Pelin toteutus Unity-pelimoottorilla

Case: Battles of Ancient Israel

Ville-Veikko Haapalehto

Opinnäytetyö
Maaliskuu 2014
Tietojenkäsittelyn ko.
Ohjelmistotuotanto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn ko.
Ohjelmistotuotanto

HAAPALEHTO VILLE-VEIKKO:
Pelin toteutus Unity-pelimoottorilla
Case: Battles of Ancient Israel

Opinnäytetyö 67 sivua, joista liitteitä 19 sivua
Maaliskuu 2014

Tietokone-, mobiili- ja konsolipelimarkkinat niin Suomessa kuin muuallakin maailmalla kasvavat koko ajan. Pelejä on monille eri alustoille: mobiileille, tietokoneille ja konsoleille. Käyttöjärjestelmiäkin on monia erilaisia jokaiselle alustalle, esimerkiksi on Windows, Mac ja Linux. Pelin kehittäminen natiivina jokaiselle alustalle on kallista, joten yleensä tarvitaan pelimoottori tai jokin alustariippumaton tekniikka. Pelimoottorit kääntävät yleensä ohjelman monelle alustalle, kuten Windowsille, Linuxille ja Macille.

Opinnäytetyön tavoitteena oli kehittää pelien kautta tapahtuvaa kristillistä evankeliointia. Opinnäytetyön tarkoitus oli toteuttaa peli Israelin historiasta Raamatun mukaan. Suunnittelun toteutti toinen TAMKin opiskelija.

Opinnäytetyössä päädyttiin tekemään peli Unity-pelimoottorilla. Unityssä on paljon asioita valmiina, kuten valmis törmäystarkastelu, Unityn omat kirjastot sekä 3D-maailma. Näistä syistä peliä ei tarvitse ohjelmoida aivan kokonaan alusta. Myös joidenkin asioiden ohjelmoiminen on helpompaa. Unity kääntää myös suoraan pelin eri alustoille, kuten selaimelle, Windowsille, Linuxille ja Macille. Unity on hyvä kehitystyökalu pienille ja keskisuurille peleille. Kun Unityä käyttää pelimoottorina, täytyy totutella sen käyttöliittymään ja ohjelmointiin, sillä Unity tuo paljon uusia luokkia C#-kieleen. Unityn omilta sivuilta löytyy myös hyvät ohjelmointiohjeet.

Unity on hyvä pelimoottori, kunhan käyttöliittymän oppii. Pienen strategiapelin tekemiseen Unity on riittävä. Ainoa asia, jota Unityssä kaivattaisiin strategiapeliin valmiina komponenttina, on polunetsintä. Pelissä tämä hoidettiin kuitenkin kolmannen osapuolen tekemällä ilmaisella polunetsinnällä.

Tavoitteeseen päästiin hyvin. Pelistä tuli odotuksien mukainen. Pelin kehitystä tosin jatketaan vielä opinnäytetyön jälkeenkin. Tilaaja on peliin tyytyväinen, mutta haluaa jatkokehittää sitä. Ainakin 3D-grafiikka uusitaan ja peliin voidaan myös liittää uusia ominaisuuksia.

Asiasanat: unity, natiivi, tietokone, mobiili, konsoli, 3D-maailma, 3D-grafiikka, pelimoottori, C#

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
Option of Software Development

HAAPALEHTO VILLE-VEIKKO:
Developing a Game in the Unity Game Engine
Case: Battles of Ancient Israel

Bachelor's thesis 67 pages, appendices 19 pages
March 2014

Computer, mobile and console game markets continue to grow in Finland and around the world. There are games for multiple platforms: mobiles, computers and consoles. Operating systems also vary from platform to platform. For example the most popular computer operating systems are Windows, Mac and Linux. Developing a game from from scratch for each platform can be expensive, so a gaming engine or some cross-platform technology is usually needed. This way the games become platform independent. Gaming engines also usually convert programs to multiple platforms such as Windows, Mac or Linux

The goal of this Bachelor's thesis was to enhance evangelization through gaming. The goal was to develop a game about the history of Israel according to the Bible. The designing of the game was carried out by another Tampere University of Applied Sciences student.

Unity gaming engine was selected as the development tool for the game. Unity already has many things built in, for example a collision checking tool, Unity's own databases and 3D world. With these tools it is not necessary to program a game from the ground up and programming some things is much easier. Unity also makes a game platform independent, so you can play it with different platforms (browser, Windows, Mac, Linux). Unity is a great tool for developing small to medium sized games. When Unity is used as a development tool, one must to get used to its user interface and programming because Unity brings lots of new categories to C# code. There are also great instructions for coding available on the Unity website.

Unity is a great tool, once the user interface becomes familiar to the user. Unity is sufficient for making a small strategy game. The only big thing Unity is really missing is a pathfinding component. In the game, however, this was done with a free third-party pathfinding component.

The goal was reached with good results. The game turned out as expected. Development of the game is continued even after the bachelor's thesis. The commissioner was satisfied with the game but wants to keep developing it further. At least the 3D graphics will be renewed and new features may also be added.

Key words: 3D, unity, cross-platform, game engine, computer, mobile, console, c#

SISÄLLYS

1 JOHDANTO ... 7
2 TAUSTAT ... 8
2.1 Tavoite ja tarkoitus ... 8
2.2 Suunnitelmat ... 8
2.2.1 Graafinen ilme ... 8
2.3 Pelien yleinen ohjelmointi ... 9
2.4 Pelimoottorit ... 9
2.4.1 Unity 3D ... 9
2.4.2 Adobe Flash ... 10
2.4.3 HTML5 ja JavaScript ... 10
2.4.4 UDK ... 11
2.5 Pelialustat ... 12
2.5.1 Tietokone ... 12
2.5.2 Konsolit ... 13
2.5.3 Mobiili ... 13
2.6 Pelin julkaisutavat ... 14
3 MENETELMÄT ... 15
3.1 Unity 3D ... 15
3.2 C# ... 16
3.3 Cinema 4D ... 16
4 PELIEN OHJELMOINTI UNITYLLÄ ... 17
4.1 GameObjectin tekeminen ... 18
4.2 Ohjelmoiminen Unityllä ... 19
5 PELIN OHJELMOINTI ... 21
5.1 GameManager ... 22
5.2 Resurssit ... 22
5.3 Maasto ... 23
5.4 PathFinding ... 23
5.5 Rakennukset ... 24
5.6 Yksiköt ... 24
5.7 Käyttöliittymä ja menut ... 26
5.8 Ongelmat ... 29
6 PELAAJAN OHJELMOIMINEN ... 31
6.1 Rakentaminen ... 31
6.2 Yksiköiden liikuttaminen ... 32
6.3 Resurssienhallinta ... 33

6.4 Sotiminen ... 34
6.5 Ongelmat ... 35
7 TEKOÄLYN OHJELMOINTI ... 38
7.1 Tekoälyn toiminta ... 38
7.1.1 Rakentaminen ... 38
7.1.2 Liikkuminen ... 38
7.1.3 Resurssienhallinta ... 39
7.1.4 Sotiminen ... 39
7.2 Ongelmat ... 40
8 TEHTÄVIEN RAKENTAMINEN ... 41
8.1 Tavoitteiden asettaminen ... 41
8.1.1 Tavoitteet yleisesti ... 41
8.1.2 Tavoitteiden ohjelmoiminen ... 42
8.2 Ongelmat ... 45
9 JOHTOPÄÄTÖKSET ... 46
LÄHTEET ... 47
LIITTEET ... 49
Liite 1. GameDesing Document ... 49

LYHENTEET JA TERMIT

GameObject	Pelimaailmassa oleva asia tai esine, esimerkiksi rakennus, yksikkö, puu jne.
Komponentti	GameObjectissa kiinni oleva ”komponentti” joka saattaa tarkoittaa Collideria, C#-luokkaa tai muuta vastaavaa.
Collider	GameObjectien välisiä törmäyksiä hallitseva komponentti.
Event	Pelitapahtuma, jonka laukaisijana toimii joko Trigger Collider tai muu pelimaailman tapahtuma.
Trigger Collider	Ei hallitse varsinaisesti törmäyksiä GameObjectien välillä, vaan kun törmäys tulee alueelle, niin Trigger Collider laukaisee eventin.
Pathfinding	Tarkoittaa sitä, miten yksikkö etsii tai löytää paikasta paikkaan.
MainMenu	Pelin päävalikko, josta pääsee peliin sisälle ja pois. Muita ominaisuuksia ovat pelistä riippuen esim. tekijät, versionumero jne.
Minimap	Strategiapeleistä käytännössä aina löytyvä palikka, jossa voi katsoa missä menee oma ruutu ja missä on omia joukkoja. Joukkoja voi komentaa myös tätä kautta.
Rigidbody	Huolehtii fysiikoista, kuten esim. tippumisesta.
Frames per second	Kertoo, montako kertaa peli pyörii sekunnissa.
Frame	On yksi kokonainen pelisilmukan pyörähdys.
Statsit	Yksikön elämä, puolustus ja hyökkäys.
GameDesing	Pelin käsikirjoitus. Kertoo, kuinka peli etenee ja kuinka se toimii. On selitetty dokumentissa.
Skene	3D-alue/-maailma, jossa jokin pelin tapahtuma on.
PathFinding	Tarkoittaa polunetsintää. Miten yksikkö löytää paikasta toiseen.

1 JOHDANTO

Suomessa pc-, konsoli- ja mobiilipelimarkkinat ovat tällä hetkellä kovassa vauhdissa. Rovio nousi vuonna 2009 maailmankartalle huimaa vauhtia, kun se julkaisi Angry Birds -pelin. Pari kuukautta sitten Supercell myi itsestään 51 prosenttia japanilaiselle teleoperaattorille ja peliyhtiölle. Monet lehdet ovat myös sanoneet, etteivät nämä tule jäämään ainoiksi pelimaailman tähdiksi Suomessa. Myös muualla maailmalla pelien markkinat ovat jo suuret ja yhä kasvavat.

Toimeksiantaja on Nuotta.com, joka toimii Sley-Media Oy:n alaisena. Sley-Median päätehtävänä on kustantaa ja levittää kristillisiä kirjoja. Nuotta.com julkaisee Nuotta-lehteä sekä julkaisee uutisia http://nuotta.com/. Toimeksiantaja ei antanut peliin budjettia, joten emme voi hankkia peliin mitään maksavaa asiaa.

Tilaajan haluamia pelejä ei ole markkinoilla. Pelejä, jotka evankelioivat ja levittävät kristinuskoa, ei käytännössä esiinny, ellei lasketa joitain pieniä flash-pelejä. Siksi tilaaja haluaa meidän suunnittelevan ja toteuttavan evankelioivan kristillisen pelin. Itse keskityn pelissä toteutukseen ja ohjelmoimiseen, kun taas Heikki Vihervaara suunnittelee pelin.

Pelin suunnittelija päätyi evankelioivaan strategiapeliin, jossa kerrotaan Vanhan Testamentin tunnetuimmat tapahtumat. Suunnittelussa otettiin huomioon myös se, että pelillä on vain kaksi tekijää, joten joistain ominaisuuksista, kuten grafiikasta, jouduttiin tinkimään.

Tässä raportissa kerrotaan, kuinka peli tehtiin. Alussa käydään läpi mahdollisia pelimoottoreita, -alustoja sekä julkaisutapoja. Tämän jälkeen moottori ja alusta päätetään niin, että peli tehdään kyseisellä moottorilla ja julkaista alustalle/alustoille. Lopussa käydään läpi pelin tekeminen ja mahdollisten ongelmien kohtaaminen ja ratkominen.

Opinnäytetyö rajataan pelin tekemisen osalta siihen, että tässä käsitellään vain pelin tekemistä ohjelmoinnin kannalta. Raportissa ei kerrota äänien tai 3D:n tekemisestä ollenkaan.

2 TAUSTAT

2.1 Tavoite ja tarkoitus

Opinnäytetyön tavoitteena on kehittää pelien kautta tapahtuvaa kristillistä evankeliointia. Jos peli onnistuu evankelioimaan ihmisiä ja saa muutkin kristilliset järjestöt kehittämään hyviä kristillisiä pelejä, niin olemme päässeet tavoitteeseemme.

Opinnäytetyön tarkoitus on toteuttaa peli Israelin historiasta Raamatun mukaan. Suunnittelu löytyy toisesta opinnäytetyöstä. Opinnäytetyö rajataan toteutuspuoleen ja Heikki Vihervaara hoitaa suunnittelupuolen.

2.2 Suunnitelmat

Pelin yleisistä suunnitelmista vastaa Heikki Vihervaara ja pelin pelikäsikirjoitusdokumentti on liitteenä. Peli on tehty ja ohjelmoitu pelikäsikirjoituksen mukaisesti. Joitain suunniteltuja asioita ei voitu toteuttaa ajasta sekä muista ongelmista johtuen. (liite 1)

2.2.1 Graafinen ilme

Pelin graafinen ilme tulee muistuttamaan hieman piirrosmaista tyyliä, koska pelillä on vain kaksi tekijää. Tekijät ovat minä, joka toteutan teknisen osuuden, ja Heikki Vihervaara, joka suunnittelee pelin. Jos projektissa olisi enemmän tekijöitä, niin pelistä olisi mahdollista saada myös graafisesti näyttävämpi. Graafista ilmettä tulee hieman heikentämään se, ettei tekijöistä löydy 3D-graafikkoa, joten grafiikka pysyy yksinkertaisena.

2.3 Pelien yleinen ohjelmointi

Pelien ohjelmointi eroaa normaalista toimistosovelluksen tekemisestä monella tavalla. Toimistosovellus odottaa, kunnes käyttäjä tekee jotain, kun taas peli pyörii silmukassa. Pelin paras pyörimisnopeus silmukassa on vähintään 30 kertaa sekunnissa. (McShaffry, M., Graham,D, 2013, 175)

Toimistosovelluksessa voi ja kannattaakin käyttää kaikkia olio-ohjelmoinnin ominaisuuksia, kuten Get- ja Set-metodeita. Peleissä näitä metodeja ei kannata käyttää, sillä ne kuluttavat ylimääräistä resurssia. Peleissä ohjelman optimointi mahdollisimman tehokkaaksi on erittäin suuressa roolissa, ja siksi se pitäisi tehdä huolella.

Peliä voi alkaa ohjelmoida, joko pelkällä ohjelmointikielellä, tai ottaa käyttöön pelimoottorin. Pelkällä ohjelmointikielellä tehtäessä kaikki pitää ohjelmoida itse, kun taas pelimoottoreissa yleensä löytyy valmiita ominaisuuksia, kuten törmäystestailut ja grafiikanpiirrot. (Unity 3D. 2013, GameEngine, 2008)

2.4 Pelimoottorit

Pelimoottorit ovat pelien ohjelmistokehys. Sen päälle pelienkehittäjät rakentavat oman pelinsä. Pelimoottorissa saattaa olla joitain ominaisuuksia valmiina, kuten fysiikkamallinnus, 2D- tai 3D-maailma ja animointi. Niiden kautta peli myös julkaistaan jollekin alustalle, kuten esimerkiksi mobiilille, tietokoneelle ja konsoleille. (GameEngine, 2008)

2.4.1 Unity 3D

Unity 3D on pelimoottori, jonka päällä toimii Unityn oma editori. Editorissa saat luotua GameObjecteja ja niille erilaisia komponentteja. Mukana Unityssä tulee myös MonoDevelop, joka on tarkoitettu ohjelmoimiseen. Editorissa GameObjectit kasataan erilaisista GameObjecteista, komponenteista ja koodin pätkistä. GameObjectin alaisia objecteja voivat olla esim. 3D-mallit. Komponentteihin voi kuulua erilaiset Colliderit. GameObjectit saadaan reagoimaan muuhun maailmaan ja ympäristöön koodilla. (Unity 3D. 2013)

2.4.2 Adobe Flash

Adobe Flash soveltuu parhaiten 2D-animaatioihin ja -peleihin. Suurin osa nettipeleistä ja mainoksista käyttää alustana juuri Flashiä. Flash ei sovellu 3D:n tekemiseen vaan sen vahvuudet ovat 2D:n puolella. Ohjelmalla saa luotua kyllä 3D:tä, mutta se vaatii hieman kikkailua tai ulkopuolisen lisäosan hankkimista. Paras vaihtoehto on luoda 2D:tä, joka muistuttaa tai näyttää ihan 3D-grafiikalta ja animoida sitä. Flash tarvitsee selaimeen erillisen lisäosan, jotta se toimii. (Adobe Flash. 2013)

Adoben Flash ohjelmistossa tulee suoraan työkalut grafiikan tekoon ja animaatioon. Grafiikan teko onnistuu siis suoraan itse ohjelmalla. Vaihtoehtoisesti voit tehdä myös jollain muulla ohjelmalla grafiikkaa ja tuoda sen ohjelmistoon. Flash käyttää vektorigrafiikkaa, joten grafiikka kannattaa tuottaa sellaisella ohjelmalla, joka tuottaa kyseistä grafiikkaa. Ei siis ole välttämättä kannattavaa piirtää grafiikkaa Paint-ohjelmalla. (Adobe Flash. 2013)

Flash-ohjelmistossa käytetään ohjelmoimiseen ActionSript-kieltä. ActionScript 3.0 tuli Flash 9 -versioon ja se toi tuen olio-ohjelmoimiselle, mikä mahdollistaa monimuotoisemman ohjelmoinnin ActionScriptillä. Ohjelmointikielellä Flashissä ohjataan animaatioiden pyörimistä, törmäystarkastelua ja pelin tapahtumia. (Adobe Flash. 2013)

2.4.3 HTML5 ja JavaScript

HTML5:n ja JavaScriptin suurin vahvuus on riippumattomuus alustasta. HTML5:een ja JavaScriptiin ei tarvita muuta kuin nykyaikainen selain. Tosin eri selainten välillä on vielä eroja siinä, mitä HTML5-ominaisuuksia ne tukevat ja miten. Esimerkkinä canvasin piirrossa on ollut joitain eroja selainkohtaisesti. Canvaksella piirretään erilaista grafiikkaa ohjelmoinnin avulla. (Canvas. 2013)

HTML:ää ja Javascriptiä ei myöskään tarvitse kääntää mitenkään, vaan selain tulkitsee koodia suoraan. Tosin ohjelmoimisessa saattaa ilmetä ongelmia, koska virheiden etsimiseen tarkoitetut työkalut voivat olla hieman puutteelliset. Suurimpaan osaan selaimista saa asennettua jonkinlaisen kehitystyökalun, joka kertoo missä virhe sijaitsee. Kuitenkin

virheen näyttämisessä saattaa olla ongelmia, koska joskus konsoli saattaa näyttää virhettä väärään paikkaan. (W3School. 2013)

HTML5:teen ja JavaScriptiin on mahdollista saada ulkopuolisia kirjastoja tai pelimoottoreita, jotka tukevat joskus myös 3D-pelin kehitystä. Sivustolta http://html5gameengine.com/ voi selailla mahdollisia HTML5-pelimoottoreita ja niiden ohjeita. HTML5 ja JavaScriptillä voi myös natiivina ilman lisäkirjastoja yrittää ohjelmoida 3D-peliä, mutta se on erittäin vaikeaa. HTML5-pelimoottorien taso ja sisään pääsemisen helppous vaihtelee hieman moottorista riippuen.

