit enter to start!", new
                Vector2(CommandText.X+3, CommandText.Y+3), Color.Black);
            spriteBatch.DrawString(Font, "Hit enter to start!", new
                Vector2(CommandText.X, CommandText.Y), Color.White);
        }
    }
}
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System.Collections.Generic;

namespace ParaatiOpen
{
    class Player
    {
        Texture2D[] PlayerTexture = new Texture2D[2];
        public Rectangle PlayerSize, PlayerAnimation, PlayerCollision;
        public string[] HoldingText = new string[] { "", "Cloak","Clock",
```

```
            "Wand", "Tuba" };
        public bool AllowScore = false, usingWand = false, ThrowWand = false,
              TubaActive = false, OutFromScreen = false, CoinCollected = false,
              ClockUsed=false;
        public int invisibleCount=0, CurrentHolding = 0;
        private int CurrentTexture=0, playerFrame = 0, playerSpeed=4,
              FrameCount=0, invisibleCountMAX=250;
        SpriteEffects mirror;
        Keys[] KeysMovement = new[] { Keys.Up, Keys.W, Keys.Left, Keys.A,
              Keys.Down, Keys.S, Keys.Right, Keys.D };
        private List<SoundEffect> SFX = new List<SoundEffect>();

        private static ContentManager content;
        public static ContentManager Content
        {
            protected get { return content; }
            set { content = value; }
        }

        public Player (GraphicsDevice graphics) // Setup
        {
            PlayerTexture[0] = Content.Load<Texture2D>("pelaajawalk1");
            PlayerTexture[1] = Content.Load<Texture2D>("pelaajawalk2");
            PlayerSize = new Rectangle(graphics.Viewport.Width, 300, 145, 230);
            PlayerCollision = new Rectangle(PlayerSize.X, PlayerSize.Y,
            ((PlayerSize.Width / 3) + 30), ((PlayerSize.Height / 3) + 30));

            for (int i= 0; i < 4; i++)
            {
                SFX.Add(Content.Load<SoundEffect>("aani"+i));
            }
        }
        // Reset player before starting the game
        public void ResetPlayer(GraphicsDevice graphics)
        {
            CurrentHolding = 0;
            invisibleCount = invisibleCountMAX;
            OutFromScreen = false;
            mirror = SpriteEffects.None;

            if (PlayerSize.X > (graphics.Viewport.Width) ||
                 PlayerSize.X < -100)
            {
                PlayerSize.Y = (graphics.Viewport.Height / 2);
                PlayerSize.X = graphics.Viewport.Width;
            }
        }

        protected bool IfKeyDown(KeyboardState state,
        params Keys[] KeysMovement) // Check If selected key(s) are pressed
        {
            bool result = false;
            for (int i = 0; i < KeysMovement.Length; i++)
            {
                result |= state.IsKeyDown(KeysMovement[i]);
            }
            return result;
        }
        // Check if key is released
        protected bool IfKeyReleased(KeyboardState state)
        {
            bool result = false;
            for (int i = 0; i < KeysMovement.Length; i++)
            {
                result = state.IsKeyUp(KeysMovement[i]);
```

```
            if (!result) // Ends loop and returns true if result is false
            {
                break;
            }
        }
        return result;
    }

    public void UpdateMovement(KeyboardState state, int SpeedBonus,
    GraphicsDevice graphics) // Keyboard controls
    {
        int addSpeed = (playerSpeed + SpeedBonus);
        PlayerCollision.X = (PlayerSize.X + ((PlayerSize.Width / 3) - 10));
        PlayerCollision.Y = (PlayerSize.Y + (PlayerSize.Height / 3) - 20);

        if (IfKeyDown(state, Keys.Right, Keys.D)) // Right
        {
            PlayerSize.X += addSpeed;
            mirror = SpriteEffects.FlipHorizontally;
        }
        else if (IfKeyDown(state, Keys.Left, Keys.A)) // Left
        {
            PlayerSize.X -= addSpeed;
            mirror = SpriteEffects.None;
        }
        else
        {
            PlayerSize.X += SpeedBonus; // Gradually move to right
        }

