Opinnäytetyö (AMK)

Tietotekniikka

Mediatekniikka

2015

Juho Makkonen

AMPUMAHIIHTOPELIN TOIMINTOJEN TOTEUTTAMINEN MOBIILILAITTEILLE

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Mediatekniikka

2015 | 41

Ohjaaja: Yliopettaja, FT, Mika Luimula

Juho Makkonen

AMPUMAHIIHTOPELIN TOIMINTOJEN TOTEUTTAMINEN MOBIILILAITTEILLE

Tämän opinnäytetyön tarkoituksena oli toteuttaa ampumahiihtokilpailun ampumaosuutta kuvaavan pelin toiminnot C#-kielellä. Teoriaosassa tutkitaan sijaintiin pohjautuvan sääominaisuuden toteutustapoja ja haasteita. Lopuksi käsitellään pelin mahdollista jatkokehitystä.

Toimiakseen sääominaisuus tarvitsee verkkoyhteyden, GPS:n ja sopivan ohjelmointirajapinnan. Data noudetaan verkosta sopivassa muodossa. Mobiililaitteen GPS lähettää sijainnin tiedot pelille, ja säätiedot liitetään verkosta peliin ohjelmointirajapinnan avulla. Ominaisuuden voi vaihtoehtoisesti toteuttaa ilman verkkoyhteyttä. Silloin tuulen nopeus ja suunta arvottaisiin.

Lopputuloksena toteutettu demo sisältää päävalikon lisäksi yhden makuuammunta- ja yhden pystyammuntatason, jotka molemmat pelaaja käy läpi kahdesti. Ampumahiihdon sääntöjen mukaan pelaaja ampuu viisi laukausta jokaisella tasolla. Lopuksi peli näyttää osumat ja laskee ampumiseen käytetyn ajan, ja mahdolliset ohilaukauksien aiheuttamat sakkominuutit. Kehitettäessä peliä eteenpäin, se voi ottaa huomioon mobiililaitteen kallistukset ja skaalautua oikein kaikille resoluutioille.

ASIASANAT:

pelinkehitys, Unity3D, GPS, ohjelmointirajapinta, C#

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Digital Media

2015 | 41

Instructor: Principal Lecturer, Ph. D. Mika Luimula

Juho Makkonen

DEVELOPMENT PROCESS OF A BIATHLON GAME FEATURES FOR MOBILE DEVICES

The purpose of this thesis was to develop a demo of a biathlon game for mobile platforms. This demo only covers the shooting rounds. The thesis also involves research on how to develop an interactive location-based weather feature and considerations for the further development of the game. The scope of this thesis covers to the programming of the game features using C#.

The weather feature would require a suitable Application Programming Interface, internet connection and GPS. The game retrieves GPS coordinates and the weather data of a player's location. Those are then used as variables in the game. Alternatively the data retrieval could be carried out without wireless connection and GPS. In that case, the speed value and direction of wind would both be random.

This game demo includes the main menu and one level for prone position and one for standing position. Players will play through both levels twice and will shoot 5 times on each level, based on rules of biathlon. Lastly the game will show how many shots hit the target, what was the total time used for it and how many penalty minutes the player got, if any. When the game is developed further, it could remain properly functional also when the device is in portrait position and scale properly to any resolution.

KEYWORDS:

Game development, Unity3D, GPS, application programming interface, C#

SISÄLTÖ

Kuvat

Taulukot

Koodit

KÄYTETYT LYHENTEET

API	Ohjelmointirajapinta
C#	Microsoftin kehittämä C++ -pohjainen ohjelmointikieli
GPS	Maailmanlaajuinen paikallistamisjärjestelmä (Global Positioning System)
GUI	Graafinen käyttöliittymä (Graphical User Interface)
HDR	High Dynamic Range. Kuvantamistapa, jossa muokattu kuva sisältää laajennetun kirkkausalueen eli dynamiikan
HTML	Hypertekstin merkintäkieli (Hypertext Markup Language)
IDE	Integroitu ohjelmointiympäristö (Integrated development environment)
JavaScript	Oliopohjainen ohjelmointikieli
JSON	Tiedostomuoto tiedonvälitykseen (JavaScript Object Notation)
PhysX	Reaaliaikainen fysiikkamoottori
QuadHD	2560 x 1440 resoluutio
Wi-Fi	Langattoman verkon kaupallinen nimitys ja Wi-Fi Alliancen tavaramerkki
XML	Merkintäkieli (Extensible Markup Language)
2D	Kaksiulotteinen grafiikka
3D	Kolmiulotteinen grafiikka

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on käydä läpi ampumahiihtopelin demon toimintojen kehitysprosessi alusta loppuun. Pelin pääasiallisena alustana tulevat olemaan mobiililaitteet. Oma osuuteni projektissa on rajattu pelin toimintojen ohjelmointiin. Työssä tutkitaan myös, miten sääpalveluista ja paikkatiedoista saatavaa dataa voisi käyttää autenttisemman pelikokemuksen luomiseen. Oppimistavoitteita ovat pelinkehitysprosessin parempi tuntemus ja toimeksiantajan toivomien ominaisuuksien toteuttaminen toimivalla ja käytännöllisellä tavalla.

Projektin toimeksiantaja on RightSpot -niminen yritys Salosta, ja projekti toteutettiin yhteistyössä Turku Game Labin kanssa. Pelin tarkoitus on kuvata ampumahiihtokisojen ampumaosuutta pelillisessä muodossa ja se on suunnattu henkilöille, jotka eivät harrasta ampumahiihtoa tai urheiluammuntaa. Itse hiihto-osuus jätettiin pelistä pois.

Projekti toteutettiin Unity3D- ja Visual Studio 2013 -ohjelmilla ja ohjelmoinnissa käytettiin C# -kieltä. Myös potentiaalisia kehitystyökaluja vertaillaan keskenään. Vertailen esimerkiksi niiden ominaisuuksia ja muita päätökseen vaikuttaneita seikkoja. Työssä käydään teoreettisesti läpi myös pelin vaatimia toimintoja ja objekteja, kuten Unity3D:n törmäystunnistimia. Pelin ei ole tarkoitus olla huomattavan realistinen, vaan helppokäyttöinen, siinä pelaajan ei tarvitse ottaa huomioon kaikkea sitä, mitä ammattilaiset joutuvat ottamaan. Pelaaja ohjaa yhtä kilpailijoista ampumapaikalla ja ampuu viiteen tauluun neljä kertaa. Peli näyttää lopuksi, kuinka monta osumaa pelaaja sai, paljonko aikaa kului ja montako sakkominuuttia kertyi. Peli ottaa myös huomioon pysty- ja makuuammunnan erot aseen heilunnan suhteen.

Työssä käsitellään demon kehitysprosessi alkaen päävalikon toiminnoista ja edetään pelinäkymän vaatimiin toimintoihin, kuten ampumatoimintoon ja siihen, miten tauluja kuvaavat peliobjektit reagoivat osumiin. Lopuksi selvennetään sitä miten lopputulokset lasketaan ja näytetään pelaajalle.

Peliin oli aluksi tarkoitus sisällyttää hieman vaihtelevia sääolosuhteita, jotka hyödyntäisivät sääpalveluista ja paikkatiedoista saatavaa dataa. Pelaajan sen hetkisen paikan, esimerkiksi Turun, sää vaikuttaisi myös pelin säähän. Mobiililaitteiden rajoitukset on otettava huomioon ja raskaampia visuaalisia elementtejä, kuten lumisadetta ei ollut tarkoitus peliin sisällyttää. Ajanpuutteen vuoksi sitä ei pelin demoon toteuteta, mutta siitä huolimatta työssä tutkitaan, miten edellä mainitun ominaisuuden voisi toteuttaa ja mitkä ovat sen haasteet. Prosessi käsitellään alkaen laitteen sijainnin määrittämisestä ja edetään tarvittavan datan hakemisprosessiin ja sen integrointiin itse peliin. Sen ohessa työssä käydään läpi sää- ja paikkatietopalveluiden teoriaa sekä sitä, miten ja mihin tarkoituksiin niitä nykyään käytetään peleissä. Lopuksi pohditaan, miten peliä voisi kehittää eteenpäin.

2 PELIMOOTTORIN JA KEHITYSTYÖKALUJEN VALINTA

2.1 Pelimoottorit

2.1.1 Unity3D

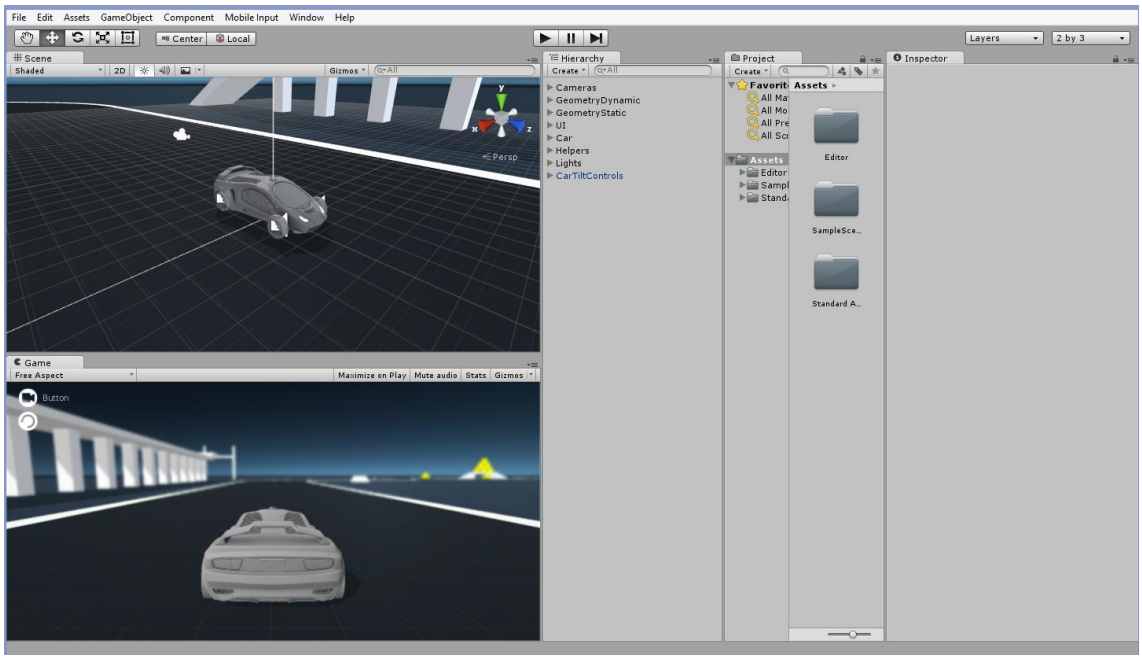
Unity3D on Unity Technologiesin kehittämä monialustainen pelimoottori ja pelinkehitykseen tarkoitettu ohjelma. Se toimii Mono -nimisellä avoimen lähdekoodin .NET-kehitysalustalla. Se sisältää sisäänrakennetun integroidun ohjelmointiympäristön (IDE) ja sillä voidaan kehittää niin kaksi- kuin kolmiulotteisia pelejä tietokoneille, konsoleille ja mobiililaitteille. Unity3D tukee muun muassa Windows, Linux ja Mac OS -käyttöjärjestelmiä sekä Android, iOS ja Windows Phone 8 -mobiilikäyttöjärjestelmiä (Windows Phone 7.x mobiilikäyttöjärjestelmiä ohjelma ei tue). Pelien kehittäminen mobiililaitteille Unity3D:n ilmaisversiolla on ollut mahdollista toukokuusta 2013 lähtien. Ohjelman uusin versio on 5.01, ja sen myötä Unity3D tukee viime sukupolven pelikonsolien lisäksi myös PlayStation 4 ja Xbox One -konsoleita. Tällä hetkellä Unity3D tukee kaikkiaan 21 eri alustaa. [1]

Version 5 myötä Unity3D sai myös monia uusia ominaisuuksia. Tärkein uusi ominaisuus on physically-based shading. Se mahdollistaa, että pelin sisältämät objektit näyttävät realistisilta kaikissa tilanteissa ja reagoivat oikein eri valaistuksiin. Kun objektit käyttäytyvät fyysisesti oikein, pelaaja kykenee tulkitsemaan esimerkiksi metallin aina metalliksi. Muita uusia ominaisuuksia ovat myös uudistettu reaaliaikainen globaali valaistus, äänien muokkaustyökalut, PhysX 3.3 -fysiikkatyökalut, HDR (High Dynamic Range) valaistus ja animaatiojärjestelmän uudistukset. [2]

Saatavilla on sekä ilmainen versio että maksullinen Unity3D Pro, joka on ominaisuuksiltaan kattavampi. Maksullinen versio ei myöskään sisällä rajoituksia sillä toteutettujen pelien kaupallistamisessa. Ilmaisversiolla kehitettyjä pelejä ei saa

lisensoida, jos pelin on kehittänyt organisaatio, jonka vuosittainen liikevaihto on enemmän kuin 100 000 dollaria. Pro-versio maksaa 75 dollaria kuussa.

