OULUN SEUDUN
AMMATTIKORKEAKOULU

Toni Kleemola

# WINDOWS PHONE -PELIN JATKOKEHITYS

# WINDOWS PHONE -PELIN JATKOKEHITYS

Toni Kleemola
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

# TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistojen kehitys

---

Tekijä(t): Toni Kleemola
Opinnäytetyön nimi: Windows Phone -pelin jatkokehitys
Työn ohjaaja(t): Juha Alakärppä
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä: *51*

---

Tämän opinnäytetyön tavoitteena oli jatkaa Windows Phone -mobiililaitteella toimivan pelin kehitystä ja saada pelille lisää sisältöä julkaistavaksi sovelluskauppaan. Opinnäytetyön alkaessa pelistä oli jo ennestään olemassa yhden pelitason mittainen kokeiluversio alkuvalikkoineen. Tarkoituksena oli suunnitella ja toteuttaa peliin lisää pelattavaa sisältöä ja ominaisuuksia, ottaen samalla hieman osittaista projektipäällikön vastuuta kehitystyön etenemisestä pienehkössä projektiryhmässä.

Projektin tavoitteena oli saada pelille lisää sisältöä julkaisuun tasaisessa ja nopeassa tahdissa. Dokumentoinnin osalta projektiin kuului pääasiassa koodin selkeä kommentointi. Näistä syistä pelin rakenteelliseen suunnitteluun ja muihin dokumentaatioihin ei käytetty paljon aikaa, vaan projektissa pyrittiin saamaan loppukäyttäjälle näkyvää sisältöä nopeasti valmiiksi. Projektin toteuttamisessa käytettiin hyväksi alkuperäistä pohjaa ja pyrittiin jatkamaan mahdollisimman pienillä muutoksilla.

Työn aikana pelin alkuperäisversioon tehtiin muutamia pieniä rakenteellisia muutoksia jatkossa toteutettavien ominaisuuksien helpottamiseksi ja niiden toteuttamiseksi. Pelille suunniteltiin ja toteutettiin erikseen tekoälytoiminnallisuutta ja pelin edetessä vaikeutuvien tasojen sisältöä useamman tason pituisesti. Tähän kuului pelitasojen sisältö normaaleine vihollisineen, niiden loppuvastustajat ja tasojen haastavuuden balansointi. Näiden lisäksi peliin suunniteltiin ja toteutettiin useita pieniä uusia peliominaisuuksia pelin mielenkiintoisuuden lisäämiseksi. Pelin alkuperäinen ulkoasu muuttui hieman työn aikana.

Tuloksena työlle tehtiin pelitasoja loppuvastustajineen useamman tason edestä varastoon. Peliin tehtiin myös erilaisia pieniä lisäominaisuuksia ja parannuksia mielenkiinnon lisäämiseksi. Näiden lisäksi alkuperäistä koodia kommentoitiin ja muokattiin helpommin luettavaksi.

---

Asiasanat: Windows Phone, C#, mobiilipelit, ohjelmointi, tekoäly

# SISÄLLYS

## SANASTO

| | |
|---|---|
| API | Application Programming Interface on rajapinta, jonka avulla sovelluskomponentit voivat keskustella keskenään |
| App Hub | Microsoftin ylläpitämä sivusto ja palvelu, jossa ohjelmistokehittäjät voivat julkaista ja myydä Windows Phone- ja XBOX360-sovelluksia |
| C# | C-sharp on Microsoftin kehittämä oliopohjainen ohjelmointikieli |
| FPS | Frames per second on määritelmä siitä, kuinka monta kuvaa näytölle piirretään sekunnin aikana |
| HUD | Heads-Up Display on läpinäkyvä komponentti muun sisällön päällä, joka voi näyttää erilaista tietoa käyttäjälle |
| IDE | Integrated Development Environment on ohjelmistoympäristö, jota käytetään sovellusten tekemiseen |
| Windows Phone | Microsoftin kehittämä mobiililaitteille tarkoitettu käyttöjärjestelmä, jolla voidaan myös viitata yleisesti Windows Phonen versiota 7.0 ja sitä uudempia käyttäviin mobiililaitteisiin |
| XNA Framework | Microsoftin tekemä peliohjelmoinnille tarkoitettu kirjasto |
| SDK | Software Development Kit on ohjelmistotyökalu, joka mahdollistaa tietynlaisien sovellusten toteuttamisen mukanaan tulevien kirjastojen avulla |
| Silverlight | Microsoft Silverlight on flashin tapainen web-ohjelmointiympäristö |
| XAML | Extensible Application Markup Languace on XML-pohjainen kieli käyttöliittymien toteuttamiseen |

# 1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli jatkaa Windows Phone -mobiililaitteella toimivan rakettipelin kehitystyötä eteenpäin pienessä projektiryhmässä. Tehtäviin kuului suunnittelun ja toteutuksen lisäksi pienen osittaisen vastuun ottaminen projektin etenemisestä eräänlaisena projektipäällikkönä. Projektiryhmä muodostui muutaman henkilön kokoisesta ryhmästä, johon kuului lisäkseni pari muuta harjoittelijaa, jotka toimivat projektin aikana muun muassa suunnittelijoina, graafikkona ja ohjelmoijana.

Pelin alkuperäinen kehitystyö oli aloitettu ennen tämän projektin alkamista. Tuona aikana pelin nimenä käytettiin itse projektin nimeä eli Project Ghost Moth, joka myöhemmin muutettiin Journey to Edeniksi. Lähtökohtana tehdyn projektin sisältöön kuului yksinkertaisella toiminnallisuudella varustettu yhden pelitason mittainen julkaisematon kokeiluversio. Kokeiluversioon kuului yhteensä päävalikko ja yhden pelitason mittaisesti pelattavaa.

Opinnäytetyön alkuperäisenä tavoitteena oli päästä nopeaan julkaisutahtiin, jossa pelille saataisiin pieniä päivityksiä tasaisin väliajoin. Tarkoituksena oli jatkaa pelin suunnittelua ja kehitystyötä eteenpäin ja saada pelille samalla lisää julkaisukelpoista sisältöä sovelluksen loppukäyttäjien iloksi.

Tässä dokumentissa käydään läpi ensiksi hieman Windows Phone -tekniikkaan ja sen ohjelmistokehitykseen liittyvää tietoa yleisesti. Toisena käydään läpi hieman peliohjelmointiin liittyviä yleisiä asioita ja peleissä käytettävien tekoälyjen yleisimpiä ratkaisuja. Kolmantena esitellään työn kohteena toimineen pelin rakennetta ja sen toiminnallisuutta yleisellä tasolla. Neljäntenä käydään läpi itse työn aikana toteutettua sisältöä lyhyiden demonstraatioiden avustamana ja selvitetään joidenkin toteutettujen ratkaisujen taustaa. Toteutuksessa käydään läpi pääsääntöisesti vain niitä asioita, joiden toteutuksen olen hoitanut itse kokonaan tai suurimmaksi osaksi. Viimeiseksi dokumentista löytyy yhteenveto, jossa kerrotaan lyhyesti työn sisältö uudestaan ja saavutetut lopputulokset.

# 2 WINDOWS PHONE -TEKNIIKKA

Windows Phone on Microsoftin kehittämä mobiililaitteille tarkoitettu käyttöjärjestelmä (kuva 1). Windows Phone on uudelleen tehty ja paranneltu versio aiemman Windows Mobile -nimellä kulkeneen mobiilikäyttöjärjestelmän tilalle. Windows Phoneksi kutsutaan yleisesti kaikkia kyseisen käyttöjärjestelmän versiota 7 ja sitä uudempia käyttäviä mobiililaitteita. Windows Mobile -sovellukset eivät ole yhteensopivia Windows Phone -käyttöjärjestelmän kanssa. (1; 2.)


*KUVA 1. Windows Phone Nokia Lumia 710- ja 800-puhelimissa (3)*

## 2.1 Historia

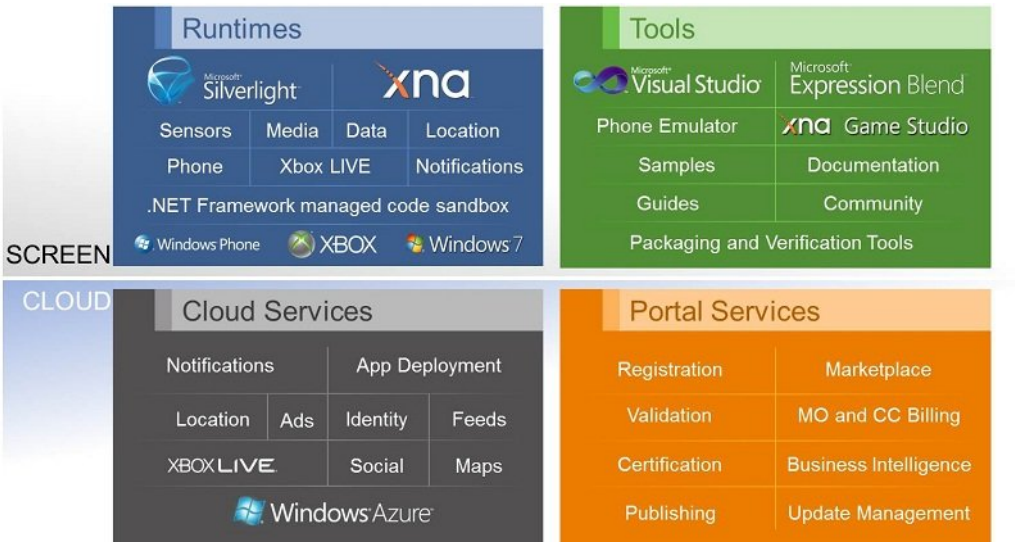
Alkuperäinen kehitystyö Windows Phonelle on arvelujen perusteella voinut alkaa jo vuonna 2004 koodinimellä Photon, mutta tällöin projekti lopetettiin hitaan etenemisen vuoksi. Virallisesti Windows Phonen kehitystyö alkoi vuonna 2008. Tällöin Microsoft järjesti uudelleen Windows Mobile -kehitystyöryhmän ja alkoi työstää uutta mobiilikäyttöjärjestelmää, joka nykyään tunnetaan paremmin nimeltä Windows Phone. (2.)

Windows Phonen ensimmäisen julkaisun oli tarkoitus tapahtua vuonna 2009, mutta erilaisten viivästysten takia sen tilalla julkaistiin väliaikaisena versiona Windows Mobile 6.5. Se piti sisällään osittaisia ominaisuuksia varsinaiseen Windows Phoneen tulevasta sisällöstä. Ensimmäinen virallinen Windows Phone 7 paljastettiin vuoden 2010 alussa ja se tuli myyntiin saman vuoden lokakuussa. Näihin aikoihin myös alkuperäisenä sarjan nimenä käytettiin Windows Phone 7 Seriesiä, josta luovuttiin ennen tuotteen tuloa myyntiin ja siirryttiin yksinkertaisempaan muotoon eli Windows Phone 7:ään. (2.)

Nykyisin käyttöjärjestelmän uusin versio on Windows Phone 7.5, jonka ensimmäiselle versiolle annettiin koodinimeksi Mango. Tälle versiolle on tehty jälkeenpäin uudistus sisältäen erilaisia parannuksia ja kulkee koodinimellä Tango. Tulevan sukupolven Windows Phone versioihin viitataan koodinimellä Apollo. (2; 4.)

### 2.2 Sovellusalusta

Windows Phonen ohjelmistoalusta on rakennettu suoraan Microsoftin työkalujen ja teknologian päälle, kuten Visual Studio, Expression Blend, Silverlight ja XNA Framework. Nämä ohjelmat tekevät ohjelmistokehityksestä helppoa kaikille niille, joille nämä teknologiat ovat jo ennestään tuttuja. Ohjelmistoalustan arkkitehtuurin koostuu pääsääntöisesti neljästä eri komponentista (kuva 2). Näitä ovat Runtimes, Tools, Cloud Services ja Portal Services. (5.)

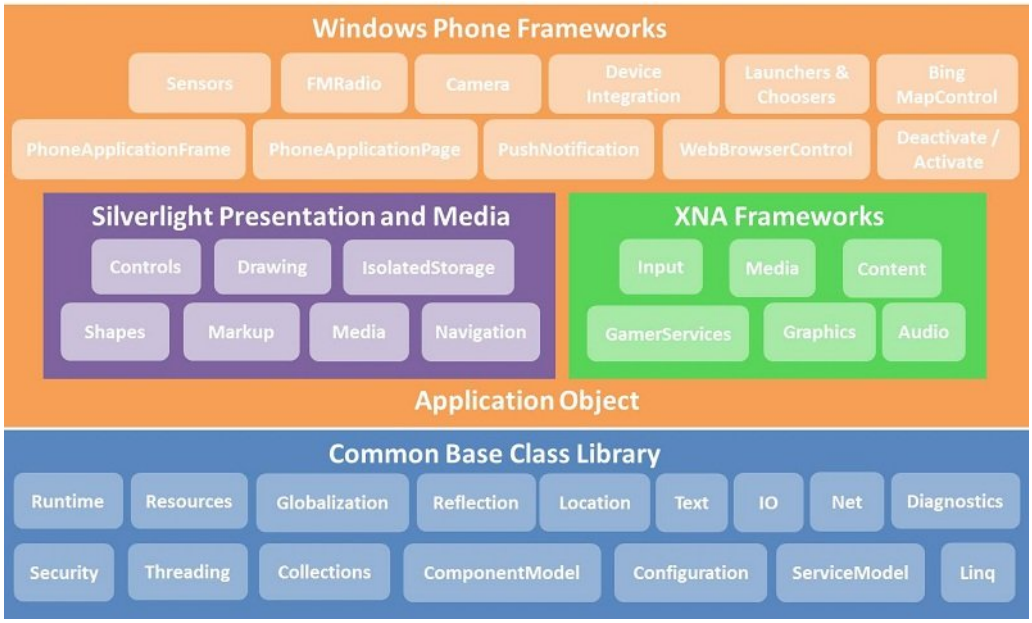



*KUVA 2. Windows Phone -sovellusalustan arkkitehtuuri (5)*

**Runtimes**

Runtimes tuo ohjelmistokehittälle valmiita alustoja sovelluskehitystä varten (kuva 3). Silverlight- ja XNA Framework -alustoilla ohjelmistokehitys tapahtuu valmiiden alustojen päällä omalla suojatulla ja eristetyllä alueella. Näin voidaan toteuttaa nopeasti uusia ja turvallisia sovelluksia Windows Phone -laitteille. Silverlight ja XNA Framework, muiden Windows Phone -ominaisten komponenttien ja yleisen pohja kirjastojen kanssa, antavat runsaat mahdollisuudet erilaisten sovellusten toteuttamiseen. (5.)