Esimerkiksi suosittu moottori Construct 2:ssa on aktiivinen kehitys ja yhteisö, mutta jos ottaa ilmaisen version, niin eventtejä voi olla vain 100. Eventeillä ohjataan moottorissa peliä. Moottori on tarkoitettu ennen kaikkea 2D-pelin kehitykseen, mutta moottorissa on myös 3D-tuki. (Construct lisenssi. 2014)

Olemassa on muitakin moottoreita, jotka tukevat 3D-grafiikkaa. Suurin osa moottoreista on kuitenkin vahvimmillaan 2D-grafiikassa. Moottoreilla voi olla myös valmis fysiikkamallinnus tai törmäystarkastelu. Tulevaisuudessa varmasti tulee markkinoille uusia HTML5-pelimoottoreita ja parempi 3D-tuki myös. Osa pelimoottoreista on täysin ilmaisia ja toiset maksaa jossain vaiheessa, tai ovat vaihtoehtoisesti täysin maksullisia.

2.4.4 UDK

UDK eli Unrealengine on pelin kehitykseen tarkoitettu 3D-pelimoottori. Moottorissa tulee valmiina fysiikat ja törmäystarkastelut. UDK:ssa tehdään töitä yleensä 3D-mallinnus ohjelmia muistuttavassa neljän ruudun tilassa, joista yksi kuvaa 3D-kameralla skeneä ja muut näyttävät sen yhdestä suunnasta 2D:nä tai vaihtoehtoisesti vain 3D-kameraa käyttäen. Yleensä jako menee neljän ruudun tilassa niin, että yksi on 3D, toinen näyttää 2D:nä ääriviivat ylhäältä ja kaksi muuta näyttävät eri sivuilta siten, että ääriviivat näkyvät. (Unreal. 2014)

Unrealissa ohjelmointi kielellä toimii UnrealScript. Se muistuttaa hieman C++-, C- ja Java-kieliä. Metodi tehdään ohjelmointikielessä näin `function int Plus(int e, int c)`. Unrealista löytyy myös ohjeet ohjelmoimiselle:

http://udn.epicgames.com/Three/UnrealScriptReference.html#Design%20goals%20of%20_UnrealScript. Kieleen pääsee sisälle lukemalla ohjeita, tosin se voi olla kielenä vaikeampi hallita kuin C# tai Java. (Unreal. 2014)

UDK muistuttaa tietyiltä ominaisuuksiltaan Unityä. Luodaan pelialue, jonne aletaan tehdä 3D-objecteja ja laitetaan ne reagoimaan maailmaan koodin ja törmäystestailun avulla. Törmäyksiin voidaan myös vaikuttaa suoraan koodilla. Esimerkiksi mitä tapahtuu, jos jokin objekti törmää johonkin. (Unreal. 2014)

UDK:lla ei ole kauppapaikkaa, mistä voisi ostaa tai latailla toisten tekemiä ratkaisuja, mutta yhteisöltä voi kysyä apua ja sieltä voi saada hyviä ratkaisuita. (Unreal. 2014)

2.5 Pelialustat

2.5.1 Tietokone

Tietokone on toiminut pelialustana jo 1980-luvulta asti. Yksi tunnetuimmista ja legendaarisimmista pelialustoista oli vuonna 1982 julkaistu Commodore 64. Ensimmäiset 3D-näytönohjaimet saapuivat markkinoille 1990-luvun puolenvälin tienoilla. Niistä ajoista tietokoneen tehot ja kyvyt ovat kehittyneet paljon. Älypuhelin on myös pieni tietokone, mutta pelien tekemisessä lasketaan omaksi alueeksi juuri tehojen ja koon vuoksi

Tietokoneelle julkaisemisessa suurin heikkous on tietokoneiden suuri tehohaarukka. On täysin eri asia, onko pelaajalla 2000 euron tehomylly vai noin 400 euron kannettava tietokone. Tästä syystä tietokoneelle julkaistessa joudutaan yleensä säätämään peliä niin, että siitä voidaan joko karsia turhista graafisista asioista tai vastaavasti lisätä niitä, riippuen koneen tehoista.

Tietokoneella on myös otettava huomioon, jos julkaisee useammalle alustalle, että se toimisi mahdollisimman hyvin julkaistavilla alustoilla. Tärkein käyttöjärjestelmä pelaamiseen on Windows ja sen eri versiot. Muita järjestelmiä, joille julkaisua kannattaa

miettiä, ovat Mac- ja Linux-tietokoneet. Useat pelimoottorit kuitenkin kääntävät ohjelman suoraan eri käyttöjärjestelmille, kuten Unity ja UDK. (Unity 3D. 2013)

Tietokoneiden vahvuuksiin kuuluvat ehdottomasti hiiri ja näppäimistö -yhdistelmä, joka tekee joidenkin genren peleistä paremmin pelattavia tietokoneella, kuin muilla alustoilla. Paras esimerkki on varmasti strategia-genre. Strategiassa hiiren ja näppäimistön yhdistelmä on ylivoimainen verrattuna ohjaimeen. Joukkojen käskytys ja muu on hiirellä paljon helpompaa kuin ohjaimella. Tämän takia parhaat strategiapelit ovat aina tietokoneella ja niistä ehkä tehdään konsoliversio, jos tehdään. Joskus konsoliversioihin on myös yksinkertaistettu näppäimiä tai toimintoja tietokoneeseen verrattuna. (PCvsConsole. 2012)

2.5.2 Konsolit

Vuonna 1983 Nintendo julkaisi ensimmäisen kotikonsolin. Historiaa on siis myös konsoleilla jo 30 vuotta. Playstation tuli mukaan markkinoille vuonna 1995 ja Xbox vuonna 2001. Nykyään on menossa jo PlayStationin neljäs ja Xboxin kolmas sukupolvi. PlayStation 4 ja Xbox One julkaistiin marraskuussa 2013.

Suurimpia vahvuuksia konsoleilla on se, että pelejä ei tarvitse erikseen optimoida monelle käyttöjärjestelmälle tai monille versioille, vaan kun peli julkaistaan esim. PlayStation 3:lle, niin se toimii varmasti kaikissa PS3-konsoleissa. (PCvsConsole. 2012)

Pelikonsoleille ohjain on samanaikaisesti vahvuus ja heikkous. Joillekin peleille ohjaimet sopivat täydellisesti ja auttavat pelaajaa parempaan kokemukseen, toisiin peleihin eivät taas ohjaimet tahdo sopia millään, esimerkkinä strategiapelit. (PCvsConsole. 2012)

2.5.3 Mobiili

Nykyaikana mobiili on todella varteenotettava pelien julkaisualusta. Kun Nokian Java-pelit, Android, IOS ja Windows Phone tulivat markkinoille, niin pelien kehitys alkoi

kunnolla myös mobiilille. Moni suomalainen studio tekee nykyään mobiilille pelinsä ja ovat luoneet menestyksensä juuri kosketusnäytöllisillä puhelimilla ja tableteilla.

Mobiilin heikkoudet ovat siinä, että alustana ne eivät vielä ole yhtä tehokkaita kuin tietokoneet tai konsolit. Kuitenkin kehitystä on tapahtunut viime vuosina ja tehot tulevat vielä nousemaan lisää. Myös mobiililaitteilla on tehoeroja riippuen mallista ja hinnasta.

Toinen rajoittava tekijä on kosketusnäyttö. Pelille pitää olla näytössä mahdollisimman paljon tilaa ja pelin ohjaus pitäisi tapahtua kosketusnäyttöä tai sensoreita hyväksi käyttäen. Peliä ei saisi mielellään ohjata ylimääräisillä nappuloilla, sillä ne vievät tilaa ruudulta. Mobiilin vahvuus on pienissä peleissä, joita pelataan pienissä erissä. Pienissä erissä pelaamista tarkoitetaan yleensä maksimissaan 15 minuutin pelisessiota.

2.6 Pelin julkaisutavat

Pelejä voidaan nykyään julkaista monella tavalla. Yleensä esim. matkapuhelimille ja tableteille on omat kauppapaikkansa riippuen käyttöjärjestelmästä. Konsoleille, Mac-koneille ja PC:lle pelejä voidaan julkaista erilaisissa kauppapaikoissa (esim. Steam), ihan perinteisesti cd- tai dvd -levyllä tai internetsivulla joko vapaaseen käyttöön tai maksua vastaan.

Tilaaja ei halua pelistä minkäänlaista maksua tai mitään muutakaan tuottoa, vaan toivoo, että peliä pelaisi mahdollisimman moni ja myös, että se herättäisi mielenkiintoa. Tämän takia peli tulee levitykseen tilaajan omille sivuille täysin ilmaisena ja vapaasti pelattavana. Peli julkaistaan ensiksi Web- ja Windows-versiona. Myöhemmin peli saatetaan julkaista myös Linuxille ja Macille, jos peli saa suuta huomiota ja paljon pelaajia. Vieläkin suurin osa tietokonepelaajista pelaa Windowsilla, kun taas Mac- ja Linux-koneille on vähemmän pelejä ja pelaajia.

3 MENETELMÄT

3.1 Unity 3D

Unity valittiin menetelmäksi paristakin syystä. Pelimoottori on tarkoitettu 3D-pelin kehitykseen ja siinä 3D toimii ilman suurempaa kikkailua tai ylimääräisten ohjelma kirjastojen latailua. Pelimoottori kääntää myös pelin useammalle alustalle kerralla, joten kun pelimme 3D-mallit ovat kooltaan yli 30 megabittiä, niin pelaajat joilla on hitaat netit, voivat ladata pelin suoraan omalle kotikoneelleen, ettei tarvitse koko pakettia joka kerta uudelleen ladata peliä pelatessaan. Kun pelin kääntää Unityn WebPlayeriin, niin se pakkaa materiaalit ja muut pienemmäksi. Tämä helpottaa siten, ettei tarvitse opetella uusia pakkausmenetelmiä pelin tekemisen ohessa.

Tulevaisuudessa, kun pelimme 3D-grafiikka jatko kehitetään 3D-grafiikkaa uudistamalla, niin emme voi tietää, paljonko 3D tulee viemään tilaa silloin. Jos 3D-grafiikan tulee viemään paljon enemmän tilaa silloin kun nykyään, niin on pakko ajaa peliä suoraan koneelta, eikä selaimen kautta. Pelaajan kannalta on mukavempaa, jos pelissä ei ole liian suuret latausajat.

Unityssä on tehty asioita paljon valmiiksi kehittäjälle. Pelimoottorista löytyvät kaikki pelin peruselementit valmiina, kuten painovoima ja kehittynyt törmäystarkastelu. Unityn editorista löytyy myös kattava AssetStore, josta saa joko ilmaiseksi tai rahaa vastaan esimerkiksi valmiita taloja ja yksiköitä (Unity 3D. 2013). Unitylle löytyi myös hyvä polunetsintä: polut löytyvät hyvin ja projekti oli ilmainen.

Unityssä pelin tekemiseen löytyy todella kattava ohjeistus aina törmäystarkastelusta 3D-objektien käyttöön. Ohjelmointiin löytyy myös laajat ohjeet Unityn ScriptReference -sivuilta. Kaikki ohjelmoitavat ominaisuudet löytyvät sivuilta ja tämän takia pelin ohjelmoiminen on helpompaa. Valmistajan sivuilla olevat ohjeet ovat laajat, niin apua ei tarvitse etsiä foorumeilta joissa kaikki ratkaisut eivät välttämättä ole toimivia.

Näiden syiden takia valitsimme Unityn kehitys alustaksemme. Unityssä on hyvät ominaisuudet ja monia asioita kehittäjälle valmiiksi tehtynä.

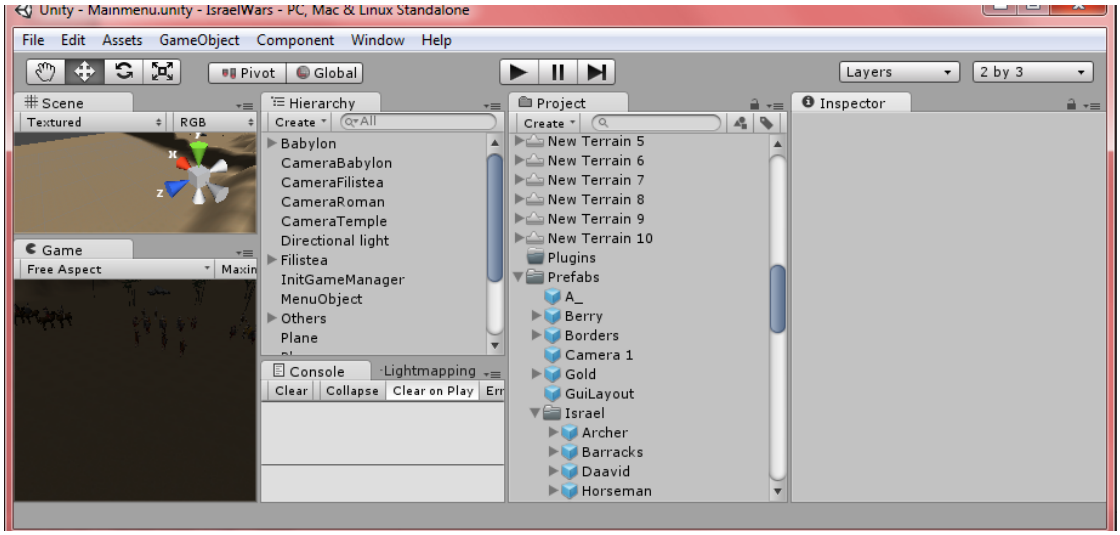
3.2 C#

Unityssä on vaihtoehtoina pari eri ohjelmointikieltä: JavaScript, C# ja Boo. Unityssä löytyy täysi tuki C#-kielelle. Ohjeet ja esimerkit löytyvät Scripting Referencestä suoraan C#-kielellä. Tämän takia valitsen ohjelmointikieleksi C#-kielen.

3.3 Cinema 4D

Cinema 4D tuottaa suoraan Unity-yhteensopivaa 3D-mallia fbx:ää. Tämän takia mallinnus tapahtuu Cinema 4D-ohjelmalla.

4 PELIEN OHJELMOINTI UNITYLLÄ



KUVA 1. Unity 3D:n perusnäkymä

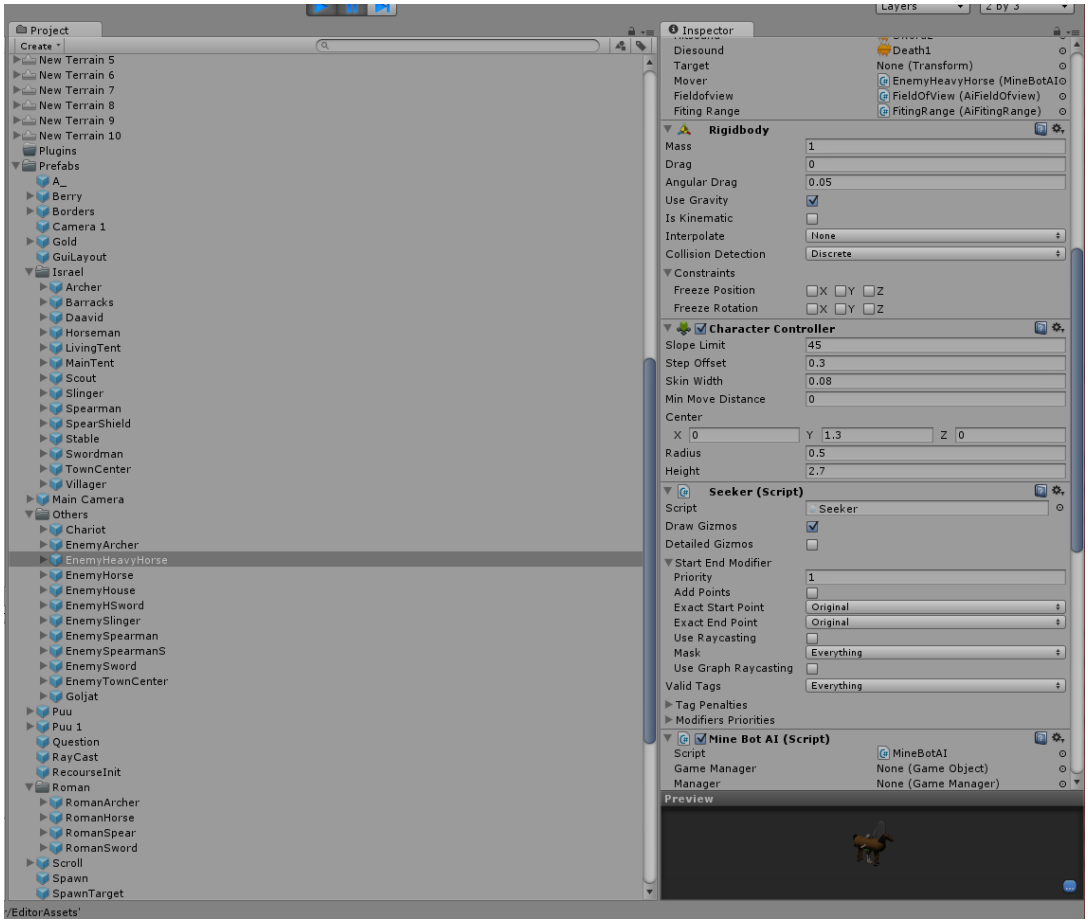
Unity-pelimoottoria käytetään aina perusnäkymässä. Ylhäällä on työkalupalkki, josta saadaan luotua GameObjecteja ja laitettua niille komponentteja. Ylhäällä on palkissa on käsi, jolla skenen näkymää voi viedä paikasta toiseen. Vieressä on nuolia, jotka siirtävät objektia 3D-maailmassa, kääntävät sitä ja skaalaavat isommaksi tai pienemmäksi. (Kuva 1) (Unity 3D. 2013)