Unity3D:n editorin voi jakaa viiteen osaan, joita ovat Hierarchy, Project Browser, Inspector, Scene ja Game. Hierarchy sisältää luettelon kaikista pelin objekteista. Project Browser sisältää projektin assets-tiedostot eli projektin käyttämät tiedostot. Inspector näyttää aktiivisen eli valitun peliobjektin sisältämät ominaisuudet. Sen avulla voi myös asettaa komponenttien julkisille muuttujille arvoja ilman ohjelmakoodiin koskemista. Scene-osiossa näkyvät pelin objektit niin kutsutussa virtuaalisessa näkymässä. Game-osio näyttää itse pelinäkymän aktiivisen kameran läpi. [3]



Kuva 1. Unity3D:n editori

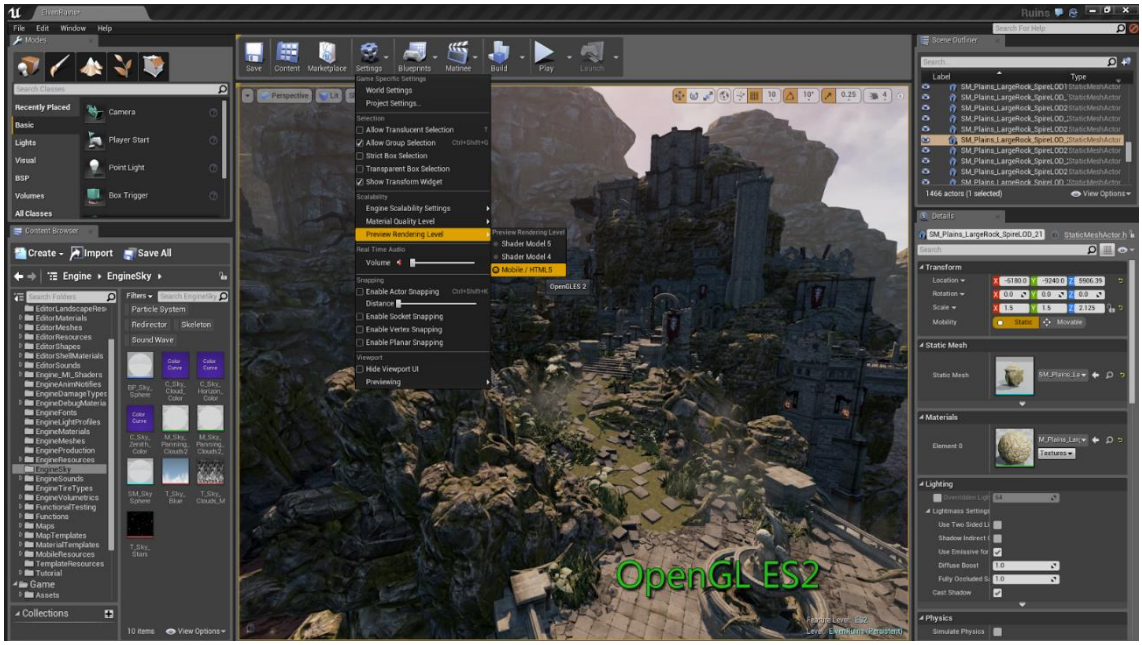
2.1.2 Unreal Engine

Tässä tapauksessa yksi varteenotettava vaihtoehto Unity3D:lle on Epic Gamesin Unreal Engine 4, joka skaalautuu hyvin myös mobiilialustoille kehitettäviin peleihin. Sen uusin versio on 4.7. Se sisälsi aiemmin kuukausimaksun, mutta siitä on

nykyään olemassa myös ilmainen vaihtoehto. Julkaistujen ilmaisversiolla kehitettyjen pelien tuotoista täytyy maksaa 5 % rojalteina Epic Gamesille. Pelimoottori mahdollistaa niin 2D kuin 3D -pelien kehittämisen joko PC:lle, pelikonsoleille tai mobiililaitteille. Työkaluiltaan se on kattava ja mahdollistaa niin käyttöliittymän, pelin eri tasojen, animaatioiden, fysiikoiden kuin verkko-ominaisuuksienkin kehittämisen. Blueprint-ominaisuuden avulla pelin toiminnot voidaan rakentaa visuaalisesti, ilman varsinaista ohjelmointia.

Unreal Engine tukee DirectX 11:sta ja 12:sta ja sisältää työkalut tuli, savu, lumi ja tomuefektien luomista varten. Se mahdollistaa myös pelin hahmojen tekoälyn laajan hienosäädön, ja sisältää oman editorin äänitiedostoja varten. Pelien objektit ja materiaalit voidaan hienosäätää näyttämään aidolta ja reagoimaan oikein kaikissa tilanteissa (physically-based shading). Pelimoottorin työkalut mahdollistavat myös monien eri animaatioiden luomisen peliin ja hahmojen liikkeiden muokkaamisen. Laajojen ulkotilojen kehittäminen ja maaston sekä kasvillisuuden muokkaaminen on myös mahdollista. Pelien visuaalista ilmettä voi käsitellä eri tehosteiden kuten bloomin tai ambient occlusionin avulla. [6] Ambient occlusion on varjostintekniikka, jolla simuloidaan valon vuorovaikutusta eri pintojen kanssa. [7] Bloom simuloi linssien toimintaa ja kirkkaiden valojen leviämistä ympäröiviin objekteihin. [8]

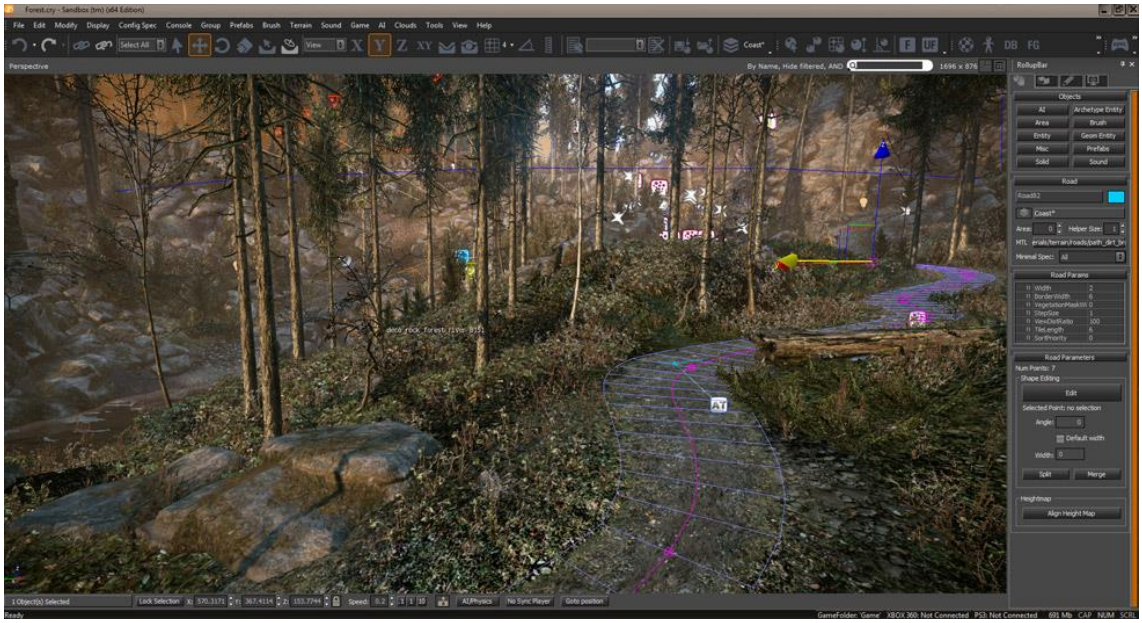
Unreal Engine on rakennettu C++ -kielellä ja sitä käytetään myös pelinkehityksessä. Se sisältää tuen myös virtuaaliselle todellisuudelle, kuten Oculus Riftille. Kehittäjillä on käytössään myös markkinapaikka, josta voi ladata sisältöä, kuten taidetta tai ääniä peleihin tai myydä omia luomuksiaan muille. Se muistuttaa paljon Unityn Asset Storea.



Kuva 2. Unreal Engine 4:n editori [9]

2.1.3 CryEngine

Crytekin kehittämä CryEngine 3 ei ole vaihtoehtona yhtä varteenotettava kuin Unreal Engine. Se mahdollistaa kyllä näyttävän grafiikan, mutta niin näyttävä visuaalinen ilme ei ollut projektin pääasiallinen tavoite. Pelimoottori on myös edellä mainittuja vaihtoehtoja haastavampi käyttää, ja sitä käyttävä yhteisö on pienempi. Se ei myöskään tue Windows Phone -käyttöjärjestelmää ja maksaa 9,90 dollaria kuussa. Uusin versio pelimoottorista on 3.7. CryEngine sisältää esimerkiksi ambient occlusionin, fysiikkatyökalut, tessellaation ja tekoälyn muokkaustyökalut. Tessellaation avulla voi hallinnoida objektien polygonirakennetta eli dynaamisesti lisätä tai vähentää yksityiskohtia 3D-objekteista reaaliajassa. CryEngine sisältää kaikki edellä mainitut Unity3D:n ja Unreal Enginen sisältämät ominaisuudet.



Kuva 3. CryEngine 3:n editori [10]

2.2 Microsoft Visual Studio

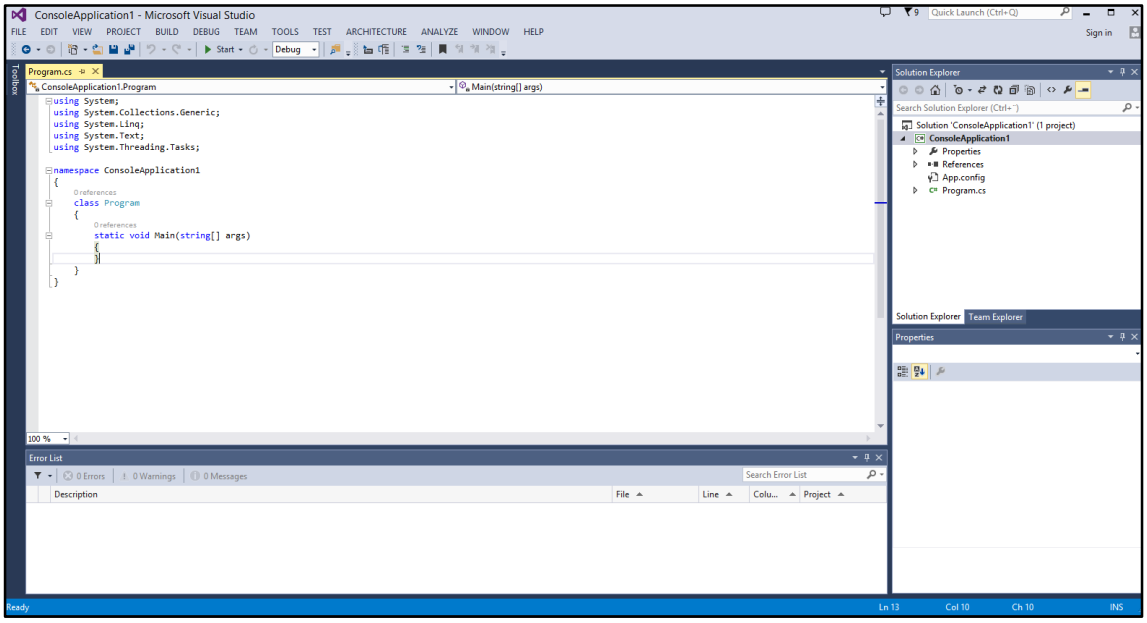
Microsoftin Visual Studio on ohjelmankehitysympäristö ja siinä on mahdollista valita useista eri ohjelmointikielistä, kuten C# ja C++. Siinä voi kehittää sovelluksia Windows-, web-, pilvi- ja mobiilialustoille. Sitä voi myös käyttää yhdessä Unity3D:n kanssa osana pelikehitystä ja siihen voi liittää pelinkehitystä helpottavia laajennuksia. Se sisältää itse editorin, virheenkorjaustyökalun (debug) ohjelmakoodin testausta varten ja graafisten käyttöliittymien kehittämiseen soveltuvat työkalut. Esimerkiksi WPF -työkalut mahdollistavat graafisten sovellusten kehittämisen. Ohjelman sisältämiä ominaisuuksia ovat esimerkiksi automaattinen koodintäydennys muuttujille, funktioille ja metodeille. Koodiin voi asettaa myös kirjainmerkkejä nopeampaa navigointia varten. Visual Studio sisältää myös Office, LightSwitch ja Node.js -työkalut. Uusin versio on Visual Studio 2013 ja siinä on paranneltu web -kehityksen ominaisuuksia, lisätty editoriin uusia metadataan perustuvia ominaisuuksia. Se sisältää myös tuen Windows 8.1 -käyttöjärjestelmälle. Visual Studio 2015 julkaistaan vuoden 2015 aikana.