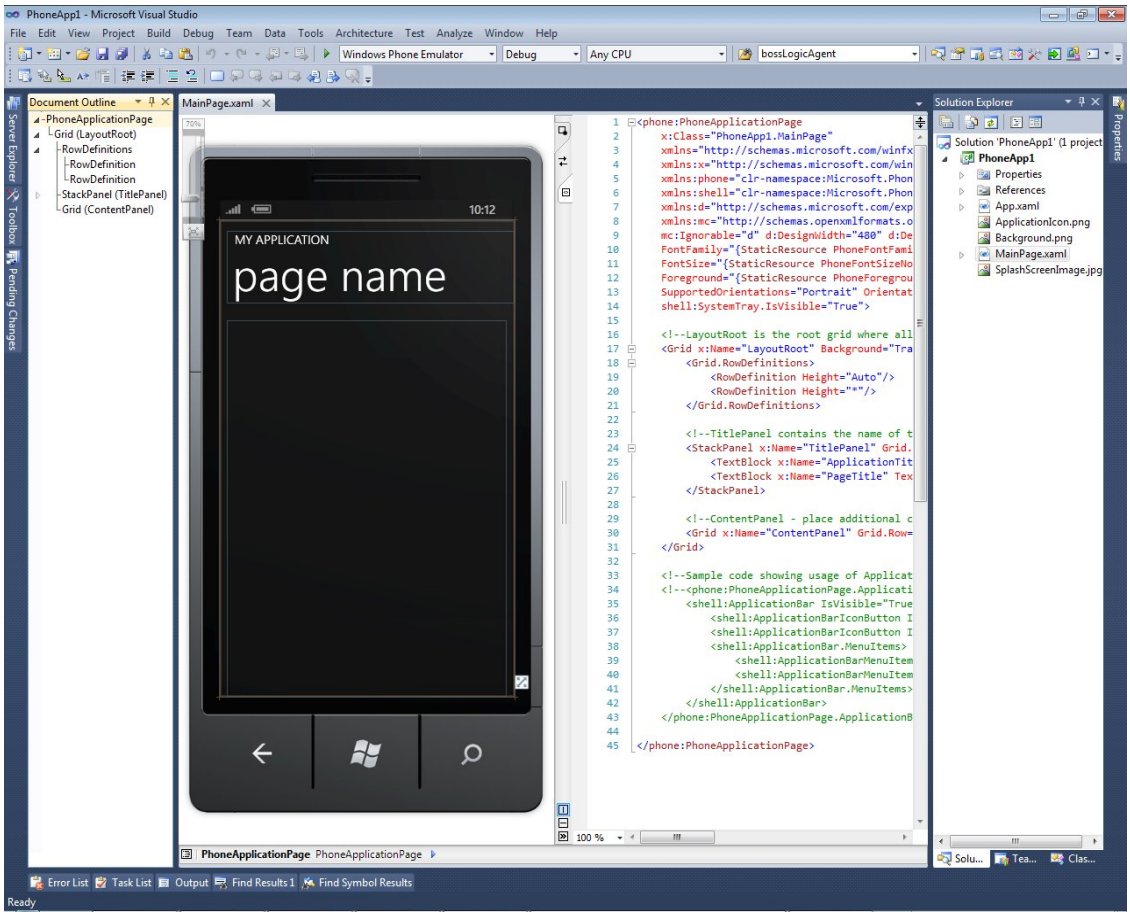
Runtimes antaa ohjelmoijalle käyttöön myös monia Windows Phone -ominaisia komponentteja. Yksi näistä on sensorit, joiden avulla laitteelta saadaan luettua monenlaista tietoa, kuten kallistusasento, mikrofoni ja kosketukset. Toisena on media komponentit, joiden avulla voidaan toteuttaa erilaisen median toisto, kuten musiikki ja videot. Datakomponentit taas mahdollistavat omat käyttöjärjestelmästä eristetyn muistin käyttämisen sovelluksen tarpeisiin. Location eli sijaintikomponentit taas antavat ohjelmistokehittäjille mahdollisuuden käyttää toiminnallisuutta, jonka avulla voidaan saada selville esimerkiksi käyttäjän sijainti. (5.)




*KUVA 3. Windows Phone Framework (5)*

## Tools

Työkaluihin kuuluvat erilaiset sovellukset, jotka mahdollistavat sovellusten kehittämisen itsessään. Näihin kuuluu Visual Studio, joka toimii ohjelmistoympäristönä Windows Phone -sovellusten toteuttamiseen (kuva 4). Tämän sovelluksen avulla ohjelmistokehittäjät voivat toteuttaa Silverlight- ja XNA Framework-sovelluksia tai käyttää molempien yhdistelmiä sovelluksissaan. (5.)




*KUVA 4. Microsoft Visual Studio 2012- ja Silverlight-sovellusten kehitysnäkymä*

Toisena on Expression Blend, joka on graafisen käyttöliittymän toteuttamiseen tarkoitettu sovellus. Sovelluksen avulla kehittäjät voi luoda XAML-pohjaisia käyttöliittymiä Windows Phone Silverlight -sovelluksille. Tämä tapahtuu täysin samalla tavalla, kuten sovelluksen avulla luodaan nettiselaimella toimivia sovelluksia. (5.)

Visual Studioon ja Expression Blendiin on integroitu mukaan Windows Phone -emulaattori (kuva 5). Emulaattorin avulla ohjelmistokehittäjät voivat testata ja poistaa virheet sovelluksistaan vaivattomasti ennen julkaisua. Emulaattori mahdollistaa täysin sovellusten asentamisen, niiden testaamisen ja ajamisen. Emulaattoriin sisältyy myös tuki GPU-emulaatiolle ja laitteen asennon emuloinnille. (5.)


*KUVA 5. Windows Phone -emulaattori*

XNA Game Studio on integroitu suunnitteluympäristö, jonka avulla ohjelmistokehittäjät voivat luoda pelejä Microsoft Windows -käyttöjärjestelmälle, Microsoft Xbox 360 -konsolille, Microsoft Zune- ja Windows Phone -laitteille. Game Studio jatkaa Visual Studiota lisäosalla, joka antaa tuen XNA Frameworkille ja sisältää myös omia työkaluja, joiden avulla voidaan tuoda media sisältöä peliin. (5.)

Näiden lisäksi ohjelmistokehittäjille on tarjolla monenlaisia eri dokumentaatiota, koodiesimerkkejä ja oppaita helpottamaan kehitystyötä (kuva 2). Ohjelmistoke-

hittäjille löytyy monenlaista blogia, keskustelupalstoja ja nettisivustoja, joilta löytyy apua sovellusten toteuttamiseen. (5.)

### Cloud Services

Sovellusalusta tarjoaa useita ominaisuuksia, jotka mahdollistavat web-pohjaisten sovellusten tekemisen. Pilvipalvelut tuovat Windows Phone -sovellusalustalle lisää toiminnallisuutta säästäen samalla laitteen akkua. Laitteelle voidaan ladata helposti dataa kolmannen osapuolen nettipalveluista. Pilvipalveluihin kuuluu muun muassa Windows Azure, Xbox Live-, ilmoitus- ja sijaintipalvelut muiden web-palveluiden mukana. (5.)

Pilvipalvelut mahdollistavat monen muun toiminnallisuuden seasta muun muassa mainospalveluiden lisäämisen sovellukseen. Tämä tapahtuu käyttämällä Microsoft Advertisement SDK for Windows Phonea. SDK:n avulla sovellusten kehittäjät voivat lisätä helposti teksti- ja kuvamainoksia sovelluksiinsa. (5.)

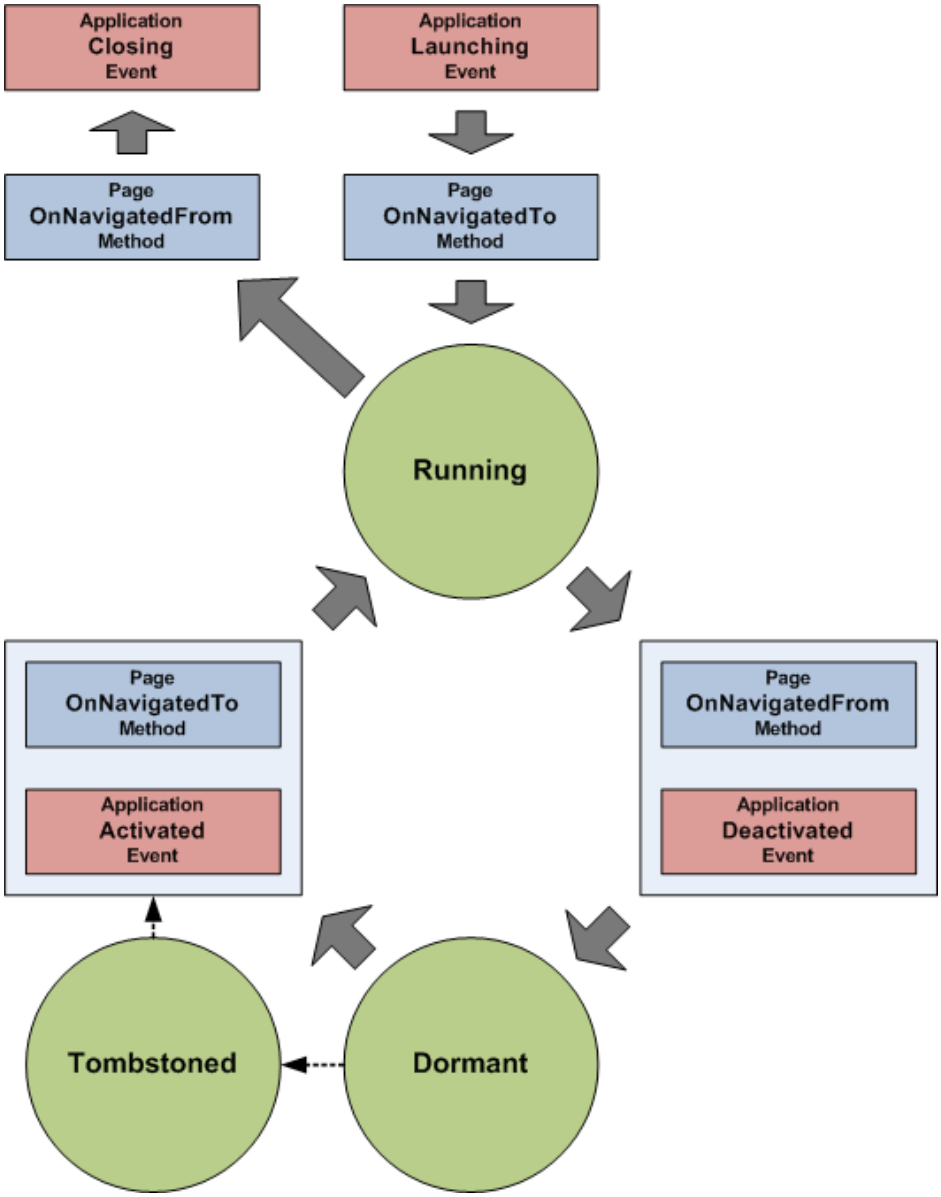
### Portal Services

Portal Services tarjoaa sovelluskehittäjille omia palveluita, joiden avulla voidaan rekisteröidä, sertifioida ja myydä sovelluksia. Sovelluskehittäjille tarjolla olevilla työkaluilla voidaan helposti lähettää sovellus sertifioitavaksi julkaisua varten Windows Phonen kauppapaikkaan. Maksulliselle sovellukselle asetettavat hinnat voidaan myös määrittää näiden palveluiden avulla. Maksutapoina on tuettuina sekä luottokortti ja mobiililiittymän tarjoajan kautta maksuttaminen. Business Intelligencen työkalujen avulla kehittäjät erilaista statistiikkaa sovelluksestaan. (5.)

## 2.3 Tilanhallinta

Windows Phonen suoritusmalli sisältää sovelluksen elinkaaren käynnistyshetkestä sen sulkemiseen. Sen tarkoituksena on antaa käyttäjälle nopea ja toimiva käyttöympäristö. Windows Phone sallii vain yhden aktiivisen sovelluksen suorituksen kerrallaan. Kun sovellus ei ole enää aktiivinen, se asettuu lepotilaan. Muistin loppuessa aktiiviselta sovellukselta käyttöjärjestelmä alkaa automaattisesti sulkea lepotilassa olevia ohjelmia aloittaen vähiten käytetystä. (6.)

Sovelluksen käynnistymishetkestä lähtien sille voidaan määrittää kolme eri tilaa. Ensimmäinen on Running-tila, joka vastaa sovelluksen normaalia aktiivista ajoa. Toisena on Dormant-tila, johon sovellus asettuu, kun se on vielä käynnissä, mutta siirretty taustalle. Kolmantena on Tombstoned-tila, jolla tarkoitetaan tilaa, jossa Dormant-tilassa oleva sovellus on suljettu, mutta joitakin tietoja löytyy vielä muistista. Tombstoned-tilassa oleva sovellus käynnistyy uudestaan alusta asti, kun se palautetaan aktiiviseksi, mutta muistista löytyvän osittaisen tiedon avulla, se voidaan palauttaa takaisin viimeiseen näkymään. Kuvassa 6 on esimerkki sovelluksen tilojen elinkaaresta. (6.)