Kuvassa näkyy myös kaksi kameraa, joista ylempi on muokkauskamera. Muokkauskameraa käytetään tekemään skene tai muokkaamaan sitä. Alempi taas näyttää pelin maincameran ja sen, miltä peli pelaajalle tulee näyttämään. Tätä kameraa tullaan käyttämään pelin testauksessa. Ylhäällä näkyvät play-, pause- ja playbyframe-nappulat. Playbyframe vie yhden peliä peliloopin eteenpäin joka painalluksella. (Kuva 1) (Unity 3D. 2013)

Hierarchy-kohdassa näkyy se, mitä on tällä hetkellä skenellä ja niitä tuplaklikkaamalla pääsee objektin luokse. Project-kohdassa näkyy kaikki, mitä tällä hetkellä projekti sisältää. Console kertoo, missä on virheitä tai huomautuksia. (Kuva 1) (Unity 3D. 2013)

4.1 GameObjectin tekeminen

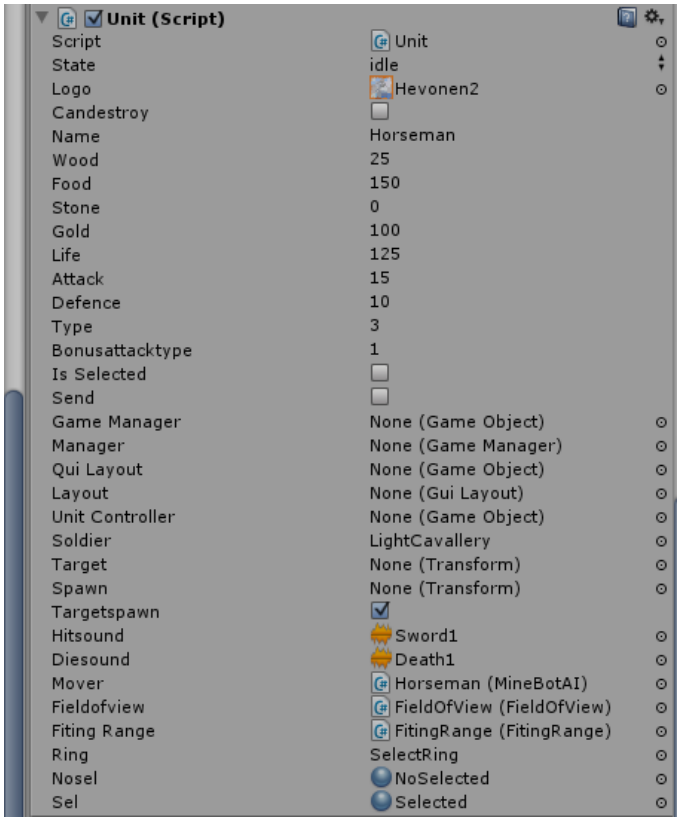
Unityssä pelimaailma koostuu GameObjecteista. Yleensä aloitetaan luomalla Unityssä tyhjä GameObjecti. Se luodaan valikosta GameObject ja valitaan create empty. Tämän jälkeen sille aletaan laittaan komponentteja. Yleisimmät komponentit, mitä GameObjectiin laitetaan ovat Collider ja fysiikkamallinnus(rigidbody). GameObjectiin laitetaan usein myös 3D-malli. GameObject saadaan reagoimaan ympäristöön luokalla, joka laitetaan komponentiksi GameObjectiin. (Unity 3D. 2013)



KUVA 2. Inspector ja project

Inspector näyttää, mitä kyseinen GameObject pitää sisällään. Inspectorista voidaan myös muokata komponenttien arvoja. Jos esimerkiksi luokassa on paljon julkisia muuttujia, niitä voidaan muokata inspectorin kautta. Tämä mahdollistaa sen, että esimerkiksi yksiköillä voi olla sama luokka mutta eri statsit: hyökkäys, elämä jne. (Kuva 2) (Unity 3D. 2013)

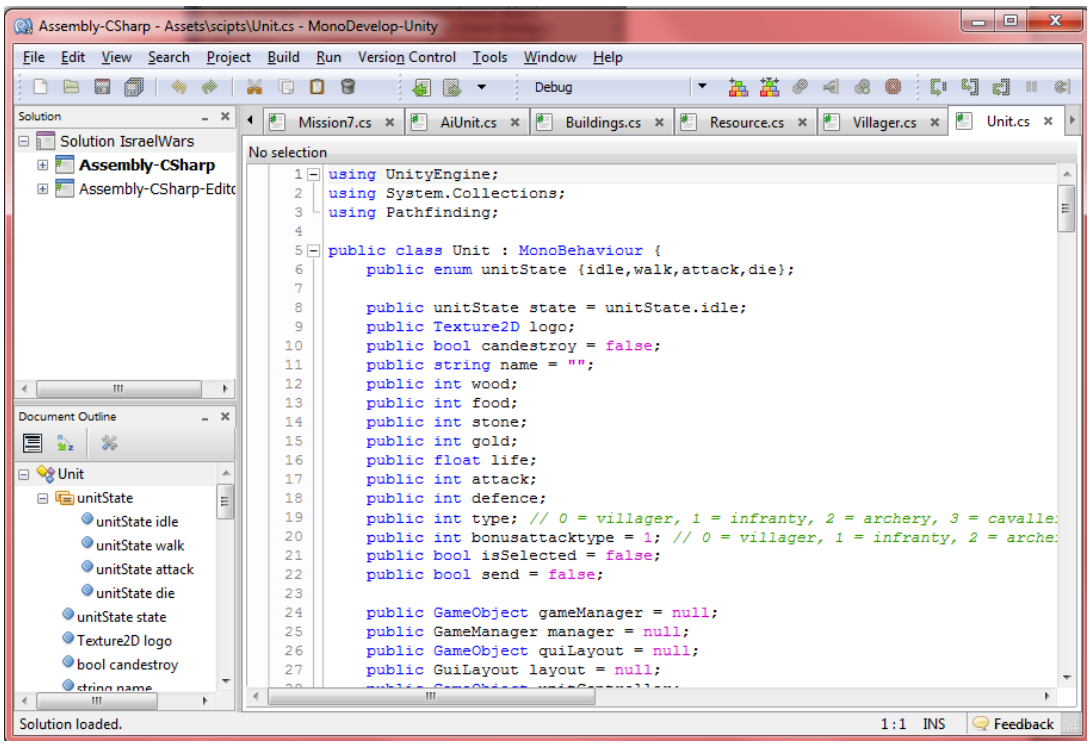
4.2 Ohjelmoiminen Unityllä



KUVA 3. C# tiedosto inspector:issa

C#-kooditiedostosta näkyvät julkiset muuttujat ja niitä pääsee suoraan Unityn mottorista muokkaamaan. Tämän kautta voidaan myös vetää julkisille muuttujille viittaukset. Esimerkiksi ”julkinen” -äänelle voidaan vetää projektista suoraan ääni, jonka jälkeen se ääni soi, kun kyseinen muuttuja soitetaan. Tämä helpottaa ohjelmointia, niin ettei kaikkia muuttujia tarvitse alustaa koodissa. (kuva 3) (Unity 3D. 2013)

Kun peliä testataan Unityn editorissa, niin testaaja tai ohjelmoija pääsee suoraan seuraamaan mitä julkisissa muuttujissa tapahtuu, ilman että tarvitsisi käyttää logia. Tämä nopeuttaa testausta ja sen huomaamista toimiiko kaikki kuten pitää. Lisäksi virheen löytyminen koodista helpottuu, kun tietää missä kohtaa peliä virhe tai ei toivottu muuttujan muunnos tapahtui.

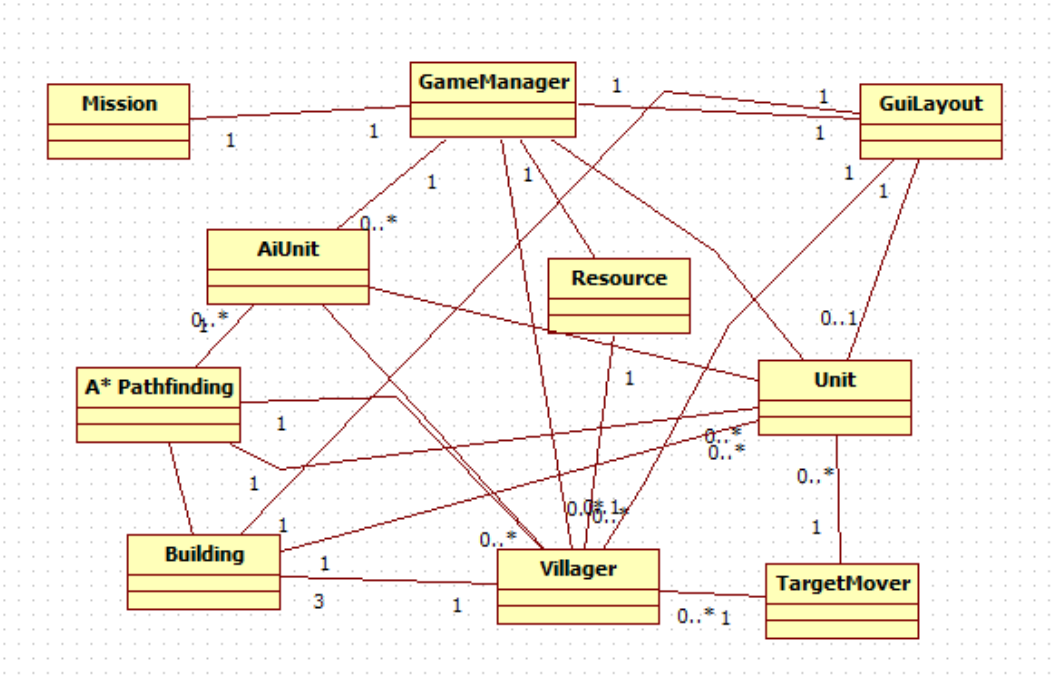


KUVA 4. MonoDevelop-ohjelma

Unityssä koodia muokataan MonoDevelop-ohjelmalla, joka tulee suoraan Unityn mukana (Kuva 4). Unityn moottori tukee kolmea kieltä, joista peliin valittiin C#. Unity-koodi tiedostot ottavat suoraan käyttöön `UnityEngine` ja `System.Collections` kirjastot ja niistä löytyy kaikki luokat ja metodit, joita Unitystä löytyy. Jos haluaa ottaa käyttöön C#:n omia perusluokkia tai ominaisuuksia, niin ne pitää importoida. Unityssä ei toimi suoraan esimerkiksi C#:n oma random, vaan pelimoottorista löytyy oma random. C# random-luokan voi kuitenkin importoida ja ottaa käyttöön, mutta silloin täytyy erikseen määritellä kumpaa randomia käyttää (esim. `UnityEngine.Random.Rage(0,1)`). (Unity 3D. 2013)

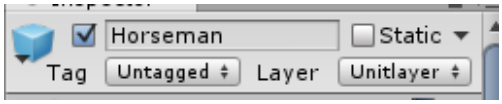
Unityn perusmetodit ovat `Start`, `Update` ja `OnGUI`. Star- metodia kutsutaan aina jokaisen skenen alussa tai GameObjectin luomisessa ja siihen kannattaa laittaa kaikki alustamiseen liittyvät koodit. OnGUI taas ajetaan yleensä kaksi kertaa framessa ja sen tehtäviin kuuluu piirtää pelaajalle käyttöliittymä. Update-metodia kutsutaan kerran framessa ja sinne kuuluu laittaa kaikki GameObjectin päivitykseen liittyvät asiat esimerkiksi animaation ajo. (Unity 3D. 2013)

5 PELIN OHJELMOINTI



KAAVIO 1. Mission

Kaaviossa näemme yhden skenaarion/tehtävän luokkarakenteen ja luokkien suhteet. GameManageri hoitaa pitkälti yleisten asioiden hoidon pelissä. Yksiköt liittyvät toisiinsa niin, että kun vihollisyksikkö on lähellä, ne alkavat taistella. Yksiköt ilmoittavat syntyessään GameManagerille, että populaatioon lisätään yksi yksikkö. Mission taas hoitaa tehtävän tavoitteet ja tarkkailee niiden täyttymistä. Kun tavoitteet täyttyvät, ilmoitetaan siitä GameManagerille. A* Pathfinding liittyy yksikköihin siten, että se ohjaa niiden liikkumista. Rakennukset tuottavat yksiköitä ja ilmoittavat Pathfindingille ettei rakennuksen läpi saa mennä. GuiLayout hoitaa skenaariossa käyttöliittymän ja valittujen yksiköiden piirron pelaajalle.



KUVA 5. Layer

Unityssä on mahdollista tehdä peliin ominaisuuksia, jotka liittyvät Unityn omiin layereihin (kuva 5). Pelissä hyödynnettiin layerin ominaisuuksia siten, että eri layerit eivät välttämättä törmää toisiinsa, eli collision-testaus on pois päältä. Tämä tapahtui kutsumalla metodia `Physics.IgnoreLayerCollision (ensimmäinen layer, toinen layer)`. Tällä tavalla saatiin pelin raskautta pois, kun turhat col-

lision-testaukset eivät toteudu ollenkaan. Tätä layer-asiaa hyödynnettiin niin, ettei esim. resurssit törmää minkään muun kuin yksikkö layerin kanssa. (Layers. 2012, IgnoreCollision. 2013)

5.1 GameManager

Gamemanager GameObject pitää sisällään luokan GameManager. GameManager-luokka huolehtii pitkälti monista yleisistä asioista pelin taustalla. Se pitää kirjaa pelaajan resursseista, pelitilasta ja mistä mihin kamera saa liikkua. (Katso liite 1)

```
6  public class GameManager : MonoBehaviour {
7      public enum GameState {InMenu,InBriefing,InGame,InEndBriefing};
```

OHJELMA 1. GameManagerin enum

Luokka myös ylläpitää Enum-pelin tilaa. Erilaisia tiloja on InMenu, InBriefing, InGame ja InEndBriefing. Nämä tilat kertovat eri GameObjecteille, mitä saa ja mitä ei saa tehdä. Esimerkiksi InMenussa näytetään vain taukomenu ja InGame pyörittää varsinaista peliä. GameManager ei myöskään tuhoudu siirryttäessä skeneltä toiseen, vaan se säilyy koska sitä on käskytetty metodilla `DontDestroyOnLoad(transform.GameObject)`. (Ohjelma 1) (Enums. 2013, Unity Script Reference. 2013)

5.2 Resurssit

Resurssit koostuvat pelissä tyhjästä GameObjectista ja sen alla olevista komponenteista ja GameObjecteista. Komponentteja suoraan tyhjän GameObjectin alla on Resurssit-luokka ja Trigger Collider. Resurssit-luokka ylläpitää tietoa siitä, paljonko kyseistä resurssia on GameObjectissa.

```
void OnTriggerEnter(Collider collision) {
    Villager villager = collision.gameObject.GetComponent<Villager>();
    if(villager != null)
    {
        villageramount++;
        villager.inrecourse = true;
    }
}
```

OHJELMA 2. Trigger Colliderin metodi

Luokka myös kuuntelee Triggeriin tulevat Colliderit (ohjelma 2). Colliderin alueelle tullessa tarkistetaan, onko se työläisyksikkö. Jos on työläisyksikkö ja se pysähtyy alueelle, niin sen tilaksi laitetaan keräys. Tämän jälkeen viiden sekunnin välein annetaan pelaajalle kyseistä resurssia. Mitä enemmän työläisiä on Triggerin alueella, sitä enemmän resursseja tulee pelaajalle. Kiveä ja kultaa tulee viisi sekä ruokaa ja puuta kymmenen yhtä työläistä kohden noin viisi sekunnin välein. (GameObject. 2013, Collider. 2013)

5.3 Maasto

Maastot on toteutettu Unitystä löytyvällä Terrain-työkalulla. Pelialueena toimii kaikissa skenaarioissa noin neljän neliökilometrin kokoinen terrain. Tämä muotoillaan hyvännäköiseksi terrain-työkalusta löytyvillä maastonmuotoilutyökaluilla. Terrainille tämän jälkeen laitetaan tekstuureja, joilla se maalataan kohtalaisen hyvännäköiseksi. Tekstuurit koostuvat aavikosta, ruohosta ja erilaisista kivimateriaaleista. (Terrain. 2013)

5.4 PathFinding

PathFindingilla tarkoitetaan polunetsintää. Eli miten yksikkö osaa maailmassa tai alueella mennä paikasta toiseen. Polunetsintä osaa väistää eteen tulevat esteet ja asiat. (A* Pathfinding Documentation. 2013)

Pahtfindingina käytetään valmista ja ilmaista komponenttia A* Pathfinding -projektia, joka toimii Unityn lisenssien pohjalta. Pelissäni ei ole varsinaista budjettia, joten ilmainen ja hyvin toimiva pathfinding löytyi juuri A* Pathfinding-projektista. Siitä on kaksi versiota, ilmainen ja maksullinen. Maksullisessa tulee ominaisuuksia enemmän, mutta ilmaisella myös pärjää tässä pelissä hyvin. Lisäksi budjetin takia ei olisikaan ollut mahdollista hankkia maksullista. Maksullisessa versiossa tosin olisi tullut muun muassa parempi päivitys liikkumiseen pelin aikana ja enemmän ominaisuuksia polunetsintään ja niin edelleen. (A* Pathfinding Documentation. 2013)

A* Pathfindingiin päädyin juuri ominaisuuksien ja sen ilmaisen version vuoksi. Etsin ja vertailin joitakin vaihtoehtoja, mutta tämä oli ainoa ilmainen. Kuitenkin sain huomata, että ilmaisella versiolla pärjää tämän kokoisessa pelissä hyvin ja siinä on tarpeeksi ominaisuuksia. (A* Pathfinding Documentation. 2013)

Rakennukset päivittävät paikkansa aina, kun ne luodaan ja yksiköidenkin piti liikkuessaan päivittää missä menevät, mutta valitettavasti tämä yksiköiden päivitys osoittautui liian raskaaksi manuaalisesti tehtäväksi. Tähän ratkaisuna olisi ollut ostaa pro-versio, jossa paikan päivitys on tehty automaattiseksi. (A* Pathfinding Documentation. 2013)

Jokainen skene, johon halutaan pathfinding, pitää asettaa A*-komponentti, joka huolehtii siitä, missä ja miten terrainilla voi liikkua. Se skannaa alueen sille asetetuin parametrein, missä voi liikkua ja mitä pitää väistää. Sille myös määrätään, mitkä layerit skannataan, kuinka korkealta ja kuinka paljon yksiköt voivat kiivetä. Tämän perusteella se sitten luo Grid Graph, jossa yksikkö voi liikkua. (A* Pathfinding Documentation. 2013)

Kaikki, jotka liikkuvat, tarvitsevat kaksi luokkaa: Seeker ja MineBotAI, jotka huolehtivat oikean polun löytämisestä ja esteiden väistämisestä. Tämän lisäksi luokat tarvitsevat liikkujille Rigidbodyn ja erikoisen Character Controller -Colliderin. (A* Pathfinding Documentation. 2013)

5.5 Rakennukset

Rakennuksia on pelissä kahta eri tyyppiä. Ensimmäinen rakennustyyppi on vain 3D-malli, jossa on Collider, jotta objektit eivät menisi vahingossa rakennusten läpi. Toinen tyyppi on sellainen, jossa on kiinni rakennusluokka sekä pari tyhjää GameObjectia sotilaiden luomista varten.