Ohjelmasta on olemassa 4 versiota eli Professional, Premium, Ultimate ja Test Professional. Professional on niin kutsuttu perusversio ja Premium on tarkoitettu

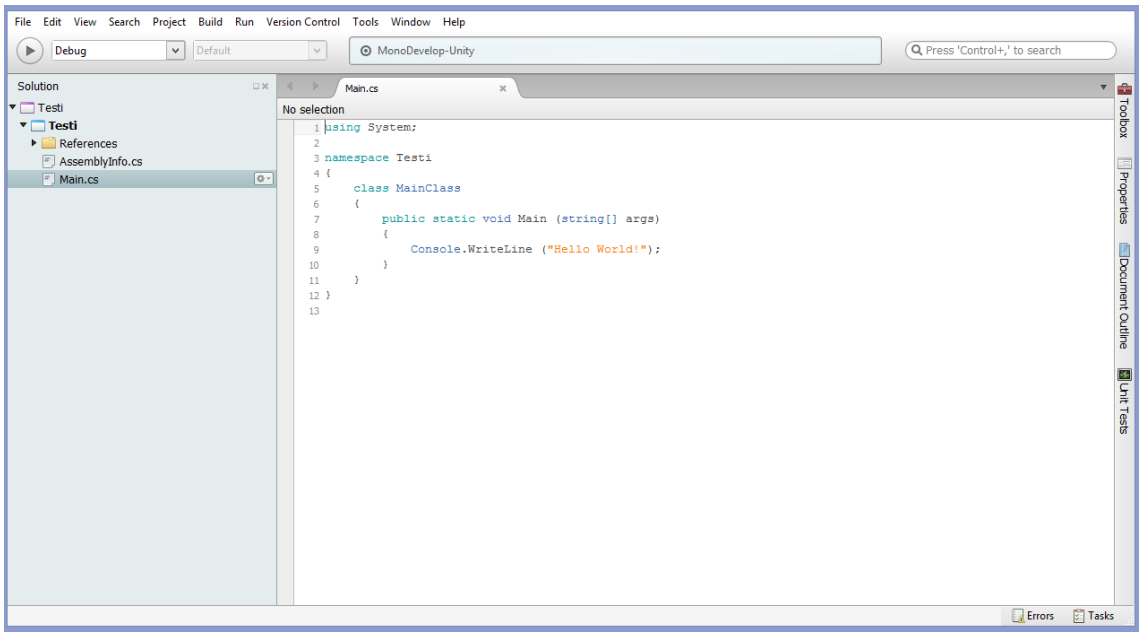
ohjelmistotyöryhmille. Ultimate on kaikkein kattavin versio ja on myös tarkoitettu pääasiassa ohjelmistotyöryhmille. Test Professional on suunnattu pääasiassa ohjelmistotestaajille.

2.3 MonoDevelop

MonoDevelop on avoimen lähdekoodin kehitysympäristö, jonka avulla voi kehittää sovelluksia esimerkiksi Windowsilla ja Linuxilla. Käyttöliittymältään se muistuttaa Microsoftin Visual Studiota ja on vaihtoehto ohjelmalle. Se sisältää paljon samoja projektin kannalta tärkeitä ominaisuuksia kuin Visual Studiokin. Kuten automaattisen koodin täydennyksen ja graafisen käyttöliittymän ja tukee esimerkiksi C#, C++, Visual Basic ja Java -kieliä. [11] Se ei kuitenkaan sisällä samanlaisia työkaluja graafisten käyttöliittymien luontiin kuin Visual Studio. Pelikehityksessä ne eivät kuitenkaan ole kovin tarpeellisia, koska käyttöliittymä luodaan pelimoottorin avulla. Ohjelman voi asentaa Unity3D:n asennuksen yhteydessä tai erikseen.



Kuva 4. Microsoft Visual Studio



Kuva 5. MonoDevelop

2.4 Päätelmät

Pelimoottorin täytyi sisältää mahdollisuus luoda 3D-grafiikkaa mobiilialustoille ja sisältää työkalut niin käyttöliittymän luontiin kuin äänienkin muokkaukseen. Myös tuki Windows Phone -käyttöjärjestelmälle oli tarpeen, koska toimeksiantajan

pyynnöstä peliä testattiin aluksi lähinnä Windows Phone -laitteilla. Ohjelman oli hyvä toimia yhteen esimerkiksi Microsoftin Visual Studion kanssa. Helppokäyttöisyys ja laaja pelimoottoria käyttävä yhteisö on myös eduksi. Aiempien käyttökokemusten perusteella Unity3D on luonteva valinta. Se sopii tarkoitukseen hyvin, koska mobiilipelien kehitys on sillä helppoa, ja siitä on olemassa ilmainen versio, joka ei sisällä rojaltien maksua Unity Technologiesille.

Vaikka Unreal Engine sekä CryEngine sisältävät runsaasti ominaisuuksia ja ovat graafisilta ominaisuuksiltaan Unity3D:tä selvästi edellä, niin tässä projektissa Unity3D:n sisältämät ominaisuudet ovat aivan riittävät projektin tarpeisiin. Myös Unityn ympärille kasvanut yhteisö on näistä laajin. Unreal Engine ja CryEngine eivät tue vielä Windows Phone -mobiilikäyttöjärjestelmää, joten niitä ei siitäkään syystä voinut valita. Unity3D on näistä pelimoottoreista helppokäyttöisin, joten opetteluun ei kulu aikaa, ja sillä on myös kattava dokumentaatio sekä runsaasti ohjeita verkossa. Pelimoottorit tukevat pääosin samoja alustoja ja pelikonsoleita. Tosin vain Unity3D tukee Windows Phonea. Käyttökokemukset, tuki Windows Phonelle, helppokäyttöisyys, ohjelmointikieli, yhteisö, dokumentaatio, lisenssipolitiikka ja projektin aikataulu vaikuttivat päätökseen valita Unity3D. Taulukossa 1 on vertailtu keskenään näitä pelimoottoreita.

Taulukko 1. Pelimoottorien vertailu

	Unity3D	Unreal Engine	CryEngine
Omat kokemukset	1	2	3
Helppokäyttöisyys	1	2	3
Graafiset ominaisuudet	3	1	2
Pelilliset ominaisuudet	3	1	2
Yhteisö	1	2	3
Dokumentaatio	1	2	3
Lisenssi	1	2	3

Ohjelmankehitysympäristöistä Microsoft Visual Studio on ominaisuuksiltaan hieman MonoDevelopia kattavampi, ja soveltuu käyttökokemusten perusteella paremmin laajempiin projekteihin. Visual Studio myös ilmoittaa virheistä ohjelmakoodissa nopeammin ja selkeämmin, jolloin virheiden korjaaminen sujuu nope-

ammin. Se mahdollistaa suoran siirtymisen virheen aiheuttavaan kohtaan ohjelmakoodissa. Projekti olisi kuitenkin ollut toteutettavissa MonoDevelop -ohjelmankin avulla, koska ja se toimii Unity3D:n kanssa yhteen Visual Studiota paremmin.

2.5 Tarpeellisia pelinkehityskomponentteja

Kun Unity3D on valittu, niin pelin toimintoja kehittäessä seuraavat pelinkehityskomponentit ovat välttämättömiä toimivan kokonaisuuden kannalta. Esimerkiksi ampumatoiminto tarvitsee lähtöpisteen luodille eli tässä tapauksessa säteelle ja tauluissa on oltava törmäystunnistimet, jotta peli rekisteröi osuman tauluun. Samoja toimintoja voi hyödyntää myös demon päävalikossa, jossa nappien reagointi painamiseen toteutettiin säteen ja törmäystunnistimien avulla.

Edellä mainittujen törmäystunnistimien avulla määritetään peliobjektin muoto törmäyksiä varten. Unity3D:ssä niistä käytetään nimitystä Collider. Sen ei tarvitse olla täysin samanmuotoinen kuin itse objektin, vaan kohtalaisen tarkasti objektin muotoa jäljittelevä riittää. Vähiten prosessoria kuormittavat ovat perusmuotoja jäljitteleviä. 3D-grafiikassa ne ovat ellipsi, kuutio, kapseli, objekti ja rengas. 2D-grafiikassa neliö ja suorakulmio. Yksittäiseen objektiin voi lisätä useammankin ja asettamalla ne oikein, niillä voidaan jäljitellä objektin muotoa ilman suurta kuormitusta. Peliobjektien aliobjekteihinkin voi liittää tunnistimia. Tarkimmat ovat objektikohtaisia törmäystunnistimia (mesh collider), jotka vastaavat täysin liitettävän peliobjektin muotoa. Samalla ne tosin kuormittavat prosessoria enemmän ja usean sellaisen käyttö ei ole suositeltavaa. Ne eivät myöskään reagoi toisiinsa törmätessään. Tason geometriaan voi käyttää objektikohtaisia tunnistimia ja liikkuviin objekteihin perusmuotoja jäljitteleviä. [4]

Raycast on eräänlainen säde, joka alkaa ennalta määrätystä pisteestä (esimerkiksi tyhjästä peliobjektista) ja osuessaan peliobjektiin käynnistää määritetyn toiminnon, esimerkiksi osumaa kuvaavan äänen. Säteen nopeus ja kulkema matka voidaan määrittää itse. Ampumapeleissä säde on luotia jäljittelevää peliobjektia parempi vaihtoehto, koska säde ei suurillakaan nopeuksilla kulje objektin läpi reagoimatta siihen. Luotia jäljittelevä erillinen 3D-objekti voi niin tehdä.

iTween on lisätyökalu Unity3D:hen, jonka avulla voi helposti luoda animaatioita Unityssa. Pohjimmiltaan se on interpoloiva järjestelmä, joka määritetyn ajanjakson aikana siirtää valittua objektia annetun arvon verran. Se on yksittäinen C# -tiedosto, jota voi käyttää kaikilla Unity3D:n tukemilla ohjelmointikielillä. [5]

GUI-elementit (Graphical User Interface) ovat pelin käyttöliittymän graafisia osia, kuten ruudulla näkyviä nappeja tai pelaajalle informaatiota näyttäviä elementtejä. Tässä pelissä ne voivat näyttää esimerkiksi luotien ja osumien määrän pelaajalle. Versioon 4.5 asti GUI-elementtien toteutus onnistui vain ohjelmakoodin kautta OnGUI() -funktiota käyttäen. Versiosta 4.6 alkaen toimivan käyttöliittymän voi luoda uusilla työkaluilla koskematta itse koodiin.