*KUVA 6. Sovelluksen elinkaari (6)*

## 2.4 Moniajo

Windows Phone 7 -käyttöjärjestelmässä on ollut tuki moniajolle alusta lähtien, mutta sen käyttö oli ensiksi rajoitettu vain sisäisten sovellusten käyttöön. Windows Phone 7.5:n julkaisun mukana tuli muutamia ominaisuuksia, joiden avulla myös kolmannen osapuolten sovellukset pystyivät käyttämään moniajoon tarvittavaa toiminnallisuutta. (7.)

Pääsääntöisesti Windows Phone sallii vain yhden sovelluksen olemisen aktiivisena kerrallaan. Tämän tyylinen toiminnallisuus on toteutettu, jotta resursseja saataisiin säästettyä ja mobiililaite pysyisi helposti reagoivana. Tähän on kuitenkin määritelty joitakin poikkeuksia kuten musiikin toistaminen ja tiedonsiirto. Nämä toiminnallisuudet pystyvät pyörimään taustalla myös toisen sovelluksen ollessa aktiivisena. (7.)

Kolmannen osapuolen sovelluksien moniajo Windows Phone -laitteilla tapahtuu erilaisten Live Agentien avulla. Live Agentin avulla sovellukset pystyvät käyttämään kahta eri tapaa taustalla pyörimiseen. Ensimmäinen on säännöllisesti ajoitettu lyhytkestoinen jakso, jonka sovellus saa käyttöönsä. Toinen on samalla periaatteella toimiva pidempi aikajakso, joka on tarkoitettu enemmän resursseja vaativien toimintojen ajamiseen. Käyttöjärjestelmä kuitenkin rajoittaa itse taustalla ajettavan sovelluksen toimintaa, jotta se ei hidastaisi aktiivisena olevaa sovellusta. (7.)

## 2.5 Ohjelmistokehitys

Windows Phonen käyttämä sovellusalusta tarjoaa ohjelmistokehittäjille kaksi eri mahdollisuutta ohjelmien toteuttamiseen. Näitä ovat pelikehitykseen tarkoitettu XNA Framework ja enemmän yleiseen ja XAML-tyyliseen sovelluskehitykseen tarkoitettu Silverlight (kuva 3). Ohjelmistokehittäjä voivat myös yhdistää sovelluksissaan nämä kaksi teknologiaa ja täten luoda monipuolisia sovelluksia. (5.)

Ohjelmointi Windows Phone -laitteille tapahtuu pääasiassa Microsoftin omalla Visual C# -ohjelmointikielellä, mutta mukaan on lisätty jälkeenpäin tuki myös Visual Basic -ohjelmointikielelle. Tämän lisäksi XAML-kieltä käytetään Silverlight-sovelluksien toteutuksessa. Ohjelmistokehitys tehdään Visual Studio työkalujen

avustamana tai muiden aiemmassa kappaleessa mainittujen työkalujen avustamana. Windows Phone Developer Tools toimii ainoastaan Windows Vista SP2:lla tai uudemmalla käyttöjärjestelmällä (2).

### 2.6 Sovelluskauppa

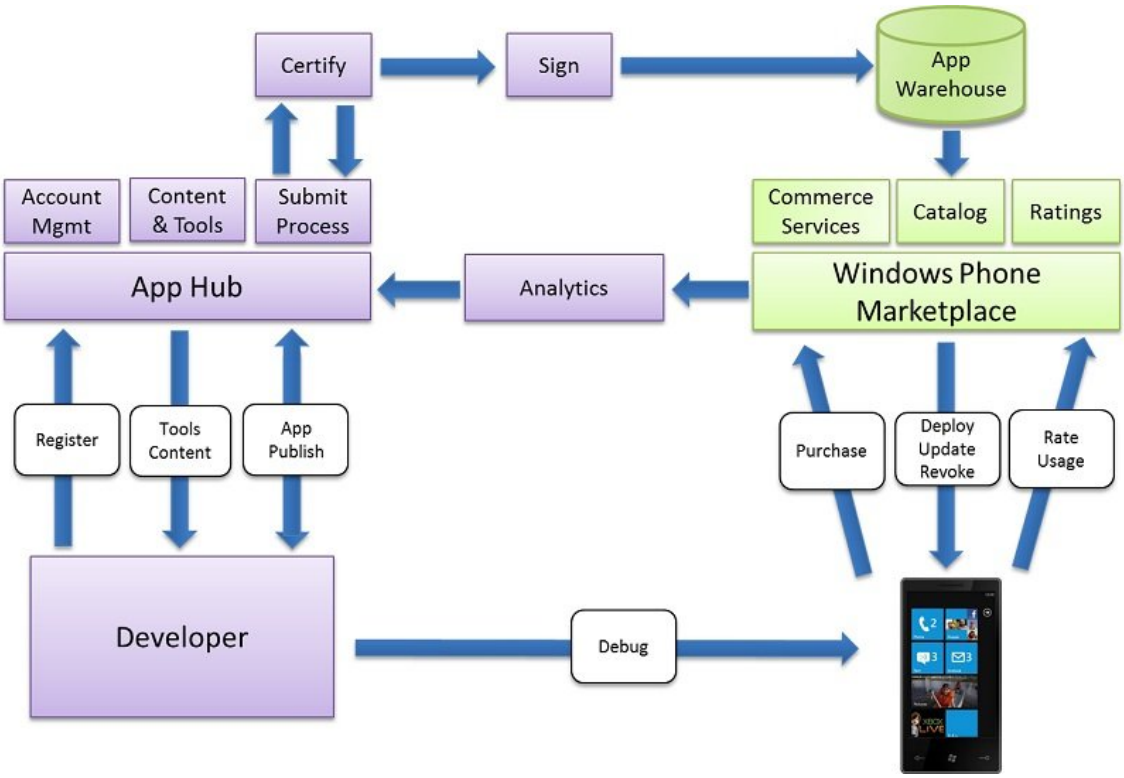
Windows Phone Marketplace on Microsoftin ylläpitämä ja tarjoama palvelu Windows Phone -laitteille. Palvelun avulla laitteiden käyttäjät pystyvät selaamaan ja lataamaan erilaisia tarjolla olevia kolmannen osapuolen toteuttamia ohjelmia. Windows Phone -laitteiden käyttäjät voivat palvelun avulla ladata käyttöönsä erilaisia ohjelmia, pelejä ja mediasisältöä. Marketplace jakautuu kahteen osaan: Windows Phone Marketplace ja Zune Marketplace. Windows Phone Marketplace on tarkoitettu enemmän kolmannen osapuolen sovellusten kuten pelien jakoon. Zune Marketplace taas hoitaa mediapuolen, johon kuuluvat musiikki- ja videotiedostot, ja sen käyttäminen tapahtuu Zune-sovelluksen avulla. (8.)

Sisältöä ladataan sovelluskaupasta pääasiassa valmiina laitteesta löytyvän Marketplace-toiminnallisuuden avulla. Lataamista varten käyttäjän täytyy aktivoida itselleen Windows Live ID. Sisältöä voi ladata laitteelle myös suoraan tietokoneelta Zune-sovelluksen avulla. Maksutapoina sovelluskaupassa tuetaan luottokortti- ja operaattorimaksutapoja. (5.)

Sovelluskehittäjän halutessa sovelluksia sovelluskauppaan täytyy hänen ensiksi rekisteröityä sovelluskehittäjäksi Microsoftin App Hub -sivustolla. Tämä maksaa yrityksille 99 dollaria ja toimii vuosittaisena maksuna. Tämän jälkeen sovelluskehittäjäksi rekisteröitynyt voi lähettää 100 kappaletta ilmaisia sovelluksia sovelluskauppaan. Määrän täyttyessä aletaan uusista sovelluksista periä 19,99 dollarin suuruinen lisämaksu. DreamPark-opiskelijat voivat rekisteröityä ja käyttää palvelua ilmaiseksi. (8; 9.)

Sovelluskehittäjä saa itse valita halutessaan hinnan sovelluksellensa 0,99 dollarin minimihinnan ja 499,99 dollarin väliltä. Sovelluksen ollessa maksullinen Microsoft perii itselleen 30 % voitosta ja loput päättyvät valmistajalle. Tämän lisäksi sovelluksen vähittäismyynnin täytyy ylittää 200 dollarin minimisumma, jotta tekijälle maksetaan voittoa sovelluksesta ollenkaan. (10; 11.)

Sovelluksen lähettäminen kauppaan on helppoa. Valmistajan täytyy vain lähettää XAP-tiedosto sertifioitavaksi, täyttää muutama tekstikenttä kuvineen ja jäädä odottamaan vastausta. Tämän sijaan Microsoftin sertifiointirutiini itsessään noudattaa tiukkoja tarkistuksia ja standardeja, jotka sovelluksen täytyy läpäistä. Muussa tapauksessa sovellus hyllytetään ja valmistaja joutuu korjaamaan viat ja lähettämään uuden korjatun XAP-tiedoston sertifioitavaksi, saadakseen sen sovelluskauppaan näkyville. Tarkempia ohjeistuksia ja vaatimuksia sertifiointiin löytyy App Hub -sivustoilta. Kuvassa 7 näytetään graafisesti etenemisprosessi sovelluskauppaan lähetettävälle sovellukselle.




*KUVA 7. Windows Phonen ohjelmistokehityksen elinkaari (5)*

# 3 PELIOHJELMOINTI

## 3.1 Pelimoottori

Pelimoottorilla tarkoitetaan pelin päärakennetta, jota käytetään pelien toteuttamisessa. Tämä kattaa monenlaisen ja pelien välillä toistuvan pelimekaniikan hallinnan, johon kuuluu esimerkiksi grafiikan moottori, fysiikka moottori, törmäystarkistus, äänet, animointi ja niin edelleen. (12.)

## 3.2 Pelisilmukka

Pelisilmukalla tarkoitetaan yleisesti yhtä toistuvaa silmukkaa, jonka aikana taustalla pyöritetään koko ajan pelin varsinaista toiminnallisuutta. Rakenteeltaan pelisilmukassa käydään yleensä läpi muutamia asioita järjestyksessä yksi asia kerrallaan. Näitä voivat olla esimerkiksi pelaajan syötteiden tarkistus ja vihollisten sijainnin päivittäminen. Esimerkissä 1 näkyy yksinkertainen pelisilmukka. (13.)

```
// yksinkertainen pelisilmukka
while(run)
{
   check_user_input();
   update_ai();
   move_enemies();
   check_collision();
   draw_graphics();
   play_sounds();
}
```

*ESIMERKKI 1. Yksinkertainen pelisilmukka (13)*

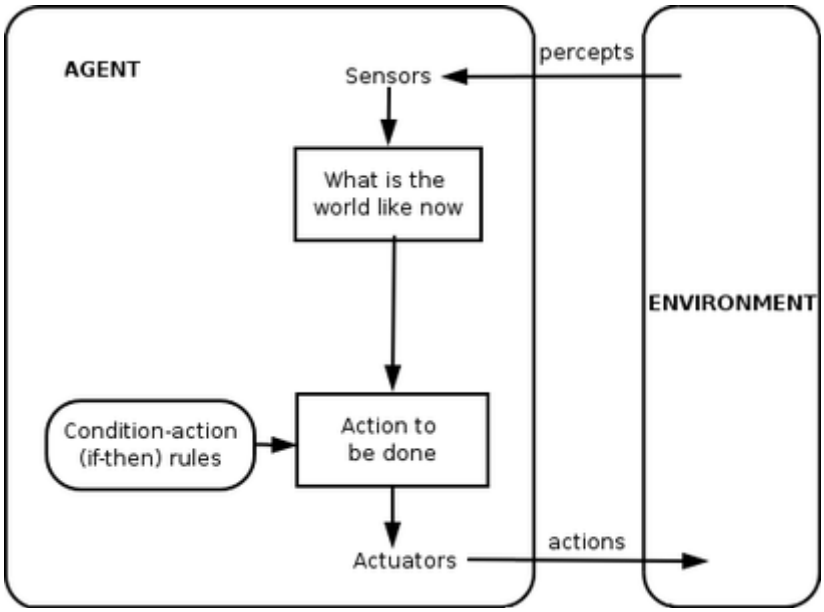
XNA Framework hoitaa itse pelisilmukan pyörittämisen ja ohjelmoijan täytyy vain täyttää siltä perittävät Update- ja Draw-metodit. Update-metodin tarkoituksena on päivittää pelilogiikka ja Draw-metodi taas hoitaa jokaisen framen eli kuvan piirtämisen. Näitä metodeja kutsutaan automaattisesti kerran jokaista framea kohti, mutta niiden ajastusta voidaan myös muuttaa. Draw-metodi on mahdollista laittaa pyörimään myös eri tahdissa kuin Update-metodi. Windows Phonen pelit pyörivät oletuksena 30 fps:n nopeudella. (14.)

### 3.3 Tekoäly

Tekoälyllä viitataan yleisesti tietokoneistettujen hahmojen itsenäiseen päätöksentekoon. Tekoälyjen toteutukseen voidaan käyttää useampia eri tapoja tilanteen mukaisesti. Tekoälyjä voidaan myös lajitella useampaan eri ryhmään niiden älykkyyden perusteella. (15.)

#### 3.3.1 Intelligent Agent

Intelligent Agentilla tarkoitetaan tekoälyssä itsenäistä toiminnallisuutta ja logiikkaa, jolla tietokoneistettu hahmo tekee päätöksiä. Näihin voi kuulua erilaisia sensoreita ja monimutkaisia laskukaavoja, joiden avustamana päätökset tehdään (kuva 8). Yksinkertaisena esimerkkinä voisi olla sensori, joka laskee etäisyyden lähimpään seinään ja kääntää hahmoa poispäin seinän lähestyessä. (16.)



