VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Ali Shadkami

GREEN PUZZLE CITY:

IOS MOBIILIPELI

Tekniikka
2017

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Ali Shadkami
Opinnäytetyön nimi	Green Puzzle City : iOS Mobiilipeli
Vuosi	2017
Kieli	suomi
Sivumäärä	60 + 1 liite
Ohjaaja	Jukka Matila

Opinnäytetyön tavoitteena oli luoda hyvin yksinkertainen logiikkapeli mobiililaitteille, alustavasti iOS-alustalle, kuten iPhonelle ja iPadille. Pelin aiheena olisi puhdas energia. Peligrafiikka olisi selkeä ja hieno. Pelattavuus olisi yksinkertaista, jotta peli sopisi kaikenikäiselle.

Pelin toteuttamiseen käytettiin Unity-pelimoottoria ja C#-ohjelmointikieltä. Pelimoottoriympäristö on minulle entuudestaan tuttua sillä olen aikaisemmin kehittänyt muutamia pelejä Unityllä.

Tämä on ensimmäinen logiikkapeli, jonka tulen toteuttamaan. Tärkeimmät osa-alueet, kuten grafiikan suunnittelu ja käyttöliittymät, tulen tekemään itse. Musiikin ja ääniefektit tulen ottamaan Unityn Asset Storesta, syy tähän on se, että ääniefekteihin ja musiikin luomiseen ei riitä aikaa.

Opinnäytetyön alussa käydään läpi mitä tekniikoita käytin, miten peli-idea syntyi ja kuinka lähdin tekemään tätä projektia. Käyn toteutusvaiheen osittain läpi ja kerron tärkeimmät osat pelin tekemisestä esittelemällä kuvia sekä käyttämiäni koodinpätkiä. Pelin haastavin osuus tulee olemaan pelidatan tallennus JSON-tiedostoon ja AES (Rijndael)-kryptaus.

Pelin koko toiminnallisuus esitetään pelattavan demon avulla.

Avainsanat Unity, C#, iOS, peliohjelmointi, mobiilipeli

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Ali Shadkami
Title	Green Puzzle City : iOS mobile game
Year	2017
Language	Finnish
Pages	60 + 1 Appendice
Name of Supervisor	Jukka Matila

The aim for my thesis was to create a very simple logic game for mobile devices, firstly to iOS-platform, such as iPhone and iPad. The subject of the game would be ”pure energy”. Game graphics should be clear and nice looking. Gameplay should be simple, so that the game could suit every agegroup.

I used Unity-game engine and C#-programming language to create the game. Game engine environment is familiar to me, because I have used Unity to create some games before.

This is the first logicgame, that I will be making. Parts of the game, such as graphics desing and user interface I will be creating myself, only the music and soundeffects I will download from Unity’s Asset Store. The reason for doing this is simply because I do not have the time to create my own sounds and music.

In the beginning of my thesis I will introduce the techniques which I used, how the gameidea was born and how I began working on this project. I will go through the executionprocess part by part and tell the most important parts of the making of the game by presenting pictures and parts of the code I used. The most challenging part will be to savet he gamedata to the JSON-file and the AES (Rijndael) crypting.

Whole functionality of the game will be displayed in a playable demoversion.

Keywords Unity, C#, iOS, game programming, mobile game

SISÄLLYS

TIIVISTELMÄ
1 JOHDANTO 9
2 MÄÄRITTELY 10
2.1 Vaatimusmäärittely 10
2.2 Toiminnallinen määrittely 11
2.3 Ideointi 13
2.4 Käyttötapauskaavio (User Case Diagram) 15
3 KÄYTETYT TEKNIIKAT 16
3.1 Unity 16
3.1.1 Unity Ads 17
3.2 IDE - ohjelmointiympäristö 18
3.2.1 Visual Studio Community 2015 18
3.2.2 MonoDevelop 18
3.2.3 Xcode 18
3.3 Käytetyt ohjelmointikielet 19
3.3.1 C# 19
3.3.2 JSON 19
3.4 MagicaVoxel 19
3.5 MagicaVoxel Qubicle To Unity 20
3.6 Inkscape – käyttöliittymän grafiikat 21
4 SUUNNITTELU JA PROTOTYYPPI 22
5 TOTEUTUS 24
5.1 Pelilogiikka 24
5.1.1 Unityn funktioiden suoritusjärjestys 24
5.1.2 Singleton-objektit 25
5.1.3 Pelimuodot 26
5.1.4 Luokkakaaviot 27
5.1.5 PlayerInputScript.cs 31
5.1.6 EventManagerit 33
5.1.7 ScriptableObject 34
5.2 Kenttien suunnittelu 36

5.3 Näkymät - Scenes 41
5.4 Pelidatan tallennus 45
5.4.1 JsonUtility 45
5.4.2 AES kryptaus (RijndaelManager) 46
5.5 Voxel To Unity 49
5.6 Pelimusiikki ja ääniefektit 50
5.7 UnityAds 50
5.7.1 UnityAds-kooditoteutus 51
6 TESTAUS 53
6.1 Koodin testaus editorissa 53
6.2 Unity Profiler 53
6.3 Unity Remote 5 54
6.4 Xcode Instrument 55
6.5 IOS-laite 57
7 JATKOKEHITYS 59
8 YHTEENVETO 60
LÄHTEET 61

KUVA- JA TAULUKKOLUETTELO

Kuva 1. Peliluonnos. 12
Kuva 2. Pulmapeliluonnos. 14
Kuva 3. Pelaajan käyttötapauskaavio 15
Kuva 4. Unity 5.6.0f3-editori 17
Kuva 5. MagicaVoxel 0.98.1 editori. 20
Kuva 6. Ei-optimoitu ja optimoitu voxel obj-tiedosto 21
Kuva 7. Inkscape-editori 21
Kuva 8. Main camera komponenttiasetukset 22
Kuva 9. Pelin prototyyppi 23
Kuva 10. Unity C# oletusskripti. 25
Kuva 11. Singleton objekti. 26
Kuva 12. Packages eli hakemistot, joissa kaikki luokat on luokiteltu 27
Kuva 13. ScriptableObject-luokat. 27
Kuva 14. Audioluokat. 28
Kuva 15. Data-luokat. 28
Kuva 16. Manager-luokat. 29
Kuva 17. Player-luokat 30
Kuva 18. Tile-luokat. 30
Kuva 19. Utility-luokat. 31
Kuva 20. TouchGameObject-funktio. 32
Kuva 21. TileMovement-funktio. 32
Kuva 22. Pelieventit 33
Kuva 23. Eventin lisäys AddListener-funktioon. 33
Kuva 24. Eventin kutsu PostNotification-funktioon. 33
Kuva 25. OnEvent-funktio. 34
Kuva 26. GAME_INPUT-eventin lisäys ja kutsu. 34
Kuva 27. ScriptableObjectin Audioevent abstrac-luokka. 35
Kuva 28. ScriptableObjectin SimpleAudioEvent-luokka. 35
Kuva 29. Uuden AudioEvent ScriptableObjectin luominen. 36
Kuva 30. SimpleAudioEvent-luokan referenssi. 36
Kuva 31. Ääniefektin toistaminen. 36