3 INTERAKTIIVINEN SÄÄOMINAISUUS

3.1 Sää- ja paikkatietopalvelut

Peliin suunniteltiin sisällytettäväksi tuulensuunnasta riippuen eri suuntaan liehuvia lippuja. Tuulensuunnista tosin vain länsi ja itä olisivat varmuudella mukana. Peliin voisi sisällyttää myös aurinkoisen tai pilvisen taustan, mutta ei sadetta. Liehuvat liput ovat erillisiä animaatioita, ja sääpalveluista saatua dataa käyttäen peli valitsee oikean animaation. Objektin valintaan voisi käyttää sopivaa ehtolausekketta. Niihin olisi määritelty ehdot ja käytettävän datan ilmoittaessa tuulen suunnan valitsisi peli oikean objektin. Jos tuulen navakkuus otettaisiin huomioon, niin pelissä voisi olla muutamia eri objekteja, joissa liput liehuvat joko enemmän tai vähemmän. Tuulettomalle tai hyvin heikkotuuliselle säällekin voisi olla oma peliobjektinsa eli esimerkiksi staattinen 3D-objekti liikkuvan sijasta. Sen voisi asettaa oletusvalinnaksi, ellei tietoa jostain syystä ole saatavilla. Toimiakseen toteutus vaatisi verkkoyhteyden ja pelaajan luvan käyttää laitteen sijaintia. Jos pelaaja ei halua niitä antaa valitsee peli oletusvaihtoehdon. Pelaajan sen hetkinen kaupunki riittäisi tässä tapauksessa eikä tarkempaa sijaintia tarvittaisi. Myöskään nopeaa päivitysfrekvenssiä ei tarvita, sillä peli tarkistaisi pelaajan sen hetkisen sijainnin vain kerran eli pelin alkaessa. Eräs vaihtoehto olisi sisällyttää pelin ennalta valitun kisapaikan sää eikä pelaajan sen hetkistä sijaintia.

3.1.1 Perustietoja

Paikkatietopalvelut ovat sovelluksia, jotka tarvitsevat puhelimen sijainnin toimiakseen oikein. Siihen tarvitaan [12]

- palveluntarjoajan sovellus
- mobiiliverkko datan siirtoa varten
- asiaankuuluvaa sisältöä loppukäyttäjälle tarjoava taho
- paikannuksen mahdollistava komponentti, kuten GPS
- loppukäyttäjän mobiililaite.

Paikannus on tärkeä osa paikkatietosovelluksia, koska juuri se mahdollistaa laitteen sijainnin paikantamisen. Siihen voi käyttää satelliittipohjaisia tekniikoita, kuten GPS:ää (Global Positioning System), mutta GSM -pohjainen paikannus on sekin tarkkuudeltaan riittävä. Ulkona käytettäviä paikannustekniikoita ovat muun muassa GPS, A-GPS (Assisted Global Positioning System), D-GPS (Differential Global Positioning System) ja Wi-Fi. Sisätiloissa käytettäväksi soveltuvat näistä muut paitsi GPS, A-GPS ja D-GPS. Bluetooth soveltuu ainoastaan sisätiloissa käytettäväksi. [13]

GPS on maailmanlaajuinen paikallistamisjärjestelmä, joka tarkkuudeltaan noin kymmenen metriä, ja sen päivitysfrekvenssi on noin sekunti. GPS-järjestelmä muodostuu kolmesta segmentistä, jotka ovat avaruus, kontrolliverkko ja käyttäjäosa. GPS-vastaanotin mittaa aikaa, joka signaalilta kuluu sen siirtyessä satelliitilta vastaanottimelle. Vastaanottimen sijainti määritetään eri satelliiteista tulevien signaalien aikaeron perusteella, kunhan satelliitin sijainti tiedetään. [14] A-GPS ja D-GPS mahdollistavat tehokkaamman ja tarkemman lopputuloksen. A-GPS eli avustettu GPS on matkapuhelimia varten kehitetty versio GPS:stä, jossa paikantamista avustetaan matkapuhelinverkoista saatavilla tiedoilla. D-GPS eli differentiaalinen GPS on GPS-paikannuksen alueellinen tarkennusmenetelmä. Se käyttää tarkkaan paikannettuja kiinteitä maa-asemia, jotka mittaavat signaaleista alueellisen paikannusvirheen. [15]

Wi-Fi on mikä tahansa langaton verkko, jonka avulla laitteen voi yhdistää verkkoon. Se käyttää radiotaajuuksia yhdistämään laitteen verkkoon. Se on tarkkuudeltaan 5 – 10 metriä ja toimii parhaiten alueilla, joissa on kattava Wi-Fi -verkko. Tukiasemat ovat toimivan verkon kannalta hyvin tärkeitä. [16] Bluetooth mahdollistaa alle viiden metrin tarkkuuden. Se on laitteiden väliseen kommunikointiin tarkoitettu avoin standardi. Laitteiden on kuitenkin oltava lähietäisyydellä toisiaan. Se perustuu myös radiotekniikkaan. Uusin versio on 4.2, joka on edeltäjiään nopeampi, ja sisältää muun muassa paranneltuja turvallisuusominaisuuksia. [17] Käyttäjä voi myös itse ilmoittaa sijaintinsa, mutta se on epätarkkaa ja päivittyy hitaasti.

Markkinoinnissa käytetään Geo-fence -paikannusta, jossa kartalla olevan kohteen ympärille luodaan virtuaalinen aita. Seuraavaksi tarkistetaan onko alueen sisällä joku, ja saadun vastauksen perusteella käynnistetään ennalta määrätty toiminto. Tekniikka voi olla joko staattinen tai dynaaminen tai muiden käyttäjien sijaintiin suhteutettu. [18]

3.1.2 Käyttö peleissä

Paikkatietoja käyttävät mobiilipelit hyödyntävät laitteen pelaajasta välittämää paikkatietoa pelaajan paikantamisessa, pelimaailman luomisessa sekä vuorovaikutuksen luomisessa muiden pelaajien kanssa. Pelimaailmaa luodessa voi yhdistellä ulkomaailmaa ja virtuaalisia elementtejä keskenään. Paikannustietoja voidaan esimerkiksi yhdistää perinteisiin ulkoilmapeleihin ja luoda uudenlainen pelikokemus. Peli voi selvittää pelaajan sijainnin esimerkiksi GPS:n avulla, jos käytössä on toimiva verkkoyhteys ja pelaajan lupa. Kaikkia pelimäisiä elementtejä käyttäviä sovelluksia ei kuitenkaan kannata kutsua peleiksi. Uusia käytössä olevia tekniikoita ovat paikannusteknologiat, kuvien tunnistaminen, langaton verkko ja lisätty todellisuus. [19]

Markkinoilla olevat pelit voidaan kategorisoida seuraavasti:

- aarteenmetsästyspelit
- paikan puolustus- ja valtauspelit
- tavaroiden keräyspelit
- roolipelit
- eri teemoja yhdistelevät pelit.

3.2 Datan integroiminen peliin

Tarvittavan datan noutamisessa oleellinen osa on joko XML (Extensible Markup Language) tai JSON (JavaScript Object Notation). XML on merkintäkielien standardi. Sitä voi käyttää formaattina tiedonvälitykseen tai tiedon tallentamiseen eri

järjestelmien välillä. XML-dokumentti voi kuvata verkkosivua tai muita tietoja. Ulkoasultaan XML on tekstimuotoista ja muistuttaa HTML-kieltä (Hypertext Markup Language). Se ei kuitenkaan ole sivunkuvauskieli, vaan metakieli, jolla kuvataan tiedon rakennetta.

JSON on tiedonvälitykseen tarkoitettu avoimen standardin tiedostomuoto. Sillä voi korvata XML:n. Alkujaan se oli sidonnainen JavaScriptiin, mutta on nykyään siitä riippumaton. Se koostuu avain-arvo -pareista. Sen sisältämiä tyyppejä ovat numerot, teksti, boolean (tosi tai epätosi), taulukko, objekti ja tyhjä.

Tarvittavan tiedon siirtäminen peliin voisi käyttää ohjelmointirajapintaa (Application Programming Interface). Sen olisi syytä olla ilmainen, ja sisältää tarvittavat tiedot, kuten tuulen nopeuden ja suunnan. Datan siirto Unity3D:n kautta peliin onnistuisi joko XML tai JSON muodossa. Nämä rajapinnat ovat ensisijaisesti verkkosivuille tai sovelluksiin tarkoitettuja, mutta dataa voisi hyödyntää Unity3D:ssä, vaikka XML-lukijaa käyttäen. AccuWeather on suosittu sääpalvelu, mutta heidän ohjelmointirajapintansa vaatii maksullisen jäsenyyden, ja ainoastaan sitä kautta saatavan erillisen avaimen. OpenWeatherMapin ohjelmointirajapinta nimeltään Current Weather Data on ilmainen ja sisältää tarvittavat tiedot, mutta vaatii kuitenkin rekisteröitymisen. [20] Forecast.io:n tarjoama Forecast API palauttaa tämän hetkisen säätilan tiedot, ja tarjoaa myös sääennusteen seuraavan tunnin ajalle, ja aina seuraavien 7 päivän ajalle asti. Se sisältää 2 pyyntöä, joista ensimmäinen palauttaa sääennusteen, ja toinen mahdollistaa ennusteen erikseen määritetylle ajalle. Rajapinnan käyttö vaatii erillisen avaimen, jota käytetään käyttäjän todentamiseen. [21] Jo mainittujen lisäksi on olemassa muun muassa Weather Undergroundin, WeatherBugin, Yahoon ja World Weather Onlinen tarjoamat rajapinnat. Yhteistä niille on käyttäjän todentaminen avaimen avulla, ja formaatti, jossa haettava data on. Eroja on lähinnä hinnoitteluissa ja pyyntöjen sallitussa enimmäismäärässä päivää kohden. [22]

Alla olevaan taulukkoon 2 on listattu rajapintojen ominaisuuksia. Tässä tapauksessa maksulliset voi pudottaa pois. Forecast.io on maksullinen vain jos pyyntöjen määrä päivässä ylittää 1000, ja silloinkin jokainen pyyntö maksaa vain sentin. Se on käytännössä ilmainen, kunhan pyyntöjen määrän pitää alle tuhannessa.

Jokainen vaatii avaimen todennusta varten ja avaimen saa rekisteröitymällä. Näistä projektiin sopivimmat ovat joko OpenWeatherMap tai World Weather Online. Ne ovat ilmaisia ja sisältävät tarvittavat tiedot, mutta nekin vaativat rekisteröitymisen, jonka kautta voi hakea avaimen.

Taulukko 2. Ohjelmointirajapintojen vertailu

	JSON	XML	REST	Avain	Maksullinen
AccuWeather	x			x	x
OpenWeatherMap	x	x	x	x	
WeatherBug	x	x	x	x	x
Forecast.io	x		x	x	
Weather Underground	x	x	x	x	x
Word Weather Online	x	x		x	
Yahoo		x	x	x	

Data voisi hakea JSON muodossa, koska melkein jokainen rajapinta tukee sitä. Sijaintia voi kutsua niin nimellä, tunnuksella kuin koordinaateilla. Alla oleva kuva 6 esittää OpenWeatherMapin ohjelmointirajapinnan sisältämä dataa XML -muodossa.

```
<temperature value="275.759" min="275.759" max="275.759" unit="kelvin"/>
<humidity value="94" unit="%"/>
<pressure value="1004.48" unit="hPa"/>
- <wind>
    <speed value="3.11" name="Light breeze"/>
    <direction value="275.502" code="W" name="West"/>
</wind>
<clouds value="44" name="scattered clouds"/>
<visibility/>
<precipitation mode="no"/>
<weather number="802" value="scattered clouds" icon="03n"/>
```

Kuva 6. OpenWeatherMapin ohjelmointirajapinta XML-muodossa

JSON:a käytettäessä on suositeltavaa käyttää esimerkiksi SimpleJSON pakettia, jonka avulla datan voi jäsentää luettavampaan muotoon.