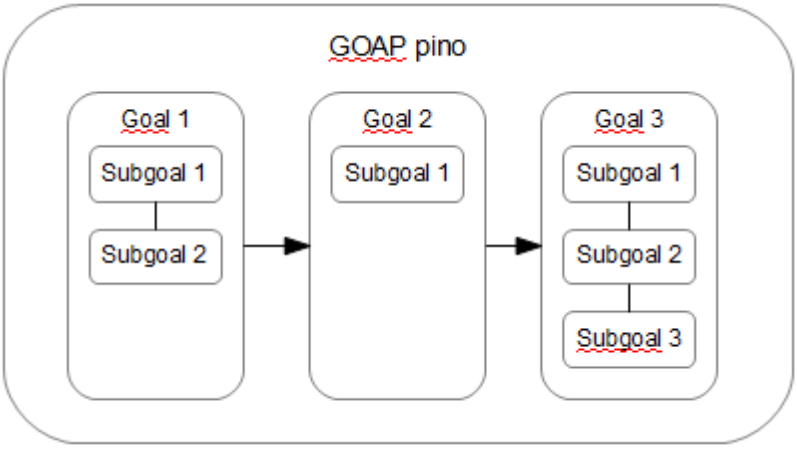

*KUVA 8. Intelligent Agent (16)*

#### 3.3.2 Goal-oriented action planning

Goal-oriented action plannilla, lyhennettynä GOAP, tarkoitetaan tavoitepohjaista tekoälyä. Tavoitepohjaisessa tekoälyssä tietokoneistetulle hahmolle annetaan oma tavoite, johon pyritään pääsemään. Tavoitteen toteuttamiseksi tekoäly toteuttaa tarpeelliset toimenpiteet ja jatkaa tämän jälkeen seuraavaan mahdolli-

seen tavoitteeseen. Tavoitteen alla pystyy myös olemaan alitavoitteita, jotka toteutetaan, kun päätavoite saavutetaan. (Kuva 9.)

Yksinkertaisena esimerkkinä tavoitepohjaisesta tekoälystä voisi käyttää tilannetta, jossa halutaan saada hahmo kiipeämään katolle ja ottamaan sieltä esine. Tässä tapauksessa hahmon päätavoitteena on päästä katolle ja toisena tavoitteena ottaa esine. Hahmo joutuu etsimään tavan päästä katolle, kuten tikapuut, ja käyttämään niitä. Tämä voidaan lukea toimenpiteisiin, joka vaaditaan katolle pääsemiseksi. Seuraavaksi hahmo joutuu etsimään katolta haluamansa esineen ja ottaa sen käyttöönsä.




*KUVA 9. Goal-Oriented Action Planning*

### 3.3.3 Pathfind

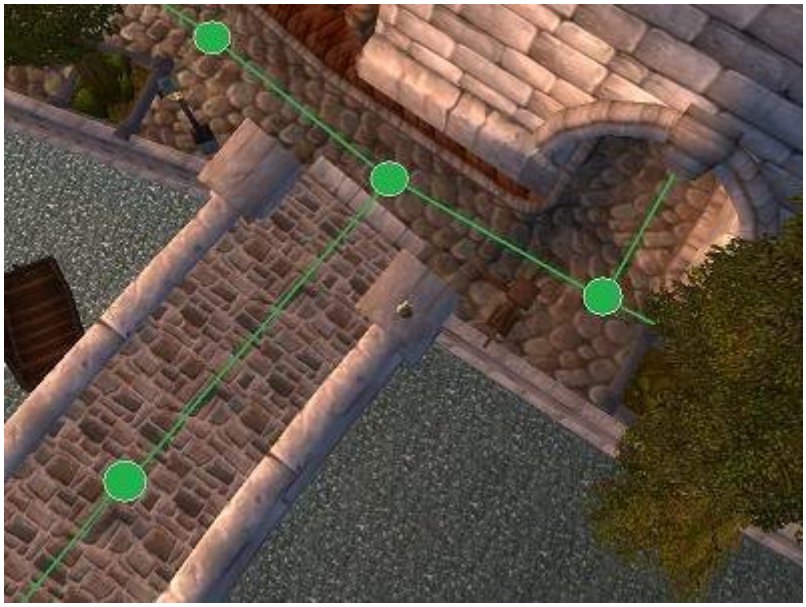
Pathfind, eli suomennettuna polunlöytämisellä, tarkoitetaan tietokoneistettujen hahmojen tapaa hakeutua tietystä pisteestä kohteena annettuun pisteeseen. Tekoälyssä käytettyjä yleisimpiä polunlöytämiseen ja liikkumiseen käytettäviä mekanismeja ovat A*-algoritmi, waypoint ja navigation mesh.

#### Waypoint

Waypointeilla tarkoitetaan ennaltaan määritettyä pisteistä koostuvien sijaintien sarjaa, joiden kautta tietokoneistettu kappale pystyy liikkumaan paikasta toiseen. Jokaisella waypointilla on oma koordinaattinsa, joka voidaan esittää kaksiulotteisena tai kolmiulotteisena käyttäen x-, y- ja z-koordinaatteja. Waypointeil-

le voidaan tarvittaessa antaa myös säde, jonka alueelta tekoälyinen kappale pystyy liikkumaan waypointin ohitse.

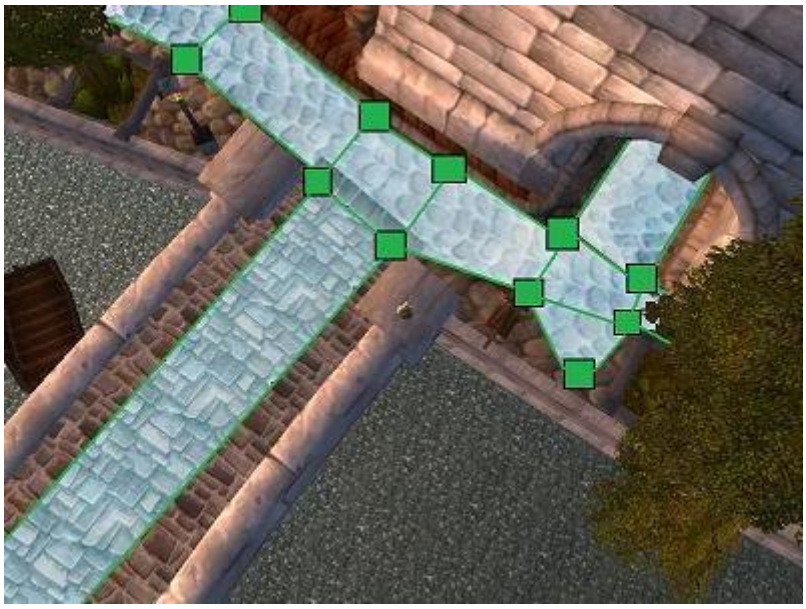
Waypointien vahvuuksiin voidaan lukea niiden yksinkertaisuus ja helppous toteuttaa. Waypointit toimivat helppona tapana saada kappale liikkumaan paikasta A paikkaan B. Waypointien heikkous taas tulee esille, kun halutaan kattaa suuri liikkumisalue. Tällöin liikuttavana toimiva alue joudutaan täyttämään usealla waypointilla ja tekoälyinen kappale liikkuu logiikkansa mukaan suoraan lyhintä waypointeista koostuvaa reittiä pitkin kohteeseen. Tämä voi tapahtua hieman ristiin ja rastiin mennen waypointien määrän ja niiden yhdyskäytävän rajoittamana. Kuvassa 10 näkyy yksinkertainen graafinen esimerkki waypointeista. (17.)



*KUVA 10. Waypoint-kaavio (17)*

**Navigation mesh**

Navigation meshillä eli navigointiverkostolla tarkoitetaan monikulmiosta muodostuvaa aluetta, jonka alueella tekoälyinen hahmo voi liikkua (kuva 11). Vahvuuksina navigaatioverkostolla on sen laaja kattavuus suuremmilla alueilla. Navigointiverkostoa voidaan verrata tietyllä tapaa waypointeihin, joilla on säde. Vastaavanlaisissa tapauksissa navigaatioverkosto kattaa kuitenkin huomattavasti tarkemmin pelialueen kuin ympyrät. Navigaatioverkostolla saadaan myös helpommin tietokoneistetut hahmot liikkumaan sulavammin ilman, että täytyy käyttää useita eri pisteitä käännöstä kohti. (17.)



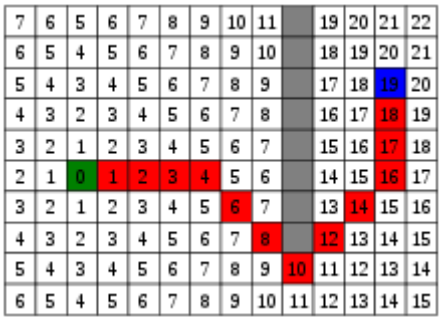
*KUVA 11. Navigation mesh (17)*

**A*-algoritmi**

A*-algoritmi eli A-tähtialgoritmillä tarkoitetaan polunlöytämistekniikkaa, jonka avulla saadaan selville lyhin reitti ruudukossa kahden pisteen välillä (kuva 12). Pisteiden välillä voi olla mahdollisia esteitä, jotka otetaan huomioon reittiä laskiessa. A-tähtialgoritmi perustuu melko yksinkertaiseen kaavaan (kaava 1). Kaavassa lasketaan tarvittava matka liikkua ruudusta toiseen, joko suoraan tai kulmittain. (18.)

$$f(n) = G(n) + H(n)$$ *KAAVA 1*

*f (n)* = matka lähtöruudusta maaliruutuun
*G(n)* = matka alkuruudusta tiettyyn ruutuun
*H(n)* = arviollinen matka tietystä ruudusta maaliruutuun



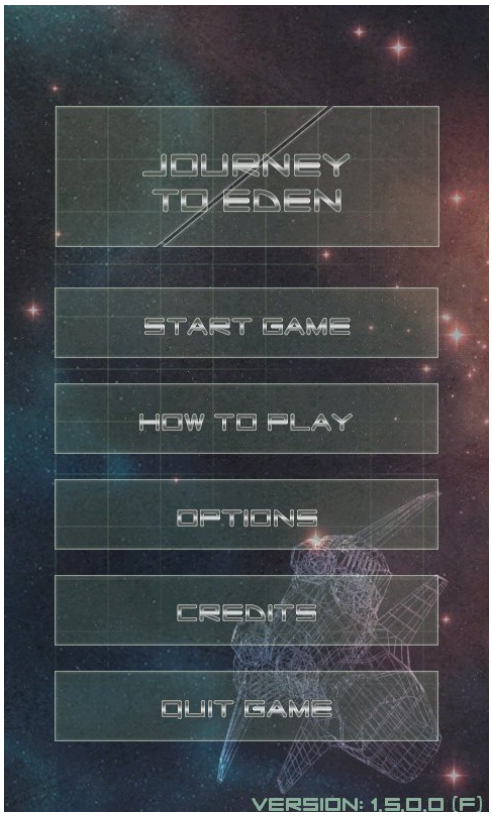

*KUVA 12. A*-algoritmin graafinen esimerkki (18)*

# 4 JOURNEY TO EDEN -PELIN RAKENNE

Työn kohteena olleen pelin nimenä oli Journey to Eden. Pelin alkuperäinen kehitystyö oli aloitettu jo ennen tämän projektin alkua. Alkuperäinen peli sisälsi päävalikon ja yhden yksinkertaisen tason loppuvastustajalla. Pelitasossa pelaajaa vastaan satoi meteoriitteja suorassa linjassa, alkaen satunnaisesta kohdasta vaakasuoralla akselilla. Tässä luvussa käydään lyhyesti läpi pelin rakennetta ja ideaa yleisellä tasolla.

## 4.1 Alkuvalikko

Peliin ensimmäisenä varsinaisena näkymänä toimii päävalikko, johon sisältyy yksinkertainen logo ja muutama painike. Ensimmäisenä on Start-painike, jonka avulla itse peli varsinaisesti alkaa. Toisena valintana on How to play -painike, jonka alta löytyy yksinkertainen näkymä, jossa pelaaja voi nähdä pelin ohjeet. Kolmantena on Options-painike, josta pelaaja pysyy muuttamaan muun muassa äänen voimakkuutta ja kalibroimaan pelattavan aluksen ohjaukseen käytettävän mobiililaitteen asentoa. Toiseksi viimeisenä on perinteinen Credits-painike, josta löytyy tietoa pelin tekijöistä, ja viimeisenä on Quit-painike. Pelin alkuperäinen päävalikko ei sisältänyt Credits-painiketta ollenkaan, ja kuvana on käytetty pelin uudemman ja hieman työn aikana muuttuneen version alkuvalikkoa. (Kuva 13.)




*KUVA 13. Pelin päävalikko*

### 4.2 Pelaaminen

Peli on tasohyppelytyylinen avaruuslentely- ja ammuskelupeli. Peliä pelataan pääasiassa ohjaamalla näytöllä leijailevaa avaruusalusta. Ohjaaminen tapahtuu mobiililaitetta kallistelemalla, mikä on myös yksi pelin pääominaisuuksista. Pelaajan tarkoituksena on väistellä vastaantulevia esteitä käyttäen samalla hyväksi kosketuksella hallittavaa laserasetta ja pelin aikana vastaantulevia samankaltaisia erilaisia lisävoimia. Peli päättyy pelaajan tuhoutuessa tai hänen päästessään pelitasot lävitse.

#### Suojakilvet

Pelaajan ohjaamalle raketille kuuluu kolme suojakilpeä jokaista pelitasoa kohti. Suojakilpien määrä palautuu aina takaisin maksimimäärään pelaajan päästessä seuraavalle pelitasolle. Pelaajan törmätessä esteisiin suojakilpien määrä vähenee ja pieni visuaalinen efekti kertoo pelaajalle tarkemmin saadusta osumasta. Peli loppuu pelaajan törmätessä vastaan tulevaan esteeseen, jos suojakilpiä ei ole enää jäljellä. (Kuva 14.)