Kuva 32. Kaikki pelilaatat. 37
Kuva 33. Sininen kytkin. 38
Kuva 34. Punaisen kytkimen OnTriggerEnter-funktio. 39
Kuva 35. Tile.cs skripti. 39
Kuva 36. CheckforCurrentTileOrderToWin-funktio. 40
Kuva 37. MainMenu-näkymä. 41
Kuva 38. SelectLevels-näkymä. 41
Kuva 39. Star Challenge Levels-näkymä. 42
Kuva 40. Star Challenge Level - 1. 42
Kuva 41. Random Challenge-näkymä. 43
Kuva 42. Random Challenge Level – 1. 43
Kuva 43. Kenttä 1 suoritettu. 44
Kuva 44. Kenttä 1 epäonnistui. 44
Kuva 45. NormalLevelData-luokka. 45
Kuva 46. Pelidatan tallennus tiedostoon. 45
Kuva 47. Muutetaan JSON-data objektiksi. 46
Kuva 48. Key-muuttuja. 47
Kuva 49. Encrypt-funktio. 47
Kuva 50. Decrypt-funktio. 48
Kuva 51. Voxel To Unity-editori. 49
Kuva 52. Voxel To Unityn luoma hakemisto. 49
Kuva 53. Unity Ads-asetukset. 51
Kuva 54. UnityAds-koodin toteutus. 52
Kuva 55. Console-ikkunan virheilmoitus. 53
Kuva 56. Profiler aikajanat. 54
Kuva 57. Unity Remote 5:n Editor asetusvalikko. 55
Kuva 58. Instrument-editori. 56
Kuva 59. Instrument-aikajana. 56
Kuva 60. Unityn Build Settings-ikkuna. 57
Kuva 61. Unityn luoma Xcode-projekti. 58

Taulukko 1. Pelin vaatimukset 10

LYHENNELUETTELO

Unity	Unity Technologiesin kehittämä pelimoottori
C#	Csharp Microsoftin kehittämä ohjelmointikieli
JSON	JavaScript Object Notation, yksinkertainen avoimen standardin tiedoston välitysmuoto
FPS	Frames per Second / Kuvataajuus
Scene	Unity-editorin näkymä
iOS	Applen kehittämä käyttöjärjestelmä
Game Center	Applen kehittämä sosiaalinen online-peliverkko
Mono	Xamarinin kehittämä avoin lähdekoodi ohjelmistokehitysympäristö
.NET Framework	Microsoftin kehittämä ohjelmistokomponenttikirjasto
Xcode	Applen kehittämä ohjelmointiympäristö
Visual Studio	Microsoftin kehittämä ohjelmistokehitysympäristö
Plugin	Liitännäinen
App Store	Applen sovelluskauppa
UI	Käyttöliittymä
Instruments	Xcoden profiilityökalu

1 JOHDANTO

Tämän opinnäytetyön tarkoitus oli luoda yksinkertainen ja hauska logiikkapeli iOS-mobiililaitteille ja pelin aiheena on puhdas energia. Projektin tekemisessä käytettiin Unityn versiota 5.5.1f1, joka on Unity Technologiesin kehittämä pelimoottori. Syy siihen, miksi valitsin tämän opinnäytetyön aiheen, oli kova kiinnostukseni mobiilipeleihin. Mobiilipeliala on tällä hetkellä suurin pelialan osa-alue ja se kasvaa kovaa vauhtia.

Nykyään pelien tekeminen ei vaadi kovin isoja työtiimejä kuten ennen, vaan yksinkin on mahdollista toteuttaa hyviä pieniä pelejä ja ne voivat jopa menestyä yhtä hyvin kuin isojen pelialan yritysten pelit. Kun on käytössä laadukas työkalu, kuten Unity, niin kaikki on mahdollista. Tässä opinnäytetyössä käyn läpi työkalut, joita käytin pelin tekemiseen, kerron miten idea syntyi, kuinka suunnittelin pelin ja kerron myös pääpiirteittäin kuinka toteutin sen.

Peligrafiikan tekemiseen käytin 3D-mallinnustyökalua nimeltä MagicalVoxel, joka on voxel-grafiikkatyökalu. UI/käyttöliittymän tekemiseen käytin Inkscape-ohjelmaa, joka on vector-grafiikkatyökalu. Näitä työkaluja on helppo käyttää ja niillä saa erittäin laadukasta graafista jälkeä lyhyessä ajassa.

2 MÄÄRITTELY

2.1 Vaatimusmäärittely

Tässä luvussa käydään läpi pelin lähtökohtaiset vaatimukset. Eli ne asiat mitä pelin tulisi sisältää ja miten se tulee toimia. Vaatimukset ovat tärkeysjärjestyksessä taulukkossa eli normaalit vaatimukset (must have), odotetut vaatimukset (should have) ja ekstrat (nice to have).