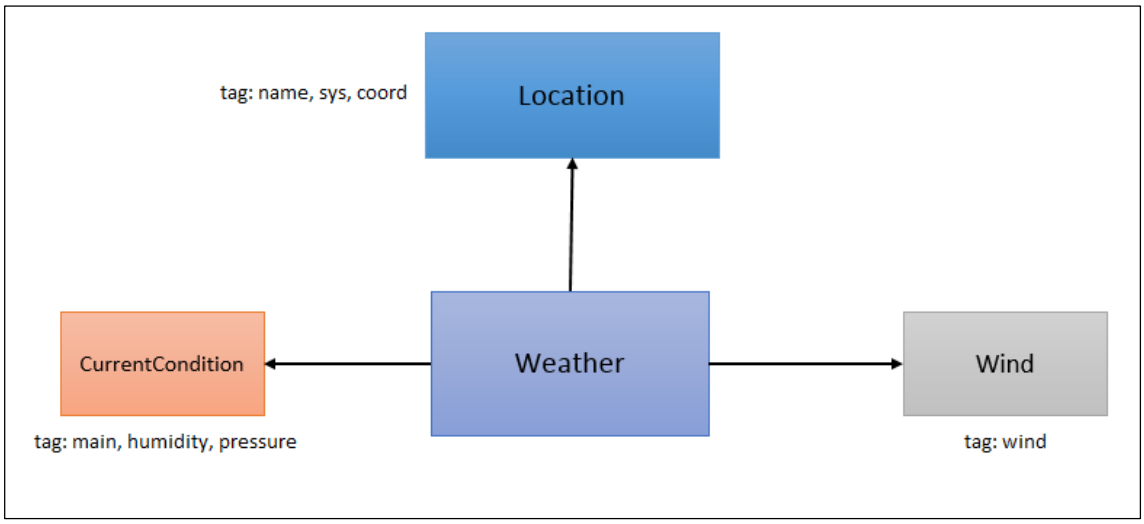
Mobiililaitteen GPS:n syöttäisi laitteen sijainnin pelille. Sitä käyttäen peli saisi ohjelmointirajapinnan kautta pelaajan sijainnin säätilan. Tuulen eri arvoja käytettäisiin muuttujina suuntaa ja nopeutta vastaavien peliobjektien valintaan. Unity3D sisältää LocationService ja LocationInfo rakenteet, joiden avulla peli hakisi GPS-

koordinaatit. [23] [24] Ne sisältävät eri muuttujia, kuten leveysasteet ja pituusasteet. Sijainnin tarkkuudeksi riittäisi pelkkä kaupunki. Tällöin peli löytäisi pelaajan sijainnin helposti myös sisätiloissa, missä GPS on yleensä epätarkempi. GPS:stä saadun datan noutamiseen voi kirjoittaa oman metodinsa, esimerkiksi alla olevan kuvan lailla.

```
void RetrieveGPSData()
{
    currentGPSPosition = Input.location.lastData;
    string gpsString = "::" + currentGPSPosition.latitude + "//" + currentGPSPosition.longitude;
    GameObject.Find("gps_debug_text").guiText.text = gpsString;
}
```

Kuva 7. Esimerkki metodista

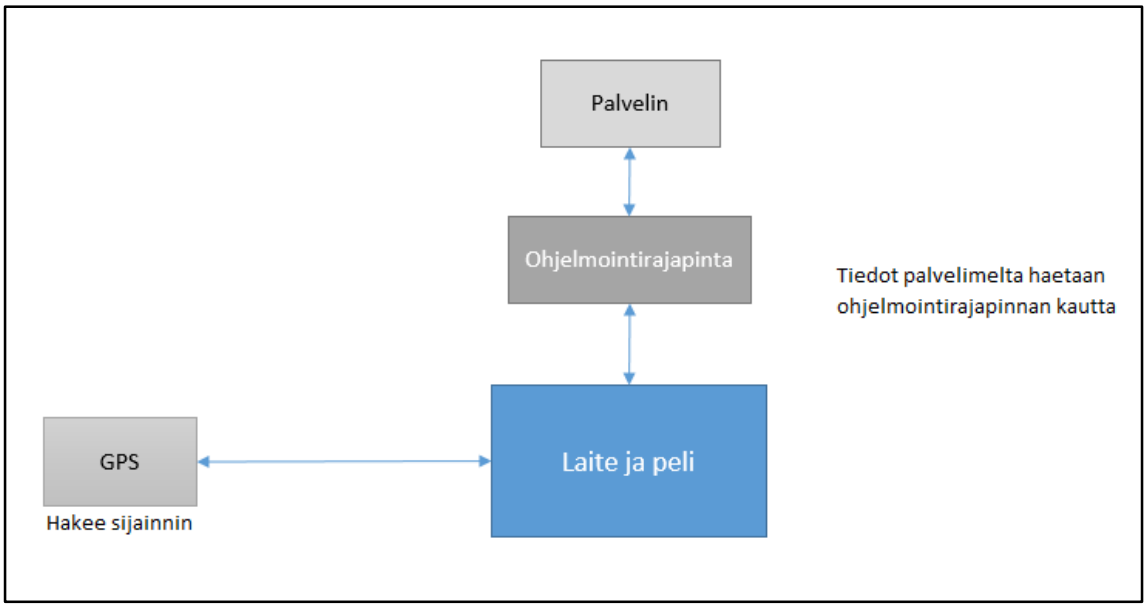
Jos peli käyttää JSON:a, niin ohjelmakoodissa peli lähettäisi pyynnön palvelimelle, ja saisi vastauksena tiedot palvelimelta. Saadut tiedot tallennettaisiin muuttujina. Pyynnön lähettämiseen voisi käyttää oma metodiaan, joka kutsutaan pelin alkaessa. Saadusta datasta luotaisiin JSON-objekti, ja seuraavaksi palvelimelta saatu vastaus voidaan jäsentää. Säähän liittyvistä tiedoista voi tehdä taulukon, johon voi tallentaa esimerkiksi sijainnin nimen. Tuulen muuttujat voi tallentaa JSON-objektiksi, johon esimerkiksi nopeus ja suunta tallennetaan. [25]



Kuva 8. Ohjelmointirajapinnan toimintalogiikka

Informaation vaihtoon palvelimen ja pelin välillä voi käyttää erillistä luokkaa. Siihen voi käyttää GET- ja POST -pyyntöjä. GET-metodin avulla pyydetään dataa määritetystä lähteestä, kuten palvelimelta. POST-metodi syöttää pyydetyn datan

määritettyyn lähteeseen. [26] Ensin avattaisiin yhteys haluttuun palvelimeen. Kun yhteys on muodostettu, OutputStream-toimintoa käyttäen kirjoitettaisiin palvelimelta haettava tieto, ja lähetettäisiin pyyntö palvelimelle. Palvelimelta saatu vastaus luettaisiin InputStream-toiminnon avulla. Parametrit kirjoitettaisiin avain ja arvo -pareina. Jos sovellus ei vastaa pyyntöön, sen voi välttää käyttämällä AsyncTask -toimintoa. Se on abstrakti luokka, joka mahdollistaa operaatioiden toteuttamisen taustalla vaikuttamatta pääsäikeeseen. Sovellus ei siis lopeta toimintaa operaation toteuttamisen ajaksi. Esimerkiksi ladattaessa palvelimelta tiedostoa, sovellus ei seisahdu, vaikka tiedoston lataus on käynnissä. [27] Ladattaessa dataa palvelimelta on suositeltavaa käyttää sitä, jotta operaation voi virheittä suorittaa loppuun.



Kuva 9. Sääominaisuuden toimintalogiikka

Ehtolausekkeeseen voisi määrittää tuulen nopeuden eri arvoja, joiden mukaan tietty objekti valitaan. Arvot voisivat esimerkiksi olla 0–5, 5,1–10 ja yli 10,1. Jos tuulennopeus on vähäinen, valitsisi peli objektin, jossa lippu liehuu vain vähän tai ei ollenkaan. Tuulen puhaltaessa lännestä, valitsisi peli itään päin liehuvan lipun. Objekteja voisi olla kuusi eli kolme (vähäinen, navakka ja kova) kummallekin tuulensuunnalle (länsi ja itä). Neljäkin kävisi eli kaksi kummallekin tuulensuunnalle, mutta se olisi minimi. Jos nopeuden arvo olisi vaikka 7,55 ja suunta idästä, peli valitsisi navakkaa tuulta esittävän lipun ja sen, jossa se liehuu länteen päin.

Jos tiedot halutaan lukea XML-tiedostosta, siihen tarvitsee XML-lukijan. Sen avulla voi lukea erillisen XML-tiedoston ja ottaa sieltä tarvittavat tiedot pelin käyttöön. Sellaisen voi tehdä joko itse tai hyödyntää valmiita. Visual Studio mahdollistaa XML-tiedoston lukemisen käyttämällä System.Xml -toimintoa, joka on osa .NET-kirjastoa. [28] Se on sisällöltään laaja, joten se kasvattaa tiedoston kokoa selvästi. XML-lukijan voi tehdä myös itse, ja räätälöidä sen omiin tarkoituksiin sopivaksi. Silloin se on kevyempi eikä juuri kasvata tiedoston kokoa. XML-tiedosto voi sijaita joko kansiossa tai palvelimella, josta sen sisältämä data ladataan käynnistettävään peliin. Tiedoston voi ladata joko käyttämällä FileIO tai WWW-luokkia. Vasta sen jälkeen tiedosto on mahdollista lukea. XML-lukija käy läpi XML-tiedoston jokaisen elementin, ja sen sisältämä data voidaan jäsentää, ja siirtää halutut arvot pelin ohjelmakoodista löytyviin muuttujiin tai taulukoihin.

3.3 Haasteet

Suurin haaste prosessissa on, että rajapinnat ovat etenkin sovelluksille tarkoitettuja eivätkä niinkään peleihin. Unity3D tai mikä tahansa pelimoottori tuo prosessiin yhden osan lisää, ja se monimutkaistaa prosessia, ja voi jopa estää kunnollisen toimivuuden. Datan siirtäminen verkosta peliin on sinällään haastavaa, ja yhteyksistä sekä paikasta riippuen toimivuus ei ole taattu. Esimerkiksi XML-tiedoston päivittäminen verkon kautta joka pelikerralla voi osoittautua haastavaksi, ja on mahdollista, että tiedosto ei päivitykään. Silloin peli käyttäisi vanhoja tietoja. Se voi olla satunnaista tai jopa jatkuvaa. XML–tiedosto voi sijaita myös palvelimella, jolloin sitä ei tarvitse sijoittaa yhteenkään pelin kansioista. Silloinkin verkon toimivuudesta riippuu voidaanko tiedot ladata palvelimelta. Myös GPS:n toimivuus luo oman haasteensa. Edes yhden sijainnin tietoja ei voida päivittää ilman päällä olevaa GPS:ää, mutta verkkoyhteyttä siihen ei tarvitse. Säätietojen noutaminen ohjelmointirajapinnan kautta tosin vaatii toimivan verkkoyhteyden.

Myös käyttöjärjestelmät muodostavat omat haasteensa. Androidille on saatavana eri tapoja hakea tietoja verkosta, ja käyttää niitä sovelluksissa, ja myös peleissä.

Kyseiset tavat eivät välttämättä toimi sellaisenaan esimerkiksi Windows Phonessa. Silloin peliä ei voisi suoraan kääntää muille käyttöjärjestelmille, vaan ohjelmakoodia olisi muokattava, jotta ominaisuudet toimisivat myös muissa käyttöjärjestelmissä. Unity3D:ssä olisi silloin luotava omat projektikansionsa kullekin versiolle, ja tehtävä jokaiseen omat muokkauksensa. Usein kehittäjä tarvitsee vain yhden valmiin pelin sisältävän projektikansion, josta Unity3D osaa kääntää eri käyttöjärjestelmille toimivat versiot.

3.4 Vaihtoehtoinen toteutustapa

Vaihtoehtoisesti ominaisuuden voisi toteuttaa sisällyttämällä tarvittavat peliobjektit peliin, ja käyttämällä niitä ilman GPS:ää ja rajapintoja. Siinä tapauksessa peli voisi arpoa milloin mitäkin elementtiä käytetään. Peliin sisällytettäisiin objektit ja animaatiot, ja tuulen nopeus arvottaisiin vaikka väliltä 0,5 ja 20. Arvon osuessa välille 0,5 – 5 valitsee peli objektin, jossa lippu liehuu vähiten tai ei ollenkaan. 5,1 – 10 tarkoittaa navakkaa tuulta ja 10,1 – 20 kovaa tuulta. Peli myös arpoisi tuulen suunnan idän ja lännen väliltä. Jos peli arpoisi tuulen suunnaksi esimerkiksi länsi, niin samalla se lukitsisi vastakkaiseen tuuleen suuntaan liitetyt objektit. Vasta seuraavalla pelikerralla ne olisivat taas avoinna.