5.6 Yksiköt

Yksiköt koostuvat niin pelaajalla kuin teköälylläkin samalla tavalla, ja ne sijaitsevat unitlayerillä. Ainoastaan käytettävät luokat ovat erilaiset. Luokkien erot ovat lähinnä siinä, että pelaaja pystyy ohjaamaan omia yksiköitään, kun taas kone tekee päätöksiä

tekoälyn yksiköille. Yksikkö varsinaisesti koostuu kolmesta Colliderista, joista kaksi on Triggereitä, kolme luokasta ja pathfinding componenteista. (Layers. 2012, Collider. 2013, GameObject. 2013)

```
1  using UnityEngine;
2  using System.Collections;
3  using Pathfinding;
4
5  public class Unit : MonoBehaviour {
6      public enum unitState {idle,walk,attack,die};
7
8      public unitState state = unitState.idle;
9      public Texture2D logo;
10     public bool candestroy = false;
11     public string name = "";
12     public int wood;
13     public int food;
14     public int stone;
15     public int gold;
16     public float life;
17     public int attack;
18     public int defence;
19     public int type; // 0 = villager, 1 = infranty, 2 = archery, 3 = cavalle
20     public int bonusattacktype = 1; // 0 = villager, 1 = infranty, 2 = arche
21     public bool isSelected = false;
22     public bool send = false;
```

OHJELMA 3. Yksikön julkisia muuttujia

Pelaajan yksiköissä on kiinni Unit-luokka. Luokka sisältää kaikki yksikön hallintaan tarvittavat asiat. Luokan tärkeimpiin tehtäviin kuuluu yksikön statsien ylläpitäminen: Paljonko on hyökkäys, puolustus, elämä ja paljonko tilaaminen maksaa ja niin edelleen. Luokassa on viittaus myös yksikön malliin, jonka animointia hallitaan enum-tiloilla. Erilaisia tiloja ovat idle, walk, attack ja die. Kun yksikkö ei ole liikkumassa tai taistelemassa, se on idle-tilassa, jolloin animoidaan idle-tilaa. Kuolema-tila on ainoastaan silloin, kun elämä on nolla tai alle, jolloin yksikkö tekee kerran kuolema-animaation ja sen jälkeen tuhoutuu 3D-maailmasta. (ohjelma 3)

Yksiköllä on myös bool, joka kertoo sen, onko kyseinen yksikkö valittu. Jos se on valittu, yksikkö kuuntelee hiirtä ja sen paikan perusteella alkaa liikkua paikasta toiseen. Yksikön liikkumista voi kuitenkin häiritä, jos vihollisyksiköitä ilmaantuu jommankumman Trigger Colliderin alueelle.

Tietokoneen yksiköt toimivat muuten samalla tavalla, paitsi niiden yksikköön on laitettu AiUnit-luokka, jolloin kone hoitaa kyseisten yksiköiden hallinnan. Tietokoneen yksiköt ovat joko paikallaan pysyviä tai aggressiivisia. Aggressiiviset tietokoneen yksiköt hyökkäävät pelaajan kaupunkiin. Tämän kertoo yksi bool-muuttuja, joka vastaa pelaajalta löytyvää valittua muuttujaa.

Villager-yksikkö on siitä erikoinen, että sillä on Unit-luokan tilalla Villager-luokka. Siellä tiloja on enemmän kuin normaalilla sotilaalla. Erilaisia tiloja ovat idle, walk, building, gathering ja die. Villagerilla on statseina vain elämä, paljonko se maksaa ruoka-resurssia ja viittaus mahdollisesti rakennettaviin rakennuksiin. Kuolema-tila on samanlainen kuin muillakin. Villagerin voi myös valita ja käskyttää paikasta paikkaan. Villager ei pysty taistelemaan ollenkaan, vaan ainoastaan ottaa osumaa jos vihollisen yksikkö pääsee hyökkäämään. Villager on hyödyllinen yksikkö, sillä ainoastaan voi kerätä resursseja.

5.7 Käyttöliittymä ja menut

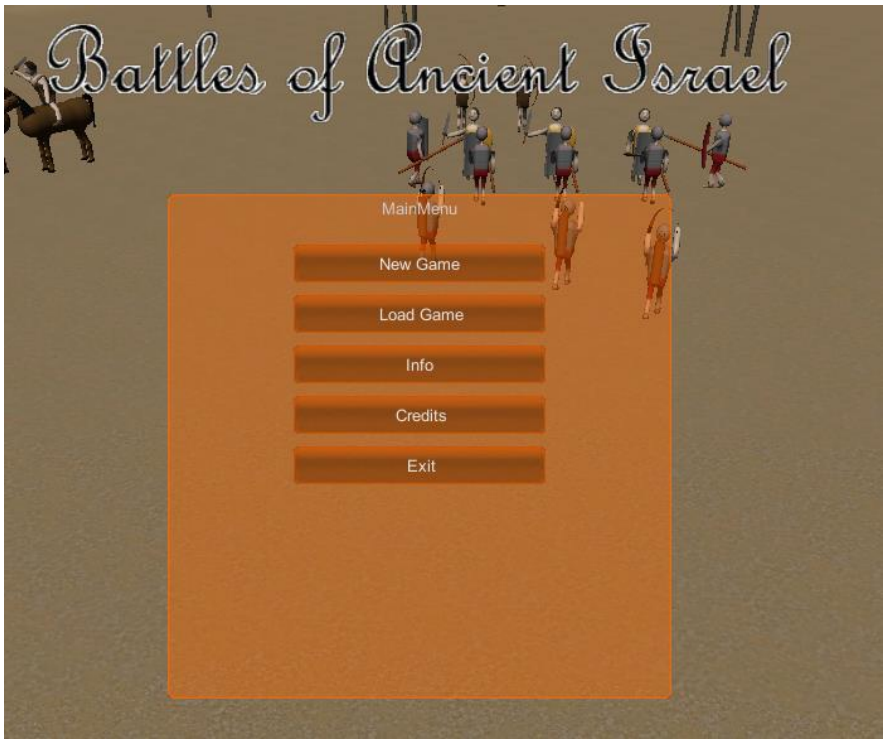
Käyttöliittymät ja menut toteutetaan Unityn omilla työkaluilla. Moottorista löytyvät hyvät työkalut tehdä esimerkiksi nappulat ja tekstilaatikot. Pelissä ei kuitenkaan käytetä pelistä valmiiksi löytyvää grafiikkaa, vaan peliin tulee omat grafiikat kaikille menu- ja käyttöliittymäelementeille. Käyttöliittymän piirron hoitaa pelissä erillinen GuiLayout-luokka. Se huolehtii, että oikeassa kohdassa piirretään oikeanlaiset menut ja nappulat. Poikkeuksena tässä toimii MainMenu, jossa on omasta piirrosta huolehtiva menu-luokka. (GUI. 2012, GUI Skin. 2013)

```
22      void OnGUI()
23      {
24          GUI.skin = style;
25          GUI.DrawTexture(new Rect(Screen.width / 2-300, 5, 600, 200), name, ScaleMode.StretchToFill);
26          GUI.BeginGroup (new Rect ((Screen.width / 2-200), 210, 700, 700));
27          if(!(story) && !(credits) && !(load)){
28          GUI.Box(new Rect(0,0,400,400), "MainMenu ");
29          if (GUI.Button(new Rect(100, 40, 200, 30), "New Game"))
30          {
31
32          Application.LoadLevel("Aavikko");
33
34          }
35          if (GUI.Button(new Rect(100, 80, 200, 30), "Load Game"))
36          {
37          load = true;
38
39          }
```

OHJELMA 4. GUI:n piirtoa

Ensiksi määritellään OnGUI:ssa sen grafiikka, joka tapahtuu `GUI.skin = tyyli`. Tämän jälkeen piirretään alue, johon käyttöliittymäkomponentit tulevat. Helpoin tapa järjestellä käyttöliittymää, on tehdä siitä ryhmä `GUI.BeginGroup(aloistusx, aloitusy, kokox, kokoy)` metodilla. Tämän jälkeen kaikki ryhmän jäsenten

aloituskohta on ryhmän aloituspisteessä ja voi lähteä siitä laskemaan mihin käyttöliittymäkomponentteja tulee. (ohjelma 4) (GUI. 2012, GUI Skin. 2013)

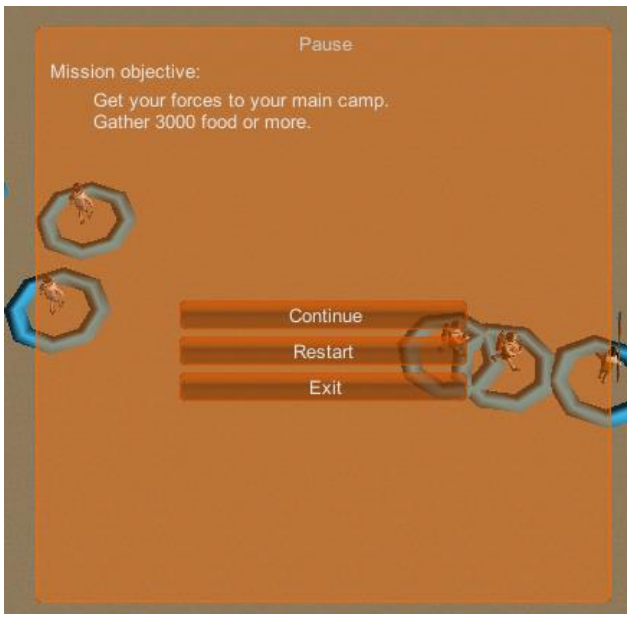


KUVA 6. MainMenu

MainMenu koostuu viidestä eri nappulasta ja kustomoidusta GUISkinistä. New Game -napista pääsee aloittamaan uuden pelin ja Load Game-painikkeesta pääsee lataamaan pelin salasanan avulla. Infosta näkee yksinkertaiset ohjeet ja sieltä löytyy myös linkki syvempiin ohjeisiin. Credits-osiosta löytyvät tekijät ja erikoiskiitokset. MainMenun yllä löytyy pelin nimi, joka on kuva ja piirretään GUI:hin. (kuva 6) (GUI. 2012, GUI Skin. 2013)

KUVA 7. Tehtävän aloitus- ja lopetusruudut

Tehtävän aloitus- ja lopetusruudut ovat GUI-laatikoiden päällä. Ne ovat kustomoitu omalla GUISkinillä. Kaikki GUI:t käyttävät samaa kustomointia. Tehtävän aloituksesta löytyy pieni alkutiedote, jossa kerrotaan tehtävän tavoitteet ja kysytään tehtävään liittyvä kysymys Raamatusta. Jos kysymykseen vastaa oikein, niin saa pienen bonuksen tehtävään (yleensä resurssipalkinto, joka on määrätty tehtäväkohtaisesti). Tehtävässä on myös lopputiedonanto, jossa kerrotaan, suoriuduitko tehtävästä vai et sekä seuraavan tehtävän salasana. (kuva 7) (GUI. 2012, GUI Skin. 2013)



KUVA 8. Taukovalikko

Taukovalikossa näkyy kyseisessä skenaariossa suoritettavat tehtävät. Taukovalikossa on myös kolme nappulaa, joista pääsee takaisin peliin (Continue), päävalikkoon (Exit) ja aloittamaan kyseisen tehtävän alusta (Restart). (kuva 8)

KUVA 9. Pelin alapalkki

Pelin alapalkista näkyy pelaajalle oma tilanne. Sieltä löytyy populaatiolimiitin tilanne sekä resurssien määrä. Alla oikealla näkyy myös minimap, joka kertoo, missä ruutu menee ja missä on omia joukkoja. Joukkojaan voi myös käskyttää tätä kautta. Alapalkki on aina 200 pikseliä korkea ja koko ruudun leveä. Minimap on tästä alueesta 200*200 pikseliä. (kuva 9) (GUI. 2012, GUI Skin. 2013)

5.8 Ongelmat

Ensimmäiset ongelmat pelin ohjelmoinnissa olivat käyttöliittymän ja menujen ohjelmoimisessa. Suurimmat ongelmat käyttöliittymässä liittyivät siihen, kuinka saada komponentit näkymään samalla tavalla erikokoisissa näytöissä. Siksi suurimmista osista

komponentteja tulikin jollain tavalla skaalautuvia tai vaihtoehtoisesti sopivan pieniä, että mahtuvat suurimpaan osaan näytöistä, sillä käyttöliittymä vaatii kuitenkin jonkin verran tilaa. Tilan vaatimusten vuoksi ei voida taata, että peli toimii todella pienillä näytöillä.

Toinen yllättävä asia oli se, ettei Unityssä toiminut C#:n oma random-arpominen vaan piti käyttää moottorin omaa randomia. Jos olisi halunnut C#:n oman randomin toimimaan, niin olisi pitänyt importoida se C#:n puolelta mukaan. (Random. 2013)

Suurimmat ongelmat tulivat yleisessä osiossa yksiköiden liikkumisesta ja törmäystestailuista. Päädyin siksi käyttämään erillisiä layereitä, jotta saan kaikki turhat törmäystestit pois päältä ja jotta kaikki törmäilyt ovat tarkoituksenmukaisia.

Yksiköiden liikkumisessa oli myös suuria ongelmia siinä. Jos yksikön oma Collider oli liian edessä, yksikön paikanpäivitys johti siihen, että yksikkö lähti aina väärään suuntaan ja ns. pakeni omaa varjoaan. Loppujen lopuksi pathfindingista jouduin yksiköiden paikan päivityksen ottamaan pois päältä, koska ilmaisversiossa se aiheutti pientä lagia noin kahden sekunnin välein.

Fog of war -ominaisuudessa oli sellainen ongelma, että jos teki liian pienet neliöt fog of wariin, niin ominaisuudesta tuli liian raskas. Tämä johtui siitä, että Collidereita tuli liikaa kentälle.

Yksiköiden tasapainotus tuottaa aina pieniä ongelmia. Tällä kertaa ennen tasapainotusta jousiampuja-yksikkö oli aivan liian ylivoimainen. Se johtui siitä, että se ampui suhteellisen kauaksi ja teki viholliseen paljon vahinkoa. Sitä tilaamalla minkä tahansa maan voi läpäistä ilman suuria ongelmia. Tästä syystä jouduin huonontamaan jousimiehiä.

6 PELAAJAN OHJELMOIMINEN

6.1 Rakentaminen

```
37      public GameObject building1;
38      public GameObject building2;
39      public GameObject building3;
40
```

OHJELMA 5. Villagerin viittaukset 3 rakennukseen.

Pelaajan rakentaminen on tehty niin, että villagerin omassa luokassa on viittaus kolmeen GameObjectiin, jotka ovat rakennuksia (ohjelma 5). Rakennukset sisältävät rakennuksen mallin, rakennusluokan ja neljä spawnauspaikkaa. Kuitenkin joissain tehtävissä olen halunnut kieltää rakentamisen. Niissä GUI:n piirtävä luokka huolehtii siitä, ettei pelaajalle tule mahdollisuutta rakentaa rakennuksia (eli otetaan pois päältä rakennusnappulat). Muuten pelaajalle näkyy kolmen rakennuksen nappulat (kuva 10).

KUVA 10. Pelin alapalkki työläinen valittuna

```
444     void OnTriggerStay(Collider other) {
445         if(movedbuildinlocation)
446         {
447             //Debug.Log("Cannot build here");
448             messageTimer = 3;
449             canbuild = false;
450             movedbuildinlocation = false;
451         }
452     }
453 }
```

OHJELMA 6. Alueen tarkistus

Kun pelaaja painaa nappulaa, niin tarkastetaan onko kyseiseen rakennukseen tarpeeksi resursseja. Jos on, niin rakennus pistetään tulemaan ja resurssit vähenevät. GUI-luokka hoitaa sen että, kun pelaaja painaa haluamaansa kohtaa, niin alue tarkastetaan (ohjelma

6). Jos alueelle voi rakentaa, niin silloin `Instantiate`-käskyllä rakennus luodaan kyseiseen kohtaan. Jos alueelle ei voida jostain syystä rakentaa, niin siitä ilmoitetaan pienellä huomautuksella pelaajalle, niin kuin kuva 11 näyttää.

KUVA 11. Huomautus, ettei voi rakentaa

6.2 Yksiköiden liikuttaminen

```
245 void Selected(float[] pos)
246     {
247     if(pos.Length == 4)
248         {
249         if( (pos[0] > this.transform.position.x && pos[1] < this.transform.position.x)
250             && (pos[2] > this.transform.position.z && pos[3] < this.transform.position.z))
251                 {
252                     isSelected = true;
253                     ring.renderer.material = sel;
254                     manager.manyChoosen++;
255                     //Debug.Log("True");
256                 }
257
258             }
259         }
260 }
```

OHJELMA 7. Yksikkö tarkistaa paikkansa

Liikuttaminen on ohjelmoitu niin, että yksiköt tulee valita ennen kuin niitä voidaan liikuttaa. Valinta tapahtuu TargetMover-luokalla, joka kuuntelee hiirtä. Kun painaa hiiren vasenta nappia pohjassa, niin ruutuun tulee valinta-alue, jonka sisällä olevat yksiköt tulevat valituiksi (ohjelma 7). Luokka siis lähettää hiiren alkukohdan ja loppukohdan `BroadcastMessage`-metodilla kaikille pelaajan yksiköille. Jolloin yksikkö tarkistaa oman paikkansa suhteessa hiiren aloitus- ja lopetuskohtaan. Ohjelman pätkässä näemme kuinka tarkistus on toteutettu. Kun ne ovat valinta-alueen sisällä, niin bool-valittu muuttuu true-arvoon. Tämän jälkeen yksiköt ovat komennettavissa. (Input. 2013, BroadCastMessage, 2013)

```
void OnGUI()
{
    if(isSelected)
    {
        if(Input.GetMouseButton(1)){

            mover.target = target;
            mover.noattack = true;

        }
    }

}
```

OHJELMA 8. Yksikkö kuuntelee hiirtä

TargetMover liikuttaa kohtaa, johon yksiköiden pitää mennä kuuntelemalla, mihin kohtaan hiiren oikeaa näppäintä on painettu maailmassa. Yksiköt, jotka ovat valittu, kuuntelevat, milloin painetaan hiiren oikeaa näppäintä (Ohjelma 8). Kun painaminen tapahtuu, niin yksiköt antavat pathfindingille käskyn liikkua kohtaan, jonka TargetMover on määrännyt. Sen jälkeen pathfinding etsii polun ja lähtee liikkeelle.