        if (IfKeyDown(state, Keys.Up, Keys.W) && PlayerSize.Y > 0) // Up
        {
            PlayerSize.Y -= addSpeed;
        }
        else if (IfKeyDown(state, Keys.Down, Keys.S) && PlayerSize.Y <
             (graphics.Viewport.Height-100)) // Down
        {
            PlayerSize.Y += addSpeed;
        }
        // Use items
        if (state.IsKeyDown(Keys.Space) && CurrentHolding != 0)
        {
            ReleaseItem();
        }
        AllowScore = !IfKeyReleased(state);
        CorrectAnimation(state, graphics);
        NonKeyActions(graphics);
    }

    public void NonKeyActions(GraphicsDevice graphics) // Automatic actions
    {
        if (invisibleCount > 0) // Player is damaged/ using cloak.
        {
            invisibleCount--;
        }
        // Outside of in game screen means game over
        if (PlayerSize.X > graphics.Viewport.Width || PlayerSize.X < -180)
        {
            OutFromScreen = true;
        }
    }
```

```
        public void EnemyCollision() // Collision with enemy
        {
            if (CoinCollected) // If the player has a coin; takes damage
            {
                CoinCollected = false;
                invisibleCount = 250;
            }
            else // otherwise; game over!
            {
                OutFromScreen = true;
            }
            SFX[1].Play();
        }

        public void ItemCollision(int itemValue) // Collision wth item
        {
            if (itemValue==0 || itemValue < 0) //Coin
            {
                CoinCollected = true;
            }
            else
            {
                if (itemValue > HoldingText.Length)
                {
                    itemValue = 1;
                }
                else
                {
                    CurrentHolding = itemValue;
                }
            }
            SFX[2].Play();
        }

        protected void ReleaseItem() // Use item
        {
            switch (CurrentHolding)
            {
                case 1: // Cloak
                    invisibleCount = invisibleCountMAX;
                    break;
                case 3: // Wand
                    if (!usingWand)
                    {
                        SFX[0].Play();
                        FrameCount = 0;
                        playerFrame = 5;
                        usingWand = true;
                    }
                    break;
                case 4: // Tuba
                    SFX[3].Play();
                    TubaActive = true;
                    break;
                default:  // Clock
                    ClockUsed = true;
                    break;
            }
            CurrentHolding = 0; // No item
        }
```

```
        protected void CorrectAnimation (KeyboardState state,
        GraphicsDevice graphics) // Animation
        {
            if (usingWand) // If player was holding a "wand" -item
            {



                if (playerFrame >= 7)
                {
                    usingWand = false;
                    ThrowWand = true;
                }
                Animation(1, 6, 8, false);
                mirror = SpriteEffects.None;
            }
            else // Check if keys pressed; Keys pressed
            {
                if (!IfKeyReleased(state))
                {
                    Animation(0, 4, 8, true);
                }
                else // No Key(s) pressed
                {
                    if (playerFrame >1)
                    {
                        playerFrame = 0;
                    }
                    Animation(1, 20, 2, true);
                    mirror = SpriteEffects.None;
                }
            }
        }
        // Executed when new game starts (but only if player
        position(X) < Screen center)
        public void IntroWalking(int SpeedBonus)
        {
            Animation(0, 3, 8, true);
            PlayerSize.X -= (playerSpeed + SpeedBonus);
        }
        //Executed when player gets "game over"
        public void GameOverAnimation(bool frame)
        {
            playerFrame = 2 + (frame ? 1 : 0);
            Animation(1, 30, 8, false);
        }

        protected void Animation(int Texture, int FrameCountMAX,
           int FrameDivider, bool IsLooping) // Handle players animation
        {
            CurrentTexture = Texture;
            if (FrameCount > FrameCountMAX || FrameCount < 0)
            {
                playerFrame = (playerFrame + 1) % FrameDivider;
                if (IsLooping)
                {
                    FrameCount = 0;
                }
            }
            else
            {
                FrameCount++; //adjust animation timer
            }
            PlayerAnimation = new Rectangle((500 * playerFrame), 0, 500, 750);
        }
```