Toinen tapa toteuttaa tämä olisi käyttää yhtä ennalta määrättyä sijaintia, ja päivittää vain sen sijainnin säätiedot. Silloin sijainnin voisi määrittää suoraan ohjelmakoodiin. Tällöin peli päivittäisi pelin alkaessa vain tämän sijainnin säätiedot. Kyseinen sijainti voisi olla jokin kisapaikka, kuten Kontiolahti.

Tavat voisi myös yhdistää sisällyttämällä peliin kaksi eri pelitilaa. Ensimmäinen toimisi ilman verkkoyhteyttä ja GPS:ää. Toinen voisi toimiakseen tarkistaa verkkoyhteyden ja GPS:n toimivuuden. Toisessa pelitilassa ei esimerkiksi olisi lainkaan tuulioloja, ja toisessa pelitilassa olisi. Internet-yhteys ei siten olisi vaatimus pelin pelaamiselle.

4 AMPUMAHIIHTOPELIN DEMON TOTEUTUS

Projektin toimeksiantaja on RightSpot Oy -niminen yritys Salosta, ja peliä oli alustavasti jo suunniteltu. RightSpot Oy suunnittelee laitteita, laitteistoja ja ratkaisuja, joilla voi parantaa tehokkuutta niin urheilussa kuin harrastuksissa. Pelin tulisi kuvata ampumahiihtokisojen ampumaosuutta pelattavassa muodossa. Pelissä pelaaja saapuisi ampumapaikalle ja ampuisi neljä kertaa viisi laukausta. Ensin ammutaan makuulta, sitten pystystä, sitten uudelleen makuulta ja lopuksi vielä pystystä. Makuuammunnassa pelaajan on osuttava pienempään tauluun, jonka halkaisija itse kisoissa on 4,5 cm. Pystyammunnassa osuma-alue on isompi, kisoissa 11 cm. Pelissä täytyy siten olla isompi ja pienempi osuma-alue luodeille. Ampumapaikan ja taulujen etäisyys kisoissa on 50 m.

Aseen heilunnan on oltava pystyssä suurempi kuin makuulta. Asiakas myös toivoi aseen heilunnan hieman lisääntyvän pelin edetessä. Pelinäkymään toimeksiantaja eivät halunneet itse asetta näkyviin, vaan etu- ja takatähtäimen. Luoti lähtisi liikkeelle tähtäinten keskeltä, ja tähtäinten heilumisliike simuloisi aseen heiluntaa. Jokaisen osuman jälkeen taulun edessä oleva läppä nousisi, ja vasta tämän jälkeen peli siirtyisi seuraavan taulun kohdalle. Ohilaukauksen jälkeen läppä ei nouse, mutta tauluun jäisi osumajälki. Myös ohilaukauksessa olisi sama viive siirtymien välillä kuin osumissakin.

Peli näyttää lopuksi, kuinka monta osumaa pelaaja sai ja mikä oli pelaajan ampumiseen käyttämä aika sekä montako ohilaukausten aiheuttamaa sakkominuuttia pelaajalle kertyi. Toimeksiantaja ehdotti pelin oikeaan yläkulmaan käyttöliittymäelementtiä, josta pelaaja näkee, mihin tauluihin hän osui ja mihin ei. Vasemmalla alhaalla näkyy aluksi 5 luotia sisältävä GUI-elementti, ja joka laukauksen myötä 1 luoti vähenee. Ampumanappi sijaitsee oikealla alhaalla ja päävalikkoon vievä nappi vasemmalla ylhäällä.

Pelin rakenne koostuu päävalikosta ja ampumatasoista. Pysty- ja makuuammunnoille on luonnollisesti omat tasonsa. Sisällöltään ja toiminnoiltaan ampumatasot ovat pääosin samanlaisia. Esimerkiksi itse ampumatoiminto on samanlainen ja

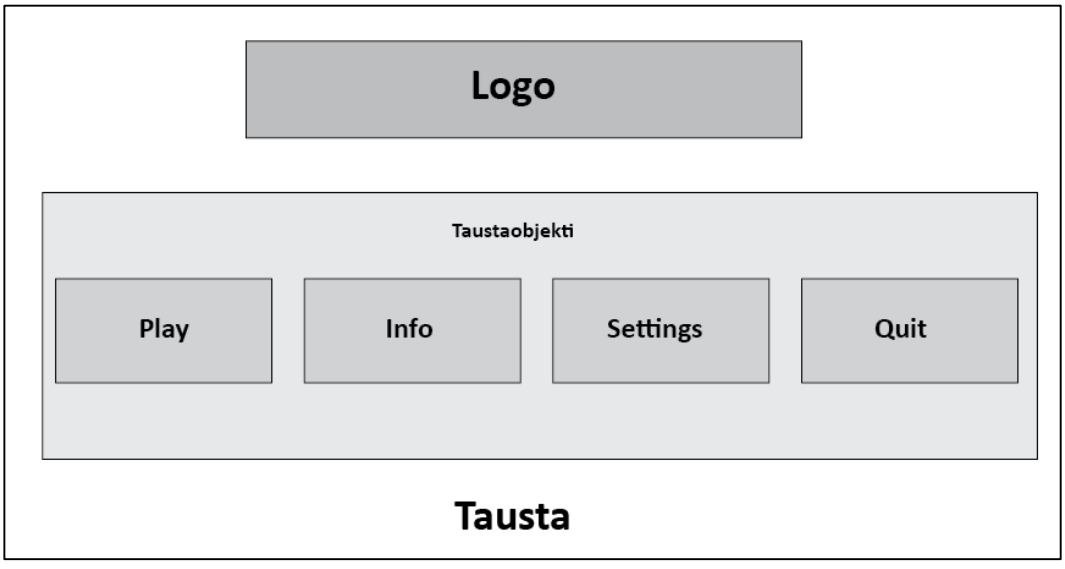
ainoastaan luodin osuma-alue tauluissa vaihtelee. Pystyammunnassa tauluihin liitetään suurempi osuma-alue, esimerkiksi suurempaa törmäystunnistinta käyttäen, ja makuuammunnassa pienempi.

4.1 Päävalikko

Ensimmäisenä avautuvaa päävalikkoa pelaaja käyttää navigointiin. Valikon on ennen kaikkea oltava käytettävyydeltään helppo ja selkeä. Lisäksi mukana on oltava pelin lopetustoiminto.

Päävalikon rakenne on seuraava:

- pelitila (Play)
- info
- asetukset (Settings)
- lopetus (Quit).



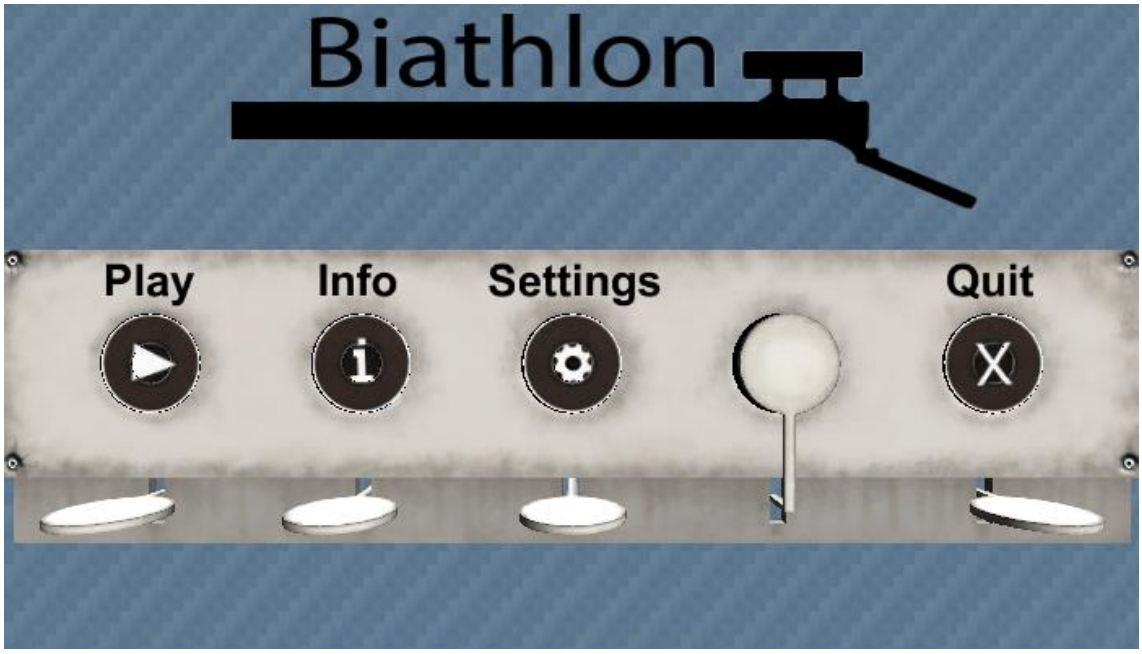
Kuva 10. Demon päävalikko suunnitteluvaiheessa

Alkuanimaation kautta pelaaja siirtyy pelitilaan Play-nappia painamalla. Info-napista aukeaa ampumahiihdosta kertova infosivu. Asetukset-tasosta käsin pelaaja pääsee muuttamaan pelin asetuksia, mutta se taso jätettiin tällä erää tyhjäksi odottamaan jatkokehitystä. Oikealla oleva Quit-nappi sulkee pelin. Valikon keskeinen elementti on maalitaulua kuvaava 3D-objekti, jonka viidestä taulusta neljä

toimii valikossa nappeina. Maalitaulut ovat erillisiä 3D-objekteja, jotka on liitetty niiden taustakehikkoa mukailevaan 3D-malliin. Yksittäinen 3D-objekti mahdollistaisi vain yhden napin neljän napin sijasta. Maalitauluille luodaan omat kosketusalueensa törmäystunnistimien avulla. Kosketusalue on hieman itse nappeja suurempi, jotta käytettävyys olisi parempaa. Raycast-toiminnon käyttäminen mahdollistaa yksittäisten objektien käyttämisen nappeina. Pelaajan painaessa sormellaan ruutua peli luo säteen, joka osuessaan tauluun käynnistää sille määrätyn toiminnon eli avaa määrätyn tason. Jos säde osuu taustaan tai taulujen kehikkoon, mitään ei tapahdu.

Päävalikko sisältää myös animaation, jossa napin painamisen jälkeen valkoinen läppä nousee peittämään maalitaulun. Valkoiset läpät ovat erillisiä objekteja ja animaatio on liitetty niihin Unity3D:ssa. Animaatio toistetaan napin painallushetkellä. Animaation päättyessä kuuluu myös kolahdus, kuten oikean metallisen läpän noustessa kuuluu. Se on äänitiedosto, joka liitetään oikeaan peliobjektiin, jonka toiminta on määritetty omassa metodissaan.

Päävalikossa käytetään neljää eri ohjelmakoodia. Kameraan on liitetty ohjelmakoodi, joka piirtää taustan ja mahdollistaa pelin lopettamisen Windows Phone puhelinten takaisin -nappia painamalla. Jokaisella taululla on myös oma ohjelmakoodinsa, ja suurin osa niiden sisällöstä on identtisiä.