Taulukko 1. Pelin vaatimukset

Prioriteetti	Vaatimukset
Normaalit vaatimukset (Must Have)	• pelin täytyy olla yksinkertainen • täytyy olla päävalikko-näkymä • täytyy olla kenttävalikko-näkymä • toimiva peliohje-näkymä • toimiva normal challenge-pelimuoto • peli pyörii iOS-alustalla 30 FPS • kosketus laattoihin on optimoitu • pelattavuus on hyvin optimoitu • sisältää 10 normaalia kenttää • sisältää 3 random kenttää • pelidatan tallennus JSON-tiedostoon, joka voidaan tallentaa iOS-laitteisiin • JSON-tiedoston kryptaus • UnityAds-integrointi

Odotetut vaatimukset (Should have)	• selkeä käyttöliittymä
	• valittavana Random Challenge-pelimuoto
	• valittavana Star Challenge-pelimuoto
	• tulee sisältää seuraavat Voxel-mallinnusgrafiikat
	  ○ aurinkopaneeli
	  ○ tuuliturbiini

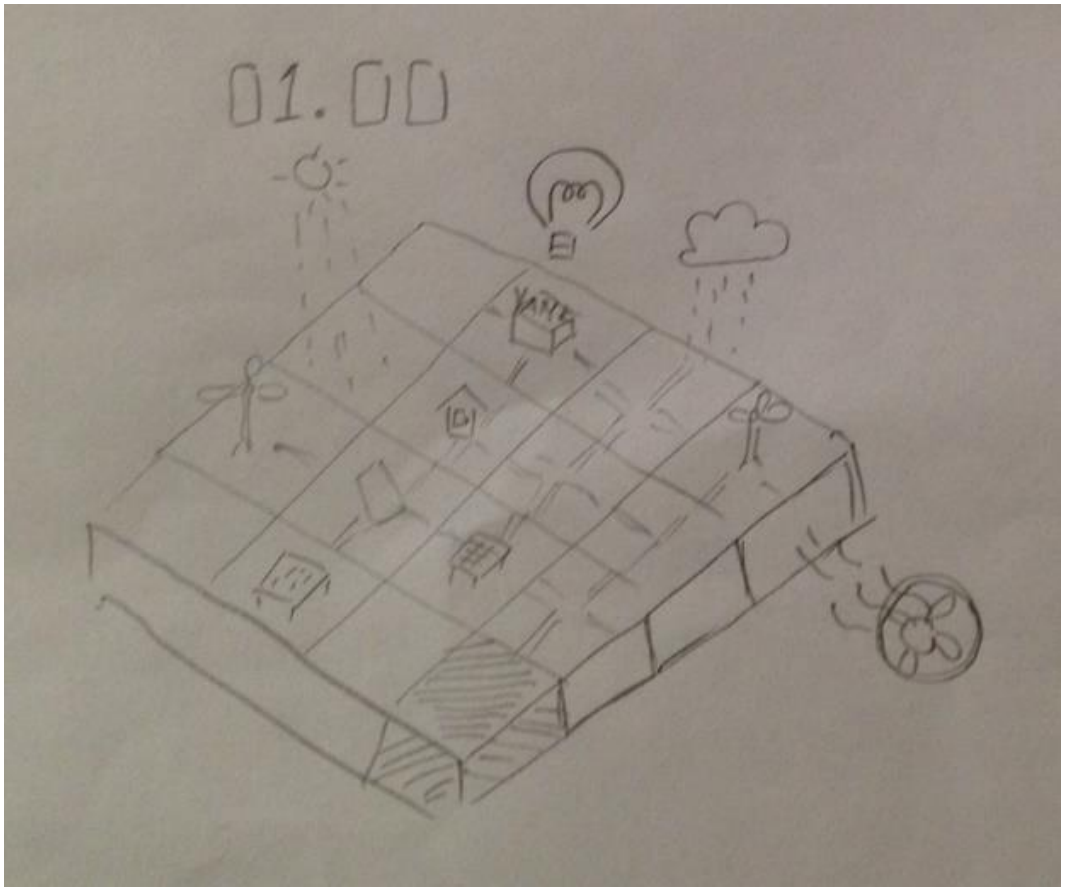
	○ 5 taloa ○ 2 kerrostaloa ○ 1 Vamk-koulu ○ puu ○ pensas ○ aita ○ laatat ○ laituri • Hint eli vihje-painike, jolla saa apua jos ei tiedä miten kenttä ratkaistaan • pelidatan resetoinnin mahdollisuus • toimiva ääniefekti ON/OFF-painike • toimiva musiikki ON/OFF-painike
Extrat (Nice to have)	• lowpoly, veden lisääminen taustagrafiikkaan • toimivat peliefektit ○ energia, energian liikkuminen pelissä ○ tuulenpuuska-efekti tuuliturbiiniin ○ liikkuvat pilvet • fade in/out eli näkymienvälinen haalistuminen • GameCenter-integrointi

2.2 Toiminnallinen määrittely

Pelin idea on ratkaista palapelimäisiä kenttiä. Pelikentän alkaessa näkyvissä on 5x5 laattaa. Laatat täytyy järjestää siten, että energia pääsee kulkemaan joko aurinkopaneeleista tai tuuliturbiineista taloihin, joissa energiantarve on. Pelilaattoja liikutellaan koskettamalla laattaa, jonka haluaa siirtyvän. Pelikentistä puuttuu aina yksi laatta. Kosketettava laatta siis siirtyy siihen kohtaan, jossa on tyhjää tilaa. Laatta ei

voi siirtyä kuin vierekkäiselle ruudulle, eli laatta siirtyy vaan jos sen ylä- tai alapuolella tai sivuilla on tyhjää. Peli toimii aikalailla samoin kuin vanhat liukupalapelit, joilla lapset leikkivät mm. -80 ja -90-luvuilla.

Kun laatat on järjestetty niin, että talo/talot saavat energiaa, olet läpäissyt kentän. Alkukentissä on liikerajoituksia, eli saat esim. 5 liikettä kentän suorittamiseksi. Jos kenttää ei saada suoritettua annetuilla liikkeillä, sama kenttä täytyy aloittaa alusta. Seuraava kenttä aukeaa pelattavaksi vasta kun edellinen on läpäisty.