6.3 Resurssienhallinta

Joka kentän alussa luokka RecourseInitialize etsii GameManager-luokan kentältä `GameObject.Find("Gamemanager")` -metodilla, jonka jälkeen se etsii luokan `gameManager.GetComponent<GameManager>()` -metodilla. Kun se on löydetty, se laittaa sille määrätyt resurssit suoraan GameManagerin pelaajan resursseiksi `manager.puu = Initpuu`. Jos bonuskysymykseen on vastannut oikein, niin tähän vielä lisätään bonuksena saadut resurssit samalla tavalla, luokkana vain toimii Bonus. (Unity Script Reference. 2013)

KUVA 12. Pelaajan resurssit alapalkista ja populaatiolimit

Muuten pelaaja joutuu huolehtimaan omasta resurssien tuotannosta villagereilla. Villagereiden tehtävänä on siis kerätä resursseja pelaajalle ja pelaajan pitää huolehtia, että niillä on jotain kerättävää. Kuvassa 12 näytetään, kuinka alapalkissa pelaajalle näkyy omat resurssit ja populaation tilanne.

Jokainen yksikkö ja rakennus maksaa pelissä resursseja. Kun tilaa yksikön tai rakentaa rakennuksen, GameManagerista vähennetään vastaava määrä resursseja. Yksiköillä on tämän lisäksi vielä piirre, että jos populaatio on täynnä, niin yksikköä ei voi tilata.

6.4 Sotiminen

```
public AiUnit target = null;
public List<AiUnit> enemyList = new List<AiUnit>();

void OnTriggerEnter(Collider other) {
        target = null;
        target = other.gameObject.GetComponent<AiUnit>();
        if(target != null)
        {
            //Debug.Log("Unit sisaa fiting");
            enemyList.Add(target);
        }
}

void OnTriggerExit(Collider other) {
    //Debug.Log("Field out");
    target = null;
    target = other.gameObject.GetComponent<AiUnit>();
    if(enemyList.Count != 0)
    {
            //Debug.Log("Unit exit fiting");
            enemyList.Remove(target);
    }
}

public void destroy(AiUnit unit)
{
    enemyList.Remove(unit);
}
```

OHJELMA 9. FieldOfView-luokan koodi

Toinen yksikössä oleva Triggeri on isompi ja kertoo ns. yksikön näköalueen. Toinen taas kertoo, mille alueelle taistelu kyky ylettyy. Kun näköalueelle tulee vihollinen, niin luokka huomioi sen automaattisesti ja pistää yksikön lähtemään aggressiivisesti vihollista kohti. Vasta kun kyseinen yksikkö tulee taistelu-Triggeriin, niin tila vaihtuu attackiin. Yksiköiden taistelualueessa ja näkemisessä on hyödynnetty layereitä niin, etteivät näkö ja taistelu-Triggerit tee törmäystarkastuksia muun kuin yksikkö-layerin kanssa. (ohjelma 9) (Collider. 2013)

Näistä Trigger Collidereista pitää huolen kaksi luokkaa: FieldOfView ja FitingRange. Kumpikin kuuntelee alueelle saapuvia Colliderin sisältäviä GameObjecteja, jonka jälkeen tarkastetaan, onko siinä vastapuolen yksikköluokkaa kiinni. Jos on, se tallennetaan Listiin (ohjelma 9). Kun joku Collider poistuu alueelta, tapahtuu sama tarkastus, mutta toisinpäin, eli jos toisen puolen yksikkö on ollut alueella, niin poistetaan se Lististä. (Collider. 2013)

Taistelussa pyörii hyökkäysanimaatio, joka kestää sekunnin. Aina, kun animaatio loppuu, se laitetaan pyörimään uudelleen ja tehdään viholliseen vahinkoa oman attack-arvon avulla. Yksiköllä on taisteluun myös kaksi int-arvoa, jotka kertovat mikä yksikkötyyppi on kyseessä ja mille se tekee paremmin vahinkoa. Tyypit ja lisävahinkotyypit ovat: 0 = ei bonusta, 1 = infranty, 2 = archery, 3 = cavallery. Toinen alijärjestelmä on se, että yksiköllä on pieni mahdollisuus tehdä joko kriittinen osuma tai heikko osuma. Tämä järjestelmä pyörii täysin unityn oman randomin päällä. Random arvo arvotaan 0-100 väliltä ja siinä on kumpaankin vahinko tyyppiin 10 prosentin mahdollisuudet. Muulloin tehdään normaali vahinko.

```
229     public void getDamage(int EnemyAttack, int bonus)
230     {
231         int nowdefence =Random.Range(0, defence);
232
233         if(nowdefence > EnemyAttack) EnemyAttack = EnemyAttack / 2;
234
235         if(bonus == type)
236         {
237         life -= (EnemyAttack + 5);
238         }
239         else
240         {
241         life -= EnemyAttack;
242         }
243     }
```

OHJELMA 10. getDamage-metodi

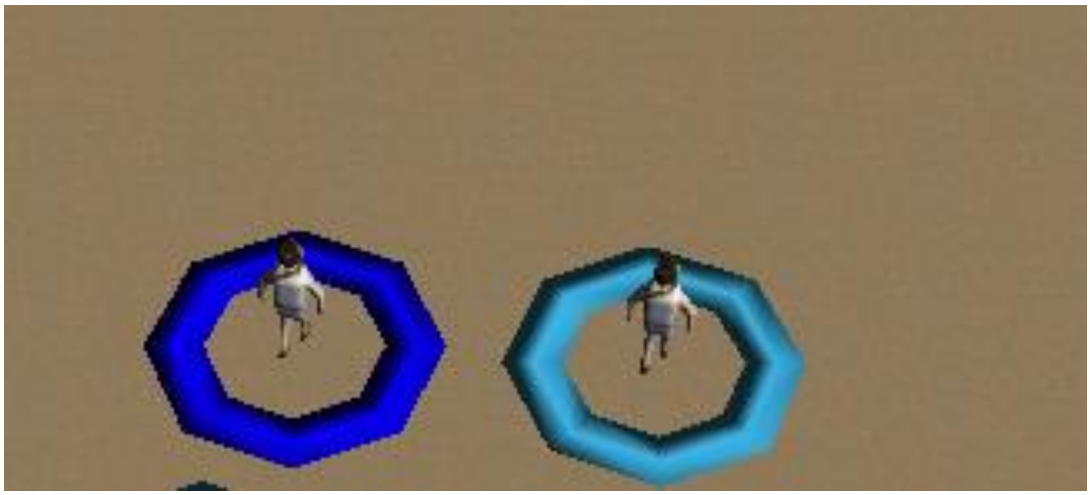
Kun on katsottu yksikön tekemä osuma, vihollisen luokasta kutsutaan metodia `getDamage(damage, bonus)`. Metodissa yksiköllä on pieni mahdollisuus puolittaa vihollisen tekemä vahinko niin, että defence-arvo randomilla laitetaan välille 0 ja defence. Jos defencen arvo on isompi kuin hyökkäyksen arvo, niin hyökkäys puolitetaan. Tämän jälkeen verrataan omaa yksikköluokkaa siihen, mihin toinen yksikkö tekee enemmän vahinkoa (bonusarvo). Jos tyypit ovat samat, damage lisääntyy viidellä ja lopullinen arvo vähennetään elämästä. (Ohjelma 10)

6.5 Ongelmat

Taistelujärjestelmä on koodattu kertaalleen jo uudestaan, sillä suurissa laumoissa taistelu alkoi vaikeutua. Ensiksi taistelu tehtiin mahdollisimman kevyeksi niin, että sotilas otti vain yhden kohteen ja lukitsi sen. Tällöin ei käytetty vielä listaa. Tämä osoittautui

kuitenkin suurissa laumoissa riittämättömäksi ratkaisuksi. Tästä johtuen siirryttiin lista-järjestelyyn. Kun vihollisyksikkö tulee alueelle, niin silloin se tallennetaan listaan ja kun se poistuu alueelta, niin se poistetaan listasta. Muita tapoja listasta poistumiseen on yksikön tuhoutuminen.

Yksiköiden erottumisessa oli alussa myös suuria ongelmia, koska tekstuuriin piirretyt pienet siniset merkit eivät riittäneet omien yksiköiden tunnistamiseen. Tähän ratkaisuksi tuli se, että omien yksiköiden alas laitetaan pieni ympyrä sinisellä värillä. Loppujen lopuksi päädyin myös siihen, että valitut yksiköt ovat tummemman sinisellä kuin valitsemattomat. Nämä ratkaisut lähtivät siitä, että pelaajalle on selkeää, ketkä ovat omia ja ketkä on valittu. (Kuva 13)



KUVA 13. Valittu ja valitsematon yksikkö

Kokeilin alussa myös paljon ominaisuuksia, jotka eivät toimineet tietyistä syistä. Näistä harmillisin poisjäänti oli taistelun jälkeen liikkuminen. Yritin tallentaa paikan, johon yksikkö oli menossa ennen taistelua, mutta suurten ongelmien jälkeen kuoppasin ominaisuuden. Ominaisuus aiheutti muun muassa valitsemattomien yksiköiden liikkumista väärään paikkaan ja muutenkin outoa käyttäytymistä yksiköiltä.

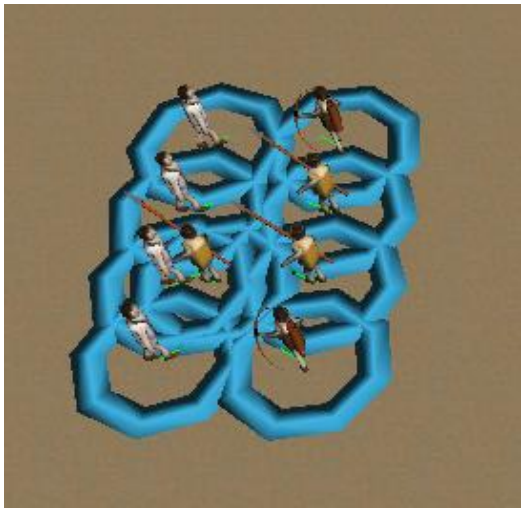
Villagerille kävi myös hassusti, kun jotain mallissa, tai jossain muualla, hajosi. Se aiheutti työläisen nykivää kävelyä ja ihmeellistä pyörimistä. Villager oli siis rakennettava alusta asti uudelleen. Uudelleenrakennuksen jälkeen se toimi jälleen ilman ongelmia.

Jossain vaiheessa kokeilin myös hevosyksiköiden keskikohdan muuttamista muualle kuin yksikön keskelle. Tämä johtui siitä, että hevosyksiköt käyttäytyivät tietyissä tilan-

teissa hieman oudosti. Tämä ei kuitenkaan ollut ratkaisu ongelmaan, vaan aiheutti samanlaisen käyttäytymisen lisäksi runsaan tärinän melkein koko ajan.

Testasin myös tiettyihin pathfinding-ongelmiin Colliderien siirtämistä yksiköillä hieman taaksepäin. Se aiheutti kuitenkin yksiköissä joskus täydellistä pyörimistä ympäri, eikä auttanut oman varjonsa pakenemiseen. Tämä siirto kävi lopullisesti turhaksi huomattuani, että yksiköiden päivitys joudutaan ottamaan kokonaan pois päältä.

Loin ominaisuuden, jossa pelaajan käskiessä useamman kuin yhden yksikön liikkumaan paikasta toiseen joukot muodostavat hassunnäköisen pallon. Tämä tarkoittaa, että yksiköt menivät vain keskikohdan ympärille oudosti. Muutin tätä kuitenkin järkevämpään suuntaan niin, että yksiköt muodostavat jonkinnäköisen neliön. Tämä tuntui järkevämmältä kuin pallomuodostelma, koska joukot ovat silloin edes jonkinlaisessa muodostelmassa. (kuva 14)



KUVA 14. Joukkojen muodostelma

7 TEKOÄLYN OHJELMOINTI

7.1 Tekoälyn toiminta

Tekoäly on pelissä kohtalaisen yksinkertainen, sillä pelin kehitysaika kävi vähiin, eikä tarvittavassa ajassa saatu tehtyä älykästä tekoälyä.

7.1.1 Rakentaminen

Tekoälylle mallinnettiin kyllä kaikki samat rakennukset kuin pelaajalle, mutta valitettavasti pelin kehitysajasta johtuen jouduin luopumaan tietokoneen rakentamisesta. Aika vain ei riittänyt niin hienon ominaisuuden tekemiseen. Joissain maissa kylläkin malleja esiintyy, mutta ainoastaan grafiikkana.

Tietokone ei myöskään kouluta yksiköitä samasta syystä. Ominaisuuteen olisi helposti tuhlautunut paljon aikaa ja siksi sen tekeminen ei ollut mahdollista. Suurimmaksi ongelmaksi tuli se, millä logiikalla se olisi tilannut yksiköitä ja missä tilanteissa. Ettei olisi aina käynyt niin, että samanlainen lauma olisi tullut hyökkäykseen.

7.1.2 Liikkuminen

Jos tietokoneen yksiköt ovat aggressiivisia, niin silloin kenttään on piilotettu paikka, jonne ne yrittävät. Tämä paikka on aina keskellä pelaajan kylää, jotta aggressiiviset joukot pyrkivät aina valtaamaan sen. Tietokoneen yksiköissä oli kiinni AiUnit-luokka, jonka teköäly toimii liikkeessä eri tavalla kuin pelaajan yksiköt osittain.

Kun yksikkö huomaa olevansa aggressiivinen, se etsii paikan minne mennä `GameObject.Find("EnemyTarget").transform`-metodilla. Kun se löytää paikan, niin tekoäly antaa käskyn liikkua paikkaan kerran. (Unity Script Reference. 2013)

Liikkumista voi kuitenkin häiritä se, että alueelle tulee pelaajan yksikkö, jolloin kohde vaihtuu siihen ja taistelu alkaa. Jos tekoälyn yksikkö selviää taistelusta hengissä, niin se antaa uudelleen käskyn liikkua paikkaan.

7.1.3 Resurssienhallinta

Tarkoituksena oli luoda tietokoneelle oikeaa resurssienhallintaa ja villagereilla keräystä, mutta valitettavasti niin kuin suurimmissa osissa muitakin tietokoneen ominaisuuksia, niin tämäkin olisi vienyt aikaa ja sitä ei ollut tarpeeksi kyseisen ominaisuuden kehittämiseen.

7.1.4 Sotiminen

Taistelu toimii tietokoneella samalla tavalla kuin pelaajallakin. On kaksi Trigger Collideria, jotka ovat näkökenttä ja taistelualue. Niistä kirjaa pitävät luokat AiFieldOfView ja AiFitingRange. Luokat vastaavat täysin pelaajan yksiköiltä löytyviä luokkia FieldOfView ja FitingRange. Erot tulevat siinä, että missä Ai-alkuiset etsivät pelaajan yksiköitä, niin pelaajan luokat etsivät tietokoneen sotilaita. (Collider. 2013)

Muutenkin taistelu toimii samalla tavalla kuin pelaajalla. Ensiksi satunnaisluvulla katsotaan, tuleeko kriittistä osumaa, jonka jälkeen kutsutaan getDamage-metodia ja annetaan damage ja bonus. Tämän jälkeen pelaajan yksikkö tekee getDamage-metodissa samat asiat kuin tietokonekin.

Ainoa poikkeus tietokoneen taistelussa on maassa, jossa Daavid kohtaa Goljatin. Goljattiin nimittäin ei voi kukaan muu kuin Daavid tehdä vahinkoa, joten Daavid on pakko saada kohtaamaan Goljat. Muuten on mahdotonta päästä maata läpi. Näiden yksiköiden kohtaamisessa Daavid kutsuu Goljatilta `getDavidDamage()`-metodia, jolla Goljat kuolee ensimmäisestä osumasta.

7.2 Ongelmat

Tekoälyn liikkuminen tuotti alussa suuria ongelmia, koska tein alussa sen niin, että se antoi paikan, johon liikkua joka framessa, kun se ei ollut menossa kohti taistelua tai taistelussa. Kun tekoälyn yksiköitä alkoi olla paljon kentällä joka sekunti kohteen lähettäminen alkoi rasittaa ja aiheuttaa häiriöitä pahtfindingissa. Joukot saattoivat yhtäkkiä lähteä väärään suuntaan ja liikkeestä tuli edestakaisin menevää. Tämä korjaantui sillä, että se lähettää paikan, johon mennä vain kerran ja silloin, kun tarvitaan. Tarvittavat kohdat olivat skenaarion aloitus ja taistelun jälkeen.

Suurin osa ongelmista ei esiintynyt tekoälyllä, koska ne ratkaistiin pelaajan ongelmina ja tulivat valmiina ratkaisuna tekoälylle. Toinen asia varmasti oli ajan riittämättömyys hienon tekoälyn tekemiseen.

8 TEHTÄVIEN RAKENTAMINEN

Suunnitelmien mukaan peliin tulee seitsemän erilaista skenaariota. Nämä kaikki saatiin toteutettua peliin. Skenaariot sisältävät erilaisia tehtäviä, jotka koostuvat Vanhan Testamentin tarinoista.

8.1 Tavoitteiden asettaminen

Suurin osa tavoitteista ohjelmoitiin peliin niin, että kun jokin asia tapahtuu, laitan boolmuuttujan true-arvoon ja pelaajalle kerrotaan, että yksi tavoite suoritettu.

8.1.1 Tavoitteet yleisesti

Pelissä löytyy tehtävät kerää, vie joukot johonkin, tuhoa vihollinen ja puolusta jotain omaa paikkaa tietyn ajan verran. Tehtävät myös vaikeutuvat hieman pelin edetessä. Pari ensimmäistä tehtävää on kohtalaisen helppoja, neljä seuraavaa keskivaikeita ja viimeinen tehtävä ennen loppuskeneä on tehty tahallaan hieman vaikeaksi.

