Vesa Ryynänen

C# mobiilipeliohjelmoinnissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

29.4.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Vesa Ryynänen C# mobiilipeliohjelmoinnissa 36 sivua + 2 liitettä 29.4.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Lehtori Juha Kämäri

Insinöörityössä tarkasteltiin C#-kielen käyttöä mobiilipeliohjelmoinnissa. Työn tarkoituksena oli selvittää, millaisia mahdollisuuksia C# tarjoaa mobiilipelikehittäjälle, ja tutkia, onko kielen osaamisesta hyötyä mobiilipelikehityksessä.

Työn alussa esitellään ensin C#-kieltä yleisesti ja selvitetään kielen käyttömahdollisuuksia eri mobiilialustoilla. Työssä tutustutaan erilaisiin työkaluihin, jotka mahdollistavat C#-pelikehityksen eri mobiilialustoille ja tutkitaan peliprojektien työstämistä kyseisillä työkaluilla.

Loppupuoliskolla tutustutaan tarkemmin C#-ohjelmointiin Unity-ympäristössä käyttäen referenssinä omaa mobiilipeliprojektia, joka on julkaistu Google Play -kaupassa. Unity-osioon sisältyy tutkimus, jossa vertaillaan C#:n ja toisen Unity-skriptauskielen, JavaScriptin suorituskykyeroja ja ominaisuuksia keskenään. Tutkimus toteutetaan kääntämällä referenssiprojektin lähdekoodi JavaScriptistä C#:lle ja vertailemalla projektien profilointeja.

Tutkimuksen tulokset yllättivät positiivisesti. Graafisesti intensiivisessäkin peliprojektissa todettiin lähdekoodin kääntämisellä JavaScriptistä C#:lle olevan vaikutusta suorituskykyyn. C#-projektin todettiin pyörivän kohdelaitteella hieman sulavammin. Lisäksi koodin ylläpidettävyys ja luettavuus helpottui Unity-ympäristön paremman C#-tuen vuoksi. Loppupäätelmänä todettiin C#-kielen soveltuvan erinomaisesti mobiilipelikehitykseen niin Unityssä kuin sen ulkopuolellakin.

Avainsanat	C#, mobiilipelit, Unity, Android, iOS, Windows Phone

Author(s) Title Number of Pages Date	Vesa Ryynänen C# in Mobile Game Programming 36 pages + 2 appendices 29 April 2014
Degree	Bachelor of Engineering
Degree Programme	Information Engineering
Specialisation option	Software Engineering
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Juha Kämäri, Senior Lecturer

This thesis examined the use of C#-language in mobile game programming. The purpose of the thesis was to find out what possibilities C# offers to a mobile game developer and to study whether C# knowledge is beneficial in mobile game development.

In the beginning of the thesis, C# is introduced in general and its possibilities in various mobile platforms are sorted out. Some different tools that enable C#-game development to different mobile platforms are explored. The reader is also instructed how to get started with game projects using these tools.

Towards the end of the study, C#-programming in Unity-environment is examined in greater detail using a self-made mobile game as a reference project. This section includes a study which compared the performance and characteristics of C# to another Unity-scripting language, JavaScript. The research was carried out by converting the source code of the reference project from JavaScript to C#, and comparing the profilings of the projects.

The study results surprised positively. Converting the source code from JavaScript to C# was found to have an impact on the performance, even in a graphic-intense game project. The C#-project was found to run a bit smoother on a target device. Moreover, the maintainability and readability of the project became easier, because of the better support of the Unity-environment for C#. As a conclusion, C# was found to be ideally suitable for mobile game programming in Unity as well as outside it.

Keywords	C#, mobile games, Unity, iOS, Android, Windows Phone

Sisällys

1 Johdanto

Mobiilipelejä on ollut olemassa 90-luvun taitteesta alkaen, kun ensimmäisiin graafisiin taskulaskimiin alettiin toteuttaa pelejä. Samoihin aikoihin myös ensimmäiset käsikonsolit ilmestyivät markkinoille. Ensimmäinen laskimeen toteutettu peli käytti vanhaa ideaa käärmeestä, joka kasvaa syödessään kentällä olevia pisteitä. Samaa ideaa käytti myös Nokia julkaistessaan ensimmäisen matkapuhelimelle esiasennetun pelin, Snaken. [1.]

Ennen vuosituhannen vaihdetta mobiilipelejä ohjelmoitiin hyvin matalan tason kielillä. Suurin osa esimerkiksi Game Boylle julkaistuista peleistä on ohjelmoitu assemblerilla. Matkapuhelinvalmistajat ohjelmoivat pelinsä suoraan laitteisiin, eikä yritysten ulkopuoliset sovelluskehittäjät voineet vaikuttaa laitteen ohjelmistosisältöön.

2000-luvun alussa osassa laitteista alkoi olla värinäyttö sekä tuki Javalle. Niissä saattoi olla myös alkeellinen WAP-ominaisuus, joka mahdollisti langattoman verkkoyhteyden. Tämä mahdollisti laajempien ja monimutkaisempien sovellusten ja pelien kehittämisen mobiililaitteille. Nokian rinnalle alkoi ilmestyä pelejä sisältäviä matkapuhelimia myös muilta valmistajilta.

Mobiililaitteet kehittyivät yhä älykkäämmiksi, ja syntyikin tarve kehittää oikeita käyttöjärjestelmiä myös matkapuhelimille. Ohjelmistovalmistajien kilpailun myötä markkinoille tuli esimerkiksi mobiililaitteille räätälöidyillä Windows- ja Linux-käyttöjärjestelmillä varustettuja laitteita. Nokian valinta oli Symbian, mikä dominoikin markkinoita lähes koko 2000-luvun. [1; 2.]

Myös Microsoft halusi saada osansa matkapuhelinmarkkinoilta omalla Windows Mobile -käyttöjärjestelmällään. Tällä käyttöjärjestelmällä varustetut laitteet olivat tarkoitettu ensisijaisesti työkäyttöön, mutta niille ohjelmoitiin myös joitakin pelejä. Käyttöjärjestelmän ohjelmat ja pelit ohjelmoitiin käyttäen vuosituhannen vaihteessa kehitettyä .NET-arkkitehtuuria ja sitä käyttävää C#-kieltä. C#-mobiilipelit eivät kuitenkaan tavoittaneet laajaa yleisöä ennen Windows Phone-käyttöjärjestelmän syntyä vuonna 2010.

Windows Phonen ilmestymisen jälkeen on C#:n suosio mobiilipeliohjelmointikielenä kasvanut nopeasti. Kielellä tuotettiin aluksi pelejä pääsääntöisesti Windows- ja Windows Phone-alustoille, mutta sittemmin on ilmestynyt tekniikoita ja työkaluja, joiden avulla on mahdollista kehittää pelejä myös muille mobiilialustoille.

Tämän insinöörityön tarkoituksena on tutkia C#:n käyttöä mobiilipeliohjelmoinnissa. Työssä tutustutaan C#:n ominaisuuksiin ja toimivuuteen pelinkehityksessä eri mobiilialustoilla sekä tarkastellaan työkaluja, joilla on mahdollista kehittää mobiilipelejä C#:lla. Tavoitteena on selvittää C#:n asemaa mobiilipeliohjelmointikielien joukossa sekä hyödyllisyyttä mobiilipeliohjelmoijalle.

Työ jakautuu kuuteen lukuun. Toisessa luvussa tutustutaan C#:n historiaan, yleisiin ominaisuuksiin, rakenteeseen ja toimintaperiaatteeseen. Kolmannessa luvussa keskitytään tutkimaan C#:n käyttöä eri mobiilialustoilla ja tarkastelemaan tähän liittyviä työkaluja. Neljännessä luvussa esitellään työn ohessa kahden miehen tiimillä kehitetty mobiilipeli, jonka avulla tutkitaan C#:n suorituskykyä ja ominaisuuksia Unity-ympäristössä. Lukuun sisältyy C#:n ja Unity-pelimoottorin JavaSript-kielen vertailu, joka toteutettiin kääntämällä JavaScript-kielellä koodattu projekti C#:lle kielelle.

2 C#

2.1 Historiaa

1990-luvun alussa Sun Microsystemsin työntekijät kehittivät uutta ohjelmointiteknologiaa seuraavan sukupolven älykkäitä laitteita varten. He harkitsivat käyttävänsä teknologiansa perustana C++:aa, mutta hylkäsivät sen ja päättivät kehittää oman ohjelmointialustan ja -kielen, joka käyttäisi tehokkaammin muistia ja olisi vähemmän altis ohjelmoijan tekemille virheille. Näin syntyi uusi ”Oak” –niminen ohjelmointikieli, jonka nimi muutettiin myöhemmin ”Java”:ksi. [3.]

Javan automaattinen roskienkeruu helpotti ohjelmoijan työtä tuntuvasti, sillä hänen ei tarvinnut enää huolehtia järjestelmämuistin hallinnoinnista itse. Javalla tuotettua koodia ei myöskään käännetä suoraan konekielelle, vaan tavukoodiksi, joka ajetaan eräänlaisessa eristetyssä ympäristössä, virtuaalikoneessa. Virtuaalikone tulkkaa tavukoodia ja tekee siitä natiivikoodia, jota prosessori ymmärtää. Tämä tekee Java-

ohjelmista turvallisempia, sillä ne eivät pääse virtuaalikoneen ulkopuolelle vaikuttamaan muihin prosesseihin. Virtuaalikoneen mukana tuli myös standardikirjasto, joka oli huomattavasti C++:n vastaavaa kirjastoa laajempi.

Microsoft huomasi Javan mahdollisuudet ja kehitti oman Java-ympäristönsä, J++:n. Avainhenkilö tämän ympäristön luonnissa oli eräs vastikään taloon palkattu Anders Hejlsber. J++:n kehitys jouduttiin kuitenkin parin vuoden jälkeen lopettamaan Microsoftin hävittyä oikeustaistelun Sun Microsystemsiä vastaan. Microsoft tarvitsikin siten oman ratkaisunsa saavuttaakseen Java-alustan tasoinen tehokkuus ohjelmistokehityksessä.

Microsoft päätti luoda kokonaan oman ohjelmistoympäristönsä, joka ottaisi oppia Javasta, mutta toteuttaisi kaiken vielä paremmin. Syntyi Next Generation Windows Services, joka muuttui myöhemmin muotoon .NET Framework. Tähän alettiin kehittää vastaavia luokkakirjastoja ja virtuaalikonetta kuin Javassa. Ajoympäristön ja virtuaalikoneen nimeksi tuli Common Language Runtime (CLR).

Tarvittiin uusi ohjelmointikieli, joka olisi yhtä helppokäyttöinen kuin Java, mutta vastaisi tehokkuudeltaan paremmin C++:aa. Tätä alkoi kehittämään J++:n luonut Hejlsberg. Kielen nimi oli aluksi Cool, lyhenteenä lauseesta “C-like Object Oriented Language”. Tavaramerkkisyistä nimi muutettiin myöhemmin muotoon C#. [4.]

2.2 Mikä on C#?

C# on kieli, joka ajetaan .NET-kehyksen virtuaalikoneessa. Se on suunniteltu alusta lähtien toimimaan yhdessä .NET-alustan kanssa. C#:n syntaksi on samankaltainen kuin C:ssä tai C++:ssa ja siihen on tuotu useita ominaisuuksia niin C++:sta kuin Javastakin.