*KUVA 14. Pelaajan ohjaama alus täysillä suojakilvillä*

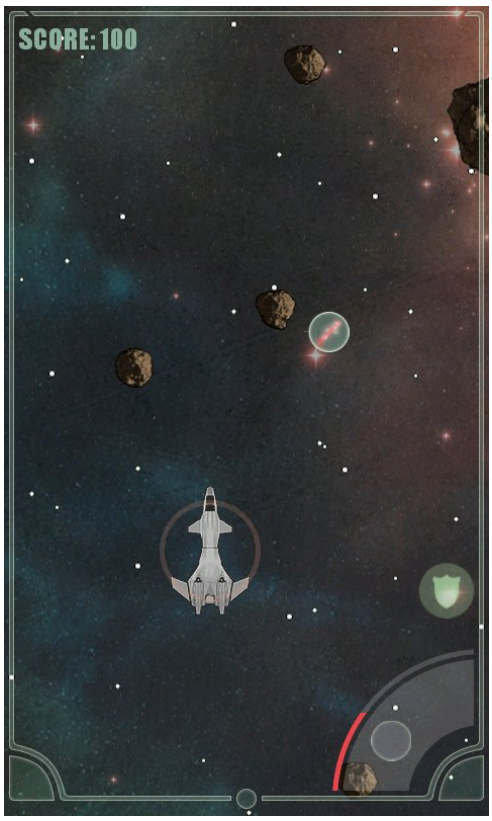
**Ampuminen**

Ampuminen pelissä tapahtuu ruudun alakulmassa sijaitsevalla pienellä, sormella painettavalla tähtäimellä. Tähtäintä painettaessa raketti alkaa ampua punaista laseria painalluskohdasta riippuvaan suuntaan, kunnes pelaaja vapauttaa sormen tai laser loppuu kesken. Lasertähtäimen vieressä pelaajalle näkyy kaarevalla punaisella viivalla käytettävän laserin määrä. Käytettävän laserin määrä vähenee koko ajan laserilla ampuessa ja latautuu koko ajan hitaasti muuna aikana. (Kuva 15.)

*KUVA 15. Tähtäin täydellä laserilla*

**Hyöty- ja haittavoimat**

Peliin kuuluu erilaisia haitta- ja hyötyominaisuuksia, joita tulee automaattisesti pelaajaa vastaan pelin edetessä. Näihin törmätessään pelaaja voi saada käyttöönsä erilaista pientä hyötyä ja peliä helpottavia ominaisuuksia tai päinvastaisesti sitä vaikeuttavia ominaisuuksia. Hyödyllisiin voimiin kuuluu esimerkiksi kuvassa 16 näkyvä laserikoni, joka palauttaa pelaajalle lisää käytettävää laseria varastoon.




*KUVA 16. Tilannekuva ensimmäisestä pelitasosta*

### 4.3 Pelitasot

Lähtökohtana käytetty peli sisälsi vain yhden pelitason, jossa näytön yläosasta päin satoi satunnaisesti pelaajaa vastaan isoja tai pieniä meteoriitteja. Pelitaso sisälsi myös loppuvastustajan, jossa pelaajan piti tuhota iso meteoriittihirviö. Loppuvastustajalla oli kolme suojakilpinä toimivaa isohkoa meteoriittia. Meteoriittihirviö liikkui hieman edestakaisin näytön yläreunassa meteoriittikilpien singahtaen välillä alaspäin, antaen pelaajalle tilaa ampua itse päävastustajaa. Loppuvastustajan toiminnallisuus muuttui hieman lopulliseen versioon, jossa kiviä oli useampia, ne olivat pienempiä ja liikkuivat sekä ylhäältä alas että alhaalta ylös.

### 4.4 Pohjarakenne

Peli pyörii XNA Frameworkin päällä käyttämällä sen tarjoamaa toiminnallisuutta. XNA Framework hoitaa pelin matalamman tason motoriikan, johon kuuluu muun muassa itse pelisilmukan pyörittäminen, syötteiden hallinta ja ruudulle piirtämi-

nen matalammalla tasolla. Tällä tavalla toteutettavaksi jää ainoastaan pelin korkeamman tason motoriikan, pelilogiikan ja pelifysiikan toteuttaminen. Pääasiassa pelin koodit kirjoitetaan XNA Frameworkin tarjoaman Update- ja Draw-metodien sisään.

Peli itsessään käyttää hyväksi omaa GameStatea eli suomalaisittain pelitilaa, jonka avulla peli tietää, osaa toteuttaa ja piirtää oikean näkymän ruudulle. GameState muutetaan menuvalintojen mukaan tai pelin päättyessä (esimerkki 2).

```
protected override void Update(GameTime gameTime)
{
   if (gameState == GameStates.menu) {
      // menu
   }
   else if (gameState == GameStates.game) {
      // play
   }
   base.Update(gameTime);
}
```

*ESIMERKKI 2. GameState yksinkertaisessa pelisilmukassa*

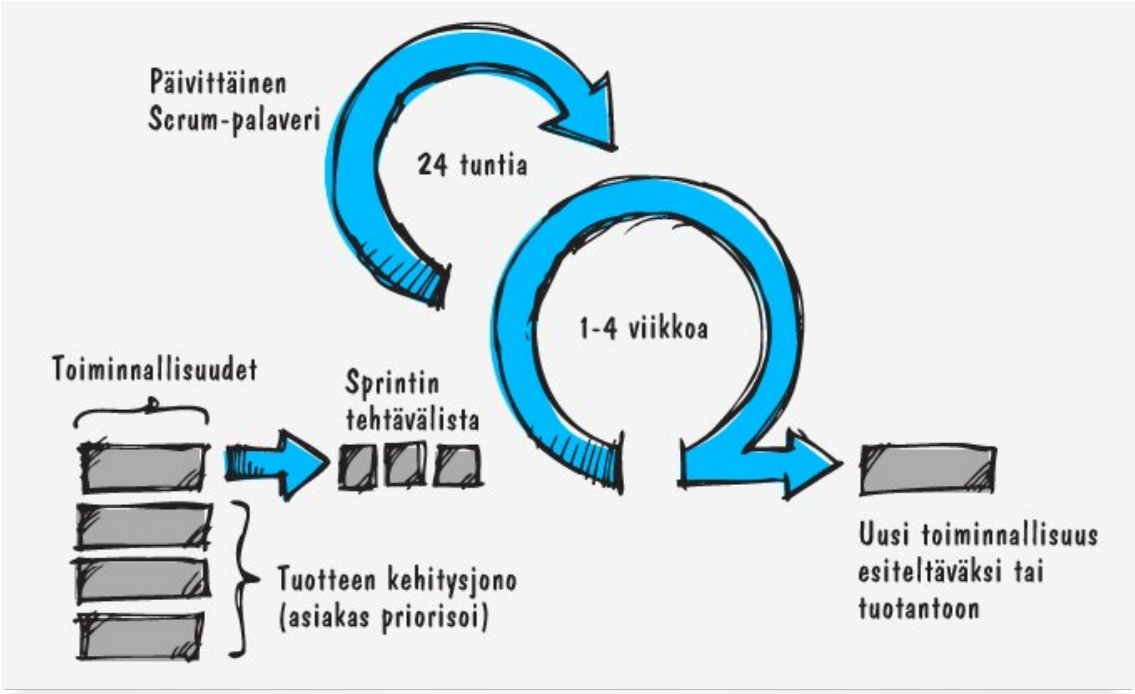
Peli on toteutettu kokonaan kaksiulotteisena, mutta toteutuksessa on käytetty hyväksi yksinkertaisia tekniikoita, joiden avulla pelaajalle annetaan visuaalinen illuusio kolmiulotteisesta sisällöstä. Grafiikka on pääsääntöisesti toteutettu kolmiulotteisina malleina, mutta pelissä ne animoidaan kaksiulotteisista kuvasarjoista. Illuusio luodaan animaatioiden ja sivusuunnista otettujen kallistuskuvien avulla, jolloin peli saadaan näyttämään hieman enemmän kolmiulotteiselta.

Pelitasot ja käyttöliittymä rakentuvat muutamasta eri kerroksesta. Näitä on alimmalla kerroksella pyörivä pelitason tausta, jonka toteutuksessa on käytetty hyväksi parallax-vieritystä. Toisella kerroksella on pelikomponentit, johon kuuluvat pelattava alus, vastaantulevat peliesteet ja muu varsinainen tasosisältö. Kolmannella kerroksella on HUD, joka hoitaa reunoilla näkyvän kehyksen, lasertähtäimen ja pistemäärän piirtämisen (kuva 15).

# 5 TYÖN TOTEUTUS

Työ toteutettiin pienessä projektiryhmässä, jossa suurin osa vastuusta projektin etenemisestä kuului minulle, osittaisen projektipäällikön vastuun mukana. Projektiryhmään kuului lisäkseni muita harjoittelijoita, jotka toimivat suunnittelijoina, graafikkona ja ohjelmoijana. Työn aikana pidettiin erillisiä projektipalavereita ja erilaisia tilannekatsauksia, joiden avulla pyrittiin saamaan projekti eteneminen yhtenäiseksi kaikkien osalta. Palavereita ja tilannekatsauksia käytettiin myös hyväksi tilaajalle raportoinnissa. Koska kaikkien projektin parissa toimineiden henkilöiden osaaminen oli hieman eri tasolla ja projektin laajuus halutun sisällön osalta melko pieni, jouduttiin työn edetessä tehtäviä hieman erottamaan toisistaan ja lopulta suuri osa toteutuksesta jäi minulle.

Työ toteutuksessa käytettiin hyväksi Scrumin ja vesiputousmallin yhdistelmiä. Jokaista pelille toteutettua päivitystä voi verrata yhteen sprint-kierrokseen Scrumissa. Päivitys taas piti sisällään vesiputousmallisen rakenteen, jossa käytiin järjestyksessä läpi, määrittely ja suunnittelu päivitykseen haluttavalle sisällölle, päivityksen toteutus ja tarvittavat testaukset. Hätätapauksissa ohjelmointivirheet priorisoitiin muiden tehtävien edelle. Työn aikana ei käytetty paljoa aikaa työaikalomakkeisiin tai vastaaviin dokumentointeihin, vaan koodin laadukas kommentointi ja palaverimuistiot toimivat työltä vaadittuna dokumentaationa. Kuvassa 17 näkyy yksinkertainen esimerkki Scrum-prosessista.




*KUVA 17. Scrum-prosessin graafinen esimerkki (19)*

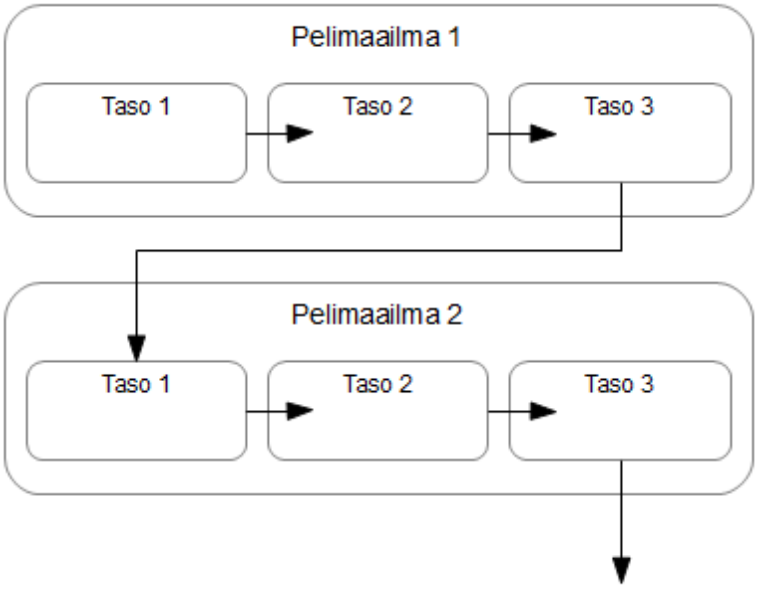
### 5.1 Suunnittelu

Työn aikana pelitasojen rakennetta muutettiin alkuperäisestä vastaamaan enemmän niiltä haluttavaa yksinkertaista eroavaisuutta. Tämän aikana pelitasojen ideaa muutettiin maailmakohtaiseksi, jossa jokainen pelimaailma noudattaa pääsääntöisesti yhtä peruskaavaa (kuva 18).

Pelimaailman ensimmäinen taso esittelee aina uuden maailman ja tuo mukaan hieman jotakin uutta edelliseen maailmaan verrattuna. Näihin kuuluvat visuaalisesti vaihtuvat vastaantulevat esteet, niiden toiminnallisuus, uudenlaiset haitta- ja hyötyvoimat sekä pelattavan maailman tausta. Nämä pienet muutokset tekevät jokaisesta maailmasta hieman erilaisen ja omalla tavallaan haastavan.

Toinen pelitaso jatkaa saman maailman teemassa, mutta tuo peliin lisää haastetta edelliseen pelitasoon verrattuna. Tähän kuuluu vihollisten monimutkaisempi toiminnallisuus ja toiselle pelitasolle ominainen loppuvastustaja, jonka pelaaja joutuu tuhoamaan tason lopussa päästäkseen pelitason lävitse. Toinen taso on periaatteessa ensimmäinen taso jaettuna kahteen pienempään erilliseen osaan, joissa tasojen pituus saadaan paremmin tasattua ja niiden väliset erot selkeämmin näkyviin.

Kolmas pelitaso taas tuo pelaajalle eräänlaisen lisätason, jossa vihollisia ja hyötyvoimia tulee vastaan huomattavasti enemmän ja nopeampaa. Viholliset sen sijaan ovat huomattavasti heikompia muihin pelitasoihin verrattuna, ja pelaajan tarkoituksena on käyttää enemmän laseria ja ampua päästäkseen kentän loppuun asti. Pelimaailman kolmannen tason vihollisten toiminnallisuus itsessään ei välttämättä eroa suuresti aiempiin tasoihin verrattuna.




*KUVA 18. Pelimaailmojen etenemisrakenne*

Näihin ratkaisuihin päätyttiin halutun tavoitteen ja pienen testailun perusteella ja omilla arvioilla siitä, kuinka kauan keskimäärin mobiililaitteen käyttäjä pelaa yhtäjaksoisesti. Näitä tapoja yhdistelemällä pelille saadaan useita eri pelitasoja helposti ja myös lisää sisältöä julkaistavaksi mahdollisimman tiheään tahtiin.