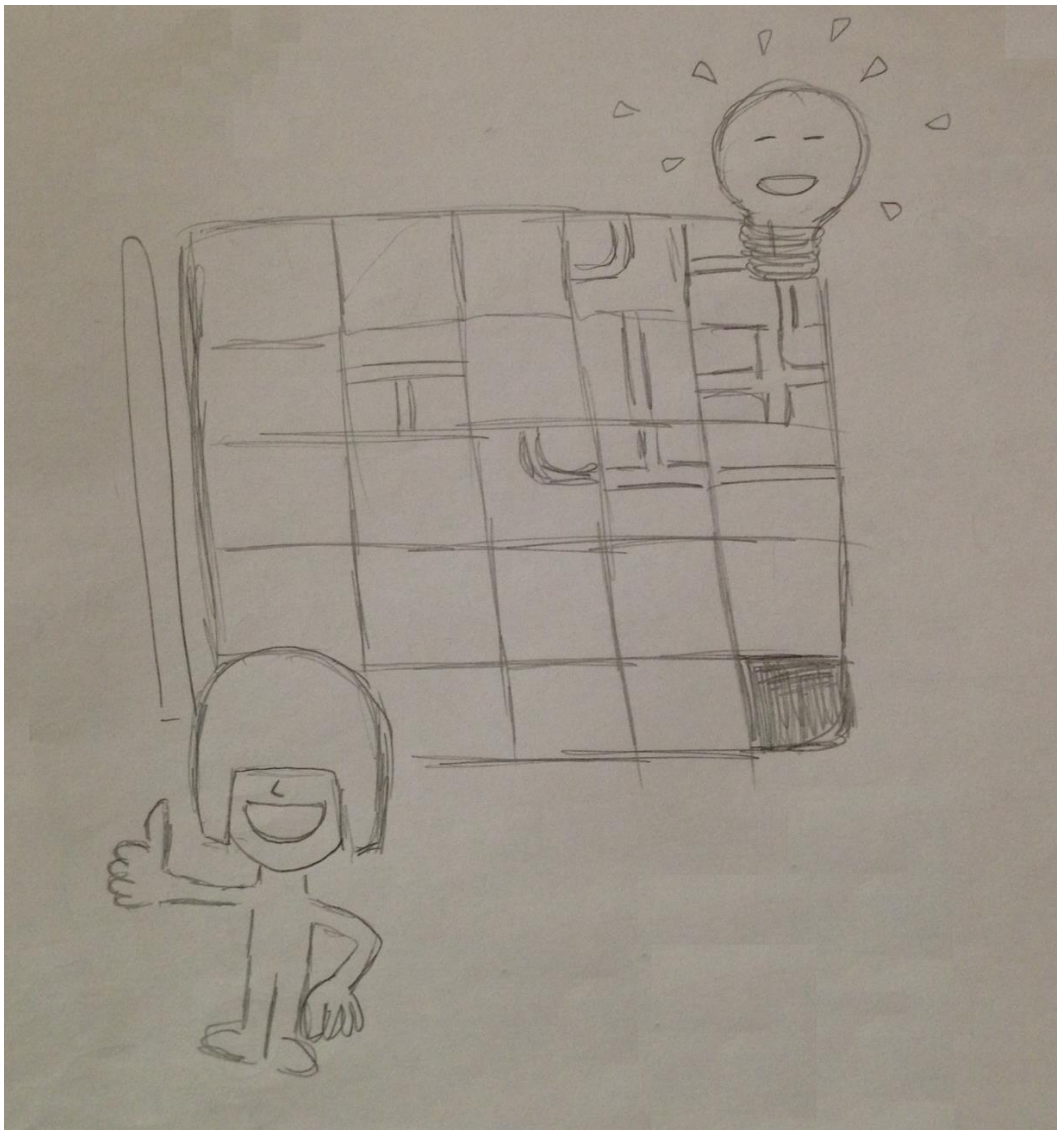
Kuva 1. Peliluonnos.

2.3 Ideointi

Pelin idea on mielestäni vaikein osa pelin tekemisessä. Yleensä hyviä peli-ideoita syntyy kun pelaa paljon erilaisia pelejä, ovat ne sitten pieniä kännykkäpelejä tai suuria pc- tai konsolipelejä. Pelaamisen lisäksi kehitin paljon erilaisia peliprototyyppejä. Hauskoja ja hyviä peli-ideoita voi olla kaikilla, mutta todellisuuden ja pelin toimivuuden näkee vasta kun ensimmäinen prototyyppi on kehitetty.

Tähän opinnäytetyöhön minulla oli muutama pelimuotoidea, yksi niistä oli pulmapeli eli (Puzzle Game) ja toinen loputon juoksupeli eli (Endless Runner Game) ja kolmas kyselypeli. Minulla oli entuudestaan jonkin verran kokemusta kehittää loputon juoksupeli ja tiesin, että jos lähden tekemään sitä, niin en tule oppimaan paljon uutta, eikä tule paljon haasteita vastaan. Kyselypeli vaikutti vain yksinkertaisesti tylsältä idealta, joten päätin kehittää ensimmäistä kertaa pulmapelin. Aiheen olin päättänyt jo, ja sen täytyi liittyä puhtaaseen energiaan. Selvitin asiaa ja tein taustatutkimusta yrittämällä etsiä sovelluskaupoista pulmapeliä, jossa aiheena olisi puhdas energia/energiamuodot, mutta lopputuloksena ei löytynyt kovin monta tuohon aiheeseen liittyvää peliä.

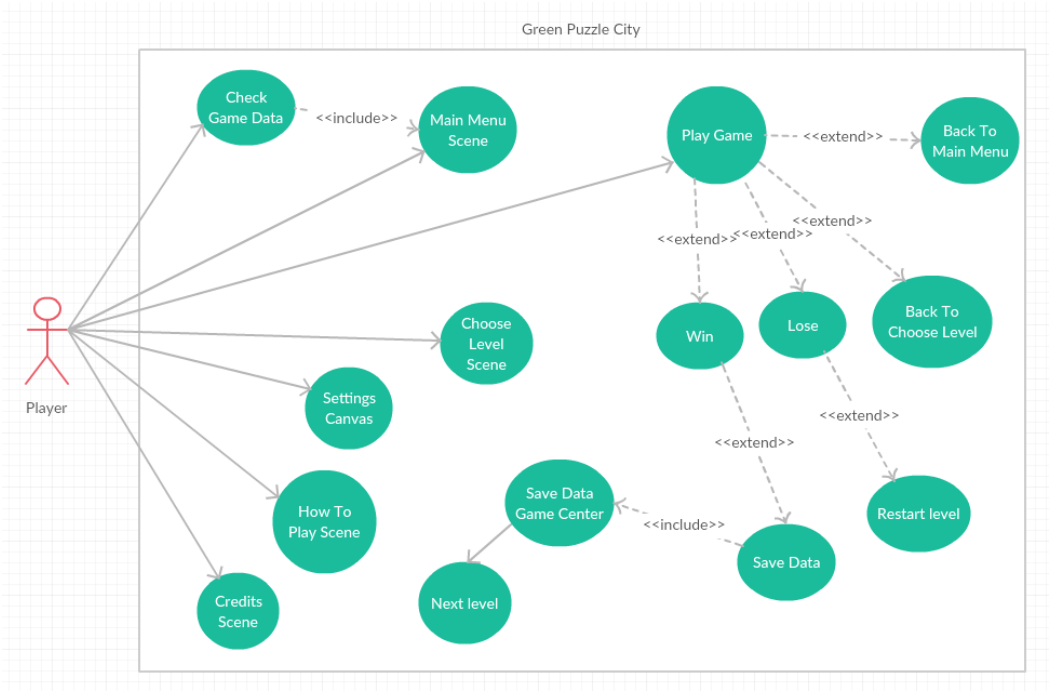
Muutaman viikon verran kokeilin erilaisia iOS-pulmapelejä ja kehitin niistä ideoista prototyyppejä. Sopivaa ideaa ei osunut kohdalle, enkä halunnut tehdä kopiopeliä jostain toisesta pelistä, mikä on jo valmiiksi sovelluskaupoissa. Sitten aviopuolisoni sai hyvän peli-idean; miksi en tee yksinkertaista sliding puzzle-tyyppistä peliä, jossa liikuttamalla palapelin laattoja yhdistetään polut toisiinsa ja sillä tavalla energia pääsee virtaamaan yhteen paikkaan. Se oli yksinkertaisen selkeä ja hauskan kuuloinen ja kaiken lisäksi sen pelattavuus olisi helppo ymmärtää. Aviopuolisoni piirsi sketsin paperilla (**Kuva 5.**) ja siitä lähdin kehittämään ideaa eteenpäin.