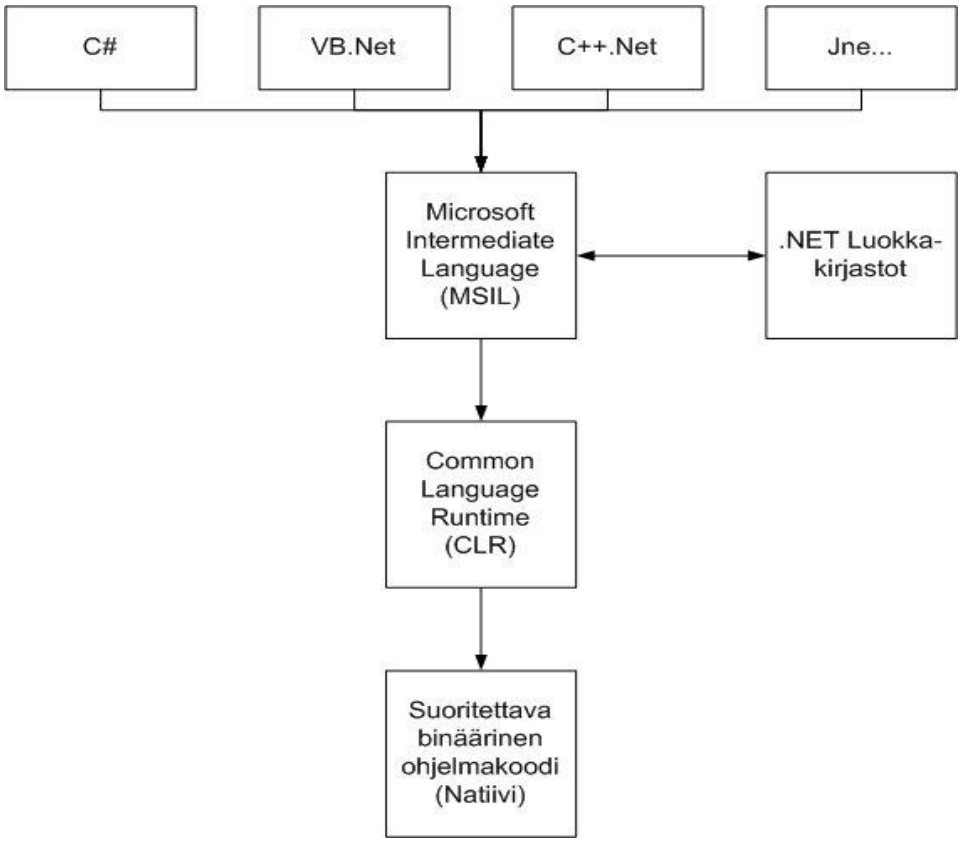
Sovellusten kehittämisestä C#:lla on pyritty tekemään yksinkertaisempaa, kuin esimerkiksi C++:lla. Siinä on Java-kielestä tuttu automaattinen roskienkeruu, mikä keventää ohjelmoijan työtaakkaa muistinhallinnan osalta, eikä esimerkiksi ”Destructor” –metodeita tarvitse välttämättä tehdä jokaiseen luokkaan. C# on tyyppiturvallinen kieli, mikä tarkoittaa, että kielellä ohjelmoidut sovellukset eivät voi vahingoittaa ympäröivää muistia muuttujiensa kautta. Käytännössä tämä tarkoittaa sitä, että muistiosoittimia ei pääsääntöisesti käytetä. Muistiosoittimia pystyy kuitenkin käsittelemään kuten C++:ssa,

mutta kyseiset koodilohkot tulee erikseen merkitä määreellä “unsafe”. Tämän määreen yhteydessä tarvitaan myös tiettyjä lupia koodin ajamiseen. Tyyppiturvallisuudesta ja muistirajoituksista johtuen koodista tulee kuitenkin helpommin ylläpidettävää ja virheiden etsiminen koodista käy helpommin.

C# on alati kehittyvä kieli. Siitä on lyhyen elinikänsä, reilun vuosikymmenen aikana ilmestynyt useita eri versioita. Kirjoitushetkellä viimeisin versio kielestä on C# 5.0, joka ilmestyi elokuussa 2012.

2.3 Toimintaperiaate

C# on tulkattu kieli, joka käännetään ensin tavukoodiksi ja natiivikielelle vasta ajon aikana. C#:n kääntyminen natiivikielelle on esitetty kuvassa 1. Kehitysympäristö esikääntää ohjelmoijan tuottaman C#-koodin Intermediate Language -muotoon, joka viedään CLR:n virtuaalikoneelle. CLR kääntää esikäännetyn ohjelmakoodin suorituksen aikana binäärimuotoon. Tällä periaatteella toimivat kaikki kielet, joita .NET-kehys tukee. Näitä onkin kuviossa esitetty muutama.



Kuva 1. C#:n kääntyminen natiivikielelle. [5.]

Kielen kääntäminen natiiviksi vasta ohjelman suorituksen aikana mahdollistaa sen, että samaa koodia ei tarvitse kääntää useiksi eri sovelluksiksi eri käyttöjärjestelmiä ja/tai suorittimia varten. Tämä vähentää ylimääräisen työn määrää, ja ohjelmoija voi keskittyä olennaisempiin asioihin koodiinsa liittyen.

2.4 Rakenteesta

Taulukossa 1 on esitetty C#:n tärkeimpiä eroavaisuuksia ja yhtäläisyyksiä verrattuna Javaan ja C++:aan. [6.] Kuten Java, C# on täysin oliopohjainen kieli, joten jokainen asia kielessä on olio. Sillä on hyvin paljon yhtäläisyyksiä C:n, C++:n ja Javan kanssa, joten näillä kielillä aiemmin ohjelmoineet oppivat käyttämään C#:aa nopeasti.

Taulukko 1. C# verrattuna Javaan ja C++:aan. [6.]

Ominaisuus	**C#**	**Java**	**C++**
Roskien keruu	On	On	Ei
Osoittimet	On	Ei	On
Sisäänrakennetut merkkijonot	On	On	Ei
Merkkijonojen käyttö Switch-lauseissa	On	On (Java SE 7)	Ei
Moniperintä	Ei	Ei	On
Operaattorien ylikirjoitus	On	Ei	On
Dynaaminen luokkien lataaminen	On	On	Ei
Luokan ominaisuudet	On	Ei	Ei
Templatet	Ei	On	On
TypeOf-tunnistus	On	Ei	On (decltype)
Foreach-iterointi	On	On	On

Taulukosta käy selkeästi ilmi, miten C#:iin on tuotu monia hyviä ominaisuuksia sekä Javasta että C++:sta. C# ei tue moniperintää, mutta tähän liittyvät ongelmat voidaan ratkaista esimerkiksi määrittelemällä tarvittavat asiat rajapintoihin, ja laittamalla luokan toteuttamaan nämä rajapinnat. Tämä on mahdollista, sillä C#-luokka voi toteuttaa usean rajapinnan kerralla. Dynaamista luokkien lataamista kutsutaan myös reflektioksi.

Sen avulla voidaan esimerkiksi suorituksen aikana luoda olioita luokista, joihin ei ole vielä viitattu projektissa. C#:n vastine templateille on geneerisyys, joka on hieman yksinkertaisempi lähestymistapa parametrisoiduille tietotyypeille. Templatet ovat kuitenkin tässä asiassa hieman joustavampia. Kuten C++:ssa, on C#-kielen muuttujilla oltava tyyppi tiedossa ohjelmaa käännettäessä. Ajettava pääohjelma tulee olla olion sisäisen ”static void Main()” -nimetyn funktion sisällä. Tunnetuimmista sukulaiskielistä poiketen aliohjelmakirjastojen ja nimiavaruuksien käyttö merkitään määreellä ”using” ennen ohjelman määrittelyä.

3 C# Mobiilialustoilla

Tässä luvussa tutustutaan C#:n käyttömahdollisuuksiin eri alustoilla mobiilipelien kannalta. Luvussa käydään lyhyesti läpi miten peliprojektin tekeminen käytännössä onnistuu C#-kielellä. Lisäksi tarkastellaan erilaisia kehitystyökaluja, jotka mahdollistavat mobiilipelien kehittämisen C#:lla eri alustoille.

3.1 Asema mobiilipeliohjelmointikielten joukossa

C# sisältää ominaisuuksia, jotka tekevät siitä varteenotettavan vaihtoehdon mobiilipelikehittäjälle. Viime vuosikymmenen aikana sen suosio on noussut tasaisesti niin pienten kuin suurtenkin peliyritysten keskuudessa. PyPL-indeksin tilastotieteilijät nimesivät C#:n vuoden 2012 ohjelmointikieleksi. [7.] Nykyään suuri osa etenkin pienemmistä mobiilipelejä kehittävistä yrityksistä edellyttää työntekijöiltään C#-kielen osaamista.

Windows 8- ja Windows Phone -käyttöjärjestelmien julkaisu ovat näytelleet kohtalaisen suurta osaa C#:n suosion leviämisessä myös mobiilialustoille. Kielen ongelmana onkin aikaisemmin ollut vahva yhteys .NET-ympäristöön, joka puolestaan sitoi sen käytettäväksi ainoastaan Windows-alustoilla. Ohjelmoijat ovat kuitenkin vakuuttuneet kielen hyödyllisyydestä niin, että on kehitetty erilaisia työkaluja ja menetelmiä, joilla ohjelmia ja mobiilipelejä voidaan kehittää myös muille alustoille.

Nykyään C#:aa voidaankin pitää yhtenä parhaimmista kielistä siirrettävyyden suhteen. Siirrettävyys ei rajoitu pelkästään mobiilialustoille, vaan myös työpöytä-, palvelin- ja

sulautettuihin järjestelmiin. Vuonna 2012 laskettiin, että C# tavoitti yli 2,2 miljardia laitetta. [8.]

3.2 Kehitystyökaluja mobiilialustoille

3.2.1 Windows Phone

Mobiilipelin kehittäminen C#-kielellä Windows Phone -alustalle on alustoista suoraviivaisin ja yksinkertaisin prosessi, sillä myös kohdelaite pyörii .NET-arkkitehtuurin ympärillä. Kehittäjällä tarvitsee olla käytössään Windows Vista tai uudempi Windows-käyttöjärjestelmä sekä Windows Phone SDK -paketti. Paketti on ladattavissa Microsoftin verkkosivuilta. [9.]

Peliprojektejä varten kehittäjällä tulee lisäksi olla käytössään XNA Game Studio. XNA on Microsoftin kehittämä oliopohjainen kirjasto, joka on tarkoitettu erityisesti pelinkehitykseen. [10.] Sillä voidaan käyttää .NET-yhteensopivilla kielillä. Mobiilipelien lisäksi XNA:lla voi kehittää pelejä esimerkiksi PC:lle ja Xbox 360 -pelikonsolille.

Täydelliset ohjeet peliprojektin työstämiseen XNA:lla ja C#:lla löytyvät Microsoft API:sta. [10.] Uutta peliproketia varten tulee luoda uusi Windows Phone Game -projekti. Tämän jälkeen tulee valita kohdelaiteen Windows Phone -versio. Hyväksynnän jälkeen ohjelma luo kaikki tarvittavat perusosaset pelinkehitykseen.

Projektin luonnin jälkeen voi tuoda projektin käyttöön esimerkiksi grafiikkaa koodiesimerkin 1 mukaisesti.

```
public class Game1 : Microsoft.Xna.Framework.Game
    {

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        // Rendedöitävä tekstuuri
        Texture2D myTexture;

        // Kuvan koordinaatit
        Vector2 spritePosition = Vector2.Zero;


        protected override void LoadContent()
        {
           // Luodaan uusi SpriteBatch, jota voidaan käyttää
           tekstuurin piirtämiseen.
            spriteBatch = new SpriteBatch(GraphicsDevice);
            myTexture = Content.Load<Texture2D>("mytexture");
        }

    }
```

Koodiesimerkki 1: Grafiikan tuonti XNA-projektin käyttöön.

Koodiesimerkissä luodaan uudet tekstuuri- ja SpriteBatch-oliot. Tekstuuri ladataan sitten projektin käyttöön LoadContent-metodissa. Tekstuurin tulee olla tuotuna projektiin, jotta Content.Load()-käsky kykenee löytämään sen. Kun grafiikka on tuotu ja projektin tiedossa, se voidaan piirtää ruudulle koodiesimerkin 2 mukaan.

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

    // Draw the sprite.
    spriteBatch.Begin(SpriteSortMode.BackToFront, Blend-
State.AlphaBlend);
    spriteBatch.Draw(myTexture, spritePosition, Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

Koodiesimerkki 2 : Grafiikan piirtäminen ruudulle.

XNA:n helppokäyttöisyydestä ja tehokkuudesta huolimatta Microsoft päätti lopettaa sen päivittämisen 1.4.2014. [11.] Syyt tähän löytyvät todennäköisesti uuden pelikonsolin julkaisemisesta, muiden työkalujen suosiosta, sekä siitä, että XNA-pelit eivät ole menestyneet odotetusti Xbox Live-markkinoilla. Tätä työtä kirjoitettaessa on vielä hieman epävarmaa, miten Windows Phone-pelien kehittäminen C#-kielellä tästä eteenpäin jatkuu, mutta ainakin XNA:n tuki jatkuu ulkopuolisten tahojen toimesta. Tästä esimerkkinä on MonoGame, jota käsitellään luvussa 3.2.2. [12.]