Tulevaisuuden kannalta pelille saatiin useampia pieniä ideoita varastoon, mutta yhdeksi suurimmista voisi mainita idean, jossa pelaaja mahdollisesti pystyisi myös valitsemaan haluamansa pelimaailman ja yrittää rikkoa vanhan ennätyksensä. Toisena ovat pelin aikana tehtävät päätökset, jotka vaikuttavat sen etenemiseen. Näiden tarkoituksena on kasvattaa pelin uudelleenpelattavuuden arvoa ja tehdä pelistä paremmin lyhyille mobiilipelisessioille sopiva peli.

### 5.2 Testaus

Työn aikana kehitetyn sovelluksen testaus tapahtui suurimmaksi osaksi yrityksen sisäisillä resursseilla ja muutamilla valituilla ja vapaaehtoisilla tahoilla. Sovelluksen testauksessa avusti jonkin verran myös joitakin yliopistolle lopputyötä tehneitä henkilöitä. Pelin testaaminen toteutettiin peliä pelaamalla ja käyttämällä hyväksi saatavissa olleita Windows Phone -laitteita.

Pelin testaamisen pystyi jakamaan pääsääntöisesti kolmeen eri osaan. Ensimmäisessä pyrittiin saamaan mahdollisimman monet virheet koodista pois ennen version julkaisua. Tämän testauksen aikana muu palaute ei ollut tervetullutta, vaikka palaute siitä huolimatta lipesi usein eri alueille. Toisessa pyrittiin saamaan pelitasojen sisältö ja tasojen haastavuus loogisesti eteneväksi ja mielenkiintoiseksi. Testauksen aikana tasojen haastavuus pyrittiin saamaan paremmaksi ja pelitason toiminnallinen osuus mielenkiintoisemmaksi. Kolmas osa liittyi enemmän yleiseen palautteen saamiseen, jossa avusti osaksi myös yliopiston porukka.

Palautteen saannissa koehenkilöt testasivat peliä arvioiden pelattavuutta ja antaen palautetta ja mahdollisia uusia ideoita, jotka voisivat tehdä pelistä mielenkiintoisemman. Ideoita otettiin huomioon kaikki yhtä tärkeinä, mutta suurin osa projektin aikana tulleista ideoista päättyi loppumattoman idealistan pohjalle monien muiden tavoin erilaisien prioriteettien takia niiden alkuperästä riippumatta.

### 5.3 Pelitasot

Työn aikana peliin suunniteltiin ja toteutettiin muutaman pelimaailman mittaisesti uusia pelitasoja, joista osa on ehditty julkaista työn aikana ja loput jäivät varastoon. Näitä varten oli ensiksi toteutettava mekanismi, jonka avulla eri pelitasot saadaan ladattua, ja tämä vaati pieniä muutoksia alkuperäiseen koodiin.

Ennen varsinaista tason latausta oli myös otettava huomioon, onko kyseessä kokeilu- vai pelin kokoversio, mikä tapahtui yksinkertaisella tarkistuksella (esimerkki 3). Tarkistaminen tapahtuu suoraan XNA Frameworkin tarjoamaa Guide-luokkaa käyttäen. Myös kauppapaikan avaaminen kokoversion ostamista varten tapahtui saman luokan tarjoamaa toiminnallisuutta hyväksi käyttäen.

```
if (Guide.IsTrialMode) {
   // trial version → mainosta kokoversiota
}
else {
   // full version → lataa seuraava taso
}
```

*ESIMERKKI 3. Kokeilu- ja täysversion tarkistus*

Tason vaihtaminen toteutettiin käyttämällä hyväksi paluuarvoa, jonka jokainen taso palauttaa pelisilmukassa. Tätä varten lisättiin myös uusi GameState, jonka aikana tason lataukseen liittyvät tarkistukset tehtiin. Pelin tasot tallennettiin tässä vaiheessa jokainen erillisiin luokkiin. Tätä varten toteutettiin yksinkertainen mekanismi, jonka avulla oikea luokka saatiin ladattua tasonumeron perusteella (esimerkki 4). Esimerkissä käytetylle funktiolle täytyy vain antaa ladattavan pelitaso-luokan numero, ja tässä tapauksessa jokainen pelitaso nimettiin samanlaiseksi muotoon Level1.cs, Level2.cs.

```
// Tason lataava funktiosta
private Level InitGameLevel(int currentLevel) {
   Type type = Type.GetType("namespace.Level" + currentLevel.ToString());

   // Muodostimen parametreiksi asetettavien luokkien tyypit
   ConstructorInfo ci = type.GetConstructor(new Type[] {
      typeof(String), typeof(int) });

   // Kutsutaan muodostinta ja annetaan parametrien arvot
   Level level = (Level)ci.Invoke(new Object[] { text, number });

   return level;
}
```

*ESIMERKKI 4. Pelitason luokan lataaminen kahdella kuvitteellisella parametrilla*

Koska pelitasot käyttivät hyväkseen paljon satunnaislogiikkaa vihollisten luomisessa, täytyi tasojen balansoinnissa ottaa käyttöön erilaisia lippuarvoja näitä rajoittamaan. Vihollisia on pyritty rajoittamaan satunnaisgeneraattorin arvojen lisäksi myös maksimimäärällä, joka ruudulle mahtuu kerrallaan, ja minimimäärällä, jolla pyritään estämään liian pitkä tyhjäkäynti pelin aikana.

Valitettavasti työtä tehdessä käytössä oli hyvin rajoitettu määrä grafiikkaa pelitasoja varten. Tästä syystä työn aikana toteutettujen pelitasojen sisältö toteutettiin hyvin pitkälle käyttämällä väliaikaista grafiikkaa ja visuaalisesti samannäköiset sisällöt toistuivat myös toisissa maailmoissa niiden toiminnallisuutta lukuun ottamatta.

**Maailma 1**

Ensimmäinen pelimaailma tehtiin käyttäen hyväksi alkuperäistä pelitasoa. Ensimmäisen pelitason vihollistoiminnallisuutena käytettiin alkuperäisen pelin tasoa, josta loppuvastustaja siirrettiin seuraavalle tasolle. Toisen sisällöstä tehtiin hieman haastavampi niin, että vastaantulevien meteoriittien kurvit muuttuivat hieman hankalammiksi. Alkuperäinen loppuvastustaja liitettiin myös tähän pelitasoon. Viimeisessä kentässä käytettiin muuten samankaltaista ideaa, mutta meteoriitit olivat huomattavasti heikompia ja niiden määrä huomattavasti suurempi. Ensimmäisen pelimaailman sisällöstä löytyy esimerkki aiemmin dokumentissa mainitussa kuvassa (kuva 16).

**Maailma 2**

Työn aikana pelille toteutettiin myös toinen pelimaailma, mutta tämän toteutuksesta vastasivat suurimmaksi osaksi muut harjoittelijat. Toisen pelimaailman toteutukseen osallistuin lähinnä vain korjaamalla jälkeenpäin suurimmat ja näkyvimmät ohjelmointivirheet pois koodista, rajoittamalla joitakin suunnittelun aikana tulleita hieman ylimeneviä ideoita ja avustamalla testauksessa. Toisen pelimaailman tunnusmerkkeinä on pelin aikana nopeutuva avaruustausta, sivulta päin nopeasti tulevat raketit ja kaksi tölkkiä loppuvastustajina.

**Maailma 3**

Kolmas pelimaailma jatkoi sisällöllisesti pitkälti ensimmäisten pelimaailmojen kaavalla. Maailman suurimpina eroina olivat vastaantulevien esteiden uudenlaiset liikeradat, niiden aavistuksen verran suurempi määrä ja täysin uutena tulleet yksinkertaiset ja lyhyet vihollisaallot. Yksinkertaisia vihollisaaltoja lisäämällä on pyritty hitaasti siirtymään alkuperäisistä täysin satunnaisesti vastaantulevista vihollisista enemmän ennalta määriteltyyn ja kovakoodattuun sisältöön pelitasoissa.

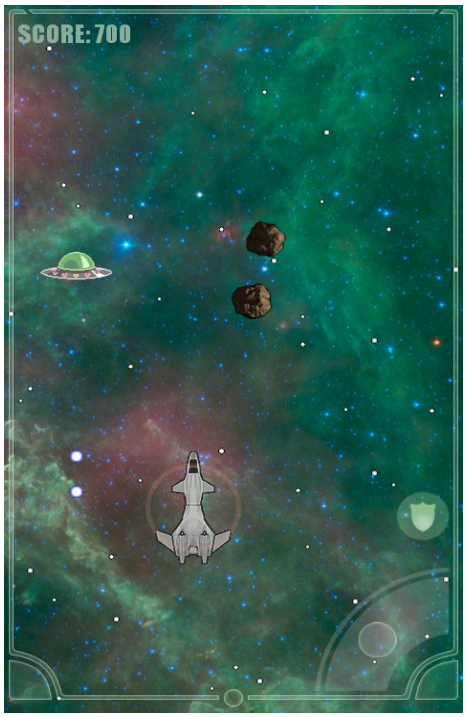
Maailman loppuvastustajana toimii etana, joka ampuu aakkosia eri suuntiin pelaajaa kohti. Loppuvastustaja on toteutettu suoraan ennaltaan olemassa olleen peli-idean mukaisesti ja sen toiminnallisuus on melko yksinkertainen. Etana ravaa edestakaisin ruudun yläosassa, välillä pysähtyen ja suuntaa vaihdellen.

Varsinainen haaste itsessään tulee etanan ampumista kirjaimista, joita pelaaja joutuu väistelemään. (Kuva 19.)


*KUVA 19. Kolmannen pelimaailman loppuvastustaja*

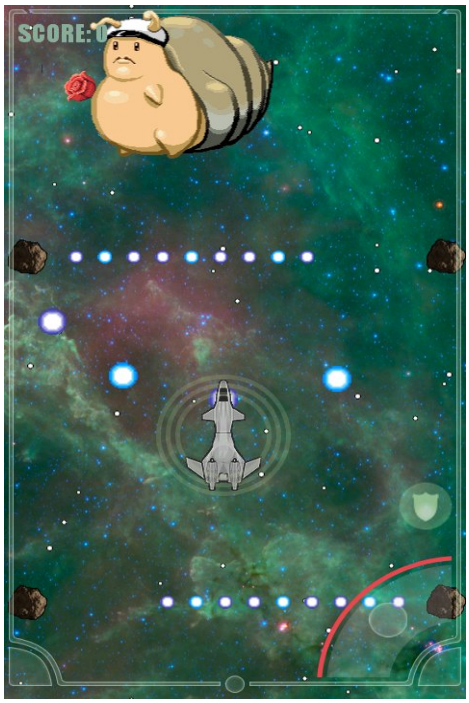
**Maailma 4**

Neljännessä pelimaailmassa mukaan tulivat osittain erilaisten liikeratojen lisäksi uudentyyppiset viholliset, jotka pystyivät myös uusien liikeratojen lisäksi ampumaan pelaajaa kohti. Ensimmäisen tason sisällössä pelaaja pääsee tutustumaan aavistuksen verran ufoihin, jotka ampuvat pieniä plasmapalloja pelialueen ohi liikkuessaan. Toisessa pelitasossa mukaan tulee hieman enemmän ampuvia vihollisia ja myös pieniä nopeita ufoja, jotka liikkuvat pelialueelta läpi ampuen nopeasti kerran tai pari. (Kuva 20.)




*KUVA 20. Ufo ampumassa avutonta pelaajaa*

Maailman loppuvastustajan toteutuksessa on käytetty paranneltua ja haastavampaa versiota aiemman pelimaailman loppuvastustajasta. Grafiikan puutteen takia loppuvastustajana vielä tässä vaiheessa jouduttiin käyttämään samaa sisältöä. Loppuvastustajaan lisättiin pientä lisätoiminnallisuutta, jossa se ampuu plasmapalloja pelaajaa kohti, ja tämän lisäksi sivuilla liikkuvat myös erilliset laserit, jotka ampuvat pelaajaa myös sivusuunnista. Sivuilla olevat laserit liikkuvat edestakaisin ylös ja alas pakottaen pelaajaa ohjailemaan rakettia ja väistelemään useaan eri suuntaan. Loppuvastustajan energiamäärän käydessä vähäiseksi alkavat sivuilla olevat laserit ampua jatkuvasti samalla liikkuen. (Kuva 21.)




*KUVA 21. Neljännen maailman loppuvastustaja väliaikaisilla grafiikoilla*

**Maailma 5**

Työn aikana toteutettiin myös viidettä pelimaailmaa, mutta se toteutettiin lähinnä erilaisten ominaisuuksien, kuten dialogien ja tekoälyn käytön esittelyä varten. Maailman vihollissisältö toteutettiin käyttämällä pelkästään valmiiksi määriteltyjä vihollisaaltoja ja käyttämällä erilaisia variaatioita tasojen välillä lopputuloksen näyttämiseksi. Maailman loppuvastustajalle toteutettiin alustavasti omaa lisäanimaatiota, jota voidaan mahdollisesti käyttää hyväksi myöhemmin.

### 5.4 Lisäominaisuudet

Pelin kehitystyön edetessä toteutettiin joitakin lisäominaisuuksia sekä pelillisen mielenkiinnon lisäämiseksi että joidenkin peliominaisuuksien toteuttamiseksi. Näkyvimmät ja suurimmat näistä käydään läpi tässä luvussa. Osa ominaisuuksista ehdittiin lisätä mukaan jo julkaistuihin versioihin ja osa jäi odottamaan seuraavaa päivitystä tai viimeistelyä. Toteutettuihin ominaisuuksiin kuului muun muassa seuraavia:

- tarvittava toiminnallisuus pelin trial- ja kokoversiota varten
- pelitason vaihtamista hallitseva toiminnallisuus

- Credits-näkymän paranneltu uusine ominaisuuksineen
- ohjauksen herkkyysasetus menupainikkeineen
- piste-ennätys ja sille kuuluva toiminnallisuus
- hahmojen väliset keskusteluruudut eli dialogit
- tason alkamis- ja päättymisvaiheessa näkyvä teksti
- tasokohtainen tilastonäkymä jokaisen pelitason päättyessä
- pelin versionumeron tulostus päävalikossa
- loppuvastustajan energiamäärän näkyminen graafisesti
- erilaisia animointivalintoja peliobjekteille käytettäväksi
- yksinkertainen tasogeneraattori toiminnallisuus
- testausasetuksilla toimiva suora tason valinta ja mahdollinen hyppäys suoraan loppuvastustajalle.
- aluksen ohjaaminen näppäimistön avulla emulaattorikäyttöön
- erilaisia tiedostostaluku- ja tiedostoonkirjoitustoiminnallisuuksia.