Kuva 2. Pulmapeliluonnos.

2.4 Käyttötapauskaavio (User Case Diagram)

Käyttötapauskaavio on UML-mallinuksen kaavio. Kaavio kuvaa toimijan toimenpiteitä tietojärjestelmässä tai tietokoneohjelmassa. Tämän kaavion kuvan avulla annetaan pelin yleiskäsitys.



Kuva 3. Pelaajan käyttötapauskaavio

3 KÄYTETYT TEKNIIKAT

3.1 Unity

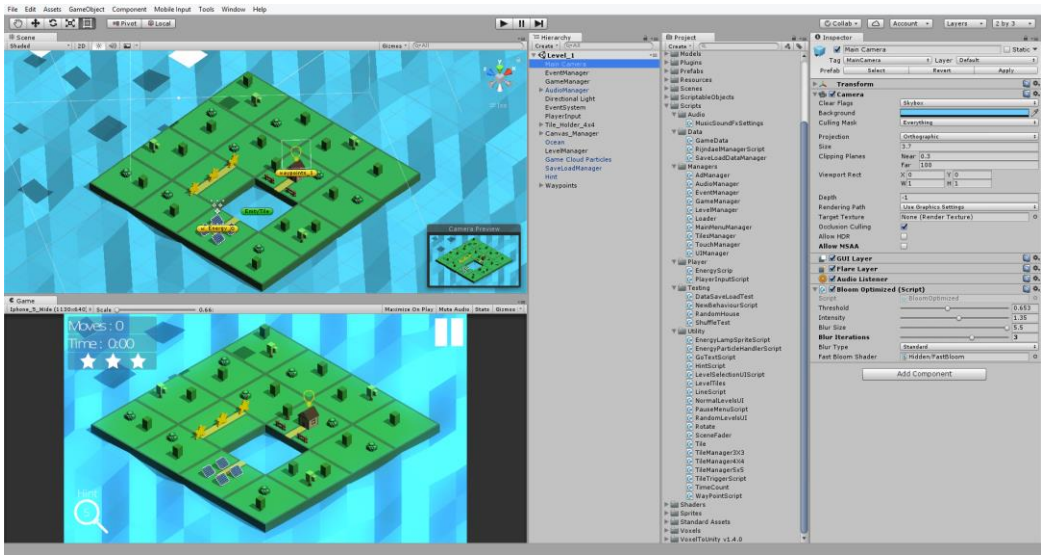
Unity on Unity Technologiesin kehittämä monialustainen pelimoottori, jolla voidaan kehittää kaksi-tai kolmeulotteisia mobiili-, konsoli-, selain- sekä PC-pelejä. Unity-editori on hyvin käyttäjäystävällinen ja käyttäjän on helppo lähteä kehittämään erilaisia peliprototyyppejä lyhyessä ajassa. Tämä on pelialalla todella tärkeä etu. Tällä hetkellä Unity tukee 27 alustaa ja alustojen määrä kasvaa jatkuvasti. Unitylla on myös erillisiä palveluja tarjolla käyttäjilleen, esimerkiksi

- UnityAds, jolla voi ansaita rahaa mainosvideoiden avulla,
- Unity Analytic, joka antaa kehittäjille pääsyn tärkeisiin pelidatoihin ja joka auttaa parantamaan itse peliä sekä tuottamaan lisää tuloja,
- Unity Collaborate, joka on hyvä työkalu, jos työskentelee tiimissä. Työkalun avulla voidaan siirtää koko projekti pilveen ja sieltä kaikki tiimin jäsenet pääsevät projektiin käsiksi ja voivat tehdä muutoksia siihen.
- Unity Cloud build, jolla voidaan luoda pelejä ja jakaa buildit automaattisesti ja
- Unity Multiplayer, jolla voidaan helposti kehittää verkkomoninpelejä.

Suurin osa palveluista on ilmaisia käyttäjille. Unity on ilmainen pelimoottori pienkehittäjille ja harrastelijoille, mutta jos kehittäjän/yrityksen liikevaihto on yli 100 000 dollaria, niin silloin yhtiö vaatii hankkimaan Plus-, Pro- tai Enterprise-version. Unity Technologies ei myöskään ota tekijänoikeusmaksuja pelien tuotoista.

Unity-editorissa on paljon työkaluja, jotka helpottavat kehittäjien työtä, tällöin kehittäjien ei tarvitse miettiä pelimoottorin teknologiaa vaan saavat keskittyä itse pelin tekemiseen. Unity-editoriin sisältyy seuraavat työkalut; animaatio, grafiikka, optimointi, ääni, 2D (Box2D) ja 3D (Physx 3.3) fysiikka, tekoäly ominaisuudet Navigation mesh ja Scriptaus, jolla voidaan ohjelmoida C# tai Javascriptillä. Itse pelimoottori on tehty C- sekä C++-ohjelmointikielillä.