3.2.2 Android

Mobiilipelikehitys Androidille C#:lla käy helpoiten Xamarin.Androidin ja MonoGamen avulla. Xamarin.Android on yhdysvaltalaisen ohjelmistoyrityksen, Xamarinin kehittämä tuote, joka mahdollistaa Android-kehittämisen .NET-ympäristössä. [13.] Xamarin.Android on osa Xamarin-pakettia. Xamarin-pakettiin kuuluu tarvittavat työkalut myös iOS-kehitykseen. Paketti on saatavilla erihintaisilla lisensseillä. Itsenäisen kehittäjän lisenssi maksaa 299 dollaria. Tämä on halvin lisenssi, jonka kanssa MonoGame toimii pelikehityksessä.

MonoGame on yksityisen MonoGame Teamin kehittämä vapaassa levityksessä oleva ohjelma, jonka alkuperäisenä tarkoituksena mahdollistaa XNA-pelikehittäjien tuotteiden julkaiseminen myös iPhonelle. Sittemmin tuettuja alustoja on lisätty, ja mukaan on tullut myös Android. Se käyttää XNA4-rajapintaa, ja sen avulla on kehitetty useita suosittuja mobiilipelejä. [14.]

Ohjelmien käyttö onnistuu Microsoft Visual Studion yhteydessä tai ilman sitä. Mikäli ei voida tai haluta käyttää Visual Studiota, on Xamarin julkaissut myös oman Xamarin Studio-ohjelman, jonka avulla kehittäminen onnistuu. Tässä työssä tarkastellaan käyttöä Visual Studion kanssa.

Xamarin-paketti on ladattavissa Xamarinin ja MonoGame CodePlexin verkkosivuilta. [15; 16.] Uudet käyttäjät saavat testikäyttöön 30 päivän kokeiluversion Business-lisenssistä, jolla kehitys MonoGamen kanssa onnistuu. MonoGamen voi ladata myös kehitystiimin omilta verkkosivuilta. Ohjelmoijan ei tarvitse huolehtia ulkopuolisten Android-pakettien asennuksista, vaan asennusohjelma hakee kaikki tarvittavat liitännäiset automaattisesti verkosta. Pakettien asennuksen jälkeen kaikki on valmista Android-kehitykseen

Peliprojektin luomisen jälkeen itse pelin ohjelmointi on päällisin puolin samankaltaista kuin Windows Phonelle kehitettäessä. Edellisen luvun koodiesimerkit käyvät sellaisenaan piirtämään grafiikkaa myös Android-laitteelle. Taustalla tapahtuvat asiat tietysti eroavat, mutta Xamarin pitää niistä huolen.

3.2.3 iOS

Mobiilipelin, kuten minkä tahansa muun ohjelman työstäminen iOS-käyttöjärjestelmälle vaatii pääsääntöisesti kehittäjän käyttöön Apple-laitteen, jolla projektia työstetään. On olemassa keinoja, jolla C#-koodia pystyy tuottamaan iOS-laitteella toimivaksi Windows-alustaa käyttäen. Kuitenkin iOS-alusta on välttämätön siinä vaiheessa, kun koodia aletaan kääntämään ja testaamaan kohdelaitteella. Näin ollen tässä luvussa keskitytään iOS-pelikehitykseen iOS-alustalta.

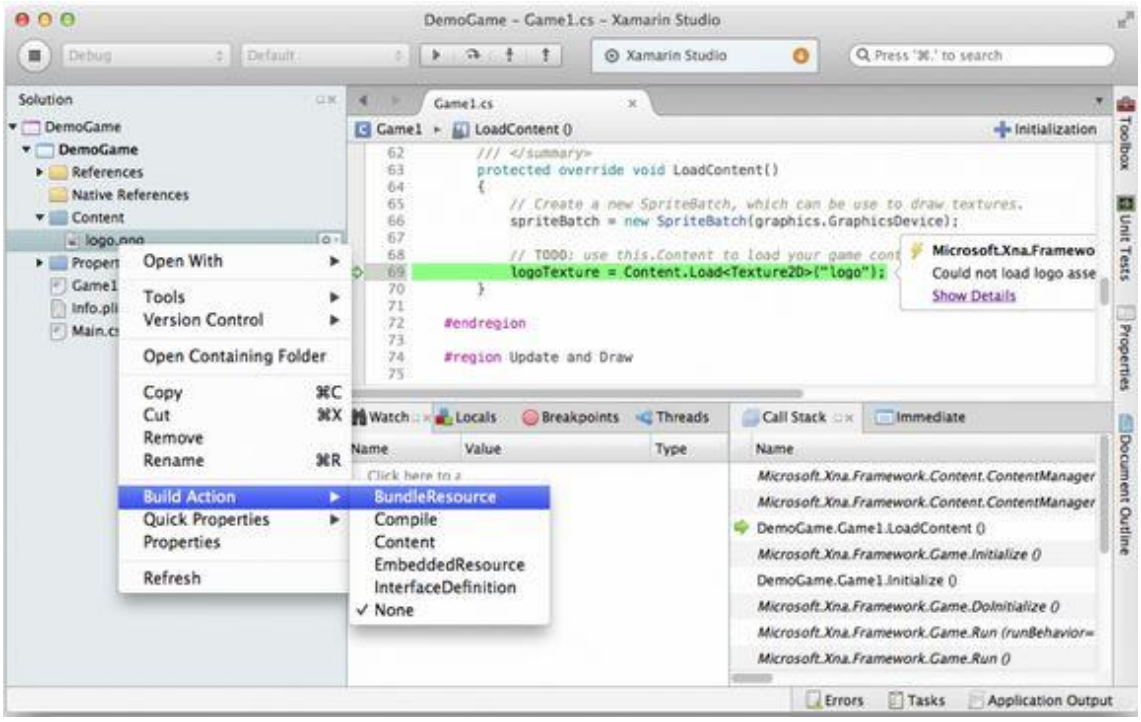
C#-pelikehitys iOS-alustalle on muutenkin hieman monimutkaisempi prosessi, ja alkuun pääseminen vaatii enemmän vaiheita kuin muille alustoille kehitettäessä. MonoGamen dokumentaatiota ylläpidetään GitHub-verkkosivustossa. Dokumentaatio sisältää oppaita, joiden avulla iOS-kehityksessä pääsee alkuun. Tulevaisuudessa prosessia tullaan todennäköisesti yksinkertaistamaan. [17.]

Pelikehitys iOS-laitteelle on helpointa Xamarinin omalla Xamarin Studio -ohjelmalla, joka on saatavilla iOS:lle. Tämän seuraksi tulee ladata ja asentaa iOS:n MonoGame for Xamarin Studio -paketti. MonoGame for Xamarin Studio on .mpack-päätteinen liitännäistiedosto, joka pitää asentaa manuaalisesti Xamarin Studion Add-in Managerin avulla.

Kun kehitysympäristö on saatu kuntoon, voidaan aloittaa uusi projekti. Uuden projektin luominen Xamarin Studiolla on kutakuinkin samankaltaista kuin Windowsin Visual Studiolla.

Xamarin.iOS ei vielä toistaiseksi kykene kääntämään MonoGame Mac -projektia kohdelaitteelle oletusasetuksilla. Tämä johtuu siitä, että projekti tuo viitattavaksi linkkikirjaston, joka toimisi ainoastaan Windowsilla. Projektin kääntämistä varten ohjelmoija joutuu lataamaan GitHub-verkkosivustosta viimeisimmän MonoGame source archive -projektin, joka käännettynä tuottaa oikean Mac-yhteensopivan linkkikirjaston. Kun kirjastoviittaukset on kunnossa ja projektia yrittää kääntää

uudelleen, tulee vielä yksi virheilmoitus puuttuvasta assetista. Tämä korjaantuu vaihtamalla kyseisen assetin Build Action-valinta kohtaan BundleResource kuvan 2 mukaisesti. [18.]



Kuva 2: Logo-assetin virheellinen asetus

Alkutoimenpiteiden jälkeen C#-koodia voi kirjoittaa Xamarin Studiolla samankaltaisesti kuin muillekin alustoille pienillä käyttöliittymän eroavaisuuksilla. Ohjelmassa on iOS-alustalla vielä pieniä puutteita, mutta ne tullaan todennäköisesti korjaamaan lähitulevaisuudessa. Xamarin.iOS-paketti sisältyy samaan lisenssiin kuin Xamarin.Android.

3.3 Muita C#-kehitystyökaluja

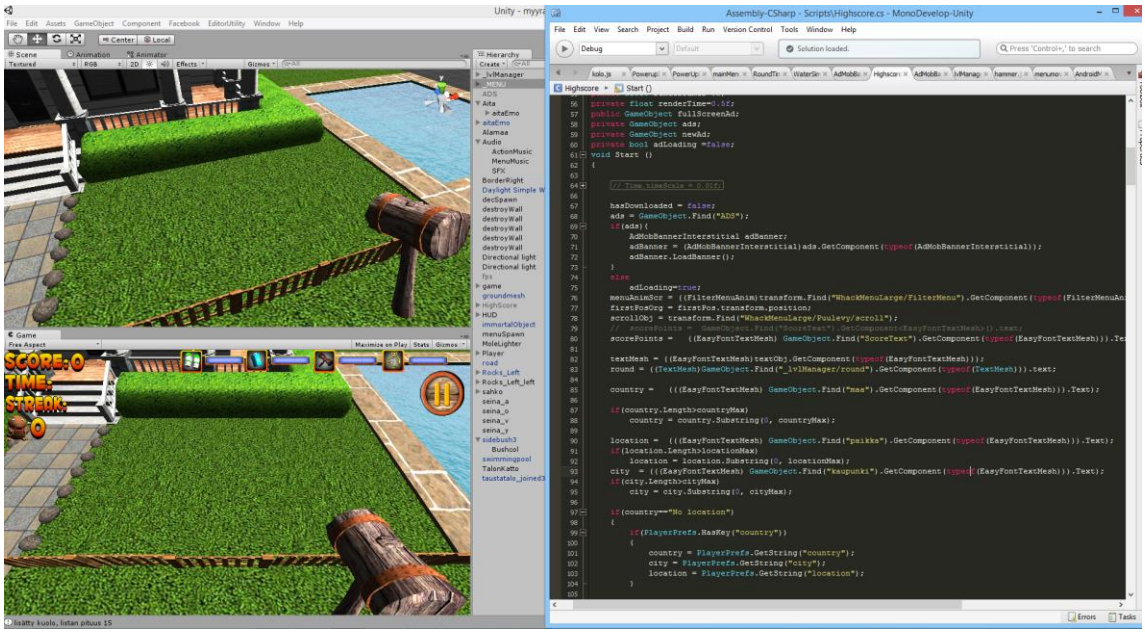
3.3.1 Unity

Unity (myös Unity3D) on tämän hetken suosituin monialustainen pelimoottori mobiilikehittäjille. [19.] Sen on kehittänyt alun perin tanskalainen Unity Technologies ja siinä on sisäänrakennettuna intuitiivinen käyttöliittymä. Tuettuja kieliä ovat C#, JavaScript ja Boo. Tämän hetken tuettuja alustoja ovat Windows, OS X, Linux, iOS,

Anroid, Blackberry 10, suosituimmat pelikosolit ja verkkoselaimet. Unityn suuren suosion takana on helppo saatavuus ja lähestyttävyys itsenäiselle pelinkehittäjälle, sekä Asset-kauppa, jossa pelinkehittäjät voivat jakaa ja myydä kehittämiään pelikomponentteja. Unity oli myös ensimmäisiä moottoreita, jossa oli täydellinen tuki iOS-alustalle. Tänä päivänä 53.1 % mobiilipelikehittäjistä käyttää Unityä. [19.]

Unity on saatavilla kahdenlaisella lisenssillä. Ilmaislisenssi tarjoaa kaikki perusominaisuudet, ja sillä voi kehittää kaupallisia pelejä tiettyyn tulorajaan asti. Ilmaislisenssi on tarkoitettu itsenäisten pelikehittäjien ja pienyritysten käyttöön. Pro-lisenssin voi hankkia 1500 dollarin kertahinnalla tai kuukausimaksuilla. Pro-lisenssillä on kaikki ominaisuudet ja työkalut käytössä, ja sillä voi tehdä pelejä ilman rajoituksia.