Ensimmäisessä tehtävässä on tarkoitus viedä vain joukot omaan kylään ja kerätä ruokaa tarpeeksi. Toisessa tehtävässä taas on tarkoitus kiertää Jerikon muurit vastapäivään ja tuhota sisällä olevat tietokoneen joukot. Kolmannessa tehtävässä pitää tuhota kolme kylää, mutta peli loppuu yllätyksellisesti jo toisen kylän jälkeen. Tämä tehtävä kuvaa Saulin kääntymistä pois Jumalasta. Seuraava tehtävä on Daavidin ja Goljatin kohtaaminen. Tehtävässä on tarkoitus viedä Daavid ensin omaan kylään, jonka jälkeen lähteä kohtaamaan Goljat ja tuhoamaan Filistean joukot. Viides tehtävä on siitä erikoinen, että se on Israelin temppelin rakennus. Tehtävässä pitää kerätä tarpeeksi resursseja temppelin rakennukseen ja rakentaa temppeli valmiiksi. Tehtävän aikana alueella on ”rosvojoukkoja”, joita joskus tulee lisää. Kuudennessa tehtävässä on tarkoitus puolustaa omaa kaupunkia babylonialaisten hyökkäykseltä ja samalla vallata babylonialaisten kaupunki itselleen. Viimeinessä tehtävässä puolustetaan 15 minuuttia omaa aluetta roomalaisten valtavalta hyökkäykseltä. Jotta tehtävä ei olisi liian helppo, roomalaisia tulee paljon ja ne ovat pelin parhaat yksiköt yleiseltä tasoltaan. (liite 1)

8.1.2 Tavoitteiden ohjelmoiminen

```
35      void OnTriggerStay(Collider collision) {
36
37          Villager villager = collision.gameObject.GetComponent<Villager>();
38          Unit unit = collision.gameObject.GetComponent<Unit>();
39
40          if(unit != null || villager != null)
41          {
42              objective1 = true;
43          }
44
45      }
```

OHJELMA 11. TriggerStay ensimmäisessä tehtävässä

Ensimmäinen tehtävä sisältää yhden Trigger Colliderin, joka on asetettu kylän kohdalle. Kun pelaajan yksikkö tulee alueelle, niin silloin ensimmäinen tavoite muuttuu tehdyksi (ohjelma 11). Toinen tavoite on, että ruokaa kerätään enemmän kuin 3000. Tehtävä siis tarkkailee GameManageria, kunnes pelaajan ruoka-arvo on yli 3000, jonka jälkeen tulee voittoruutu. (Collider. 2013)

```
4   public class JerichoRound : MonoBehaviour {
5
6       public bool before = false;
7       public bool path = false;
8
9
10      void OnTriggerEnter(Collider collision) {
11
12          Unit unit = collision.gameObject.GetComponent<Unit>();
13          if(unit != null && before) path = true;
14
15      }
16
17  }
```

OHJELMA 12. JerichoRound-luokka

Toisessa tehtävässä kaupunkikierros on tehty luokalla JerichoRound ja Mission2. Mission2 pitää huolta siitä, että kierto tapahtuu oikein päin muuttamalla bool-muutujia JerichoRound-luokissa. GameObjecteja, joka sisältää luokan JerichoRound ja Trigger Collidereita on neljä kaupungin ympärillä, yksi jokaisessa nurkassa. JerichoRound kuuntelee bool-arvoa, joka kertoo, koska saa kuunnella Triggeriin tulevia pelaajan yksiköitä (ohjelma 12). Mission2 kuuntelee kaikkia neljää JerichoRound-luokkaa järjestyksessä. Kun ensimmäiseen tulee pelaajan yksikkö, bool muuttuu true-arvoksi ja seuraavalle JerichoRoundille annetaan lupa alkaa kuunnella Trigger Collideria. Kun kaikki neljä on kierretty, niin muurit sorrutetaan vain tuhoamalla ne. Tämän jälkeen Mission2 alkaa kuunnella GameManageria, koska tekoälyn populaatio on nolla. Kun se tapahtuu, niin tehtävä numero kaksi on suoritettu, ja näkyviin tulee lopputekstit.

```
16      // Update is called once per frame
17      void Update () {
18
19          timer -= Time.deltaTime;
20
21          if(timer < 0 && !destroyed)
22          {
23              bool findunit = false;
24              for(int i = 0; i < unit.Length;i++)
25              {
26                  if(unit[i] != null)
27                  {
28                      findunit = true;
29                      break;
30                  }
31
32
33              }
34
35              if(!findunit) destroyed = true;
36
37              timer = 5;
38
39              if(destroyed) missio.destroyVillage();
40          }
41      }
42  }
43
```

OHJELMA 13. Mis3Village-luokka tarkistaa taulukon viiden sekunnin välein

Kolmannessa tehtävässä tapahtumista pitää kirjaa Mission3-luokka. Tehtävässä on kolme kylää, joissa on kiinni Mis3Village-luokka. Mis3Villagessa on sisällä taulukko, johon on linkitetty kaupungissa olevat joukot. Kun kaikki taulukon arvot ovat null, kylä on tuhoutunut ja se kutsuu Mission3-metodia `destroyVillage()` (ohjelma 13). Mission3 laittaa seuraavaksi yhden tehtävän boolean arvoista trueksi. Tässä tehtävässä on se erikoisuus, että vaikka kyliä on kolme ja kaikki pitäisi tuhota, niin vain kahden tuhotun kylän jälkeen tulee voitto.

```
26      void Update () {
27
28          if(!objective2 && Goljat == null)
29          {
30              objective2 = true;
31              Destroy(wall);
32          }
33          if(manager.vcurrentpop < 1)
34          {
35              objective3 = true;
36          }
37
38          if(objective1 && objective2 && objective3)
39          {
40              manager.victory = true;
41              manager.inEndBriefing();
42
43          }
44          if(Daavid == null)
45          {
46              manager.inEndBriefing();
47          }
48      }
```

OHJELMA 14. Missio4 Update-metodi

Neljännen tehtävän tavoitteista pitää huolta Missio4-luokka. Ensimmäinen tehtävä on viedä Daavid omaan kylään. Tämä on toteutettu niin, että kylässä on Trigger Collider, joka kuuntelee koska alueelle tulee Unit-luokka. Kun Unit-luokan name-muuttuja on Daavid, niin ensimmäinen tavoite on tehty. Tämän jälkeen Daavidin ja Goljatin pitäisi kohdata taistelussa. Missio4:ssä on kumpaakin viite, kun Goljat-viite muuttuu null-arvoksi, seuraava tavoite on tehty. Viimeisestä tavoitteessa Missio4 katsoo, koska vihollisen populaatio on nolla, jolloin tehtävä on suoritettu ja tulee voitto. Mutta jos Daavid kuolee ja Mission4:n Daavidiin viittaava muuttuja muuttuu null-arvoksi, tehtävä merkitään hävityksi. (ohjelma 14)

Viidennessä tehtävässä tehtävän tavoitteiden suorittamista tarkkailee Mission5-luokka. Sen apuna toimii BuildTemple-luokka. BuildTemple-luokka hoitaa temppelin rakentumisen. Kun temppelin rakennusprosentti on 100, temppeli on valmis ja BuildTemple laittaa Mission5:n bool objective1 arvon trueksi. Kun bool-arvo muuttuu trueksi, tehtävä on suoritettu ja tulee lopun briefing. Temppelin rakennus kuitenkin vaatii resursseja: puuta 50, ruokaa 10, kultaa 20 ja kiveä 50 per prosentti. Rakennuksella on myös rakennusalue, jossa pitää olla vähintään yksi villager, jotta rakentaminen menee eteenpäin. Jokaista villageria kohtaan rakennus etenee yhden prosentin aina viiden sekunnin välein, jos on tarpeeksi resursseja.

Kuudennessa tehtävässä tehtävästä pitää huolta Mission6 ja Mission6Losing. Tehtävässähän on tarkoitus etsiä vihollisen kylä ja pitää sitä hallussa, kunnes pelaaja saa kaupungin haltuunsa. Vihollinen yrittää tehdä saman pelaajan kylälle. Kummatkin luokat tekevät saman asian, mutta vain toiselle puolueelle. Mission6 alkaa tiputtaa prosentti kerrallaan vihollisen kaupunkia, jos pelaajalla on yksiköitä viholliskaupungin alueella. Mission6Losing tekee saman omalle kaupungille, mutta ainoastaan jos vihollisia on alueella. Alueen kuuntelijana toimii Trigger Collider.

Seitsemännestä ja viimeisestä tehtävästä huolta pitää Mission7. Voittoaikana siinä toimii se, että 15 minuuttia pitää puolustaa omaa kylää. Jos yksikin vihollinen on kylänalueella, niin oma hallinta alkaa laskea ja kun hallinta menee nollaan, maan on hävinnyt. Mikäli kuitenkin pystyy puolustamaan omaa aluettaan 15 minuutin ajan ja oman kaupunki on pysynyt omassa hallinnassa, niin voittaa maan.

8.2 Ongelmat

Ongelmia oli monesti tehtävien koodaamisessa siinä vaiheessa, kun piti saada Trigge Collider ja yksiköt kohtaamaan oikealla tavalla ja oikeaan aikaan. Joskus Triggerit olivat liian pieniä ja eivät aiheuttaneet mitään. Joskus taas koodauksessa oli virheitä törmäystestailussa. Suurin osa ongelmista selvisi, kun hieman suurenti Trigger Collidereita ja laittoi Debug.Log()-metodiin tarkastuksia, että mitä milloinkin tapahtui. (Collider. 2013)

9 JOHTOPÄÄTÖKSET

Pelistä tuli sellainen kuin oli suunniteltu, mutta vielä ei ole päästy tavoitteeseen. Joitakin asioita jouduttiin jättämään pois, koska aika ei millään olisi riittänyt. Ominaisuuksien karsiminen jäi harmittamaan varsinkin, kun jouduin karsimaan tekoälyn ominaisuuksia. Peliä ei ole vielä julkaistu, joten tavoitteeseen ei päästy tämän opinnäytetyön aikana. Kuitenkin pelin kehitys jatkuu, kun peli julkaistaan, se tulee pääsemään tavoitteeseen.

Pelin kehitystä on päätetty jatkaa opinnäytetyön jälkeenkin. Grafiikkaan hankitaan 3D-osaaja ja muitakin ominaisuuksia hiotaan vielä eteenpäin. Neljän kuukauden kehitysaika osoittautui kahdella kehittäjällä lyhyeksi ajaksi. Tilaaja on tyytyväinen nykyiseen versioon, mutta haluaa kuitenkin hioa peliä vielä paremmaksi. Peliä jatkokehitettään niin, että siihen tulee näyttävämpi 3D-grafiikka ja ehkä uusia tehtäviä ja ominaisuuksia. Mielenkiintoista olisi myös kokeilla internetin välityksellä olevia 1 vs. 1 -matseja, joissa pelaaja saisi valita parin valtion välillä ja ottaa mittaa toisistaan skenaariossa.

Pelin kehitys onnistui kaikissa niissä alueissa, missä suurimmat odotukset olivatkin. Pelin saatiin toteutettua monia hyviä ominaisuuksia ja paljon pientä pelattavaa. Tehtävät kestävät noin 15 minuuttia, jota tavoiteltiinkin. Pelin 3D-grafiikka olisi voinut olla parempaa ja monipuolisempaa, mutta valitettavasti projektissa ei ollut yhtään mallintajaa, joten jouduin suurimmat karsimiset tekemään graafisesta puolesta. Pelissä oli myös paljon ongelmia, mutta niistä selvittiin kohtalaisen helposti pelitestauksella ja koodin muokkaamisella.

Unity on kehitysalustana hyvä. Pelimoottorista löytyy paljon hyviä ominaisuuksia, eikä kaikkea tarvitse toteuttaa itse. Unity sopii varsinkin pienille ja keskisuurille peliprojekteille mainiosti. Unityn editori on helppo oppia, ja moottorista on todella kattava materiaali Unityn omilla sivuilla. Kaikista paras koodauksen apuväline on tietenkin Scripting Reference, josta löytyy kaikki ominaisuudet, joita voi ohjelmoida. Unityn sivuilla on myös hyvät tutorialit, joilla pääsee alkuun pelimoottorin käytössä.

Lähteiden käyttö oli tässä opinnäytetyössä hieman vaikeaa. Kun on niin kattava materiaali Unityn sivuilla, niin Unityn koodausoppaisiin tutustuminen ei ollut niin kannattavaa. Siksi suurin osa koodaukseen liittyvistä lähteistä on Unityn sivuilta.

LÄHTEET

A* Pathfinding Documentation. 2013. Doxygen. Luettu 1.8.2013
http://arongranberg.com/astar/docs/

Adobe Flash. 2013. Flash. Luettu 30.11.2013
http://www.adobe.com/fi/products/flash.html

BroadCastMessage. 2013. Unity3D. Luettu 1.11.2013
https://docs.unity3d.com/Documentation/ScriptReference/Component.BroadcastMessage.html

Canvas. 2013. W3School. Luettu 20.1.2014
http://www.w3schools.com/html/html5_canvas.asp

Collider. 2013. Unity3D. Luettu 1.8.2013
http://docs.unity3d.com/Documentation/ScriptReference/Collider.html

Construct lisenssi. 2014. Construct 2. Luettu 20.2.2014
https://www.scirra.com/store/construct-2

Enums. 2013. Microsoft. Luettu 20.11.2013
http://msdn.microsoft.com/en-us/library/sbbt4032.aspx

GameEngine, 2008, GameCareerGUIde, Luettu 20.1.2014
http://www.gamecareerGUIde.com/features/529/what_is_a_game_.php?page=1

GameObject. 2013. Unity3D. Luettu 1.8.2013
http://docs.unity3d.com/Documentation/ScriptReference/GameObject.html

GUI. 2012. Unity3D. Luettu 20.11.2013
http://docs.unity3d.com/Documentation/Components/GUIScriptingGUIde.html

GUI Skin. 2013. Unity3D. Luettu 25.11.2013
http://docs.unity3d.com/Documentation/Components/class-GUISkin.html

Input. 2013. Unity3D. Luettu 28.10.2013
http://docs.unity3d.com/Documentation/ScriptReference/Input.html

McShaffry, M., Graham,D. 2013. Game Codin Complete. Boston : Course Technology.

PCvsConsole. 2012. DSUNLOCLED. Luettu 20.1.2014
http://www.digitalstormonline.com/unlocked/pc-gaming-vs-console-gaming-idnum25/

Random. 2013. Unity3D. Luettu 10.11.2013
http://docs.unity3d.com/Documentation/ScriptReference/Random.html

Layers. 2012. Unity3D. Luettu 20.11.2013
http://docs.unity3d.com/Documentation/Components/Layers.html

IgnoreCollision. 2013. Unity3D. Luettu 20.11.2013
http://docs.unity3d.com/Documentation/ScriptReference/Physics.IgnoreCollision.html

Terrain. 2013. Unity3D. Luettu 1.8.2013
http://docs.unity3d.com/Documentation/Manual/Terrains.html

Unity Script Reference. 2013. Unity3D. Luettu 29.5.2013.
http://docs.unity3d.com/Documentation/ScriptReference/

Unity 3D. 2013. Unity 3D. Luettu 29.5.2013.
http://unity3d.com/

Unreal. 2014. Unreal engine. Luettu 20.2.2014
http://www.unrealengine.com/udk/

W3School. 2013. HTML5. Luettu 1.12.2013
http://www.w3schools.com/html/html5_intro.asp

LIITTEET

Liite 1. GameDesing Document

Battles of Ancient Israel

Laatija: Heikki Vihervaara

Tämä pelikäsikirjoitus on dokumentti Raamatun mukaista Israelin historiaa noudattavan pelin suunnittelusta.

Sisällys

1 Esittely ... 52
1.1 Genre ... 52
1.2 Alusta ja vaatimukset ... 52
1.3 Kohdeyleisö ... 52
1.4 Keskeisimmät pelimekaniikat ... 52
2 Pelin yleisidea (High Concept) ... 53
3 Keskeisimmät pelielementit ... 54
3.1 Pelin yleinen eteneminen ... 54
3.2 Pelaajan motivointi ... 54
3.3 Pelin alijärjestelmät ... 55
3.3.1 Tekoäly ... 55
3.3.2 Gamemanager ... 55
3.3.3 UI (käyttöliittymä) ... 56
3.4 Resurssit ... 56
3.5 Maasto ... 56
3.6 Rakennukset ... 56
3.7 Yksiköt ... 56
3.8 Käyttöliittymä ja menut ... 57
3.9 Pääkamera ... 57
3.10 AI ... 57
4 Grafiikka ja käyttöliittymä suunnittelu ... 57
4.1 Menut ... 57
4.1.1 Main menu ... 58
4.1.2 Briefing ja debriefing ... 58
4.1.3 Pause menu ... 59
4.2 Käyttöliittymä ... 60
5 Kuvakaappaukset ... 60
6 Skenaariot ... 62
7 Yksiköiden ja Rakennusten attribuutit ... 63

1 Esittely

1.1 Genre

Battles of Ancient Israel-pelin genre tulee olemaan RTS (Real Time Strategy, tosiaikastrategia). Peli ottaa vaikutteita ja sen esikuvina käytetään Age of Empires ja Empire Earth-pelisarjoja. Pelin taustalla tulee tästä johtuen pyörimään yksinkertainen kivi-paperi-sakset-taistelujärjestelmä. Tässä taistelujärjestelmässä miekkamies aiheuttaa hyvin vahinkoa muulle jalkaväelle, mutta on heikko ratsuväkeä vastaan. Jousiampujat ovat puolestaan vahvoja etäältä, mutta heikkoja lähitaistelijoita.

1.2 Alusta ja vaatimukset

Pelin alustana toimii PC ja Mac -koneet, sillä sitä pelataan selaimessa. Pelin pystyy tarvittaessa kääntämään paikallisesti suoritettavaksi sovellukseksi. Selaimeen on ladattava ennen pelin aloittamista Unity-plugini. Jotta peli latautuisi kohtalaisen nopeasti olisi suositeltavaa, että nettiyhteyden latausnopeus on vähintään 1Mbit/s. Tietokoneessa tulisi myös olla tuplaydin prosessori ja kaksi gigatavua RAM-muistia. Näyttökortiksi kelpaa jopa emolevyyn integroidut näytönohjaimet.

1.3 Kohdeyleisö

Pelin kohdeyleisö on enimmäkseen nuoret pelaajat, joita pelin avulla yritetään saada kiinnostumaan raamatusta ja raamatun tapahtumista. Pelin tarkoituksena on myös evankelioida ei-kristittyjä.

1.4 Keskeisimmät pelimekaniikat

Pelin keskeisimmät pelimekaniikat ovat resurssien keräys ja hallinta, yksiköiden hallinta ja niiden vuorovaikutus muiden yksiköiden kanssa. Rakennusten rakentaminen ja niiden hyötykäyttäminen, sekä karttakohtaisten tavoitteiden suorittaminen skenaarion läpäisemiseksi ovat myös tärkeitä pelimekaniikkoja.