```
        public void Draw(SpriteBatch spriteBatch)
        {
            if (invisibleCount % 2 == 0)
            {
                spriteBatch.Draw(PlayerTexture[CurrentTexture], PlayerSize,
                PlayerAnimation, Color.White, 0.0f, Vector2.Zero,
                mirror, 0.0f);
            }
        }
    }
}
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;

namespace ParaatiOpen
{
    class ParadeObject // Base class
    {
        private Texture2D[] image = new Texture2D[8];
        private Rectangle[] Collision = new Rectangle[8];
        protected Texture2D paradePerson;
        public Rectangle personSize;
        protected Rectangle PersonAnimate;
        protected int Speed, Points, CurrentFrame, MaxFrames,
               TextureWidth, TextureHeight;
        protected int frameCycle, Framedalay = 4, addSpeed = 0,
               LastFrame = 0, newLocation = 200, AliveTimer=0;
        public bool AllowRemove = false, ReturnScore = false;
        protected Random Unlock = new Random();

        public ParadeObject(Texture2D image, int newSpeed,
               int newPoints, int maxAnim)
        {
            paradePerson = image;
            TextureWidth = (image.Width / maxAnim);
            TextureHeight = image.Height;
            personSize = new Rectangle(-100, 0, (TextureWidth / 3),
                 (TextureHeight / 3));
            Speed = newSpeed;
            Points = newPoints;
            MaxFrames = maxAnim;
        }
        // Default object behaviour
        public virtual void Update(int GameSpeed, GraphicsDevice graphics)
        {
            addSpeed = (Speed + GameSpeed);
            Animation(addSpeed);
            Movement(addSpeed);
            OutFromScreen(graphics);
        }

        public virtual void Movement(int CurrentSpeed) // Default movement
        {
            personSize.X += CurrentSpeed;
        }

        public virtual void Animation(int Speed) // Default animation
        {
            PersonAnimate = new Rectangle(TextureWidth * CurrentFrame, 0,
                   TextureWidth, TextureHeight);
            frameCycle++;
```

```
            if (frameCycle >= Framedalay)
            {
                CurrentFrame = (CurrentFrame + 1) % (MaxFrames - LastFrame);
                frameCycle = 0;
            }
        }
        // If out from screen/ damage teken
        public virtual void OutFromScreen(GraphicsDevice graphics)
        {
            if (personSize.X > graphics.Viewport.Width ||
                personSize.X < -260 || AliveTimer > 20)
            {
                ReturnScore = true;
            }
        }
        // Returns "Points" value to main class and removes object
        public virtual int returnPoints()
        {
            AllowRemove = true;
            return Points;
        }

        public virtual void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(paradePerson, personSize,
            PersonAnimate, Color.White);
        }

        public object Clone()
        {
            return this.MemberwiseClone();
        }
    }

    class Items : ParadeObject  //Normal items
    {
        public int ItemValue;
        public Items(Texture2D image, int newSpeed, int newPoints, int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim) // Setup
        {
            personSize = new Rectangle(-100, 0, (TextureWidth / 2),
                   (TextureHeight / 2));
            Framedalay = 50;
        }

        public virtual void AddCopy(List<Items> ItemClones,
              Items ItemClone, int Y, int X, int ID) //Create copy
        {
            var itemClone = ItemClone.Clone() as Items;
            itemClone.personSize.Y = Y;
            itemClone.personSize.X = X;
            itemClone.ItemValue = ID;
            ItemClones.Add(itemClone);
        }
    }

    class PlayersWand : Items // a Wand item used by player
    {
        SoundEffect SFXenemy;
        public PlayersWand(Texture2D image, int newSpeed, int newPoints,
        int maxAnim, SoundEffect Sound)
        : base(image, newSpeed, newPoints, maxAnim)
        {
            Framedalay = 3;
            SFXenemy = Sound;
        }
```

```
        public virtual void PlaySFX()
        {
            SFXenemy.Play();
        }

        public override void Movement(int CurrentSpeed)
        {
            personSize.X -= 15;
        }

        public override void OutFromScreen(GraphicsDevice graphics)
        {
            if (personSize.X > graphics.Viewport.Width ||
            personSize.X < -260 || AliveTimer > 20)
            {
                AllowRemove = true;
            }
        }