Skriptaamiseen Unity käyttää oletuksena hieman muokattua versiota MonoDevelopista, mutta muitakin työkaluja voidaan käyttää. MonoDevelop ja Unity toimivat yhteistyössä tehokkaasti, sillä MonoDevelopissa on mukana koodivihjeet ja mahdollisuus askellettuun virheenjäljittämiseen ohjelman suorituksen aikana. Se on myös aktiivisessa kehityksessä yhdessä Unity-ympäristön kanssa. Myös muita työkaluja, kuten Visual Studiota voidaan käyttää Unity-skripteihin. MonoDevelopia suositaan kuitenkin etenkin itsenäisten pelikehittäjien keskuudessa, sillä se on ilmainen. Kuvassa 3 on esitetty Unity-käyttöliittymä yhdessä MonoDevelopin kanssa. Neljännessä luvussa tutustutaan tarkemmin C#-ohjelmointiin Unityssä.



Kuva 3: C#-ohjelmointia Unityssä MonoDevelopilla

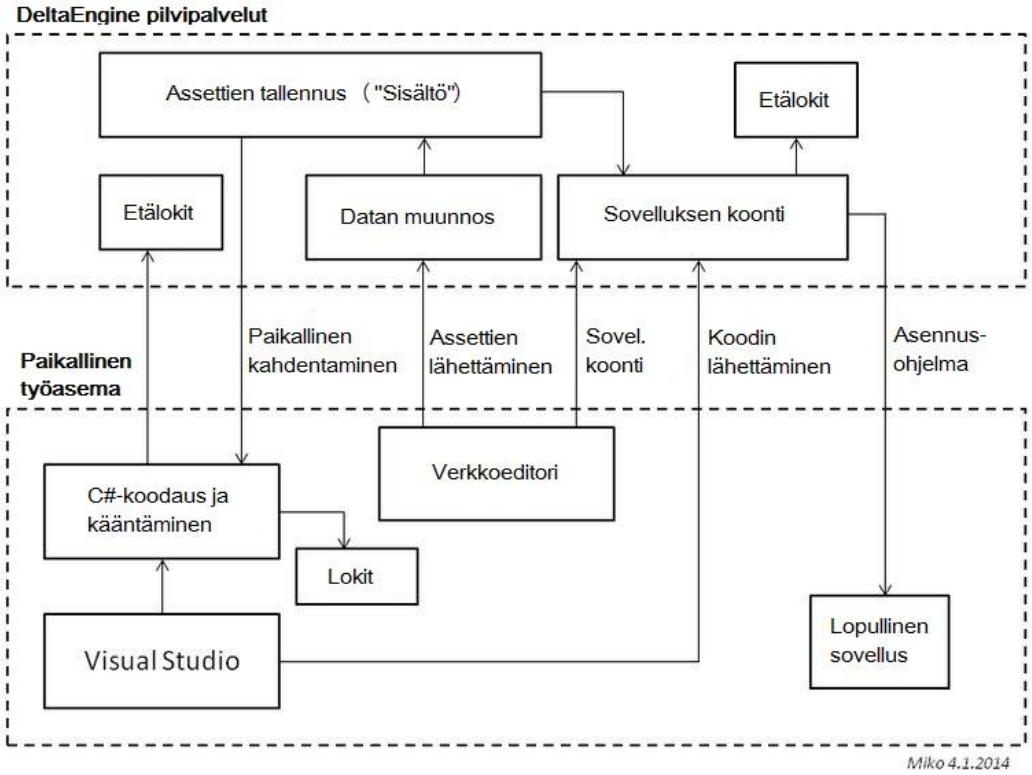
3.3.2 Delta Engine

Delta Engine on saksalaisen Delta Engine GmbH:n kehittämä pelimoottori, joka mahdollistaa pelinkehityksen C#:lla usealle alustalle. [20.] Projektin tavoitteena on tehdä peliohjelmoinnista ja monialustakehityksestä mahdollisimman yksinkertaista. Toisin kuin Unity, Delta Engine perustuu täysin avoimeen lähdekoodiin, joka on saatavilla GitHubista. Ohjelmoija voi vapaasti muokata ja kehittää pelimoottorin ominaisuuksia eteenpäin. Delta Engine on kirjoitettu kokonaan C#:lla ja tukee yleisimpiä .NET-pohjaisia kehitystyökaluja, kuten Visual Studiota.

PC:lle kehittäminen Delta Enginellä alle viiden hengen ryhmällä on täysin ilmaista. Jos kehitystiimi on suurempi tai on tarvetta kehittää muille alustoille, joutuu hankkimaan kuukausihintaisen lisenssin. Peruslisenssi mahdollistaa Android- ja HTML5-kehityksen ja yrityslisenssi edellisten lisäksi kehityksen iOS, Windows Phone, Windows 8 Store sekä Linux-alustoille. Yrityslisenssin ominaisuudet ovat vielä työn alla, mutta ovat valmistumassa lähiaikoina. [20.]

DeltaEngine on siitä mielenkiintoinen pelimoottori, että monialustakehityksen vaatimat käännös- ja sisällön prosessointioperaatiot tehdään DeltaEnginen palvelimien avulla pilvessä. Ohjelmakoodi ja muu sisältö lähetetään tietokoneelta DeltaEnginen pilvipalveluille, joissa ne käännetään valitulle kohdealustalle. Valmis sovellus

lähetetään takaisin pilvestä kehittäjälle. Tämä vähentää ohjelmoijan työtaakkaa huomattavasti, ja tuloksena on hyvin pitkälle optimoitu sisältö. Sisällön optimointi mahdollistaa pienen asennustiedoston koon, mikä on tavoiteltava asia mobiilipeliä kehitettäessä. Moottorin toimintaperiaatetta on selvennetty kuvassa 4.



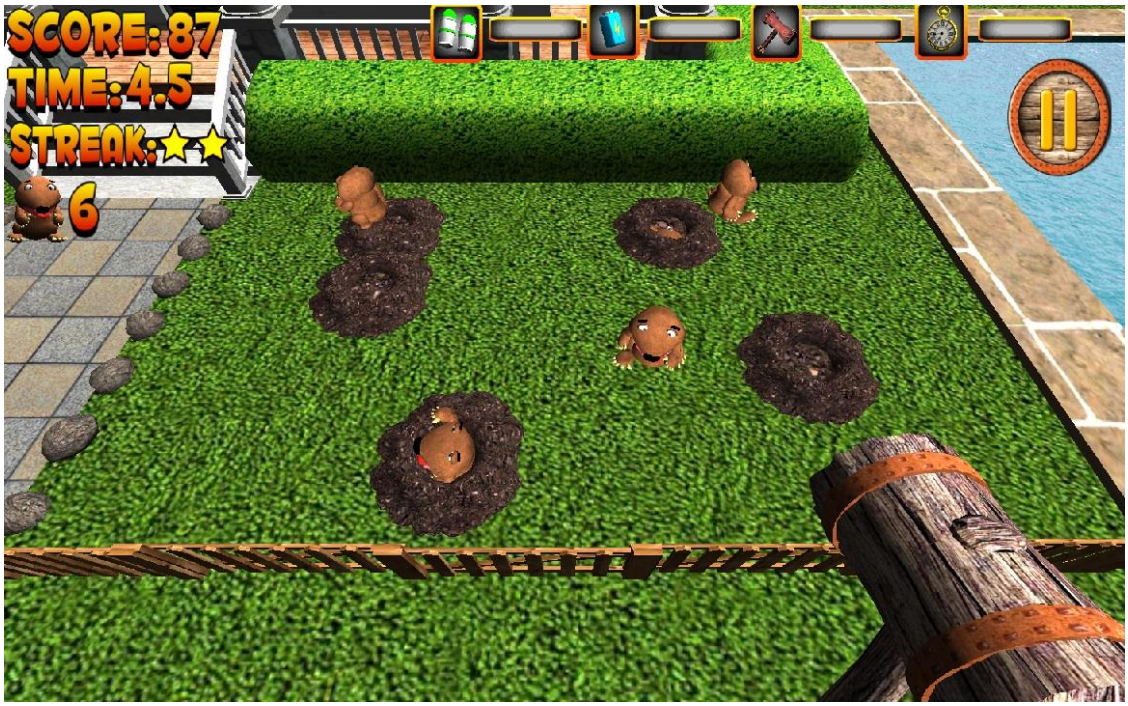
Kuva 4: Delta Enginen toimintaperiaate. [20.]

4 Oma Mobiilipeliprojekti

Tässä luvussa tutustutaan kirjoittajan ja hänen opiskelutoverinsa työstämään Android-peliin, Whack!:iin. Tarkoituksena on tutkia C#-ohjelmointia Unityssä oman peliprojektin avulla. Tavoitteena oli selvittää, mitä etuja tai hankaluuksia C#-kielen käytöllä on projektissa ja vertailla kieltä toiseen Unity-ohjelmoinnissa käytettyyn kieleen, JavaScriptiin. Tutkimusten tuloksena oli tarkoitus saada vastaus kysymykseen, kannattaako mobiilipeliä työstää Unityssä C#-kielellä vai ei?

4.1 Whack!

Whack! on kolmiulotteinen Android-alustalle suunnattu toimintapeli. Pelaajan tavoite on käyttää hallussaan olevaa vasaraansa mahdollisimman tehokkaasti hakkaamalla myyriä, jotka tekevät reikiä pihamaalle ja juoksevat niistä ulos. Pelin konsepti on ottanut vaikutteita 70-luvun pelihalliklassikosta, *Whac-A-Mole*:sta. Klassikkopelissä pelaaja hakkaa vasaralla myyriä, jotka tulevat aika ajoin ulos valmiiksi asetetuista rei'istä ja painuvat hetken päästä takaisin alas. Whack!-pelissä myyrät eivät kuitenkaan ikinä palaa koloihinsa, vaan yrittävät juosta pelialueelta ulos. Mikäli myyrä pääsee karkaamaan eikä pelaaja ehdi lyödä sitä, menettää pelaaja yhden elinvoimapisteen. Jos pelaaja lyö satunnaisesti ilmestyvään pommiin, menettää hän usean elinvoimapisteen. Elinvoimapisteitä saa kerättyä takaisin poimimalla kentältä satunnaisesti ilmestyviä kultaisia pähkinöitä. Pelaajalla on käytettävissään neljää erilaista power-up:ia, jotka auttavat pelaajaa eteenpäin. Kuva 5 on kuvankaappaus pelitilanteesta.

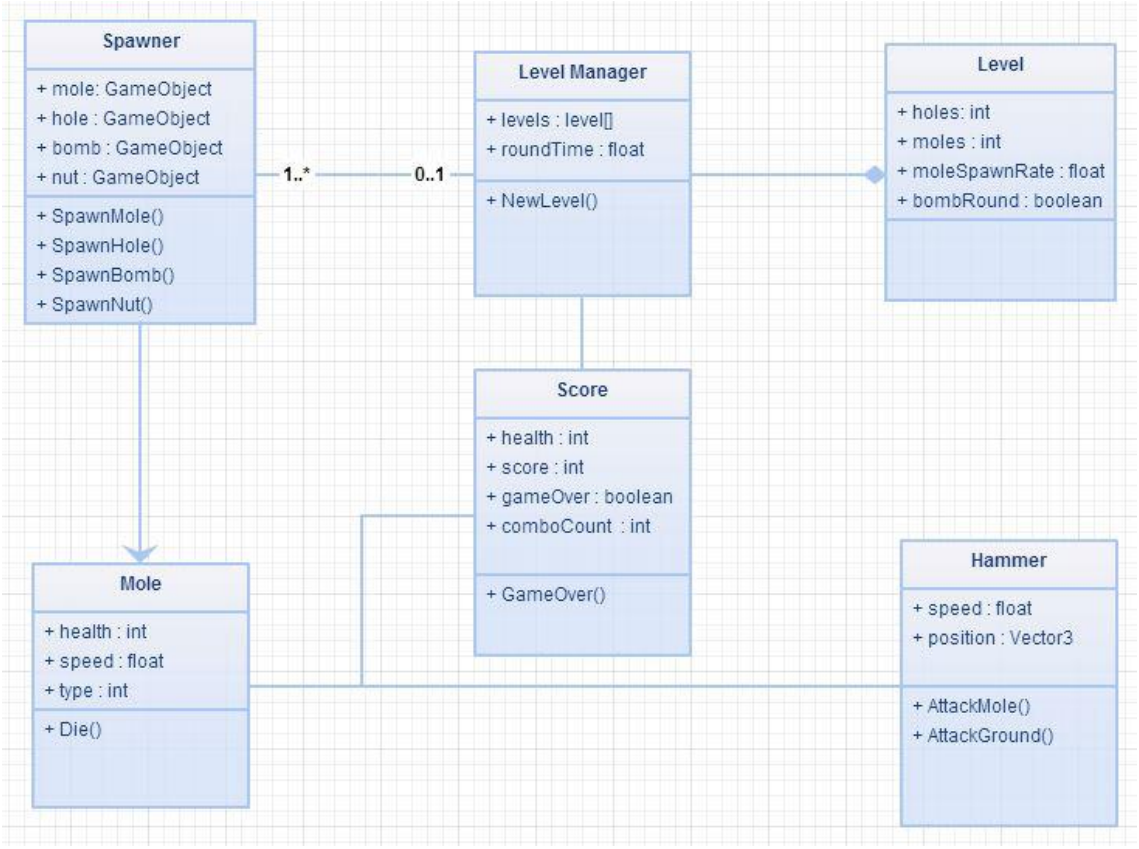


Kuva 5: Whack!