2 Pelin yleisidea (High Concept)

Pelin inspiraatio on saatu tosiaikastrategiapeleistä. Tämän genren isoimpiin pelisarjoihin kuuluvat esim. Age of Empires, Command & Conquer, Starcaft jne. Näitä pelejä on pyritty käyttämään esikuvina ja moni asia onkin kohtalaisen samalla tavalla toteutettu kuin niissä. Pelaaja siis kerää resursseja, rakentaa valtiotaan ja hallitsee omaa armeijaansa.

Battles of Ancient Israel-pelisessio on pelaajalla yleensä puolipitkä tai pitkä (15+ min). Tämä johtuu yksittäisen tehtävän pituudesta, joka on vähintään 15 min.

Pelin ideana on suorittaa erilaisia Raamattuun pohjautuvia tehtäviä, jotka sijoittuvat nykyisen Israelin alueelle ennen ajanlaskumme alkua.

Pelissä tulee olemaan muutama erilainen aikakausi, jotka rajoittavat yksiköiden kouluttamista. Suunniteltuja aikakausia on kolme (1: Israelin erämaavaellus, 2: Kaananin maan valtaus, 3: Israelin kuningas kunta ja pakkosiirtolaisuus, sekä sen jälkeinen aika). Peliin on myös suunniteltu seitsemästä erilaisesta skenaariota koostuva kampanja, jossa pelaajan tulee suorittaa erilaisia tehtäviä. Tähän peliversioon ei ole tulossa kuitenkaan skirmish-pelimuotoa, jossa pelaaja saisi pelata vapaammin.

Pelin grafiikka on hieman sarjakuvamaista ja räikeää, ei siis yritetä lähteä tekemään uutta Empire: Total waria, vaan tyydytään hieman kevyempään grafiikkaan ja hauskaan pelattavuuteen.

Pelaajalle esitellään Raamatun historian mukainen Israel ja pelaajan tarkoitus on johtaa Israelin kansaa erilaisissa tärkeissä tapahtumissa.

Pelissä tulee olemaan tallennusjärjestelmä, joka on kuin suoraan 80 tai 90 -luvulta. Haluttuun skenaarioon pääsee suoraan tietämällä sen salasanan. Salasanat tulevat olemaan Raamattuaiheisia. Jos kuitenkin tämä tallennusjärjestelmä osoittautuu riittämättömäksi, niin siitä yritetään tehdä hieman kehittyneempi.

Jokaisen skenaarion alussa on pieni bonusjärjestelmä, jossa pelaajalle esitellään seuraavaan skenaarioon liittyvä kysymys. Pelaajan vastatessa oikein hän ansaitsee bonuksen. Skenaarioihin olisi myös tarkoitus piilottaa ennustuksia Jeesuksesta. Pelin läpäisyn yhteydessä ruudulle ilmestyy risti ja jotain evankelioivaa tekstiä (esim. Mutta kaikista suurin taistelu ja voitto käytiin ristillä jonne meidän vapahtajamme kuoli ja nousi 3 päivänä henkiin). Nämä ominaisuudet ovat tarkoitettu evankeliointiin ja siihen, että pelaaja saataisiin kiinnostumaan Raamatusta.

3 Keskeisimmät pelielementit

3.1 Pelin yleinen eteneminen

Peli alkaa sillä, että pelaaja käynnistää pelin ja hänen eteensä avautuu main menu. Main menusta löytyy 4 nappia, jotka ovat: New Game, Password, Credits ja Info. New Game-painiketta painamalla pelaaja pääsee aloittamaan uuden pelin. Password-valikossa pelaaja pääsee jatkamaan haluamastaan skenaariosta salasanan avulla. Credits-osiossa esitellään tekijät ja tekijöiden avustajat. Infosta löytyy pelin pikaohjeet.

Pelaaja aloittaa joko uuden pelin tai jatkaa vanhaa kirjoittamalla salasanan. Sen jälkeen peli lataa itsensä skenaarioon ja antaa pelaajalle pienen briefingin päätavoitteiden kera. Pelaaja painettua ok-painiketta hänelle annetaan bonus-kysymys, johon oikein vastaamalla hän saa bonuksen. Bonus-kysymys liittyy aina skenaarion aiheeseen ja vastaus löytyy Raamatusta. Oikein vastannut pelaaja saa bonuksena esimerkiksi resursseja tai yksiköitä.

Pelaaja pelaa peliä lintuperspektiivistä. Sieltä hän ohjailee omia yksiköitään tekemään haluamiaan asioita. Pelaaja voi siis laittaa työläisyksiköt tekemään töitä (Esim. keräämään ruokaa, hakkaamaan puuta tai rakentamaan rakennuksia). Sotilasyksiköillä pelaaja taas yrittää tehdä annettuja tehtäviä ja tuhota vihollisen armeijan, joskus aikarajan kanssa.

Kun annetut tehtävät ovat suoritettu ja skenaario on voitettu, niin pelaajalle annetaan jonkinlainen statistiikka/debriefing taulukko/teksti, mistä näkee miten tehtävä on mennyt (statistiikka/debriefing-ruutu sisältää myös seuraavan skenaarion salasanan). Tämän jälkeen pelaaja viedään uuteen skenaarioon. Viimeisen skenaarion läpäistyään pelaajalle esitetään loppukohtaus.

3.2 Pelaajan motivointi

Pelissä panostetaan laatuun ja siihen, että kaikki toimii mahdollisimman sulavasti ja luo näin pelaajalle mukavan ja ihanan pelikokemuksen. Tämän olisi tarkoitus saada pelaaja nauttimaan pelistä niin paljon, että hän myöhemmin palaisi pelaamaan uudelleen. Pelin tarina rakennetaan siten, että pelaajan mielenkiinto säilyy ja hän haluaa pelata pelin loppuun. Pelimekaniikat pyritään luomaan ja tasapainottamaan siten, että kaikki on sopivan haastavaa ja näin nostattaa pelaajan mielenkiintoa. Grafiikan laatu tulee olemaan sellaista, ettei se häiritse pelikokemusta joskaan se ei välttämättä tuo mitään hirveästi lisääkään. Skenaarioista löytyy joistain paikoista aina pieniä paperikääröjä, jotka sisältävät ennustuksia Jeesuksesta.

Tämän olisi tarkoitus motivoida pelaajaa etsimään ja tutkimaan pelialueita, koska kääröt saattavat sisältää myös jonkun bonuksen.

3.3 Pelin alijärjestelmät

Tässä osiossa selitetään hieman peliin tulevista alijärjestelmistä. Selityksen piiriin kuuluu yleiskuva asiasta ja siitä miten sen voisi teknisesti toteuttaa.

3.3.1 Tekoäly

Tekoäly-alijärjestelmän tehtäviin kuuluu yksiköiden käyttäytymisen hallinta (pathfinding [polunlöytäminen], automaattinen ympäristöön reagoiminen, jne.) ja tietokonevastustajan käyttäytyminen.

3.3.1.1 Vihollisen tekoäly

Vihollisen tekoälyn toiminta riippuu hieman tehtävästä, mutta tietyt perustoiminnot ovat ja pysyvät. Tietokone pyrkii pitämään noin 20 työntekijää, jotka keräävät resursseja tietokoneen puolueelle. Perusyksiköiden kouluttaminen noudattaa tiettyä kaavaa. Sen taktiikka on pelaajaa kohtaan tehtävästä riippuen joko aggressiivinen tai puolustava.

Ainoa tapa millä tekoäly tulee huijaamaan pelaajaa on se, että työläiset ovat ilmaisia ja se ostaa niitä aina 20:een kappaleeseen asti. Viimeinen selviytymistaktiikka koneen tekoälyllä on, että se yrittää saada oman resurssien tuotannon pakkokäyntiin ostamalla ilmaiseksi työläisiä. Näillä keinoilla pyritään välttämään tietokonetta jäämästä ”jumiin”.

3.3.1.2 Yleinen tekoäly

Pathfinding toimii yksiköillä siten, että se pyrkii ottamaan huomioon maaston, muut yksiköt, rakennukset ja luonnon (puut, joet, kivet yms.). Näiden huomioon ottamisen jälkeen yksikkö valitsee mahdollisimman lyhyen reitin kiertäen eteen tulevat esteet.

Yksiköiden automaattinen ympäristöön reagoiminen tarkoittaa sitä, että kun vihollisyksikkö tulee tarpeeksi lähelle, niin ne alkavat taistelemaan keskenään.

3.3.2 Gamemanager

Pitää kirjaa resursseista, yksiköistä, rakennuksista, kentästä, triggereistä ja tehtävistä/tavoitteista.

Gamemanager tarkistaa aina kunkin skenaarion alussa että mitä tehtäviä/tavoitteita pelaajalla on ja seuraa niiden onnistumista. Jotkut pelaajan tekemät asiat voivat laukaista triggerin eli eventin, josta tapahtuu joko jotain hyvää tai pahaa pelaajalle.

Resurssien kirjanpito tapahtuu niin, että se seuraa pelaajan ja tietokoneen resurssien keräämistä ja tuhlaamista. Käyttöliittymä hyödyntää näitä tietoja.

Gamemanager pitää myös kirjaa yksiköistä ja rakennuksista siten, että yksiköitä ei saa olla pelaajalla tai tietokoneella yli 50. Se pitää kirjaa myös rakennuksista, joista pelaaja tai tietokone voi tilata yksiköitä. Gamemanager myös ilmoittaa pelaajalle, jos hänen rakennukset/yksiköt joutuvat hyökkäyksen kohteeksi. Se pitää myös kirjaa siitä, että kauanko jonkun rakennuksen/yksikön rakentamiseen menee aikaa. Tämän lisäksi se pitää kirjaa kaikkien yksiköiden sijainnista ja gamemanager pääseekin jokaiseen yksikköön koska tahansa kiinni.

3.3.3 UI (käyttöliittymä)

Käyttöliittymän on tarkoitus rekisteröidä napinpainallukset ja esittää pelaajalle pelin arvoja ymmärrettävässä muodossa.

Rakennuksilla on oma käyttöliittymäpalikka ja ne ovat rakennuksesta riippuvaisia.

Yksiköiden UI:ssa esitellään yksikön attribuutit ja jos on monta yksikköä valittuna, niin esitellään ensimmäisen yksikön attribuutit.

3.4 Resurssit

Niitä kerätään pelimaailmasta ja ne säilyvät kunkin osapuolen varastossa. Resursseja käytetään esimerkiksi yksiköiden ostoon ja rakennusten rakentamiseen.

3.5 Maasto

Maasto luodaan pelimaailmaan pelimoottorin omalla terrain-työkalulla ja siitä eivät yksiköt pääse putoamaan läpi. Maasto pitää sisällään pelialueen yleismuodot ja sen päälle rakennetaan kenttä eri peliobjekteista (puut, kivet, rakennukset jne). Maastolla on tietyt rajat, jonka ulkopuolelle eivät yksiköt tai pelaajan kamera pääse.

3.6 Rakennukset

Rakennuksia voi olla valmiina kentällä tai niitä voidaan rakentaa kyläläisillä. Omat rakennukset voi valita ja osasta voi tilata erilaisia yksiköitä. Vihollisen rakennukset voi tuhota ja vihollinen voi tuhota pelaajan rakennukset.

3.7 Yksiköt

Rakennuksista tilataan yksiköitä ja ne voi valita ja käskeä erilaisiin tehtäviin (Liikkuminen, resurssien keruu ja hyökkäys). Yksikkö valitaan painamalla hiiren vasempaa painiketta sen yläpuolella. Pelaaja voi valita monta yksikköä vetämällä valintalaatikon niiden päälle pitämällä hiiren vasenta painiketta pohjassa. Vapaut-

tamalla hiiren vasemman painikkeen pelaaja valitsee valintalaatikon alle jääneet yksiköt. Valitut ja valitsemattomat yksiköt erottaa niiden valintakiekon väristä.

3.8 Käyttöliittymä ja menut

Menuissa on asetuksia ja lisävalikoita. Valikoissa pystyy navigoimaan toisiin valikoihin ja menemään peliin tai pois pelistä.

Käyttöliittymä sisältää minimapin, resurssi tiedot ja tietoa valitusta yksiköstä tai rakennuksesta. Tämän lisäksi riippuen valitusta yksiköstä tai rakennuksesta näytöllä näkyy mitä eri toimintoja ne voivat suorittaa. Käyttöliittymässä näkyy myös skenaarion tavoitteet ylänurkassa.

3.9 Pääkamera ja Fog of War

Pelaaja näkee pelimaailman kamera-objektin kautta. Tätä kameraa pelaaja voi liikuttaa sulavasti WASD-näppäimillä ja zoomata hiiren rullalla. Pelaajan tulisi myös olla mahdollista asettaa kamera haluamaansa paikkaan minimapin avulla. Kameraa pystyy liikuttamaan ainoastaan skenaarioissa tiettyjen rajojen sisäpuolella. Maaston yläpuolella tulee olla pelaajan yksiköiden näkökentän ulkopuolella näköeste, jota genren pelit kutsuvat Fog of Wariksi. Tämä estää pelaajaa näkemästä sellaisiin paikkoihin, joissa hänellä ei ole yksiköitä. Fog of Warin tarkoituksena on kannustaa pelaajaa tutkimaan maastoa vihollisen, resurssien ja kätkettyjen asioiden toivossa.

3.10 AI

Pelaajan vastustaja, joka kerää omat resurssinsa, rakentaa omia yksiköitä & rakennuksia ja yrittää estää pelaajaa tehtävien suorittamisessa eri keinoin (hyökkää pelaajaa vastaan, yrittää puolustaa omia linjoja, jne).

4 Grafiikka ja käyttöliittymä suunnittelu

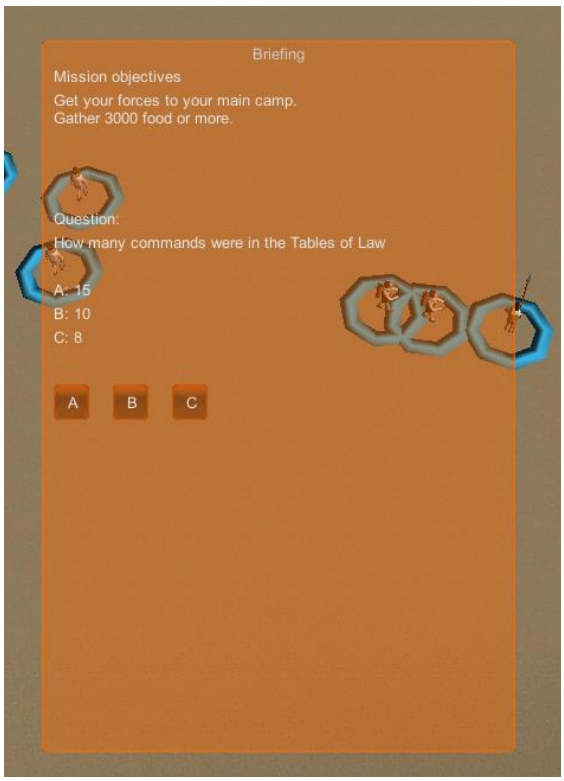
4.1 Menut

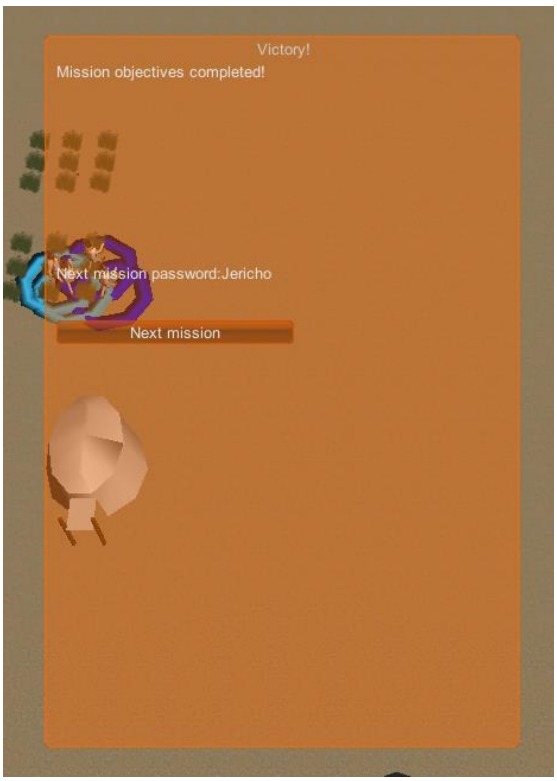
Pelissä löytyy main menu, briefing, debriefing ja pause menu. Kaikki menut noudattavat samaa tyyliä. Tyyli suunniteltiin olemaan melko puumainen. Tyylissä on myös läpinäkyvyyttä jokaisessa komponentissa, etteivät värit ole liian vahvoja.

4.1.1 Main menu

Main menuun suunniteltiin viisi nappia. New Game-napista pääsee aloittamaan täysin uuden pelin. Load Game:ssa tullee esiin tekstikenttä, johon voit laittaa skenaarion salasanan. Antamalla oikean salasanan pelaajaa pääsee pelaamaan kyseistä skenaariota. Info-napin takana löytyy pelin pikaohjeet ja linkki perusteellisimpiin ohjeisiin. Credits-osiossa näkyy pelin tekijät ja erityiskiitokset. Exit sulkee pelin. Menun taustalla käydään ikuista taistelua, jonka kamerakulma vaihtelee tasaisin väliajoin.