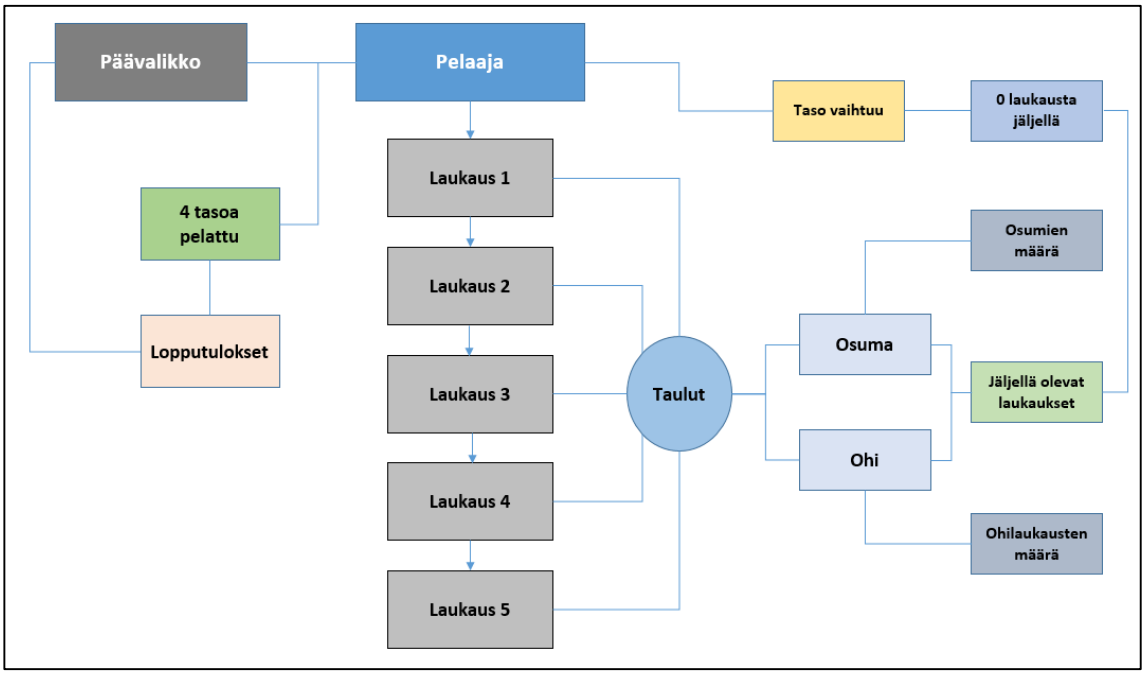
Kuva 11. Demon valmis päävalikko

4.2 Pelitila

Pelaaja ampuu sekä makuu- että pystyammuntatasot yhtenä pelisessiona. Ensin makuulta, seuraavana pystystä ja sitten sama toistetaan. Tasoja on kaksi, mutta kummassakin tasossa pelaaja käy kahdesti. Tasoja on siis käytännössä neljä, ja jokaisessa pelaaja ampuu viisi kertaa eli yhteensä 20 laukausta. Aseen heilunta hieman lisääntyy taso tasolta ollen suurempaa pystyammunnan aikana.

4.2.1 Pelinäkymä

Pelin vaatimat 3D-objektit ovat etu- ja takatähtäin, maalitaulut sekä tausta. Aseen runko on jätetty pois näkyvistä. Tähtäimet on sijoitettu keskelle ruutua ja kamera lähelle takatähtäintä. Maalitaulut on sijoitettu siten, että ensimmäinen taulu on pelin alkaessa vastapäätä tähtäimiä. Maalitauluja on yhteensä 5, ja ne on sijoitettu osaksi kehikkoa. Koska peli on suunniteltu mobiililaitteille, niin se vaatii kosketusnäytöille soveltuvat käyttöliittymäelementit. Ruudulla on kaksi nappia ja yksi vie takaisin päävalikkoon ja toinen nappi on ampumista varten. Nappien lisäksi luotien määrä, osumat ja ohilaukaukset sekä kulloinkin käynnissä oleva taso näytetään GUI-elementteinä ruudulla.



Kuva 12. Pelinäkymän toimintalogiikka

4.2.2 Makuu- ja pystyammunta

Tasot vaativat seuraavat toiminnot toimiakseen.

- säteen lähtö suoraan kohti taulua, kun ammuntanappia painetaan
- säteen osuessa objektiin se rekisteröidään joko osumaksi tai ohilaukaukseksi
- osumien, ohilaukausten ja ajan tallennus tyhjään peliobjektiin
- tietojen tyhjennys pelin lopussa
- lyhyt tauko laukausten välillä
- rekyyli
- osumapisteet
- tähtäimien ja kameran siirtymä oikeaan suuntaan laukausten jälkeen
- Jäljellä olevien luotien määrän väheneminen yhdellä joka laukauksen jälkeen
- laukausten määrä per taso ei saa ylittyä
- tähtäinten heiluminen
- äänet
- GUI-elementtien toimivuus

- tasojen alkuruudut
- siirtymä seuraavalle tasolle viimeisen laukauksen jälkeen.

Ampuminen on toteutettu Raycast -toiminnon avulla, koska nopeasti liikkuvat 3D-objektit saattavat välillä kulkea muiden objektien lävitse. Silloin peli ei tunnistaisi osumaa osumaksi. Säde osuu aina 3D-objekteihin ja sitä käytettäessä erillistä 3D-luotia ei tarvita, vaan säde itsessään riittää. Ampumanappia painettaessa säde (luoti) alkaa etutähtäimen eteen sijoitetusta tyhjästä peliobjektista ja kulkee määritetyn matkan verran määrätyllä nopeudella. Samalla kuuluu myös laukauksen ääni. Tauluun osuessa kuuluu myös ääni, mutta osuman ja ohilaukauksen äänet ovat hieman erilaiset. Äänet ovat erillisiä äänitiedostoja, jotka on liitetty haluttuihin peliobjekteihin ja hetki jolloin äänet kuuluvat on määritetty koodissa. Osumat jättävät myös osumapisteet tauluihin. Ne ovat peliobjekteja, jotka luodaan osumakohtiin.

Joka laukauksen jälkeen rekyyli siirtää tähtäimiä 1,2 yksikköä ylös. Tauko laukausten välillä on asetettu 1,5 s:iin eli tuona aikana pelaaja ei voi ampua. Sillä estetään se, että pelaaja ampuisi kaikki laukaukset hetkessä. Ampuminen on toteutettu kahtena ohjelmakoodina. Makuu- ja pystyammunnalle on omansa. Pystyammuntatason ohjelmakoodi sisältää pelin lopetusnäkymän avaavan metodin, koska pelin viimeinen taso on pystyammuntaa. Muilta osin ohjelmakoodit ovat samanlaisia.

Koodi 1. Ampumatoiminnon ohjelmakoodi

```
   public void shootRay()
   {
      RaycastHit hit = new RaycastHit();
      Physics.Raycast(this.transform.position, Vector3.forward * 900f,
out hit);
      Debug.DrawRay(this.transform.position, Vector3.forward * 900f,
Color.red);

      GameObject bullethole = Instantiate(bulletHole, hit.point,
bulletHole.transform.rotation) as GameObject;

      bullethole.transform.position = new
Vector3(bullethole.transform.position.x,
bullethole.transform.position.y, bullethitz.transform.position.z);

      if (Counter.shots == 5)
      {
         Counter.roundCounter = Counter.roundCounter + 1;
         Debug.Log("Kierros " + Counter.roundCounter + " pelattu");
         nextLevel = true;

      }

      if (hit.collider != null)
      {
         if (hit.collider.GetComponent<Animator>())
         {

hit.collider.GetComponent<Animator>().SetTrigger("standUp");

         }
      }

      if (hit.collider != null)
      {

         if (hit.collider.tag == tagCheck)
         {
            Debug.Log("Raycast hit object " + hit.transform.name);
            targetScript.instance.recoilMove();
            Counter.score = Counter.score + 1;
            hitCount++;
            playAudio1();
            Debug.Log("osuma");
            updateTargetUi();
            if (hitCount >= 5)
            {
               audioFinished = false;
               this.audio.Play();
            }
            Invoke("callMoveTarget", 1);
         }

         else
         {
            targetScript.instance.recoilMove();
            playAudio2();
            Debug.Log("ei osu");
            Invoke("callMoveTarget", 1);
         }
```

Joka laukauksen jälkeen pelaaja siirtyy automaattisesti seuraavan taulun kohdalle. Siirtymä on toteutettu iTweeniä käyttäen. Tähtäimiä ja kameraa siirretään x-akselilla 9,9 yksikköä oikealle ja y-akselilla 1,2 yksikköä alas. Tähtäimet eivät

siirry kuitenkaan heti, vaan sekunnin tauon jälkeen. Tuona aikana pelaaja voi vilkaista mihin kohtaan osui. Jokainen laukaus tulee myös vähentämään jäljellä olevien luotien määrää yhdellä. Maksimi jokaisella tasolla on 5 luotia. Tason vaihtuessa luotien määrä palautuu takaisin viiteen. Luotien määrä näkyy pelaajalle 2D-grafiikkana näytön vasemmassa alareunassa. Se on 5 kuvaa sisältävä taulukko. Ensimmäisessä kuvassa on 5 luotia ja seuraavissa 1 vähemmän. Joka laukauksen myötä taulukko näyttää seuraavan kuvan.

Oikeaan yläkulmaan on sijoitettu GUI-elementti, joka näyttää osumat ja ohilaukaukset pelaajalle. Se on kuvia sisältävä taulukko, jossa mustat ympyrät on sijoitettu valkoiselle taustalle. Aluksi ympyrät ovat mustia ja vaihtuvat valkoisiksi vain jos pelaaja osuu tauluun. Jos pelaaja esimerkiksi ampuu ohi toisen ja neljännen laukauksen, niin toinen ja neljäs kuva taulukossa jäävät ennalleen. Loput vaihtuvat, koska niihin tauluihin osuttiin. Ohjelmakoodissa on määritetty kuinka oikea kuva vaihtuu osuman hetkellä. Ruudun vasemmassa reunassa keskellä on valkoinen silhuettikuva ampumahiihtäjästä. Makuuammuntatasoilla kuva on makuuasennossa oleva urheilija ja pystyammuntatasoilla pystyasennossa oleva.

Tähtäimiin on liitetty aseen heilumista jäljittelevää liikettä. Heiluntaliike on vähäisempää makuuammunnan ja suurempaa pystyammunnan aikana. Liike myös lisääntyy taso tasolta jäljitellen kisan aiheuttaman rasituksen vaikutusta urheilijaan. Ensimmäisellä tasolla eli makuuammunnan aikana se on vähäistä ja seuraavan tason eli pystyammunnan aikana ase heiluu enemmän. Kolmannella tasolla eli toisen makuuammunnan aikana heiluntaliike on vähäisempää verrattuna pystyammuntaan, mutta suurempaa verrattuna ensimmäiseen makuuammuntatasoon. Viimeisellä tasolla eli toisen pystyammunnan aikana aseen heilunta on suurinta. Heiluntaliike on oma ohjelmakoodinsa, joka on liitetty tähtäinobjekteihin ja kameraan Unity3D:ssa.

Siirtymä seuraavalle tasolle ei vielä demossa tapahdu automaattisesti, vaan joka tason viimeisen laukauksen jälkeen ampumanapin toiminta vaihtuu ja mahdollistaa siirtymän seuraavalle tasolle. Tämä siksi, että ruutua ei tarvitsisi täyttää liian monella napilla. Samalla napin ikoni vaihtuu nuoleen, joka osoittaisi pelaajalle

mitä nappia painaa seuraavaksi. Seuraavan tason alussa ampumanapin toiminta palautuu normaalitilaansa.

Kuva 13. Demon pelinäkymä

4.2.3 Lopputulokset

Pelaajan ammuttua kaikki laukaukset aukeaa erillinen loppunäkymä. Se näyttää, kuinka monta osumaa 20:sta pelaaja sai, montako sakkominuuttia kertyi ja paljonko aikaa kului. Osumat, ohilaukaukset ja aika on kerätty tyhjään peliobjektiin, joka säilyttää tiedot tasojen vaihtumisen jälkeenkin. Normaali käytäntö Unity3D:ssa on, että peliobjektit tuhotaan tason vaihtuessa ja luodaan uudelleen seuraavan tason käynnistyessä. Silloin tiedot häviävät muistista, mutta sen voi välttää esimerkiksi DontDestroyOnLoad -komennolla. On kuitenkin syytä tarkistaa, että kyseinen objekti ei kopioidu joka pelikerralla ja ala hidastaa peliä. Täytyy myös varmistaa, että tiedot nollautuvat pelin päätyttyä. Näin laitteen muisti ei täyty eivätkä vanhat tiedot sotke myöhempiä pelikertoja. Aika näkyy pelaajalle jo pelin aikana ruudun oikeassa reunassa. Loppunäkymä sisältää myös napin uudelleen pelaamista varten, jotta pelaajan ei tarvitse kiertää päävalikon kautta.