Peli on julkaistu Google Play Storessa sekä SlideME-kaupassa. [21; 22.] Julkaisuhetkellä suurin osa skripteistä oli kirjoitettu JavaScript-kielellä.

4.1.1 Toteutus

Whack!-pelin ohjelmakoodi käsittää julkaisuhetkellä noin 5500 riviä JavaScript-koodia, joka on jakaantunut noin viiteenkymmeneen erilliseen skriptiin. Suurin osa skripteistä on lyhyitä, alle 100-rivisiä skriptejä, jotka kattavat jonkin pienen yksityiskohdan toiminnallisuuden. JavaScript-koodin lisäksi projektissa on liitännäisinä neljä C#-skriptiä, joilla hoidetaan pelaajan pisteiden lähettäminen palvelimelle, asetusten tallentaminen laitteeseen, mainosten näyttäminen ja fontin sävytys.



Kuva 6: Pelin luokkakaavio

Kuva 6 esittää yksinkertaistettua Whack! –pelin luokkakaaviota. Level Manager-luokka pitää sisällään taulukon, joka sisältää Level-olioita. Level-oliot pitävät sisällään tiedot kyseisen kierroksen reikien määrästä, sekä eri peliobjektien syntymisnopeuksista ja todennäköisyyksistä syntyä kyseisellä kierroksella. Level Manager luo kunkin kierroksen alussa Spawner-olion, joka saa kierroksen tiedot Level Managerilta. Tietojen perusteella Spawner-olio luo pelikentälle oikean määrän reikiä, myyriä, pommeja ja pähkinöitä. Kierroksen loputtua Spawner-olio tuhotaan. Spawner-olion luomat myyrät

pitävät tiedon muiden muassa omasta kestävyydestään, nopeudestaan ja tyypistään. Kun vasara hyökkää onnistuneesti myyrään, kutsutaan myyrän kuolinmetodia, joka tuhoaa myyrän ja kerryttää pelaajan pisteitä. Pisteet-skripti pitää sisällään tiedon myös pelaajan elinvoimasta. Mikäli myyrät ehtivät elossa alueen rajoille, menettää pelaaja yhden elinvoimapisteen. Kun elinvoima vähenee nollaan tai sen alle, kutsutaan Pisteet-skriptin GameOver –metodia. Tällöin välitetään Level Managerille tieto pelin loppumisesta. Level Manager pysäyttää pelin etenemisen, tyhjentää kentän ja aktivoi nimikentän pisteiden palvelimelle lähettämistä varten. Todellisuudessa prosessi on paljon monimutkaisempi ja sisältää useita tarkistuksia ja välivaiheita.

4.2 Unityn ohjelmointikielet

4.2.1 JavaScript

JavaScript oli ensimmäinen Unity-pelimoottorin tukema kieli. Se on edelleen laajalti käytössä, ja monen vasta-alkajan ja harrastelijan valinta helpon luettavuutensa ja yksinkertaisen syntaksinsa vuoksi. Myös verkkosovelluksia ohjelmoineet kokevat JavaScriptin tutummaksi muihin kieliin verrattuna. Oikeammin JavaScriptiä Unityssä kutsutaan nimellä UnityScript, sillä sitä on muokattu tuntuvasti alkuperästään Unityä varten. Ensinnäkään, perinteisessä JavaScriptissä ei ole luokkia, vaan perintä tapahtuu dynaamisten olioiden kautta. Tämä tarkoittaa, että olion ominaisuuksia voidaan laajentaa olion luonnin jälkeen JavaScriptin ”prototype”-ominaisuuden avulla. Koodiesimerkki 3 selventää JavaScript-olion laajentamista.

```
function Kone(x) {
   this.kind = ["trukki", "juna", "auto"][x];
}

var c = new Kone(2);
print(typeof c.announce); // "ei määritelty"

Machine.prototype.announce = function() {
   print("Olen "+this.kind+".");
};

print(typeof c.announce);
c.announce(); // tulostaa "Olen auto"
```

Koodiesimerkki 3 : Perintä JavaScriptissä

UnityScriptissä on luokat, mikä helpottaa koodin luettavuutta ja tekee siitä helpommin lähestyttävän muilla kielillä ohjelmoineelle ohjelmoijalle. Myöskään dollarimerkkejä ei tueta UnityScriptissa standardin mukaisen JavaScriptin tapaan. Hieman yllättäen UnityScript muistuttaakin eniten Microsoftin Jscript.NET-kieltä. Tästä eteenpäin insinöörityössä käytetään termiä UnityScript viittaamaan Unityssä käytettyyn JavaScriptiin. [23.]

4.2.2 C#

C#-tuki tuli Unityyn hieman UnityScriptin jälkeen. Se oli aluksi oheiskielen roolissa, mutta suosio on noussut huomattavasti viime vuosien aikana, kun sitä on alettu käyttää peliohjelmoinnissa enemmän myös Unityn ulkopuolella. C# on vasta-alkajalle hieman hankalampi kieli opetella kuin UnityScript, mutta se tarjoaa enemmän mahdollisuuksia ohjelmoijalle. Nämä ominaisuudet eivät ole välttämättömiä valmiin pelin tekemiseen, mutta ovat hyödyllisiä suuremmissa peliprojekteissa ja ammattimaisessa pelikehityksessä. Ominaisuuksia eritellään tarkemmin luvussa 4.4. C# Unityssä eroaa sekin hieman tavallisesta C#:sta, sillä Unity on lisännyt joukkoon liudan omia metodeitaan ja toimintojaan. Nämä löytyvät kaikki Unityn skriptausdokumentaatiosta. [24.]

4.2.3 Boo

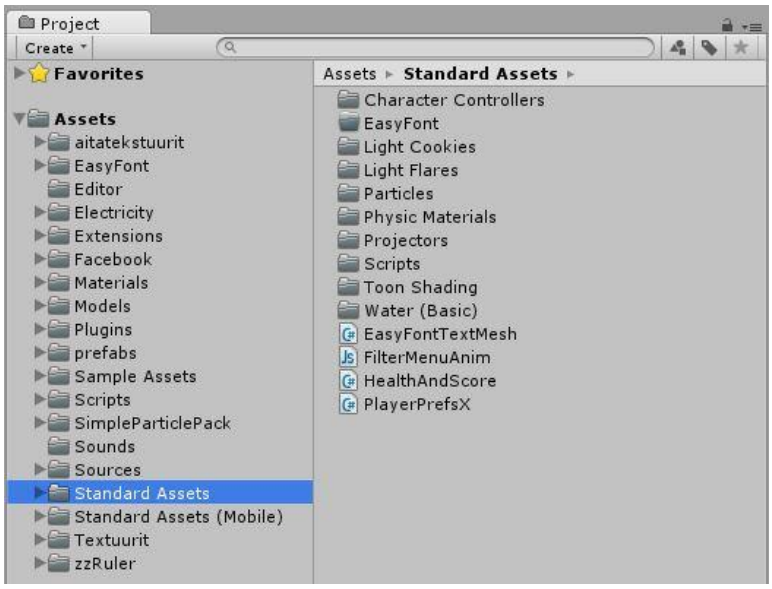
Boo on Unityn kolmas ja vähiten käytetty kieli, jota Unity tukee. Sen syntaksi on samankaltainen kuin Pythonissa, joten sitä käyttävät enimmäkseen muutamat Python-taustaiset ohjelmoijat. Kielen suorituskyky on hyvä, ja kielen tuki tulee ilmaiseksi MonoDevelopin kylkiäisenä, joten se on pysynyt tuettujen kielien joukossa. Verkosta ei ole kuitenkaan saatavilla kovinkaan paljon materiaalia Boo-kieleen liittyen Unityssä, joten kieltä käyttävien peliohjelmoijien osuus pysyy alhaisena. Näiden seikkojen vuoksi tässä insinöörityössä keskitytään vertailemaan UnityScript- ja C#-kieliä keskenään.

4.3 Kielten yhteistoiminta peliprojektissa

Unity tarjoaa mahdollisuuden käyttää useampaa ohjelmointikieltä samassa projektissa. Tämä mahdollistaa esimerkiksi C#-skriptin käyttämisen ja kommunikoinnin UnityScript-skriptin tai skriptien kanssa. Unity-yhteisön keskuudessa ollaan sitä mieltä, että tätä tulisi välttää, sillä kommunikoinnin toteuttaminen on usein hieman hankalaa ja käy yhä hankalammaksi projektin laajentuessa. Suositeltavaa olisikin, että kaikki peliprojektin koodi olisi kirjoitettu samalla kielellä. [25.]

Tällainen tilanne on kuitenkin tullut vastaan omassakin peliprojektissa, jossa suurin osa skripteistä oli julkaisuhetkellä kirjoitettu UnityScriptillä. Osa avaintoiminnoista on sen sijaan kirjoitettu C#:lla. Saumattoman toimivuuden vuoksi harkinnassa oli ensin kääntää kaikki C#-skriptit UnityScriptiin pienemmän työmäärän vuoksi. Tämä on kuitenkin pidemmässä juoksussa ongelmallista, sillä on olemassa asioita, joita C#-kielellä voi tehdä ja UnityScriptillä ei. Näitä eroavaisuuksia käsitellään luvussa 4.4.2. Näin ollen C#:ia ei aina pystytä täydellisesti kääntämään UnityScriptiksi. Ongelma ei ole läheskään yhtä suuri käännettäessä UnityScriptiä C#:ksi.

Unity kääntää eri kansioissa olevat skriptit tietyssä järjestyksessä. Jotta C#-skriptin toimintoja voi käyttää UnityScriptistä, tulee C#-skriptin olla käännettynä ennen kuin UnityScript-skripti käännetään. Tämä saavutetaan, kun asetetaan C#-skripti kansioon, joka käännetään ennen muita. Näitä kansioita ovat Standard Assets, Pro Standard Assets, Plugins ja Editor -kansiot. Kuvassa 7 on esimerkki kansiojärjestelystä. [26.]



Kuva 7: C#-skriptejä Standard Assets-kansiossa.

4.4 C# vastaan UnityScript

Kuten jo luvussa 4.2 mainittiin, on C#-kielen käyttö noussut Unity-peliohjelmoijien keskuudessa huomattavasti viime vuosina. Aluksi verkossa oleva dokumentaatio ja suurin osa opetusmateriaalista oli UnityScriptiä, mutta kelkka on kääntymässä yhä enemmän C#:n suuntaan. Nykyään voitaisiin karkeasti yleistää, että UnityScript on käytössä osittain itsenäisillä harrastelijoilla ja pienillä peliyrityksillä, kun taas C#:ia käyttävät enemmän ammattilaiset ja suuremmat yritykset. Osasyynä tähän on myös se, että UnityScript on käytössä ainoastaan Unityssä, mutta C#-kieltä käyetään laajasti myös Unityn ulkopuolella.