        public virtual void AddWandCopy(List<PlayersWand> ItemClones,
        PlayersWand ItemClone, int Y, int X)
        {
            var itemClone = ItemClone.Clone() as PlayersWand;
            itemClone.personSize.Y = Y;
            itemClone.personSize.X = X;
            ItemClones.Add(itemClone);
        }
    }
    // Regular Enemies, no player needed for act.
    class ParadeFolk : ParadeObject
    {
        public bool isDamageTaken = false;
        public Rectangle personCollision;
        public ParadeFolk(Texture2D image, int newSpeed, int newPoints,
             int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {
            personCollision = new Rectangle(personSize.X, personSize.Y,
            ((personSize.Width / 2) + 30), ((personSize.Height / 2) + 30));
            LastFrame = 1;
            AliveTimer = 0;
        }

        public virtual void Update(int GameSpeed, GraphicsDevice graphics,
        List<ParadeFolk> ParadeClones, List<Items> ItemClones, int X, int Y)
        {
            addSpeed = (Speed + GameSpeed);
            personCollision.X = (personSize.X + ((personSize.Width / 3) - 10));
            personCollision.Y = (personSize.Y + (personSize.Height / 3));

            if (isDamageTaken)
            {
                CurrentFrame = (MaxFrames - LastFrame);
                AliveTimer++;
            }
            else
            {
                Movement(addSpeed);
            }
            Animation(addSpeed);
            UpdateItemDrop(ParadeClones);
            OutFromScreen(graphics);
            PlayerFound(X, Y);
        }
```

```
        public virtual void UpdateItemDrop(List<ParadeFolk> ParadeClones)
        {
        }

        public virtual void PlayerFound(int playerLocationX,
        int playerLocationY)
        {
        }

        public virtual void AddCopy(List<ParadeFolk> ParadeClones,
        ParadeFolk PersonClone)
        {
            var paradeperson = PersonClone.Clone() as ParadeFolk;
            paradeperson.newLocation = Unlock.Next(200, 600);
            paradeperson.personSize.Y = paradeperson.newLocation;
            ParadeClones.Add(paradeperson);
        }
    }

    class RowFolk : ParadeFolk
    {
        protected int getLocation=200;
        public RowFolk(Texture2D image, int newSpeed, int newPoints,
        int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {

        }

        public override void AddCopy(List<ParadeFolk> ParadeClones,
        ParadeFolk PersonClone)
        {
            int getLocation = Unlock.Next(200, 600);
            for (int i=0; i <3; i++)
            {
                var paradeperson = PersonClone.Clone() as RowFolk;
                paradeperson.personSize.Y = (getLocation + (i*50));
                ParadeClones.Add(paradeperson);
            }

        }
    }

    class FloatyGhost : ParadeFolk
    {
        public FloatyGhost(Texture2D image, int newSpeed, int newPoints,
        int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {
        }

        public override void Animation(int Speed)
        {
            PersonAnimate = new Rectangle(TextureWidth * CurrentFrame, 0,
                   TextureWidth, TextureHeight);
            CurrentFrame = 1;
        }
        public override void Movement(int CurrentSpeed)
        {
            personSize.X += CurrentSpeed;
            personSize.Y += (int)(10 * Math.Sin((personSize.X * 0.2) *
              0.5 * 2 / 20));
        }
    }
```

```
class Squirrel : ParadeFolk // Enemies that Drop items
{
    protected int Waiting, ChangeAnimation=900;
    public Wand SquirrelsWand;
    public Squirrel(Texture2D image, int newSpeed, int newPoints,
         int maxAnim)
    : base(image, newSpeed, newPoints, maxAnim)
    {
        Waiting = 0;
    }

    public override void Movement(int CurrentSpeed)
    {
        if (personSize.X >= ChangeAnimation)
        {
            personSize.X += 3;
            Waiting++;

            if (Waiting >= 60)
            {
                Waiting = 61;
            }
        }
        else
        {
            personSize.X += CurrentSpeed;
        }
    }

    public override void Animation(int AddSpeed)
    {
        PersonAnimate = new Rectangle(TextureWidth * CurrentFrame, 0,
           TextureWidth, TextureHeight);
        if (isDamageTaken)
        {
            CurrentFrame = MaxFrames;
        }
        else
        {
            if (personSize.X >= ChangeAnimation)
            {
                //Wand is not returning, play different animation
                if (Waiting >= 60)
                {
                    CurrentFrame = 8;
                }
                else
                {
                    CurrentFrame = 7;
                }
            }
            else
            {
                frameCycle++;
                if (frameCycle >= Framedalay)
                {
                    CurrentFrame = (CurrentFrame + 1) % (MaxFrames - 3);
                    frameCycle = 0;
                }
            }
        }
    }
```