Yksi Unityn parhaimmista ominaisuuksista on Unity-yhteisö, joka auttaa ja helpottaa kehittäjiä. Jos on ongelmia jonkun asian kanssa liittyen pelikehitykseen, ohjelmointiin tai itse Unity-editoriin, niin yhteisön foorumeilla on aina joku jolla on vastaus ongelmaan. Erityisesti tämä tekee Unitysta hyvän työkalun.



Kuva 4. Unity 5.6.0f3-editori

3.1.1 Unity Ads

Unity Ads on videomainosverkko, jolla voidaan tuottaa kehittäjälle rahaa mainosten avulla. Pelaajat katsovat videoita pelin sisällä ja tällä tavalla voivat ansaita lisäpelimerkkejä, jotka auttavat pelaajia etenemään peleissä nopeammin tai ansaita lisäbonuksia yms. Tällä hetkellä Unity Ads on yksi suurimmista videomainostusverkoista, joita isot yritykset käyttävät. Näihin yrityksiin lukeutuvat pelialan jäteistä mm. Supercell, Rovio, Sega, Next Games, Halfbrick ja moni muu nimekäs yritys. Unity tekee todella helpoksi integroida Unity Ads:in peleihin, ollessaan nopea ja helppokäyttöinen.

3.2 IDE - ohjelmointiympäristö

3.2.1 Visual Studio Community 2015

Visual Studio Community on Microsoftin kehittämä IDE / -ohjelmankehitysympäristö, jossa voidaan ohjelmoida erilaisilla ohjelmointikielillä, kuten Visual Basic, C++, C#, F#. Työkalulla voidaan tehdä esimerkiksi Windows-, Web- ja mobiilisovelluksia. Unity 5.2.0 versioon on integroitu natiivi Visual Studio Community 2015 ja se on parantanut pelin kehittämisen laatua. Visual Studio tuo mukanaan tehokkaat ominaisuudet kirjoittaa C#-ohjelmia nopeasti (Intellisense). Ohjelman avulla voi navigoida helposti scriptien välillä. Visual Studion debuggaus-työkalua voi käyttää myös Unityssä. Tämä debuggaus-työkalu on mielestäni huomattavasti parempi kuin Unityn oma. Ainoa huono puoli tällä hetkellä on, että Visual Studio 2015 tukee vain Windows-käyttöjärjestelmää.

3.2.2 MonoDevelop

MonoDevelop on Mono Communityn kehittämä avoimen lähdekoodin ohjelmakehitysympäristö, joka on integroitu Linuxiin, macOSiin ja Windowsiin. MonoDevelop tunnetaan myös uudesti brändättynä nimellä Xamarin Studio. Unity käyttää mukautettuna versiona Monodevelopia oletusohjelmointiympäristönä. MonoDevelopin ensisijaisena tavoitteena on kehittää projekteja käyttäen Monoa ja .NET Frameworkia.

3.2.3 Xcode

Xcode on Applen macOslle kehittämä ohjelmointiympäristö. Xcode:n avulla voi tehdä komentoriviohjelmia (Unix), Java-appletteja, macOs-ohjelmia, iOS, watchOS sekä tvOS. Xcode sisältää ohjelmointiympäristön, projektimanagerin, debuggerit ja kääntäjät sekä iPhone- ja iPad- simulaattorit. Xcode tukee erilaisia oh-

jelmointikieliä, kuten C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby ja Swift. Koska tässä projektissa kehitetään iOS-alustalle peli, niin Xcode on aika iso tekijä projektissa. Xcodessa on myös hyvin tärkeä työkalu nimeltä Instruments (tästä työkalusta laajemmin kohdassa 6.4), jolla voi analysoida sekä visualisoida pelin suorituskykyä.

3.3 Käytetyt ohjelmointikielet

3.3.1 C#

C# on Microsoftin kehittämä ohjelmointikieli, jonka tarkoituksena on olla yksinkertainen, moderni ja yleiskäyttöinen olio-ohjelmointikieli. C#-kieli julkaistiin kesäkuussa 2000. Kieli kehitettiin yhdistämällä C++:n tehokkuus ja Java-kielen helppokäyttöisyys. C# tämän hetkinen versio on C# 7.0 joka julkaistiin 2017 Visual Studio 2017 mukana. C# on Unityn pääohjelmointikieli, tästä syystä tein pelin pääosin sillä.

3.3.2 JSON

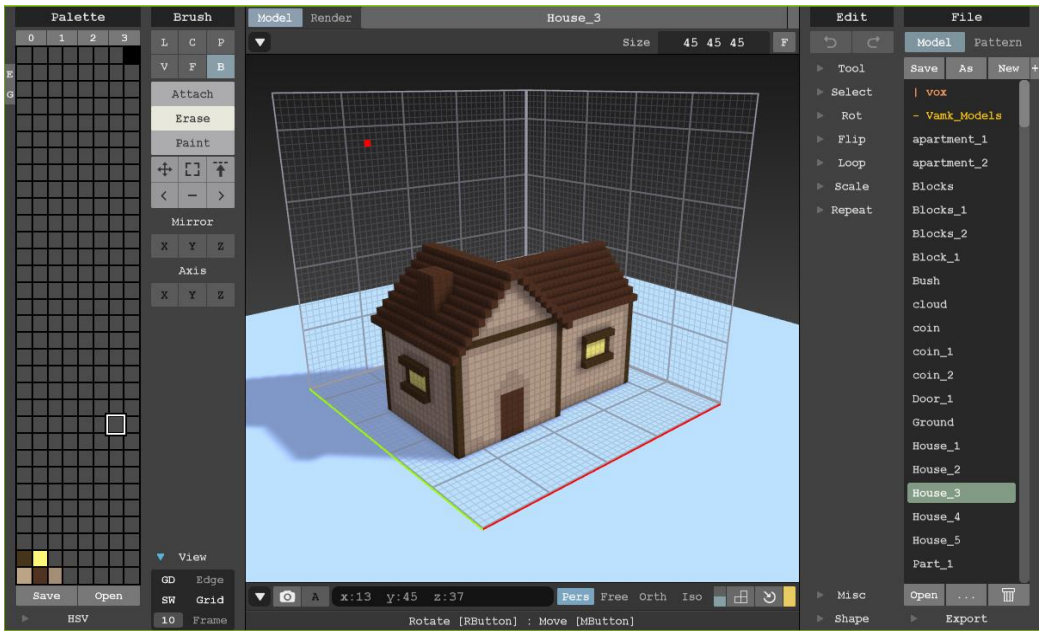
JSON on yksinkertainen avoimen standardin tiedoston välitysmuoto. Ohjelmoijat käyttävät sitä tietoa siirtääkseen palvelimen tai ohjelman välillä. JSON on riippumaton ohjelmointikielestä ja sen tietoja pystyy lukemaan melkein millä tahansa ohjelmointikielellä. JSONssa voidaan käyttää numeroita, boolean-arvoja, listoja, stringejä ja null-arvoja.