4.4.1 Ulkoiset eroavaisuudet

Koodiesimerkeissä 4 ja 5 on esitetty sama toiminnallisuus toteutettuna C#:lla ja UnityScriptillä.

```
using UnityEngine;
using System.Collections;

public class ExampleSyntax : MonoBehaviour
{
    int myInt = 5;

    int MyFunction (int number)
    {
        int ret = myInt * number;
        return ret;
    }
}
```

Koodiesimerkki 4: C#-esimerkki

```
#pragma strict

var myInt : int = 5;

function MyFunction (number : int) : int
{
    var ret = myInt * number;
    return ret;
}
```

Koodiesimerkki 5: UnityScript-esimerkki

Esimerkeistä huomataan, että luokkajulistus C#:ssa on näkyvä, kun taas UnityScriptissä se on piilotettu. C#:ssa ilmoitetaan muuttujan tyyppi ennen muuttujan nimeä, kun UnityScriptissä tyyppi tulee nimen jälkeen. UnityScriptissä on myös mahdollista luoda muuttuja käyttämällä epäsuoraa (implisiittinen) julistusta jättämällä tyypin kokonaan pois. Tyypin ilmoittamatta jättäminen voi kuitenkin kostautua myöhemmin etenkin isommissa projekteissa, kuten koodiesimerkistä 6 huomataan.

```
var omaMuuttuja;

function Update()
{
    if (omamuuttuja > 0)
    {
        // tee jotain
    }
}
```

Koodiesimerkki 6: Epäsuoran tyypityksen riski

Esimerkissä ohjelmoija on kirjoittanut Update-funktion ehtolauseeseen virheellisesti ”omamuuttuja”, vaikka julistus on muodossa ”omaMuuttuja”. Tällöin kääntäjä luo uuden paikallisen muuttujan, joka alustetaan nollaksi. Näin ollen ehtolause ei koskaan toteudu riippumatta siitä, mikä muuttujan ”omaMuuttuja” arvo kulloinkin on. Kääntäjä ei ilmoita virheestä, ja ohjelmoijalla voi mennä todella kauan ennen kuin virhe on löytynyt. Tämä on hyvä esimerkki siitä, miten UnityScriptillä kirjoitettu koodi vähäsanaisuudestaan huolimatta vaatii loppujen lopuksi usein enemmän työtä ohjelmoijalta, kuin C#:lla kirjoitettu vastaava toiminnallisuus.

UnityScriptissä muuttujat ovat oletusarvoisesti julkisia, kun taas C#:ssa ne ovat yksityisiä. Jos C#-muuttujaa haluaa käsitellä toisesta skriptistä, sen eteen tulee kirjoittaa määre ”public”. UnityScriptissä tätä ei tarvitse tehdä. Julkisten muuttujien arvot ovat myös automaattisesti muokattavissa Unityn käyttöliittymällä.

4.4.2 Erot ominaisuuksissa

Tässä luvussa eritellään seuraavaksi lyhyesti niitä C#-kielen ominaisuuksia, joista Unity-ohjelmoija jää paitsi kirjoittaessaan koodia UnityScriptillä.

Laajennusmetodit

Laajennusmetodeilla voidaan lisätä toiminnallisuuksia tiettyyn luokkaan muokkaamatta itse luokkaa. Esimerkiksi luokka ”Transform” on sisäänrakennettu Unityyn, joten ohelmoija ei pääse muokkaamaan sen lähdekoodia. Mikäli ohjelmoija haluaisi esimerkiksi kirjoittaa ”Transform”-luokkaan metodin, joka alustaisi kyseisen objektin

sijainti-, kierto- ja skaala-arvot alkuarvoihin, tulevat laajennusmetodit tarpeeseen. Koodiesimerkissä 7 on esimerkki luokasta, joka sisätää ”Transform”-luokan laajennusmetodin:

```
using UnityEngine;
using System.Collections;

public static class ExtensionMethods
{
    public static void ResetTransformation(this Transform trans)
    {
        trans.position = Vector3.zero;
        trans.localRotation = Quaternion.identity;
        trans.localScale = new Vector3(1, 1, 1);
    }
}
```

Koodiesimerkki 7: Laajennusmetodi

Sekä luokka että sen metodit tulee julistaa staattisiksi. ”this”-sanan käyttö metodin parametreissä tekee metodista laajennusmetodin staattisen metodin sijaan. Luotua laajennusmetodia voidaan nyt käyttää luokan ulkopuolelta koodiesimerkin 8 mukaan.

```
using UnityEngine;
using System.Collections;

public class SomeClass : MonoBehaviour
{
   void Start () {
      transform.ResetTransformation();
   }
}
```

Koodiesimerkki 8: Laajennusmetodin käyttö.

”ResetTransformation()”-metodi on nyt siis ikään kuin osa ”Transform”-luokkaa.

Goto-käsky

C#-kielellä ohjelmoitaessa voidaan käyttää goto-käskyä. Käsky on jokseenkin ikääntynyt, ja kokematon ohjelmoija saa usein tarpeettomasti sotkettua koodinsa käyttämällä sitä. Yleisesti goto-käskyn käyttöä neuvotaankin välttämään ja etsimään vaihtoehtoinen ratkaisu toiminnallisuudelle. Tämän esitti jo vuonna 1968 tietojenkäsittelytieteilijä Edsger W. Dijkstra kuuluisassa artikkelissaan ”Go To Statement Considered Harmful”. [27.] Oikein käytettynä käskystä voi kuitenkin olla apua tietyissä tilanteissa.

Käännösaika

Unity-yhteisön keskuudessa on tehty tutkimuksia C#-koodin käännösajasta JavaScriptiin verrattuna. Eräs projektinsa UnityScriptista C#:lle muuntanut käyttäjä totesi käännösajan olevan neljä kertaa nopeampi, kuin käännösaika UnityScriptillä. Käännösajan todettiin myös kasvavan eksponentiaalisesti sitä mukaa, kun UnityScript-koodin määrä projektissa kasvoi. Sitä vastoin C#-koodin käännösaika pysyi kutakuinkin vakiona. [28.] Tämä on luonnollisesti varteenotettava seikka etenkin työstettäessä laajempaa peliprojektia.

Nimiavaruudet

C# tukee nimiavaruuksia, mikä on tärkeää isommissa projekteissa. Tuen tärkeys korostuu sitä mukaa, mitä laajemmaksi projekti kasvaa ja mitä useampia ohjelmoijia on työstämässä samaa projektia. Esimerkiksi jos kahdesta ohjelmoijasta toinen kirjoittaa koodia pelaajan ohjaamiseen ja toinen hallitakseen vihollisen liikkeitä, saattavat molemmat nimetä pääskriptinsä luokan ”Kontrolleri”-nimellä. Kun tuotokset yhdistetään projektiin, syntyy ristiriita. Tiettyyn pisteeseen asti tämän voi välttää nimeämällä luokat uudelleen ristiriitojen ilmentyessä. Ongelmat alkavat kasaantua kuitenkin silloin, kun on luotu muuttujia kyseisillä nimillä ja jokainen näistä muuttujista pitäisi nimetä uudelleen. Nimiavaruuksilla pystytään kokoamaan luokkia nimiavaruuden alle koodiesimerkin 9 esittämällä tavalla.

```
namespace Vihollinen{
            public class Kontrolleri1 : MonoBehaviour {
            //toiminnallisuus
            }
            public class Kontrolleri2 : MonoBehaviour {
            //toiminnallisuus
            }
}
```

Koodiesimerkki 9: Nimiavaruudet

Nimiavaruuden luokkia voidaan sitten käyttää ulkopuolelta kirjoittamalla skriptin alkuun määre ”using Vihollinen”.

UnityScriptistä puttuvia ominaisuuksia voisi listata vielä paljon pidemmälle. Kaiken kaikkiaan voidaan kuitenkin päätellä, että vasta-alkajankin voisi olla viisainta käyttää C#-kieltä Unity-ohjelmointiin saman tien, vaikka oppimiskäyrä on hiukan jyrkempi. Jossakin vaiheessa suurempaa projektia UnityScriptin rajoitukset tulevat todennäköisesti vastaan, ja ohjelmoija joutuu vaihtamaan kieltä joka tapauksessa.

4.4.3 Suorituskyky

Unity-ohjelmoijien keskuudessa on ollut eriäviä mielipiteitä Unity-ohjelmointikielten suorituskykyeroista. Joidenkin mielestä suorituskykyerot ovat selviä, kun taas toisten mielestä niitä ei juurikaan ole. [28.]

Ne, joiden mielestä suorituskykyerot ovat olemattomat, perustavat väitteensä sille, että Unity-moottorin kielet ovat kaikki .NET-kieliä. Näin ollen ne käännetään kaikki ensiksi CIL-kieliksi, joka käännetään varsinaisiksi käskyiksi vasta suoritusajossa (.NET-kielten toimintaperiaate, luku 2.3). Käskyt toimivat tämän varjolla siis yhtä nopeasti, eikä koodin alkuperällä ole merkitystä.

Vasta-argumentiksi on esitetty, että UnityScriptin tavukoodikääntäjä ei olisi yhtä tehokas kuin C#:n vastaava, sillä kehittäjät panostavat enemmän C#:n kehitykseen. Painavammat perustelut löytyvät kuitenkin jo todetuista UnityScriptin ja C#:n ominaisuuksista. UnityScript on vapaampi kieli kirjoittaa, ja se tekee paljon asioita ohjelmoijan puolesta. Tämän vuoksi UnityScriptillä tulee helposti kirjoittaneeksi

tehottomampaa koodia kuin voisi olettaa. Voi esimerkiksi syntyä tilanne, jossa epäsuorasti määriteltyä muuttujaa käsitellään aluksi Array-tyyppisenä taulukkona ja muualla koodissa jonain muuna tyyppinä. Tällöin kutsuttaessa kyseiseen muuttujaan esimerkiksi Push()-metodia syntyy virhe, sillä kääntäjä ei tunnista muuttujaa taulukoksi. Epäsuorien tyyppimuuttujien ongelmiin UnityScript tarjoaa mahdollisuuden käyttää ”#pragma strict” -määrettä skriptien yhteydessä, mikä periaatteessa estää ohjelmoijaa käyttämästä epäsuoria julistuksia muuttujissa. Väittämän mukaan C#-koodi on ”#pragma strict” -määreestä huolimatta vielä tiukempaa ja sen vuoksi myös suorituskykyisempää kuin UnityScript.

4.5 Projektin kääntäminen C#-kielelle

Insinöörityön yhtenä tavoitteena oli tutkia C#:n ja UnityScriptin välisiä eroavaisuuksia oman projektin avulla. Tässä luvussa käsitellään peliprojektin kääntämisprosessia C#:iin ja verrataan saatua lopputulosta alkuperäiseen, UnityScriptillä kirjoitettuun tuotokseen.

4.5.1 Kääntäminen

Koko projektin kääntämiseksi C#:lle oli käännettävänä noin 50 yksittäistä UnityScript-skriptiä. Kääntämisen tukena käytettiin Unityn Asset kaupasta saatavaa työkalua, joka kääntää UnityScriptiä C#:lle. [29.] Työkalu osasi kääntää kohtalaisen hyvin yksinkertaisimpia asioita , kuten skriptien perusrakenteet ja muuttujajulistukset. Suuren osan koodista joutui kuitenkin käymään läpi omin käsin.

```
#pragma strict
var health:int;
var score:int;
var highScore:GameObject;
var roundTextScript : RoundText;
var gameOverDelay : float = 4;


function Start () {

    roundTextScript = GameObject.Find("HUD/RoundInfo").GetComponent(RoundText);
}

function Update () {

    if(roundTextScript.gameOver){

        gameOverDelay-=Time.deltaTime;

    }

    if(gameOverDelay<0){

        highScore.SetActive(true);
        roundTextScript.gameObject.SetActive(false);

        var hudArray = FindGameObjectsWithLayer(12);
            for(var i:int=0;i<hudArray.Length;i++){

                for ( var r : Renderer in hudArray[i].GetComponents(Renderer))
                {
                    r.enabled = false;
                }
            }

        gameOverDelay = 4.0;
    }
}
```

Koodiesimerkki 10: Score.js

Kääntöprosessin havainnollistamiseksi on koodiesimerkissä 10 esitetty osa pistelaskuskriptin toiminnallisuudesta UnityScriptillä. Esitetty toiminnallisuus on pelin päättymiseen liittyvä toiminto, jossa tarpeettomat käyttöliittymäelementit piilotetaan pelaajalta ja tuodaan esille nimikenttä, jotta pelaajan tulos voidaan lähettää palvelimelle. Koodiesimerkissä 11 on vastaava toiminnallisuus käännettynä C#-kielelle. Koodiesimerkeistä havaitaan eroja luokkarakenteessa ja muuttujajulistuksissa, sekä taulukoiden ja silmukoiden käsittelyssä. Esimerkeissä on tässä yhteydessä poistettu tarpeettomia muuttujia ja metodikutsuja lukemisen helpottamiseksi.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class score : MonoBehaviour {


int health;
double curScore;
RoundText roundTextScript;
public float gameOverDelay = 4;
GameObject highScore;

void Start (){

    roundTextScript = GameObject.Find("HUD/RoundInfo").GetComponent<RoundText>();