```
        public override void UpdateItemDrop(List<ParadeFolk> ParadeClones)
        {
            if (Waiting == 2)
            {
                AddCopyItem(ParadeClones);
                Waiting = 3;
            }
        }

        public virtual void AddCopyItem(List<ParadeFolk> ParadeClones)
        {
            var squirrelsWand = SquirrelsWand.Clone() as Wand;
            squirrelsWand.personSize.X = (this.personSize.X -10);
            squirrelsWand.personSize.Y = (this.personSize.Y -20);
            ParadeClones.Add(squirrelsWand);
            squirrelsWand.Setup();
        }
    }
    class Wand : ParadeFolk
    {
        public Items WandCollect;
        protected int DirectionThrow = 0, DirectionThrowMax = -30,
        SpeedValue = -2, Length;
        public Wand(Texture2D image, int newSpeed, int newPoints, int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {
            personSize = new Rectangle(0, 0, (TextureWidth / 2),
            (TextureHeight / 2));
            Framedalay = 7;
            LastFrame = 0;
        }

        public virtual void Setup()
        {
            Length = Unlock.Next(0, 5);
            if (Length > 2)
            {
                DirectionThrowMax = -200;
            }
        }

        public override void Update(int GameSpeed, GraphicsDevice graphics,
        List<ParadeFolk> ParadeClones, List<Items> ItemClones, int X, int Y)
        {
            personCollision.X = personSize.X;
            personCollision.Y = personSize.Y;
            if (DirectionThrow < DirectionThrowMax)
            {
                SpeedValue = 2;
            }

            DirectionThrow += SpeedValue;
            Movement(DirectionThrow/2);
            Animation(GameSpeed);
            OutFromScreen(graphics, ItemClones);
        }

        public virtual void OutFromScreen(GraphicsDevice graphics,
        List<Items> ItemClones)
        {
            if (personSize.X > graphics.Viewport.Width)
            {
                ReturnScore = true;
            }
            else if (personSize.X < -150)
            {
                AllowRemove = true;
```

```
                WandCollect.AddCopy(ItemClones, WandCollect,
                personSize.Y, -100, 3);
            }
        }
    }

    class TubaPlayer : ParadeFolk
    {
        public Items TubaInstru;
        public TubaPlayer(Texture2D image, int newSpeed, int newPoints,
               int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {

        }

        public override void Update(int GameSpeed, GraphicsDevice graphics,
        List<ParadeFolk> ParadeClones, List<Items> ItemClones, int X, int Y)
        {
            addSpeed = (Speed + GameSpeed);
            personCollision.X = (personSize.X + ((personSize.Width / 3) - 10));
            personCollision.Y = (personSize.Y + (personSize.Height / 3));

            if (isDamageTaken)
            {
                CurrentFrame = (MaxFrames - LastFrame);
                AliveTimer++;

                if (AliveTimer == 15)
                {
                    TubaInstru.AddCopy(ItemClones, TubaInstru,
                    this.personSize.Y, this.personSize.X + 10, 4);
                }
            }
            else
            {
                Movement(addSpeed);
            }
            Animation(addSpeed);
            OutFromScreen(graphics);
        }
    }
    // Classes that need players position to act.
    class WatchingPlayer : ParadeFolk
    {
        protected bool isActive;
        protected float Distance;
        protected Vector2 Getlocation;
        public WatchingPlayer(Texture2D image, int newSpeed, int newPoints,
        int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {
            personSize = new Rectangle(-100, 0, ((TextureWidth / 2) - 10),
            (TextureHeight / 2) -10);
        }

        public override void PlayerFound(int playerLocationX, int
        playerLocationY) //Method for Torpedo, Wand and HidingGhost enemies
        {
            Getlocation.X = playerLocationX;
            Getlocation.X = playerLocationY;
        }
    }
```