**Piste-ennätys**

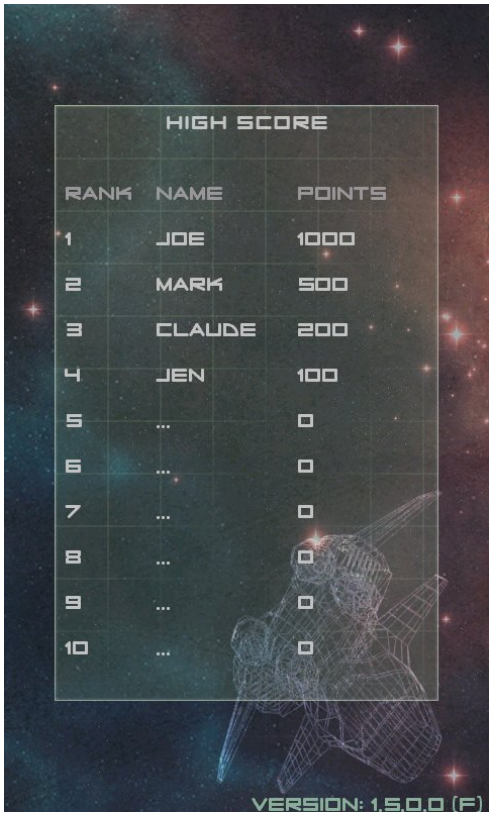
Työn aikana peliin toteutettiin alustavasti valmiiksi piste-ennätyksiä varten toiminnallisuus, joka tosin oli vielä lukittuna julkisesta versiosta. Pelin aikana pelaajalle kertyy pisteitä hänen tuhotessaan laserilla vastaantulevia esteitä. Pelin päättyessä ohjelma tarkistaa ensiksi, onko pelaaja saanut uuden ennätyksen. Uuden ennätyksen syntyessä pelaajalle aukeaa näkymä, johon hän voi kirjoittaa oman nimensä. Tämän jälkeen uusi ennätys tallentuu automaattisesti tiedostoon mobiililaitteelle käyttäen Isolated Storagea (esimerkki 5). Piste-ennätyksiä varten toteutettiin myös oma näkymä, jossa käyttäjä pääsee selaamaan senhetkisiä parhaita ennätyksiä (kuva 22).

```
try {
   // Isolated Storage alustukset
   IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
   StreamWriter sw = new StreamWriter(new IsolatedStorageFileStream(fileName,
   FileMode.Create, isf));

   // Kirjoitetaan string-taulukon sisältö tiedostoon rivi riviltä
```

```
    for (int i = 0; i < data.Count; i++) {
        sw.WriteLine(data[i]);
    }
    // Suljetaan yhteys
    sw.Close();
}
```

*ESIMERKKI 5. Piste-ennätysten tallentaminen tiedostoon*




*KUVA 22. Ennätyslistan näkymä*

### Credits

Alkuperäinen Credits-valikko sisälsi kaikki tekstit ja nimet kovakoodattuina rivi riviltä näkymään ilman rullautuvaa ominaisuutta. Työn aikana credits-valikon tekstit laitettiin latautumaan suoraan omasta sovelluksen mukana tulevasta tekstitiedostosta ja tulostus muutettiin tapahtumaan yhdellä silmukalla (esimerkki 6). Tekstien tulostamisessa otetaan huomioon yhdellä lippuarvolla varustettu fontin tyyli, joka annettiin rivin ensimmäisenä kirjaimena.

```
// Asetetaan luettava tiedosto
Stream streamPath = TitleContainer.OpenStream("Content/Data/credits.txt");
string line = string.Empty;

using (StreamReader sr = new StreamReader(streamPath)){
    // Luetaan tiedostoa kunnes vastaan tulee ;-merkki
```

```
   while ((line = sr.ReadLine()) != ";") {
      // Lisätään rivi taulukkoon
      creditsList.Add(line);
   }
}
```

*ESIMERKKI 6. Tiedostosta lukeminen*

Credits-näkymälle toteutettiin myös automaattisesti vierivät tekstit, jotka liikkuvat ylös ja takaisin alas näyttäen listan nimiä pelin aiemmista ja nykyisistä tekijöistä. Nimet tulostuvat niille varatulle omalle tulostus alueelle ja poistuvat näkyvistä nimien mennessä vihreän ruutualueen ulkopuolelle (esimerkki 7). Näkymään yhdistettiin myös ominaisuus, jolla tekstien sijaintia voidaan liikutella suoraan kosketuksen avulla (esimerkki 8). (Kuva 23.)

```
// Alustetaan asetukset leikkausaluetta varten
spriteBatch.GraphicsDevice.RasterizerState = new RasterizerState()
      { ScissorTestEnable = true };

// Asetetaan suorakulmio jonka ulkopuolelle ei piirretä
spriteBatch.GraphicsDevice.ScissorRectangle = new Rectangle( ... );

// Aloittaa sprite batch operaation leikkuualueen kanssa
spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend, null, null,
      spriteBatch.GraphicsDevice.RasterizerState);
```

*ESIMERKKI 7. Piirrettävän alueen rajaaminen*

```
foreach (TouchLocation tl in touchCollection)
{
   // Creditsien liikuttaminen sormella
   if (tl.State == TouchLocationState.Moved) {
      // Lasketaan liikemäärä Y-akselissa ja muutetaan tekstin sijaintia
      tapCurrentPosition = tl.Position;
      creditsOffsetY += (tapCurrentPosition.Y - tapPosition.Y) / 2;
      tapPosition = tapCurrentPosition;
      ...
```

*ESIMERKKI 8. Tekstin sijainnin siirtäminen kosketuksen avulla*


*KUVA 23. Credits-näkymä*

**Puhedialogit**

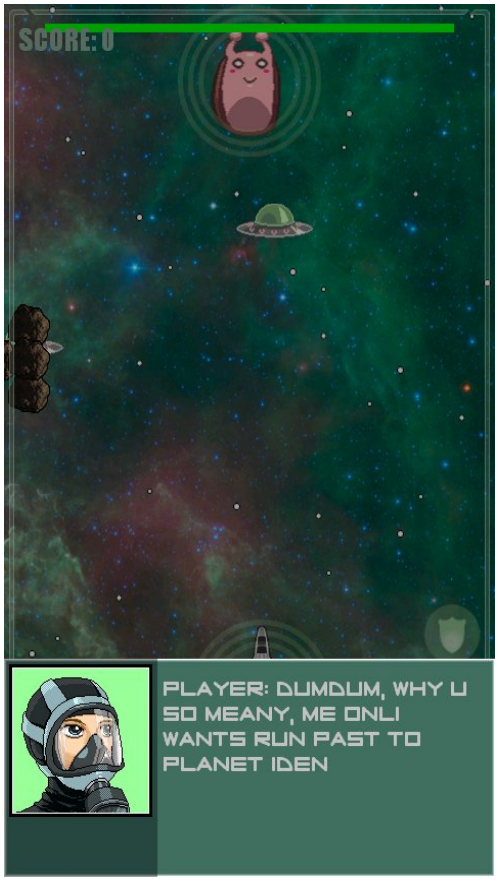
Pelille toteutettiin varastoon myös yksinkertainen puhedialogitoiminnallisuus. Puhedialogin suunnittelussa otettiin huomioon se, että ne olisi mahdollisimman helppo käyttää ja koodirivien määrä pysyisi vähäisenä käytettäessä niitä uudelleen pelitasoissa. Toteutuksessa päätyttiin script-tyyliseen ratkaisuun, jossa puhedialogit ikoneineen luetaan suoraan XML-tiedostosta (esimerkki 9). Tämä tapahtuu dialogisarjan UID:n avulla ja sieltä niitä hallitseva luokka tulostaa ne näytölle puhekupla kerrallaan erottaen eri osat yksinkertaisella parserilla.

```
<!-- 1. Esimerkki keskustelu -->
<Dialog uid="0">
   <dlg>0|Player: Let's chat?</dlg>
   <dlg>1|Monster: Grrrr... Ugha Bugha Jug-Jug.</dlg>
</Dialog>

<!-- 2. Mahdollinen parannusidea -->
<Dialog uid="1">
   <dlg icon="0" name="Player">Let's chat?</dlg>
   <dlg icon="1" name="Monster">Grrrr... Ugha Bugha Jug-Jug.</dlg>
</Dialog>
```

*ESIMERKKI 9. Keskustelu XML-muodossa*

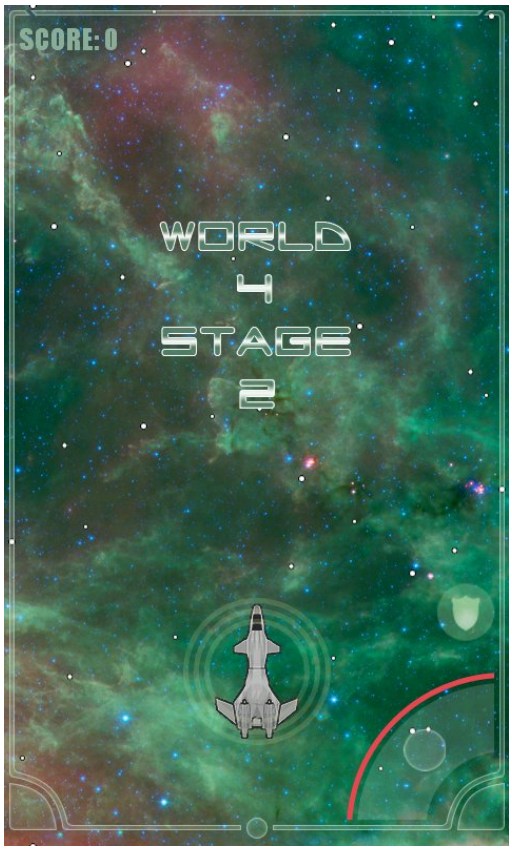
Puhedialogiin kuuluu puhujan kuva ja alue, johon teksti tulostuu ilmestyen kirjain kirjaimelta. Dialogien aikana muu pelialue hämärtyy ja muu pelin eteneminen pysähtyy väliaikaisesti. Varsinaista tarinaa ja hahmojen välisiä keskusteluja ei työn aikana juuri suunniteltu sen tarkemmin ja esimerkeissä on käytetty vain yksinkertaisia lyhyitä väliaikaisia tekstejä. (Kuva 24.)




*KUVA 24. Dialogi pelin varhaisversiollisella grafiikalla*

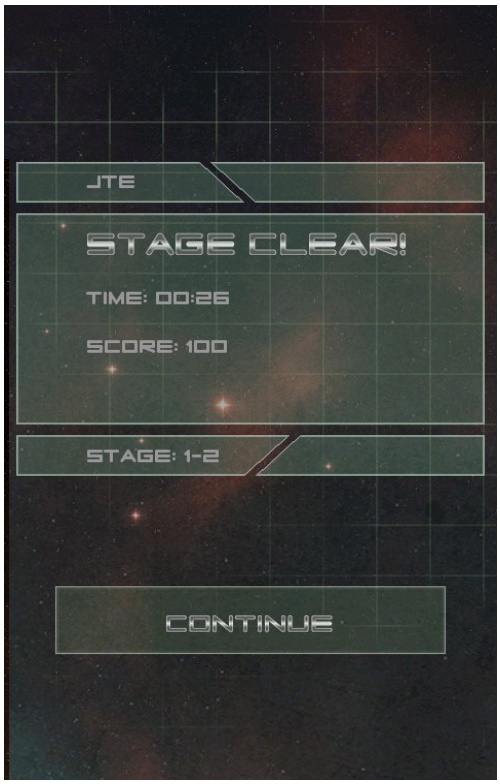
**Tasonäkymät**

Tasoja varten toteutettiin erillinen toiminnallisuus, jolla saadaan tulostettua lisäefekteillä muotoiltu teksti näytölle tason alkaessa ja päättyessä. Tätä varten toteutettiin yksinkertainen bitmap font -tyylinen toiminnallisuus, jolla oikeat numerot saatiin tulostettua ulos kuvasta. Teksti toistuu jokaisen tason alussa kertoen sen hetkisen maailman ja tason numeron. Tason päättyessä pelaajalle näkyy erillinen teksti, joka ilmoittaa tason päättymisestä. (Kuva 25.)




*KUVA 25. Pelitason alkuvaihe*

Pelille toteutettiin tason päättyessä myös saman idean mukainen tasokohtainen statistiikkanäkymä. Tämän näkymä ilmestyy pelaajalle hänen päästessään senhetkisen pelitason lävitse. Näkymässä ilmoitetaan yksinkertaisesti kerääntyneet pisteet ja kulunut peliaika. Toteutuksessa näkymään piilotettiin myös lisästatistiikkaa, jota on tarkoitus parannella mahdollisesti käyttää tulevaisuudessa jatkoideana. (Kuva 26.)




*KUVA 26. Tason päättymisen jälkeinen statistiikkanäkymä*

### 5.5 Rakennemuutokset

Työn aikana joidenkin ominaisuuksien toteuttamiseksi jouduttiin tekemään pieniä muutoksia, lisäyksiä ja rakenteellisia korjauksia. Näihin kuului muun muassa pelitason alkuperäinen rakenne, joka muutettiin moneksi pelitasoksi, ja vihollisten käyttämä pohjarakenne. Vihollisille toteutettiin kokonaan oma pohjaluokka, jonka erityyppiset viholliset perivät. Tätä varten toteutettiin myös kokonaan oma animaatioluokka, jota uudet viholliset käyttivät hyväksi.