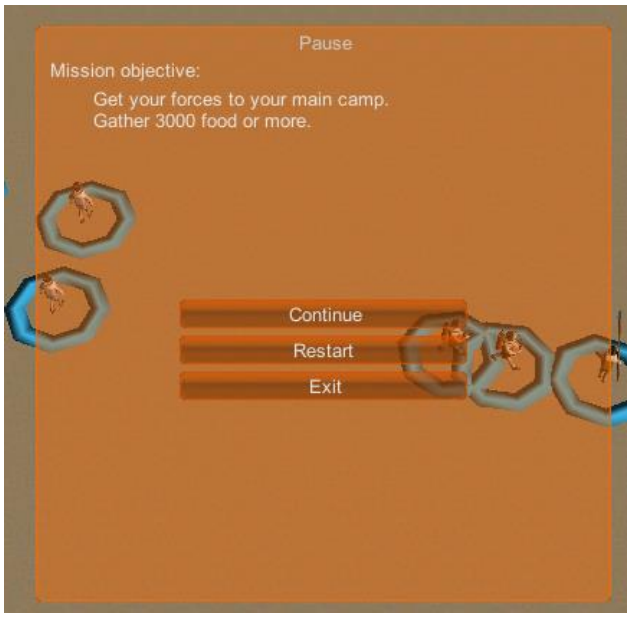
4.1.2 Briefing ja debriefing





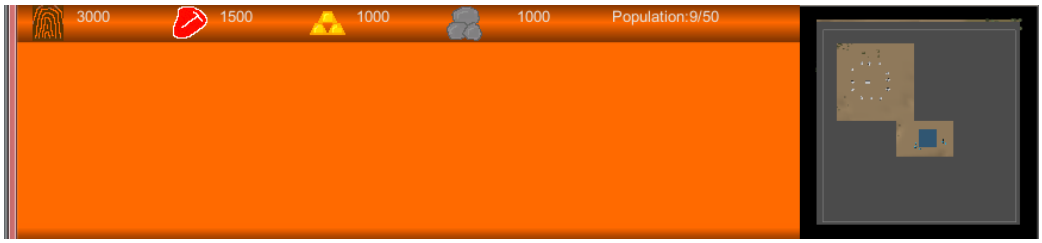
Briefing menussa näkyy tehtävän tavoitteet ja bonus-kysymys. Eteenpäin menusta pääsee vastaamalla kysymykseen. Kun kysymykseen on vastattu, näytetään pelaajalle että vastasiko hän oikein vai väärin. Continue–nappia painamalla pääsee jatkamaan peliin. Debriefing menusta selviää, että voittiko vai hävisikö pelaaja. Voiton johdosta pelaajalle annetaan seuraavan skenaarion salasana ja painike, josta pääsee kyseiseen skenaarioon. Pelaajan hävittyä skenaarion hänelle esitetään tehtävän tavoitteet ja painike skenaarion aloittamiseen alusta.

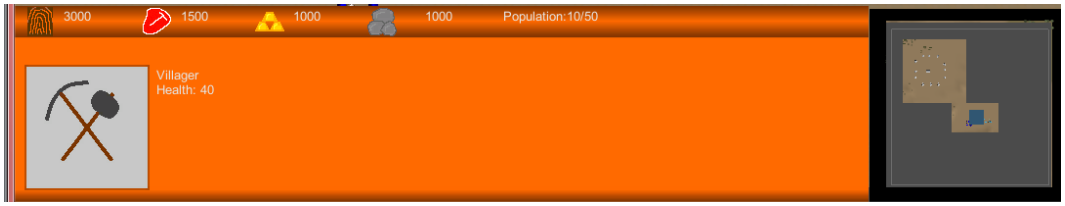
4.1.3 Pause menu

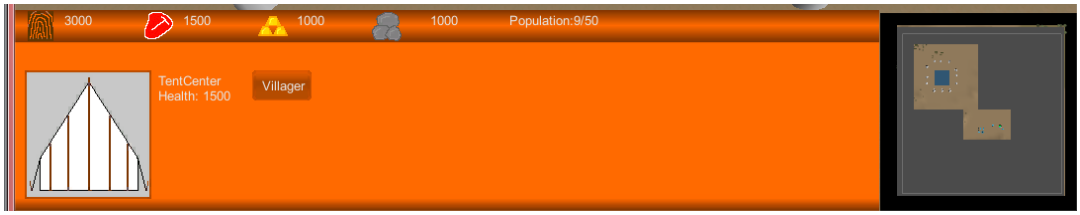


Pause-menussa näkyy skenaarion tavoitteet ja niiden alla nappulat, joista pääsee jatkamaan, aloittamaan tehtävän uudelleen tai poistumaan tehtävästä.

4.2 Käyttöliittymä



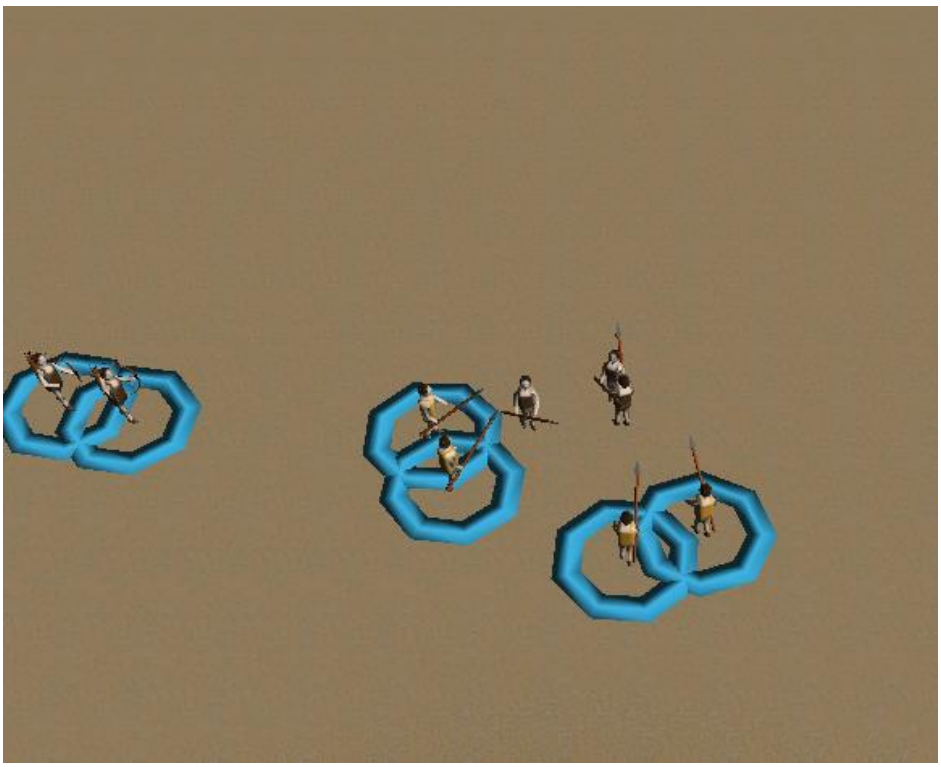




Käyttöliittymään kuuluu myös minimap, jonka kautta on mahdollista liikuttaa kameraa ja yksiköitä.

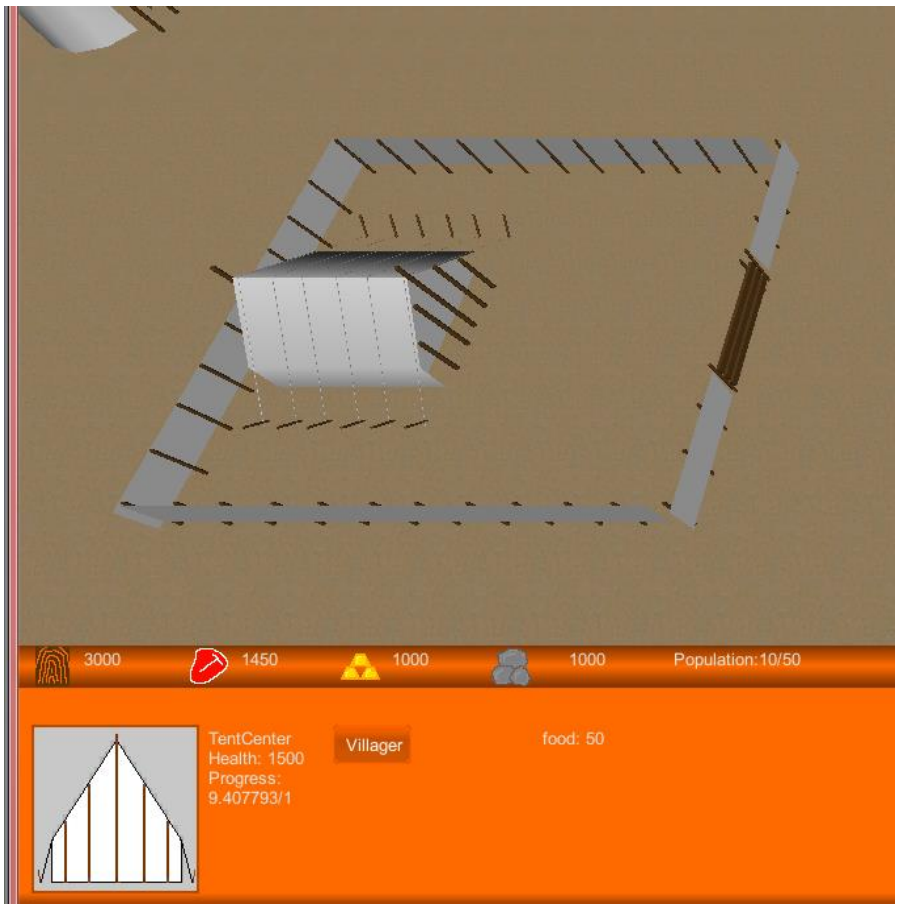
Käyttöliittymän ylälaidassa näkyy pelaajan resurssit ja populaatio. Resurssien alapuolella näkyy valittu yksikkö tai rakennus, sekä sen attribuutit. Jos valittuna on rakennus, tuodaan esille myös sieltä tilattavissa olevat yksiköt.

5 Kuvakaappaukset



Kuvassa näkyy pelin taistelua, jossa yksiköt käyvät automaattisesti toistensa kimppuun.

Kuvassa työläsyksikkö keräämässä ruokaa.



Kuvassa näkyy rakennus, joka on myös valittu. Rakennuksesta on juuri tilattu yksikkö ja sen valmistumisen aikalaskuri etenee kohti nollaa. Kuvassa näkyy myös yksikön koulutukseen vaadittavat resurssit.

6 Skenaariot

1. Tutoriaali

- Kertoo israelin erämaavaelluksesta
- Opetellaan pelin perusteet (liikkuminen, rakentaminen, resurssien keräys ja taistelu)
- Bonus kysymys (Mitä jumala tarkoitti kun sanoi että mooseksesta tulee israelin jumala ja aaronista hänen profeettansa, Montako käskyä oli lain tauluissa, Montako vitsausta iski egyptiin ennekuin israel vapautettiin orjuudesta?)

2. Jerikon muurit

- kertoo kun israel valtaa jerikon (jumala sortaa muurit)
  -> Yksikkö pitää viedä kerran jerikon ympäri
- Bonus kysymys (Kuinka monta kertaa israel kiersi jerikon muurit ja kuinka monessa päivässä)

3. Saul sotii Amalekia vastaan

- 3 kylää jotka pitäisi tuhota, mutta viimeisen kylän kohdalla otetaankin talteen lampaat ja lehmät yms.
- Samuel kertoo että saul on tehnyt väärin jumalan mielestä
- Bonus kysymys (Monesko Israelin kunigas Saul oli?, Ketkä olivat israelin 3 ensimmäistä kuningasta?)

4. Daavid ja Goljat
- Daavid on sankariyksikkö
- Vie Daavid farmilta kohtaamaan goljat, boonuksena aika limit
- Bonus kysymys(Mihinkä sukuun messias syntyy?)

5. Tempelin rakennus
- Tehtävänä on rakentaa herran temppeli(vaihtoehtoinen tehtävä on rakentaa temppeli tiettyyn aikaan ja kukistaa rosvojoukot)
- Bonus kysymys (Sodittiinko salomon aikaan yhtään sotaa, Montako vaimoa ja sivuvaimoa on salomolla, Koska israelin kuningas kunta oli rikkaimmillaan)

6. Israelin pakkosiirtolaisuuteen vieminen
- Puolusta tiettyä aluetta tietyn aikaa
- Bonus kysymys (Miksi israel vietiin pakkosiirtolaisuuteen tai jotain)

7. Rooma hyökkää israeliin
- Puolusta tiettyä aluetta tietyn aikaa ja menetä vain tietyn verran ykiköitä
- Bonus kysymys (Kun rooma valtasi israelin monenko vuoden päästä Jeesus syntyi tästä?)

7 Yksiköiden ja Rakennusten attribuutit

Rakennukset:

Päärakennus
Resurssit: (Puu)1000 ,(Ruoka)0 ,(Kivi) 250 ,(Kulta) 100
Elämä: 1500
Ominaisuudet: Kouluttaa työläisiä, (kerätyt resurssit saapuvat rakennukset).

Parakki
Resurssit: (Puu) 500,(Ruoka) 0,(Kivi) 0,(Kulta) 0
Elämä: 800
Ominaisuudet: Kouluttaa jalkaväkeä.

Talli
Resurssit: (Puu) 650,(Ruoka) 0,(Kivi) 100,(Kulta)0
Elämä: 1000
Ominaisuudet: Kouluttaa ratsuväkeä.

Rooma:
Miekkamies (kilpi+miekka+panssari)
Resurssit: (Puu) 25,(Ruoka) 100,(Kivi) 0,(Kulta) 50
Elämä: 120
Hyökkäys: 15
Puolustus: 10
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset rakennuksia ja muuta jalkaväkeä vastaan

Keihäsmies (kilpi+keihäs+panssari)
Resurssit: (Puu) 50,(Ruoka) 100,(Kivi) 0,(Kulta) 25
Elämä: 100

Hyökkäys: 10
Puolustus: 15
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset ratsuväkeä vastaan

Jousimies (Normi jousi+panssari)
Resurssit: (Puu) 100,(Ruoka) 60,(Kivi) 0,(Kulta) 35
Elämä: 80
Hyökkäys: 15
Puolustus: 5
Nopeus: Normi
Ominaisuudet: kaukotaistelu

Kevytratsumies (miekka+kilpi+panssari)
Resurssit: (Puu) 25,(Ruoka) 200,(Kivi) 0,(Kulta) 100
Elämä: 150
Hyökkäys: 20
Puolustus: 12
Nopeus: Nopea
Ominaisuudet: Lähitaistelu ja bonukset jalkaväkeä vastaan

Israel

Keihäsmies (Keihäs)
Resurssit: (Puu) 50,(Ruoka) 50,(Kivi) 0,(Kulta) 0
Elämä: 70
Hyökkäys: 8
Puolustus: 2
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset ratsuväkeä vastaan

Kivilinko
Resurssit: (Puu) 50,(Ruoka) 50,(Kivi) 5,(Kulta) 0
Elämä: 60
Hyökkäys: 6
Puolustus: 1
Nopeus: Normi
Ominaisuudet: kaukotaistelu

Miekkamies (kilpi+miekka)
Resurssit: (Puu) 10,(Ruoka) 100,(Kivi) 0,(Kulta) 40
Elämä: 100
Hyökkäys: 12
Puolustus: 8
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset jalkaväkeä vastaan

Keihäsmies (kilpi+keihäs)
Resurssit: (Puu) 50,(Ruoka) 100,(Kivi) 0,(Kulta) 20
Elämä: 90
Hyökkäys: 8
Puolustus: 10
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset ratsuväkeä vastaan

Jousimies (Normi jousi)
Resurssit: (Puu) 100,(Ruoka) 50,(Kivi) 0,(Kulta) 30
Elämä: 70
Hyökkäys: 12
Puolustus: 3
Nopeus: Normi
Ominaisuudet: kaukotaistelu

Hevostiedustelija(miekka)
Resurssit: (Puu) 0,(Ruoka) 150,(Kivi) 0,(Kulta) 10
Elämä: 100
Hyökkäys: 10
Puolustus: 6
Nopeus: Tosi Nopea
Ominaisuudet: Lähitaistelu

Kevytratsumies(miekka+kilpi (ei haarniskaa))
Resurssit: (Puu) 25,(Ruoka) 150,(Kivi) 0,(Kulta) 100
Elämä: 125
Hyökkäys: 15
Puolustus: 10
Nopeus: Nopea
Ominaisuudet: Lähitaistelu ja bonukset jalkaväkeä vastaan

Muut viholliset

Keihäsmies (Keihäs) (ei babylon)
Resurssit: (Puu) 50,(Ruoka) 50,(Kivi) 0,(Kulta) 0
Elämä: 60
Hyökkäys: 8
Puolustus: 2
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset ratsuväkeä vastaan

Kivilinko (ei babylon)
Resurssit: (Puu) 50,(Ruoka) 50,(Kivi) 5,(Kulta) 0
Elämä: 50
Hyökkäys: 6
Puolustus: 1
Nopeus: Normi
Ominaisuudet: kaukotaistelu

Miekkamies (kilpi+miekka+panssari) (Filistea)
Resurssit: (Puu) 25,(Ruoka) 100,(Kivi) 0,(Kulta) 50
Elämä: 110
Hyökkäys: 13
Puolustus: 11
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset jalkaväkeä vastaan

Miekkamies (kilpi+miekka) (babylon)
Resurssit: (Puu) 10,(Ruoka) 100,(Kivi) 0,(Kulta) 40
Elämä: 100

Hyökkäys: 14
Puolustus: 6
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset jalkaväkeä vastaan

Keihäsmies (kilpi+keihäs) (Babylon ja amalekia)
Resurssit: (Puu) 50,(Ruoka) 100,(Kivi) 0,(Kulta) 20
Elämä: 90
Hyökkäys: 9
Puolustus: 9
Nopeus: Normi
Ominaisuudet: Lähitaistelu, bonukset ratsuväkeä vastaan

Jousimies (Normi jousi) (kaikki)
Resurssit: (Puu) 100,(Ruoka) 50,(Kivi) 0,(Kulta) 30
Elämä: 70
Hyökkäys: 12
Puolustus: 3
Nopeus: Normi
Ominaisuudet: kaukotaistelu

Kevytratsumies (kaikki)
Resurssit: (Puu) 25,(Ruoka) 150,(Kivi) 0,(Kulta) 100
Elämä: 125
Hyökkäys: 15
Puolustus: 12
Nopeus: Nopea
Ominaisuudet: Lähitaistelu ja bonukset jalkaväkeä vastaan

Raskasratsumies (Amalekia)
Resurssit: (Puu) 50,(Ruoka) 250,(Kivi) 0,(Kulta) 150
Elämä: 150
Hyökkäys: 15
Puolustus: 15
Nopeus: Normaali ++
Ominaisuudet: Lähitaistelu ja bonukset jalkaväkeä vastaan

Sotavaunu (Babylon)
Resurssit: (Puu) 100,(Ruoka) 300,(Kivi) 0,(Kulta) 150
Elämä: 200
Hyökkäys: 10
Puolustus: 20
Nopeus: Nopea
Ominaisuudet: Lähitaistelu ja bonukset jalkaväkeä vastaan

Erikoiset:

Daavid
Elämä: 350
Hyökkäys: 15
Puolustus: 15
Nopeus: Normaali ++

Ominaisuudet: Lähitaistelu ja bonukset kaikkia vastaan, antaalähellä oleville taistelu bonuksen ja ainoana maailmassa pystyy erikois iskulla tappamaan goljatin

Goljat

Elämä: 2000
Hyökkäys: 50
Puolustus: 120
Nopeus: Normaali
Ominaisuudet: Erikoisuutena antaa multidamagea. Kuolee ainostaan daavidin erikois iskuun.