5 AMPUMAHIIHTOPELIN JATKOKEHITYS

Osa tässä luvussa esille tulevista ominaisuuksista oli alustavasti ideoitu suunnitteluvaiheessa, mutta niitä ei aikataulun vuoksi demoon toteutettu. Päävalikkoa voi muokata sen verran, että se ottaa huomioon mobiililaitteiden kallistukset. Demoversiossa päävalikko on suunniteltu käytettäväksi mobiililaitteen ollessa sivuttain (landscape). Mobiililaitteen ollessa pystyssä (portrait) napit ovat kyllä allekkain, mutta logo jää sivulle, joten sen pitäisi jäädä näytön yläreunaan. Myös tekstin ja ikonien pitäisi kääntyä sopimaan pystyssä olevan laitteen näyttöön. Maalitaulua esittävän 3D-objektin pitäisi hieman skaalautua pienemmäksi, jotta logon saisi mahtumaan näytön yläosaan. Nyt se on kooltaan koko ruudun verran. Myös käyttämättömät napit voidaan ottaa käyttöön ja lisätä niihin tarvittavat toiminnot. Päävalikon voi suunnitella kokonaankin uudelleen ja vaihtaa elementit toisiksi ja sijoittaa ne toisin.

Itse pelinäkymän voi rakentaa siltä osin uusiksi, että sekin ottaa huomioon mobiililaitteen asennon. Demo on suunniteltu näkymään pelaajalle oikein vain mobiililaitteen ollessa sivuttain. Pelin objektit ja elementit voisivat sijoittua oikein myös mobiililaitteen ollessa pystyssä. Pelin objektit pitäisi silloinkin asettaa niin, että kaikki tarvittava on aina näkyvissä. Tähtäimien ja taulujen etäisyys toisiinsa ja kameraan ei saa kuitenkaan kasvaa liian isoksi tai pieneksi. Jos objektit näkyvät näytöllä joko liian pieninä tai suurina, niin pelaaminen hankaloituu tai peli ei näytä visuaalisesti kovin hyvälle. Silloin peli ei myöskään kuvaa kisatilannetta oikein. Toisaalta, pelinäkymän rakentaminen toimimaan oikein myös laitteen ollessa pystyssä on hankalaa, koska silloin se helposti kapenee liikaa eikä vastaa sitä mitä urheilija näkee kisatilanteessa. Parempi ratkaisu olisi varmistaa se, että peli kuvaa kisatilanteen oikein ja se onnistuu parhaiten, kun laite on sivuttain.

Myös käyttöliittymäelementtien pitäisi mahtua näytölle loogisiin paikkoihin. Kyseiset elementit voisi korvata esimerkiksi spriteillä tai 3D-objekteilla, jotta ne skaalautuvat oikein eri resoluutioille. Sivuilla näkyvän graafisen kuvion voisi myös poistaa. Demo suunniteltiin lähinnä laitteille joiden resoluutio on varsin pieni. Pe-

lin jatkokehityksessä on otettava suuretkin resoluutiot huomioon, koska markkinoilla on jo mobiililaitteita, joissa on QuadHD -resoluutio. Unity3D:n versiossa 4.6 käyttöliittymän työkaluja oli kehitetty selvästi paremmiksi, joten sillä versiolla pelin käyttöliittymän päivittäminen on helpompaa kuin edeltävissä versioissa. Myös Unity3D:n uudessa versiossa 5 uudistetut käyttöliittymätyökalut ovat mukana.

Mahdolliseen kokoversioon voisi liittää tulosten tallentamisen erilliseen listaan ja näyttää tulokset joko omalla tasollaan tai pelatun kisan päätteeksi. Erilliseen tasoon pääsisi päävalikosta. Peli voisi kisan päätteeksi tallentaa tuloksen joko automaattisesti tai kysyä pelaajalta. Tulokset listattaisiin parhaimmasta huonoimpaan. Jos peli vaatisi käyttäjätunnuksen luomisen, niin tulos liitettäisiin kulloinkin kirjautuneeseen käyttäjätunnukseen. Tuloksen ohella myös siihen liitetty käyttäjätunnus näytettäisiin. Jos peli ei vaatisi käyttäjätunnusta, niin tulokset listattaisiin ilman tunnuksia. Tulokset voisi listata joko osumien tai ajan perusteella. Mitä enemmän osumia, sitä parempi on tulos ja mitä nopeampi aika, sitä parempi tulos. Ohilaukauksista tosin kertyy sakkominuutteja lopulliseen aikaan, joten hyvään osumatarkkuuteen on syytä pyrkiä, jos haluaa hyvän ajan.

Peliin voisi liittää myös palkintotoiminnon. Xbox-peleissä on saavutukset (achievement) ja PlayStation-peleissä palkinnot (trophy). Pelissä voisi jakaa palkintoja, kun pelaaja on saavuttanut määrättyjä saavutuksia, kuten suorittanut kisan alle määrätyn ajan ja silti osunut jokaiseen tauluun. Jos pelissä olisi palkintohuone tai kaappi, niin se voisi olla oma tasonsa, jossa pelaajan saavuttamat palkinnot olisivat nähtävissä. Palkinnot ja tulokset voisi myös yhdistää samaan tasoon. Saavutukset voisivat olla listattuna joko arvokkaimmasta vähiten arvokkaimpaan tai ajankohdan mukaan. Tällöin viimeisin saavutus olisi listassa ensimmäisenä ja vanhin viimeisenä. Pelissä voisi myös olla mahdollista listata saavutukset vanhimmasta uusimpaan. Mitään kovin graafista palkintohuonetta ei tarvitsisi tehdä, vaan tekstiin painottuvan ja selkeän.

Peliin voisi lisätä myös uusia tasoja ja vaihtamalla pelissä jo olevaan tasoon taustan. Taustat voisivat jäljitellä oikeita kisapaikkoja. Pelaaja voisi pelata kaikki tasot kampanjamuodossa tai valita yksittäisen tason. Alkuanimaation voisi liittää kampanjaan, mutta ei välttämättä yksittäisiin tasoihin.

6 YHTEENVETO

Lopputuloksena on toimiva pelidemo, joka sisälsi toimeksiantajan toivomat ominaisuudet. Myös oppimistavoitteet täyttyivät. Pelinkehitysprosessi ja asiakaslähtöinen projektityöskentely ovat aiempaa tutumpia. Demo sisältää toimivan valikon ja kahdesta tasosta koostuvan pelitilan. Joka tasolla pelaaja käy kahdesti, koska kisatilanteessa urheilija ampuu kahdesti makuulta ja kahdesti pystystä. Oli parempi kehitystyön kannalta laittaa peliin vain 2 tasoa ja tavallaan kierrättää niitä. Ampumistoiminnon kannalta varsinaisen luotia esittävän 3D-objektin sijasta Raycast-säde osoittautui paremmaksi. Se on nopeampi eikä kulje taulujen läpi.

Interaktiivinen sääominaisuus vaatisi käyttäjän laitteelta toimivan verkkoyhteyden ja GPS:än. Yhdistämällä GPS:stä saadut koordinaatit esimerkiksi Current Weather Data-ohjelmointirajapintaan, voi laite hakea verkosta sijainnin säätiedot. Jäsentämällä XML- tai JSON -muodossa olevat tiedot ne voi liittää ohjelmakoodissa muuttujiin tai taulukoihin. Sitten esimerkiksi heikon tai navakan tuulen arvot voi liittää sopiviin graafisiin objekteihin. Vaihtoehtoisesti ominaisuuden voi toteuttaa sekä ilman verkkoyhteyttä että GPS:ää, ja antaa pelin arpoa esimerkiksi tuulen voimakkuuden arvo ja suunta.

Internet-yhteys ja GPS asettavat ominaisuuden toteuttamiselle omat haasteensa. Myös XML tai JSON formaateista jäsennetyn datan liittäminen peliin monimutkaistaa prosessia hieman.

Pelin jatkokehityksen kannalta pelin keskeiset toiminnot ovat pääosin toimivia jo nyt. Käyttöliittymän elementit voisi uusia, ja peli voisi ottaa paremmin huomioon mobiililaitteen asennot. Kunnollinen skaalautuminen suurillekin resoluutioille on tärkeä ominaisuus, joka demosta vielä puuttui. Tasojen määrää peliin voi lisätä tai kehittää jonkinlaisen palkintotoiminnon. Peliin voi liittää myös useamman kuin yhden pelitilan.

LÄHTEET

[1] Unity3D 2014. Unity – Game engine, tools and multiplatform. Viitattu 14.12.2014. http://unity3d.com/unity

[2] Unity3D 2015 What's new in Unity 5.0. Viitattu 20.4.2015. http://unity3d.com/unity/whats-new/unity-5.0

[3] Okita, Alex. Learning C# Programming with Unity3D. 2014. Viitattu 4.5.2015.

[4] Rani, Aava. K. Learning Unity Physics. 2014. Viitattu 4.5.2015.

[5] Pixelplacement 2015. iTween for Unity. Viitattu 18.4.2015. http://itween.pixelplacement.com/index.php

[6] Unreal Engine 2015. What is Unreal Engine 4. Viitattu 21.4.2015. https://www.unrealengine.com/what-is-unreal-engine-4

[7] Hardware.fi 2015. DirectCompute auttaa Ambient Occlusion -efektin luomisessa. Viitattu 21.4.2015. http://www.hardware.fi/artikkelit/artikkeli.cfm/mita_directcompute_tarkoittaa_pelaajien_kannalta/2

[8] Unity3D 2015. Unity – Manual: Bloom. Viitattu 21.4.2015. http://docs.unity3d.com/Manual/script-Bloom.html

[9] Unreal 2014. Unreal Engine 4.6 Released. Viitattu 24.4.2015. https://www.unrealengine.com/blog/unreal-engine-46-released

[10] CryEngine 2015. CryEngine Sandbox. Viitattu 24.4.2015. http://cryengine.com/features/sandbox

[11] MonoDevelop | MonoDevelop 2015. Viitattu 21.4.2015. http://www.monodevelop.com/

[12] Search Networking 2014. What is location-based service (LBS)?. Viitattu 15.12.2014. http://searchnetworking.techtarget.com/definition/location-based-service-LBS

[13] Geoawesomenes 2012. Location-Based Services – Technologies. Viitattu 17.12.2014. http://geoawesomeness.com/knowledge-base/location-based-services/location-based-services-technologies/

[14] Jakl, Andreas. Grün, Christoph. Krösche, Jens. Drab, Stephan A. Pushing Location Based Games Further – How to Gain End User Suitability. 2009. Viitattu 8.1.2015.

[15] Järvinen, Petteri. IT-Tietosanakirja. 2003. Viitattu 20.4.2015.

[16] Webopedia 2015. What is Wi-Fi (IEEE 802.11x)? Viitattu 22.4.2015. http://www.webopedia.com/TERM/W/Wi_Fi.html

[17] Bluetooth 2015. Fast Facts | Bluetooth Technology. Viitattu 24.4.2015. http://www.bluetooth.com/Pages/Fast-Facts.aspx

[18] Geoawesomeness 2012. Location-Based Gaming. Viitattu 13.1.2015. http://geoawesomeness.com/location-based-gaming/

[19] Paelke, Volker. Oppermann, Leif, Reimann, Christian. Mobile Location-Based Gaming. 2008 Viitattu 15.1.2015

[20] Open Weather Map 2014. Current Weather Data. Viitattu 7.1.2015. http://openweathermap.org/current

[21] Forecast.io 2015. The Dark Sky API. Viitattu 23.4.2015. https://developer.forecast.io/docs/v2

[22] ProgrammableWeb 2014. Top 10 Weather APIs. Viitattu 23.4.2015. http://www.programmableweb.com/news/top-10-weather-apis/analysis/2014/11/13

[23] Unity 2015. Unity – Scripting API. Viitattu 11.2.2015. http://docs.unity3d.com/ScriptReference/LocationInfo.html

[24] Unity 2015. Unity – Scripting API. Viitattu 12.2.2015. http://docs.unity3d.com/ScriptReference/LocationService.html

[25] Surviving w/ Android 2014. Android Weather App. Viitattu 17.1.2015. http://www.survivingwithandroid.com/2013/05/build-weather-app-json-http-android.html

[26] Surviving w/ Android 2014. Android HTTP Client. Viitattu 19.1.2015. http://www.survivingwithandroid.com/2013/05/android-http-downlod-upload-multipart.html

[27] W3Schools 2015. HTTP Methods GET vs POST. Viitattu 22.4.2015. http://www.w3schools.com/tags/ref_httpmethods.asp

[28] Microsoft Developer Network 2015. System.XML Namespace(). Viitattu 24.2.2015. https://msdn.microsoft.com/en-us/library/system.xml.aspx