```
class Agent : WatchingPlayer
{
    public Agent(Texture2D image, int newSpeed, int newPoints, int maxAnim)
    : base(image, newSpeed, newPoints, maxAnim)
    {
        isActive = false;
    }

    public override void Animation(int speed)
    {
        PersonAnimate = new Rectangle(TextureWidth * CurrentFrame, 0,
           TextureWidth, TextureHeight);
        Distance = (Getlocation.X - personSize.X);

        if (Distance <= 60 && Distance > 0)
        {
            isActive = true;
        }

        if (personSize.Y > 600)
        {
            isActive = false;
        }
        CurrentFrame = (isActive ? 1 : 0);
    }

    public override void Movement(int Speed)
    {
        if (isActive)
        {
            personSize.Y += Speed;
        }
        else
        {
            personSize.X += (Speed - 2);
        }
    }

    public override void AddCopy(List<ParadeFolk> ParadeClones,
    ParadeFolk PersonClone)
    {
        var paradeperson = PersonClone.Clone() as Agent;
        paradeperson.personSize.Y = 20;
        ParadeClones.Add(paradeperson);
    }
}

class Torpedo : WatchingPlayer
{
    private int CurrentLock, CustomSpeed;
    SpriteEffects mirror;
    private SoundEffect SFXenemy;
    public Torpedo(Texture2D image, int newSpeed, int newPoints,
       int maxAnim, SoundEffect Sound, GraphicsDevice graphics)
    : base(image, newSpeed, newPoints, maxAnim)
    {
        personSize = new Rectangle(graphics.Viewport.Width, 40,
           (TextureWidth / 2), ((TextureHeight / 2) - 20));
        CurrentLock = 1;
        CustomSpeed = -1;
        SFXenemy = Sound;
        mirror = SpriteEffects.None;
    }
```

```
        public override void PlayerFound(int playerLocationX,
           int playerLocationY)
        {
            if (playerLocationY < 80 && CurrentLock > 0)
            {
                SFXenemy.Play();
                personSize.Y = 40;
                CurrentLock = 0;
                AliveTimer = 0;
            }
        }

        public override void Update(int GameSpeed, GraphicsDevice graphics)
        {
            addSpeed = (Speed + (GameSpeed*2));
            personCollision.X = (personSize.X + ((personSize.Width / 3) - 10));
            personCollision.Y = (personSize.Y + (personSize.Height / 3));

            if (CurrentLock == 0)
            {
                if (isDamageTaken)
                {
                    CurrentFrame = (MaxFrames - LastFrame);
                    AliveTimer++;
                }
                else
                {
                    Movement(addSpeed);
                }
                Animation(addSpeed);
            }
            OutFromScreen(graphics);
        }

        public override void Movement(int CurrentSpeed)
        {
            personSize.X += (CurrentSpeed * CustomSpeed);
        }

        public override void OutFromScreen(GraphicsDevice graphics)
        {
            if (personSize.X < -300)
            {
                personSize.X = -290;
                mirror = SpriteEffects.FlipHorizontally;
                CustomSpeed = 1;
                CurrentLock = 1;
                ReturnScore = true;
            }
            else if (personSize.X > graphics.Viewport.Width + 50)
            {
                personSize.X = graphics.Viewport.Width;
                mirror = SpriteEffects.None;
                CustomSpeed = -1;
                CurrentLock = 1;
                ReturnScore = true;
            }
        }

        public override int returnPoints()
        {
            ReturnScore = false;
            return Points;
        }
```

```
        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(paradePerson, personSize,
            PersonAnimate, Color.White, 0.0f, Vector2.Zero, mirror, 0.0f);
        }
    }

    class HidingGhost : WatchingPlayer
    {

        public HidingGhost(Texture2D image, int newSpeed, int newPoints,
           int maxAnim)
        : base(image, newSpeed, newPoints, maxAnim)
        {
        }

        public override void Animation(int Speed)
        {
            PersonAnimate = new Rectangle(TextureWidth * CurrentFrame, 0,
                TextureWidth, TextureHeight);
            CurrentFrame = 0;
            Distance = (Getlocation.X - personSize.X);

            if (personSize.X < 100 || Distance <=50 || isDamageTaken)
            {
                isActive = true;
            }
            else if (personSize.X > 150)
            {
                isActive = false;
            }
            else
            {
                if ((personSize.X % 2) == 0)
                {
                    isActive = !isActive;
                }
            }
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (isActive)
            {
                spriteBatch.Draw(paradePerson, personSize,
                  PersonAnimate, Color.White);
            }
        }
    }
}
```