}

void Update (){

if(roundTextScript.gameOver){

    gameOverDelay-=Time.deltaTime;

}

if(gameOverDelay<0){
    highScore.SetActive(true);
    roundTextScript.gameObject.SetActive(false);

    GameObject[]  hudArray= FindGameObjectsWithLayer(12);

        for(int i=0;i<hudArray.Length;i++){

            foreach (Renderer rend in hudArray[i].GetComponents<Renderer>())
            {
                rend.enabled = false;
            }

        }

    gameOverDelay = 4.0f;

    }
}
```

Koodiesimerkki 11: Score.cs

Kuten jo aiemmin todettiin, antaa JavaScript paljon vapauksia ohjelmoijalle esimerkiksi muuttujatyyppien ja taulukoiden suhteen. Näihin liittyviä ongelmia tuli esiin varsin runsaasti kääntämisen yhteydessä, vaikka pääsääntöisesti jokaisessa JavaScript-skriptissä oli käytetty ”#pragma strict”-määrettä.

C# ei tunne UnityScriptin Array-luokkaa, jolla dynaamisten taulukoiden käsittely hoitui yksinkertaisesti. Nämä oli korvattava Javasta tutummilla ArrayList-taulukoilla tai tarvittaessa List-tyyppisillä Generic-luokan listoilla. Lisäksi ongelmia syntyi kielten

eriävien suojauskäsitysten vuoksi. Käännettäessä joutui miettimään uudelleen, mitkä muuttujat tarvitaan julkisiksi ja mitkä yksityisiksi.

Muutamat toiminnallisuudet joutui toteuttamaan hieman monisanaisemmin, kuten osasin odottaakin. Esimerkiksi peliobjektin ”Transform”-komponentin ominaisuuksia ei voinut enää muokata päivitysfunktiossa yhdellä rivillä, kuten UnityScriptissä:

```
transform.position.y = y0+amplitude*Mathf.Sin(hoverSpeed*Time.time);
```

Kyseinen koodi laittaa peliobjektin leijuvaan ylösalaiseen liikkeeseen, kun sijainnin y-komponentti päivitetään joka kerta kuvan piirtyessä ruudulle. C#:lla sijainnin joutui tilapäisesti tallentamaan muuttujaan ennen käyttöä:

```
Vector3 newPos = transform.position;
newPos.y = y0+amplitude*Mathf.Sin(hoverSpeed*Time.time);
transform.position = newPos;
```

Kun perusasioiden kääntäminen rutinisoitui, tuli prosessista melko suoraviivainen. MonoDevelopin parempi tuki C#:lle tuli ilmi hyvin nopeasti. Koodin automaattinen täydentäminen toimi jouhevammin, ja esimerkiksi muuttujan tyypin pystyi näkemään suoraan pitämällä kursoria hetken muuttujan päällä. Vaikka kieli onkin paikoin hieman monisanaisempi kuin UnityScript, niin koodin luettavuus kuitenkin parani ja kokonaiskuvan hahmottaminen helpottui. Taulukosta 2 nähdään yhteenvetona kääntämisestä aiheutuneet rivilukumäärien muutokset viiden suurimman skriptin osalta. Taulukosta havaitaan, että erot eivät ole kovin suuria.

Taulukko 2 : Koodirivien lukumäärät

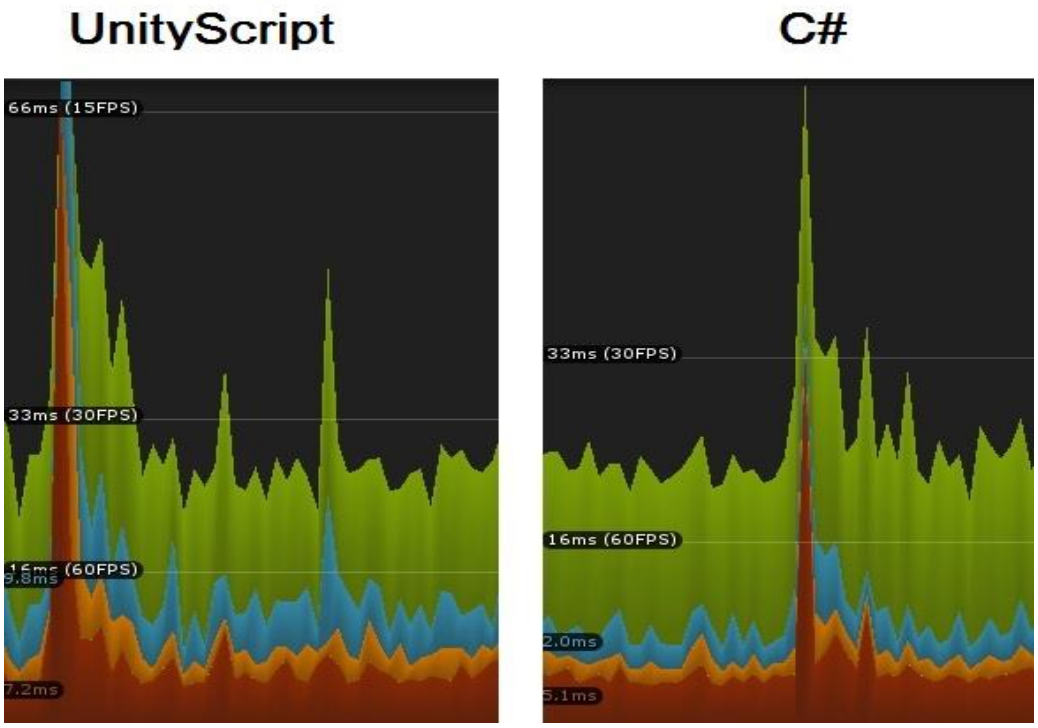
Nimi	**UnityScript**	**C#**	**Erotus**
Mole	513	558	45
MainMenu	459	470	11
Spawner	348	363	15
Hammer	282	289	7

4.5.2 Suorituskykyerojen mittaaminen

Suorituskyky- ja muiden erojen vertailua varten peliprojektista luotiin kaksi kopiota, josta toisessa kaikki toiminnallisuus oli UntyScript-koodia ja toisessa C#-koodia. Kumpaankin projektiin luotiin samanlainen testiasetelma. Asetelmassa valikkonäkymän taakse syntyvien kolojen määrä asetettiin kummassakin kahdeksaan koloon. Myyriä asetettiin tulemaan koloista yhden sekunnin välein.

Tiedot mobiililaitteen suorittimen kuormituksesta saatiin Unityyn sisäänrakennetun profilointiohjelman avulla. Profilointiohjelma on työkalu pelin optimointiin. Se ilmoittaa, kuinka paljon aikaa on käytetty pelin eri osa-alueisiin. Ohjelma asetettiin näyttämään pelkästään mobiililaitteen suoritinkäyttöön liittyvää dataa. Profilointi mobiililaitteella onnistui asettamalla Unityn Android-kääntöasetuksista valinnat ”Development build” ja ”Script Debugging” aktiiviseksi. Testilaitteena käytettiin Samsung Galaxy S II -älypuhelinta.

Liitteet 1 ja 2 esittävät otoksia profilointiohjelman näyttämästä datasta. Liite 1 on UnityScriptillä tuotettu data ja liite 2 vastaava data C#-projektista. Kuva 8 esittää suurennettuja otoksia liitteistä. Molemmissa testeissä käytettiin niin sanottua syväprofilointia, jossa kaikki mahdolliset metodikutsut profiloidaan. Tämä asetus on raskaampi, mutta tuottaa tarkemman tuloksen. Kummassakin otos on noin tuhannennesta ruudusta eteenpäin, sillä pelikenttä on siinä vaiheessa vakiintunut.



Kuva 8: Otokset profilointidatoista

Liitteitä tarkasteltaessa tulee kiinnittää huomiota kuvaajien sinisiin osioihin. Nämä osiot kertovat skriptien aiheuttamasta rasitteesta suorittimelle. Kuvaajia tarkasteltaessa havaitaan, että C#-versiossa skriptien osuus kokonaisrasituksesta on jonkin verran pienempi kuin UnityScript-versiossa. Tämän lisäksi suorittimen kokonaisrasitus on pienempi C#-versiossa, eikä suurinkaan rasituspiikki ei pudota ruudunpäivitystä alle viiteentoista kuvaan sekunnissa. Tällaisia piikkejä on UnityScript-versiossa useita. Ruudunpäivitys pysyy myös selvästi tasaisempana C#-versiossa.

4.5.3 Testin tulokset ja päätelmät

C#- ja UnityScript-projekteja vertailtaessa todettiin C#-version olevan kevyempi ja tasaisempi ruudunpäivitykseltään kuin UnityScript-versio. Ruudunpäivitys oli C#-versiossa lähes koko ajan yli kolmekymmentä kuvaa sekunnissa, kun UnityScript-versiossa tämä arvo alitettiin useita kertoja. On hankala arvioida, onko suorituskykyeron syynä yksinomaan koodin kääntäminen UnityScriptistä C#:lle, tai kuinka suuri osuus eroista aiheutuu koodin kääntämisestä. Mahdollista on, että projektia tuli huomaamatta siivottua kääntämisprosessin aikana myös muilta alueilta,

vaikka olosuhteet kummassakin projektissa pyrittiin pitämään mahdollisimman samankaltaisina.

Oleellisin ja kiistattomin informaatio löytyy kuitenkin liitteiden 1 ja 2 sinisistä alueista. Näitä tarkasteltaessa huomataan, että skriptien osuus suorittimen kokonaisrasitteesta on pienempi C#-projektissa. Skripteistä ei myöskään aiheudu yhtä paljon piikkejä suorituskykyyn. Tämän varjolla voidaan todeta, että C#-koodi Unity-pelimoottorissa on hieman suorituskykyisempää kuin UnityScript-koodi. Ero ei kuitenkaan ole kovin merkittävä. Suurempi vaikutus suorituskykyyn ainakin tässä peliprojektissa löytyykin sellaisista asioista, kuten mallien ja animaatioiden monimutkaisuus sekä tekstuurien yksityiskohtaisuus. Selkeämpiä tuloksia olisi voinut saada luomalla mobiiliprojektin, jossa 3D-grafiikalla olisi vähäisempi merkitys ja skripteissä olisi ollut enemmän raskasta laskentaa.

Varmuudella voidaan sanoa, että projektin kääntäminen C#-kielelle ehdottomasti kannatti. Peli pyörii käännöksen jälkeen sulavammin, mikä tarkoittaa laajempaa asiakaskuntaa mobiilipelimarkkinoilla. Lisäksi koodia on helpompi ylläpitää johdonmukaisemman ulkoasun ja työkalujen paremman tuen takia. Positiivista oli myös se, että apuja ohjelmointiin löytyi nyt myös Unity-yhteisön ulkopuolelta. Yleiset C#-ohjelmointiin liittyvät vinkit, ja dokumentaatiot käyvät Unity-ohjelmointiin usein sellaisenaan. Näin ei ole UnityScriptin kanssa, sillä UnityScriptillä ei ole käyttöä Unityn ulkopuolella.

5 C# ja mobiilipelit tulevaisuudessa

C# on vakiinnuttanut asemaansa mobiilipeliohjelmointikielten joukossa koko 2010-luvun ajan. Unityn, XNA:n ja MonoGamen kaltaiset työkalut ovat mahdollistaneet C#-kehityksen useille mobiilialustoille. Työkaluja päivitetään tasaisesti, ja myös uusia apuvälineitä syntyy tarpeen mukaan kehittäjäyhteisöjen toimesta.

Mobiilipeleissä koodin suorituskyky on ollut kieltä valittaessa ensiarvoisen tärkeää kohdelaitteen heikon suorituskyvyn vuoksi. Tämän takia mobiilipelikehittäjien valinta on ollut pitkään C++ ylivertaisen tehokkuutensa vuoksi. Mobiililaitteiden tullessa päivä päivältä tehokkaammiksi, alkaa kielen valinnassa näkyä kuitenkin samaa trendiä kuin PC- ja konsolipuolella. Tehokkaammille mobiililaitteille tehdään yhä monimutkaisempia pelejä ja monet haluavatkin usein uhrata pienen osan koodin suorituskyvystä

saadakseen helpommin ylläpidettävää ja luettavaa koodia. Tämä on yksi syy, miksi monet peliyritykset suosivat nykyään C#:aa C++-kielen sijaan.

Taulukko 3 : Unity-kehittäjille teetetyt kyselyt vuosilta 2010 ja 2013. [30.]

Kieli	2010	2013
Vain Boo	4.35%	3.28%
Vain C#	31.10%	40.10%
Vain JavaScript	31.77%	27.92%
Boo & C#	2.68%	1.97%
Boo & JavaScript	0.33%	0.52%
JavaScript ja C#	28.76%	25.29%
Boo, C# ja JavaScript	1.00%	0.92%

Unity-kehittäjien keskuudessa ollaan niin ikään kallistumasa C#:n puolelle. Taulukko 3 esittää vuosina 2010 ja 2013 Unity-kehittäjille teetetyn kyselyn tuloksia heidän käyttämistään kielistä. Taulukosta havaitaan, että pelkästään C#:lla ohelmoivien osuus on noussut kolmessa vuodessa noin kymmenen prosenttia, kun muita kieliä käyttävien osuudet ovat laskeneet.

Kaiken kaikkiaan C#-kielen tulevaisuus mobiilipelikehityksessä näyttää siis varsin valoisalta. Kielen jatkuva kehittäminen Microsoftin toimesta sekä alati kasvava käyttäjäkunta antavat sille hyvät eväät vuosikausiksi eteenpäin. Onkin mielenkiintoista nähdä, miten tilanne kehittyy ajan kuluessa. Vielä C++ ei ole väistymässä ykköskielen paikalta, mutta mikä on C++:n ja C#:n asema mobiilipeleissä 10 vuoden kuluttua?

6 Yhteenveto

Insinöörityössä esiteltiin C#-kieltä ja sen käyttömahdollisuuksia mobiilipeliohjelmoinnissa. Työssä käytiin läpi, miten pelinkehitys C#:lla käytännössä toimii eri mobiilialustoilla ja esiteltiin tähän liittyviä työkaluja. Lisäksi tutustuttiin tarkemmin C#-kielen ominaisuuksiin Unity-ympäristössä oman peliprojektin avulla. Käytännön tutkimuksena käännettiin UnityScript-projekti kokonaan C#:lle ja vertailtiin lopputulosta alkuperäiseen.

Käännösprosessi oli kaiken kaikkiaan onnistunut, ja tulokset yllättivät positiivisesti. Käännöksen jälkeen peli pyöri sulavammin ja lähdekoodin ylläpidettävyys parani. Oppimisprosessina projekti oli antoisa ja opetti monia uusia asioita C#-peliohjelmoinnista, millä on käyttöä niin työelämässä kuin omissakin projekteissa. Työ tarjoaa hyödyllistä tietoa kaikille, jotka ovat kiinnostuneet mobiilipeliohjelmoinnista, tai peliohjelmoinnista yleensäkin. Kaiken kaikkiaan C#-kielen opettelua ja käyttöä voi lämpimästi suositella kaikille mobiilipelikehittäjille. Ammattimaisen pelikehittäjän on kuitenkin suositeltavaa osata myös C++-kieltä, sillä ohjelmoinnin taustalla tapahtuvat seikat ja kokonaiskuva hahmottuvat sen hallitsemisen myötä ohjelmoijalle paremmin kuin pelkän C#-osaamisen avulla.

Lähteet

1 InformIt. A brief history of mobile software develompent. Verkkodokumentti <http://www.informit.com/articles/article.aspx?p=1388959> Luettu 20.4.2014.

2 Wikipedia. Symbian OS. Verkkodokumentti <http://fi.wikipedia.org/wiki/Symbian_OS> Luettu 20.4.2014.

3 Wikipedia. Oak. Verkkodokumentti < http://en.wikipedia.org/wiki/Oak_(programming_language)> Luettu 20.4.2014.

4 Stackoverflow. What's the early history of the .NET-framework? Verkkodokumentti < http://stackoverflow.com/questions/1083368/whats-the-early-history-of-the-net-framework> Luettu 20.4.2014.

5 C Sharp programming language. Verkkodokumentti <http://en.wikipedia.org/wiki/C_Sharp_(programming_language)> Luettu 4.4.2014.

6 Syrjälä, Santtu. C# Pikakurssi. Kandidaatintutkielma <http://users.jyu.fi/~vesal/kurssit/winohj/csharp/seminaari05/CS_Pikakurssi.pdf> Luettu 4.4.2014.

7 Developertech. C# language of the year. Verkkodokumentti <http://www.developer-tech.com/news/2013/jan/03/pypl-crowns-c-2012s-language-year> Luettu 20.4.2014.

8 Xamarin blog. Verkkodokumentti <http://blog.xamarin.com/eight-reasons-c-sharp-is-the-best-language-for-mobile-development> Luettu 7.4.2014.

9 Windows Phone SDK. Verkkodokumentti <http://www.microsoft.com/en-us/download/details.aspx?id=35471> Luettu 4.4.2014.

10 Getting started with XNA Game Studio Development. Verkkodokumentti < http://msdn.microsoft.com/en-us/library/bb203894.aspx> Luettu 7.4.2014.

11 Gamasutra. XNA is dead. Verkkodokumentti <http://www.gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php> Luettu 20.4.2014.

12 StackExchange Game Development. Verkkodokumentti. <http://gamedev.stackexchange.com/questions/22292/what-is-the-future-of-xna-in-windows-8-or-how-will-manged-games-be-developed-in> Luettu 7.4.2014.

13 Wikipedia. Xamarin. Verkkodokumenti <http://en.wikipedia.org/wiki/Xamarin> Luettu 20.4.2014.

14 MonoGame. Verkkodokumentti. <http://en.wikipedia.org/wiki/MonoGame> Luettu 7.4.2014.

15 Download Xamarin. Verkkodokumentti <http://xamarin.com/android> Luettu 8.4.2014.

16 Download Monogame. Verkkodokumentti <http://monogame.codeplex.com> Luettu 8.4.2014.

17 MonoGame tutorials. Verkkodokumentti <https://github.com/mono/MonoGame/wiki/Tutorials> Luettu 9.4.2014.

18 MonoGame Hello World on Mac, OSX and Xamarin Studio. Verkkodokumentti. <http://jaquadro.com/2013/09/monogame-hello-world-on-mac-os-x-and-xamarin-studio> Luettu 9.4.2014.

19 Unity wikipedia. Verkkodokumentti <http://en.wikipedia.org/wiki/Unity_Technologies> Luettu 9.4.2014.

20 Delta Engine. Verkkodokumentti <http://deltaengine.net> Luettu 10.4.2014.

21 Google Play. Whack!. Verkkodokumentti <https://play.google.com/store/apps/details?id=com.futureactor.whack> Luettu 20.4.2014.

22 SlideME. Whack!. Verkkodokumentti <http://slideme.org/application/whack> Luettu 20.4.2014.

23 Unity community wiki. UnityScript versus JavaScript. Verkkodokumentti. <http://wiki.unity3d.com/index.php/UnityScript_versus_JavaScript> Luettu 10.4.2014.

24 Unity scripting reference. Verkkodokumentti. <https://docs.unity3d.com/Documentation/ScriptReference> Luettu 10.4.2014.

25 Unity forums. Make JS and boo deprecated in Unity4. Verkkodokumentti <http://forum.unity3d.com/threads/131902-Make-JS-and-Boo-deprecated-in-Unity-4-And-non-functional-in-Unity-5/page12?p=1037432&viewfull=1#post1037432> Luettu 11.4.2014.

26 Unity Script Compile Order Folders. Verkkodokumentti. <https://docs.unity3d.com/Documentation/Manual/ScriptCompileOrderFolders.html> Luettu 10.4.2014.

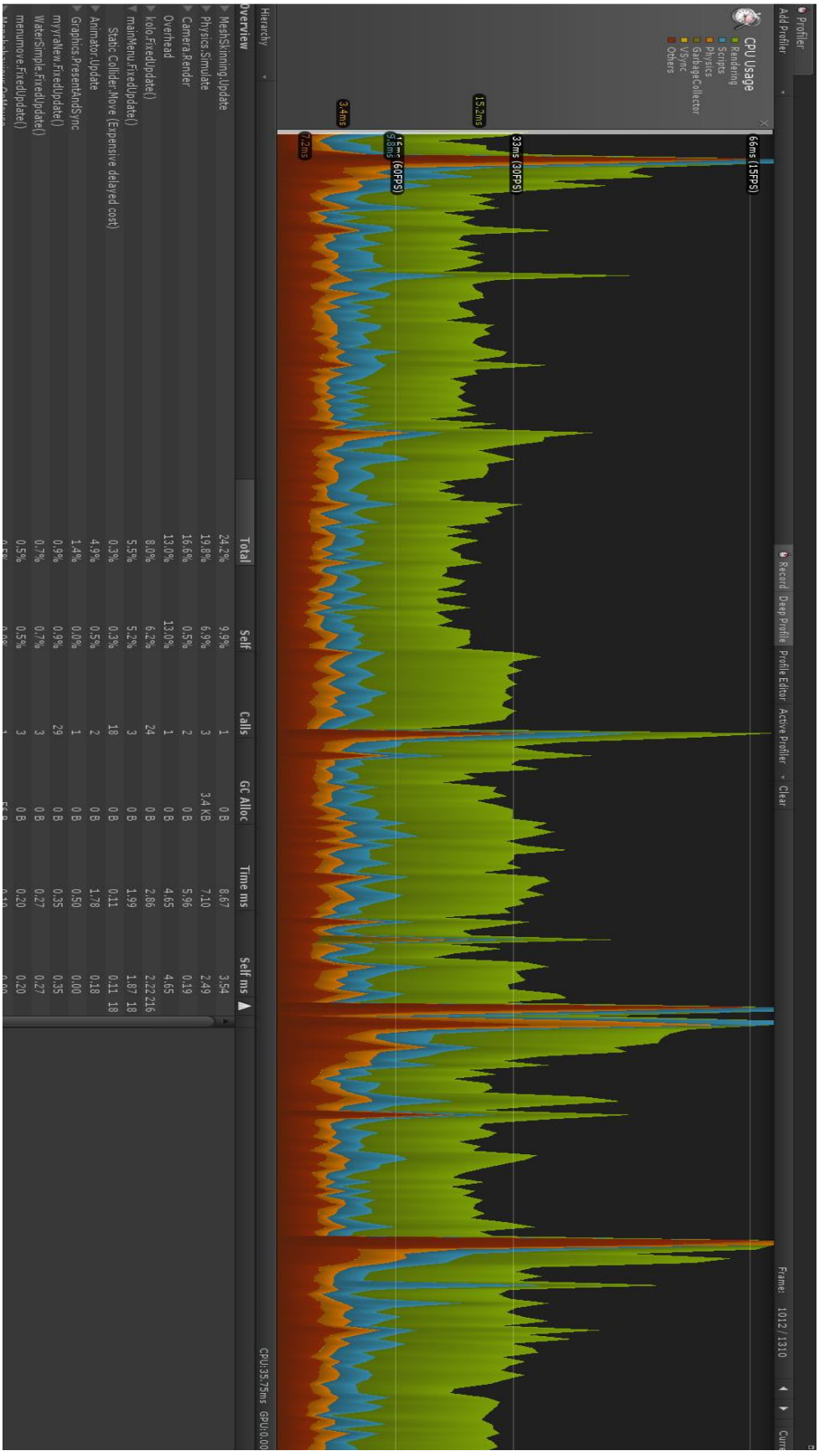
27 Texas University. A Case against the GO TO Statement. Verkkodokumentti <http://www.cs.utexas.edu/~EWD/transcriptions/EWD02xx/EWD215.html> Luettu 20.4.2014.

28 Unity3D answers. Is there a performance difference between JS and CS? Verkkodokumentti. <http://answers.unity3d.com/questions/7567/is-there-a-performance-difference-between-unitys-j.html> Luettu 12.4.2014.

29 Unity Asset Store. JS to C# Script Converter. Verkkodokumentti. <https://www.assetstore.unity3d.com/#/content/176> Luettu 20.4.2014.

30 Unity forums. Boo, C# and JavaScript experiences in Unity. Verkkodokumentti. <http://forum.unity3d.com/threads/18507-Boo-C-and-JavaScript-in-Unity-Experiences-and-Opinions/page5> Luettu 12.4.2014.

UnityScript-profilointidata



C#-profilointidata