Alkuperäisen pelitason koodin sekavuuden ja rajoittuvuuden takia pelitasoille tehtiin hieman puhtaampi pohja uudella ja selkeämmällä rakenteella. Uudet pelitasot toteutettiin tämän pohjan päälle. Tasojen latautumista varten peliin oli myös lisättävä oma toiminnallisuus, jonka avulla oikeat tasot saatiin ladattua.

Pelimaailmojen loppuvastustajia varten toteutettiin myös kokonaan omalle uudelle puhtaana tehdylle pohjalle. Loppuvastustajan tekoälyssä käytettiin pelaajasta riippumatonta tilapohjaista ratkaisua, jossa vastustaja pyörii täysin omassa tekoälysilmukassaan (esimerkki 10). Vastustajan tilojen hallintaa pyöritti yksin-

kertainen tekoälyagentti, jonka lisäksi toimintaan lisättiin muutama rajoitus estämään liian pitkät saman tilan toistomahdollisuudet. Vastustajan jokaiseen tilaan yhdistettiin oma animaatio ja toiminnallisuus (esimerkki 11).

```
// Tehdään jotakin nykyisen tilan mukaan
switch (state)
{
   case State.STOP:
      Stop(gameTime);
      break;
   case State.MOVE:
      Move(gameTime);
      break;
   case State.ATTACK:
      Attack(gameTime);
      break;
}
```

*ESIMERKKI 10. Tila-pohjainen toiminnallisuus*

```
private void Stop(GameTime gameTime)
{
   sleepTimer += gameTime.ElapsedGameTime.Milliseconds;

   // Pakotetaan toinen tila maximi idlausajan täyttyessä
   if (sleepTimer > stopTime) {
      sleepTimer -= stopTime;
      state = State.MOVE;
   }

   // Päivitetään animaatio
   if (isHit)
      UpdateDamageAnimation(gameTime);
   else
      animation.AnimateRange(new Point(0, 2), new Point(2, 2));
}
```

*ESIMERKKI 11. Yksinkertainen malli stop-tilalle*

### 5.6 Tekoäly

Alkuperäiset peliesteet pystyivät liikkumaan vain suoraan ylhäältä alas. Tätä varten työn aikana vihollisille toteutettiin omaa tekoälytoiminnallisuutta, jonka avulla ne saatiin käyttämään erilaisia liikeratoja ja lopulta myös ampumaan.

Työn tavoitteena oli saada nopeita päivityksiä, joten tekoälyn pohjarakenteen valinnassa päädyttiin käyttämään helpohkoa waypoint-ratkaisua. Kyseisellä mekanismilla peliesteet laitettiin seuraamaan ennalta määriteltyjä pelaajalle näkymättömiä pisteitä, joiden kautta ne pystyvät kulkemaan pelialueen lävitse.

### Waypoint-toiminnallisuus

Waypointeja seuratessaan vihollinen kulkee ne läpi tietyssä järjestyksessä käyttäen ominaista kääntymisnopeutta, joka myös hallitsee kurvien terävyyttä (esimerkit 12 ja 13). Kääntymisnopeuden avulla yksinkertaisista muutaman pisteen mittaisilla waypoint-sarjoilla saatiin aikaiseksi useanlaisia toisistaan eroavia liikeratoja. Waypoint-toiminnallisuutta varten lisättiin myös useanlaisia eri tarkistuksia, joiden avulla voidaan välttää haluamattomat käyttäytymiset.

```
// Tarkistetaan onko tekoälyinen kappale saavuttanut nykyisen waypointin
if (WaypointReached()) {
   // Siirrytään seuraavaan waypointiin
   waypointList.Dequeue();
}
else {
   // Päivitetään nykyistä liikkumissuuntaa kohti seuraavaa waypointia
   // ja liikutaan sen hetkiseen suuntaan
   MoveToDirection(elapsedTime);
}
```

*ESIMERKKI 12. Waypointien seuraaminen*

```
public void MoveToDirection(float elapsedTime)
{
   // Päivitetään nykyinen liikkumissuunta kohti nykyistä waypointia
   base.direction = SetDirection(elapsedTime);

   // Liikutaan nykyistä suuntaa kohti
   base.position = base.position + (base.direction * base.moveSpeed);
}
```

*ESIMERKKI 13. Liikkuminen waypointia kohti*

Toteutuksessa waypointit luetaan suoraan XML-tiedostosta dialogien tapaisella mekanismilla (esimerkki 14). Dynaamisuuden luomiseksi niille voidaan antaa suoraan yhtenä monesta, offset-arvoja, joilla erilaiset waypointeista rakentuvat liikeradat voidaan toteuttaa eri kohdilla pelialuetta. Yksinkertaisena esimerkkinä tästä toiminnallisuudesta on pelin ensimmäisen maailman toinen pelitaso, joka löytyy myös pelin virallisesta trial-versiosta.

```
public List<Vector2> getWaypointsById(int index)
{
   List<Vector2> waypointList = new List<Vector2>();
   String[] point;

   // Haetaan waypoint-elementti halutun UID:n avulla XML-dokumentista
   IEnumerable<XElement> elems = (from element in doc.Root.Elements("Waypoint")
      where (string)element.Attribute("uid") == index.ToString()
```

```
        select element);

    // Erotetaan saadun elementin alta kaikki waypointit Vector2-talukkoon
    foreach (XElement e in elems.Descendants("point"))
    {
        point = e.Value.Split(',');
        waypointList.Add(new Vector2(Convert.ToInt16(point[0]),
            Convert.ToInt16(point[1])));
    }

    return waypointList;
}
```

*ESIMERKKI 14. Waypointien hakeminen XDocumentista*

### Ampuminen

Tekoälyä varten toteutettiin myös yksinkertainen toiminnallisuus, jolla vastustajat saatiin ampumaan. Tämä tapahtui pääsääntöisesti ampumisnopeuden ja määrän säätelyn avulla. Ampumistoiminnallisuus itsessään kaipaa vielä parantelua ja sen toteutus oli vielä melko varhaisessa versiossa, tehtynä tarjolla olevien rakenteiden päälle. Ammusten luomiseen käytettiin yksinkertaista ratkaisua, jossa ammukselle voidaan antaa käytettävä tekstuuri, suunta ja nopeus (esimerkki 15).

```
Bullet bullet = new Bullet(texture, position, direction, speed);
bullets.Add(bullet);
```

*ESIMERKKI 15. Ammuksen luominen*

### Tavoitepohjaisuus

Vihollisille lisättiin työn aikana myös yksinkertainen tavoitepohjainen toiminnallisuus, jossa ne saadaan tekemään tietty sarja käskyjä. Tämä yhdistettiin waypointeihin, jonka avulla vihollinen saatiin käyttäytymään erillä tavalla. Esimerkkinä tähän voisi olla kasa tavoitteita, jotka pinotaan kasaksi. Ensimmäisenä voisi olla liikkumiskäsky, jonka saadessaan vihollinen alkaa liikkua seuraavaa waypointia kohti. Tavoitteen saavuttaessaan vihollinen tarkistaa mahdollisia lisätehtäviä, joita voi olla esimerkiksi liikkumisnopeuden muuttuminen tai pysähtyminen väliaikaisesti. (Kuva 18.)

# 6 YHTEENVETO

Työn tarkoituksena oli jatkaa mobiilipelin kehitystyötä eteenpäin ja saada pelille lisää pelillistä sisältöä loppukäyttäjille ja julkaisuja varten. Tavoitteena työn aikana oli päästä nopeaan julkaisutahtiin, jossa pelille saataisiin ulos uusia päivityksiä tasaisin väliajoin. Dokumentaation osalta työn vaatimuksiin kuului lähinnä koodin laadukas kommentointi, jonka avulla tulevaisuudessa mahdollisesti projektin parissa jatkavat pystyisivät perehtymään projektiin nopeasti ja vaivattomasti.

Työn aikana peli-ideaa kehitettiin hieman eteenpäin ja pelille suunniteltiin uutta pelillistä sisältöä useamman pelitason mittaisesti. Tämän lisäksi peliin toteutettiin useita erikokoisia lisäominaisuuksia ja pientä toiminnallisuutta tekemään loppukäyttäjän pelikokemuksesta mielekkäämpää.

Projektin aikana korjattiin myös erilaisia ohjelmointivirheitä, joita alkuperäisessä versiossa oli mukana, sekä omia ja muiden projektin parissa työskennelleiden jäljiltä jääneitä ohjelmointivirheitä. Varteenotettavimpiin näistä voisi mainita alkuperäisestä versiosta mukana tulleet ohjelmointivirheet, jotka aiheuttivat laserin menemisen usean vihollisen läpi, ja törmäystarkistuksien epäsäännöllisyyden.

Varsinaisesti nopeita päivityksiä ei projektin aikana saatu ulos aivan alkuperäisen idean mukaisesti, mutta lopputulokseen oltiin silti tyytyväisiä. Tähän asiaan vaikutti useampi pieni seikka, joita ei ollut mietitty täysin loppuun asti. Pelin kehitystyötä jatkettiin silti eteenpäin näitä asioita päätettäessä. Loppujen lopuksi pelille saatiin valmiiksi varastoon pieniä päivityksiä varten sopivaa sisältöä useamman päivityksen mittaisesti, ja tämä tulee helpottamaan mahdollisien nopeiden päivitysten toteuttamista tulevaisuudessa.

Projektin aikana vastaan ei tullut mitään pysäyttämättömiä ongelmia, mutta muun muassa käytettävän grafiikan puute rajoitti tiettyjen toiminnallisuuksien toteutusta julkaisukelpoisiksi ja käyttämään paljon väliaikaista grafiikkaa. Tämä myös teki päivityksien visuaalisesta sisällöstä hieman tylsähköä ja osa jäikin varastoon odottamaan grafiikkaansa. Koodillisesti jotkin toteutetut rakenteet ja käytetyt ratkaisut toivat hieman lisätyötä ja vaikeuksia joidenkin asioiden toteut-

tamiseen, mutta mitään pysäyttämätöntä estettä ei vastaan tullut. Tämä johtui projektin luonteesta, jossa peliä on toteutettu vähäisellä suunnittelulla ja nopeilla päivityksillä, lisäämällä uusia palikkoja aiemman kokonaisuuden päälle. Vähäisen suunnittelun luomat ongelmat tulevat vaikuttamaan projektiin suuresti luultavammin myös tulevaisuudessa.

Kaiken kaikkiaan projektin aikana tuli opittua paljon erilaista pientä asiaa peliohjelmoinnista ja pelien tekoälystä. Tämän lisäksi projektin aikana pääsi perehtymään hieman tarkemmin XNA-sovellusalustan käyttämiseen ja sen tarjoamaan toiminnallisuuteen sekä sovelluskehitykseen Windows Phone -alustalle yleisesti ja siihen vaikuttaviin tekijöihin. Projekti antoi myös aavistuksen verran lisäkokemusta pienehköstä projektipäällikkyydestä sen parissa työskennellessä.

# LÄHTEET


1. Windows Phone. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Windows_Phone. Hakupäivä 3.5.2012.

2. Windows Phone. 2012. Saatavissa: http://en.wikipedia.org/wiki/Windows_Phone. Hakupäivä 2.5.2012.

3. Windows Phone. 2012. Saatavissa: http://i.microsoft.com/global/windowsphone/fi-fi/PublishingImages/lumia_800_710_fi.png. Hakupäivä 3.5.2012.

4. Windows Phone version history. 2012. Saatavissa: http://en.wikipedia.org/wiki/Windows_Phone_version_history. Hakupäivä 3.5.2012.

5. Application Platform Overview for Windows Phone. 2011. Saatavissa: http://msdn.microsoft.com/en-us/library/ff402531%28v=vs.92%29.aspx. Hakupäivä 6.5.2012.

6. Execution Model Overview for Windows Phone. 2012. Saatavissa: http://msdn.microsoft.com/en-us/library/ff817008%28v=vs.92%29. Hakupäivä 13.5.2012.

7. Multitasking for Windows Phone. 2012. Saatavissa: http://msdn.microsoft.com/en-us/library/hh202866%28v=VS.92%29.aspx. Hakupäivä 13.5.2012.

8. Windows Phone Marketplace. 2012. Saatavissa: http://en.wikipedia.org/wiki/Windows_Phone_Marketplace. Hakupäivä 2.5.2012.

9. Membership. 2012. Saatavissa: http://create.msdn.com/en-us/home/membership. Hakupäivä 6.5.2012.

10. Payouts. 2012. Saatavissa: http://msdn.microsoft.com/en-us/library/hh202925%28v=vs.92%29.aspx. Hakupäivä 6.5.2012.

11. Pricing. 2012. Saatavissa: http://msdn.microsoft.com/en-us/library/hh202929%28v=vs.92%29.aspx. Hakupäivä 6.5.2012.

12. Game engine. 2012. Saatavissa: http://en.wikipedia.org/wiki/Game_engine. Hakupäivä 6.5.2012.

13. Game Programming. 2012. Saatavissa: http://en.wikipedia.org/wiki/Game_programming. Hakupäivä 6.5.2012.

14. Initializing a Game. 2012. Saatavissa: http://msdn.microsoft.com/en-us/library/bb203873.aspx. Hakupäivä 6.5.2012.

15. Tekoäly. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Teko%C3%A4ly. Hakupäivä 4.5.2012.

16. Intelligent Agent. 2012. Saatavissa: http://en.wikipedia.org/wiki/Intelligent_agent. Hakupäivä 5.5.2012.

17. Fixing Pathfinding Once and For All. Paul Tozour 2008. Saatavissa: http://www.ai-blog.net/archives/000152.html. Hakupäivä 5.5.2012.

18. A*-algoritmi. 2012. Saatavissa: http://fi.wikipedia.org/wiki/A*-algoritmi. Hakupäivä 5.5.2012.

19. Scrum. 2012. Saatavissa: http://reaktor.fi/assets/images/scrum_process.png. Hakupäivä 23.5.2012.