Tässä projektissa tallennetaan pelidata JSON-tiedostoon, joka tallentuu iOS-laitteisiin. Päätin käyttää tätä välitysmuotoa, sillä JSONin tiedostokoko on pieni ja siten sopii hyvin mobiilialustoille.

3.4 MagicaVoxel

Voxel (Vokseli) on niin sanotusti kolmiulotteinen pikseli. MagicaVoxel /7/ on kevyt ja ilmainen 8-bittinen voxel-editori ja renderoija, joka on tällä hetkellä Beta-versiossa 0.98.1. Tällä mallinnustyökalulla pystytään tekemään hienonnäköisiä 3D-

malleja hyvin lyhyessä ajassa. Voxel-mallien tekemiseen käytetään pelkästään kuutioita. Kuvassa 5 nähdään MagicaVoxel-editori.

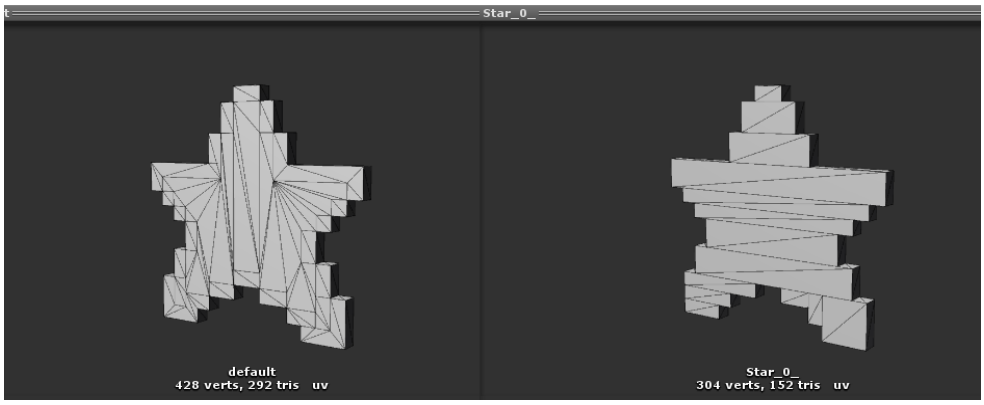


Kuva 5. MagicaVoxel 0.98.1 editori.

3.5 MagicaVoxel Qubicle To Unity

MagicalVoxel Qubicle to Unity on Unity-plugin, jolla voidaan muuttaa vox-tiedosto optimoituun obj-tiedostoon, jotta Unity tunnistaa kyseiset 3D-mallit. Tämä lisäosa toimii Unity 5.X versiolla tai uudemmalla. Green Puzzle City on tarkoitettu mobiilialustalle, joten jouduin optimoimaan kaikki grafiikat, jotta pelin suorituskyky pysyisi mahdollisimman hyvänä.

Esimerkkinä **(Kuva 6)** vasemmalla puolella nähdään ei-optimoitu obj-tiedosto ja oikealla puolella optimoitu obj-tiedosta. Kuvasta huomaa, että optimoinnin jälkeen verteksilukema laskee 428 verteksistä 304 verteksiin, sekä 292 triangeli laskee 152 triangeliin.

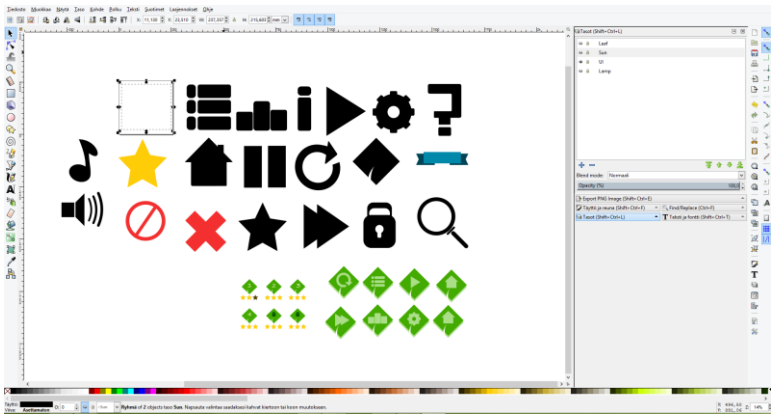


Kuva 6. Ei-optimoitu ja optimoitu voxel obj-tiedosto

3.6 Inkscape – käyttöliittymän grafiikat

Inkscape on avoimeen lähdekoodiin perustuva vektorigrafiikkaohjelmisto. Ohjelman kehitys alkoi kun ryhmä sodipodi-projektin ohjelmoijia alkoi kehittää uutta, täysin W3C:n SVG standardin kanssa yhteensopivaa vektorigrafiikkaeditoria vuonna 2003. Inkscape kehitettiin pääasiassa Linux-käyttöjärjestelmälle, mutta se toimii myös muilla alustoilla, kuten Windows ja OSX. Inkscape vastaa ominaisuuksiltaan Adobe Illustratoria tai Corel Draw'ta.

Kaikki pelin käyttöliittymän grafiikat tuli tehtyä Inkscapella, tämä siksi, että ohjelma oli minulle ennestään tuttu ja sen käyttö on helppoa. Kuvassa 7 nähdään Inkscape-editori ja muutamia vektorigrafiikoita, jotka ovat käytössä Green Puzzle Cityssä.

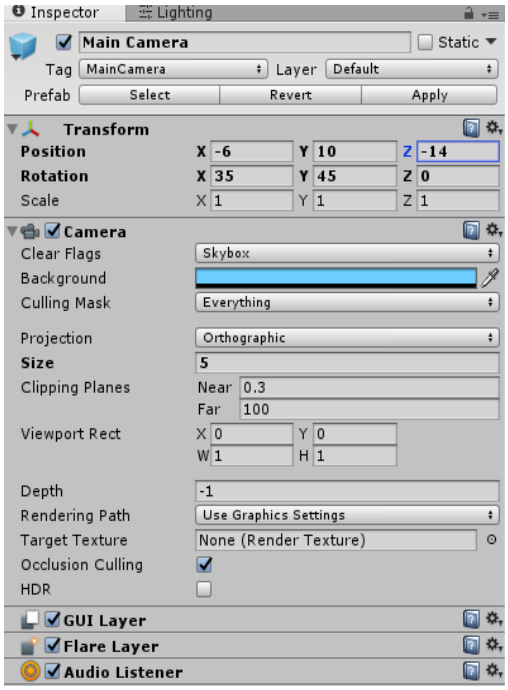


Kuva 7. Inkscape-editori

4 SUUNNITTELU JA PROTOTYYPPI

Tiesin jo ennen projektin aloittamista, että haluan pelin olevan kolmiulotteinen. Yleensä kolmiulotteiset pulmapelit tehdään joko niin, että kamera on suunnattu ylhäältä alas, tai käytetään isometristä kameraa, jolloin kamera on viistossa. Tämän lähtökohtatiedon perusteella lähdin työstämään prototyyppiä.

Joka Unity Scenessä on Main Camera ja Directional Light. Muutokset, jotka tein Main Cameraan komponenttiin, olivat seuraavat; Projection valinnan vaihdoin oletuksena olleesta Perspectivestä Orthographiciin. Valitsin kooksi Size 5 ja muutin Transform-komponentin asetuksiksi Position X : -6 , Y : 10 , Z :-14 Rotation x : 35, Y: 45 ja Z:0. Näillä asetuksilla sain isometrisen kameran käyttöön.

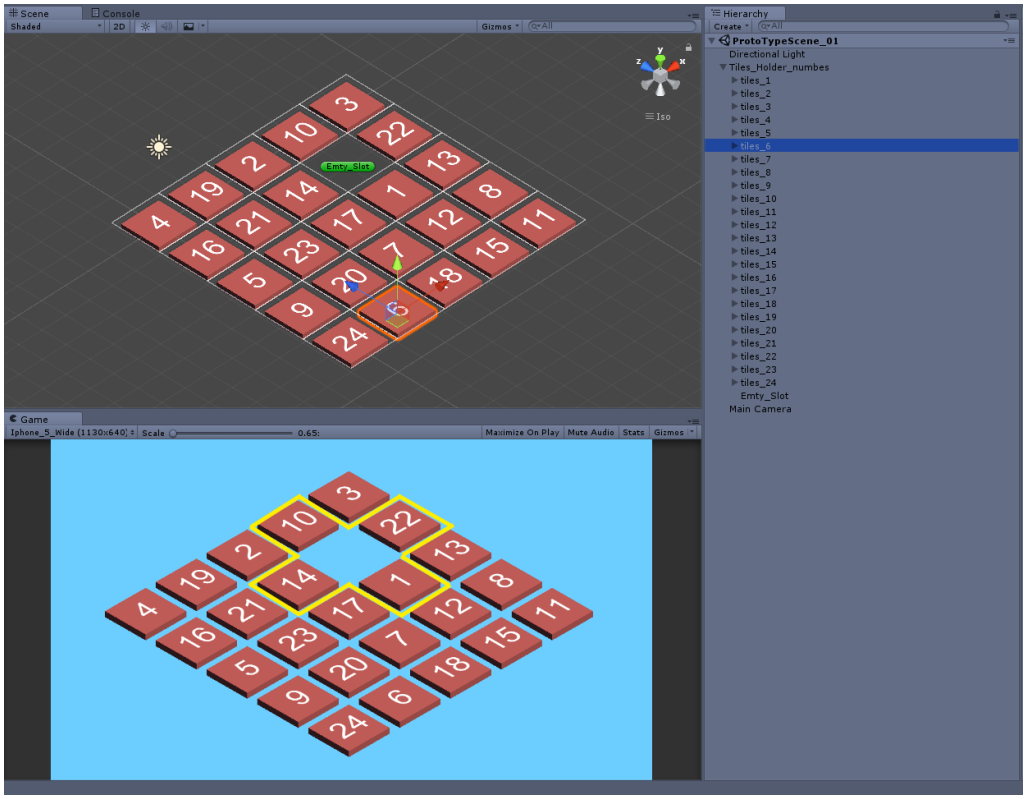


Kuva 8. Main camera komponenttiasetukset

Unityyn on valmiiksi sisäänrakennettu erilaisia peliobjekteja (GameObject), joita on kätevä käyttää prototyyppivaiheessa. Unityn kolmiulotteiset peliobjektit ovat kuutio, sylinteri, pallo, kapseli, plane ja quad. Näiden peliobjektien avulla voidaan

tehdä helposti prototyyppejä ilman, että tarvitsee kuluttaa aikaa 3D-mallien tekemiseen.

Muutin 24 kuutiota litteäksi ja annoin niille numerot 1-24. Kahden skriptin avulla sain tehtyä toimivan kentän. Ensimmäisen skriptin (PlayerInputScript.cs) avulla voidaan siirrellä kuutioita ja toisen (TileManager.cs) avulla tarkistetaan ovatko kuutiot oikeassa järjestyksessä. Käänsin pelin iOS-alustalle ja testasin sitä iPhone 5-laitteen kanssa. Kosketuksen kannalta testaus laitteessa vaikutti toimivalta ja isometrinen kamera vaikutti visuaalisesti hyvältä, joten päätin jatkaa projektia eteenpäin.



Kuva 9. Pelin prototyyppi

5 TOTEUTUS

5.1 Pelilogiikka

Kun peli alkaa, ohjelma tarkistaa onko pelidatatiedosto olemassa. Jos ei, niin se luo uuden tiedoston ja tallentaa alustavan pelidatatiedoston. Pelissä voidaan navigoida näkymien (scene) välillä. Ensimmäinen näkymä on splash screen, jonka jälkeen näkymä siirtyy päävalikkoon. Päävalikossa on viisi painiketta, joista neljästä pääsee eri näkymiin. Nämä näkymät ovat SelectLevel-, Credits-, HowToPlay- ja GameCenter. Asetus-painikkeesta aukeaa päävalikossa ikkuna, jossa voi tehdä muutoksia, esimerkiksi äänen mykistämiseen ja pelidatan resetointiin.

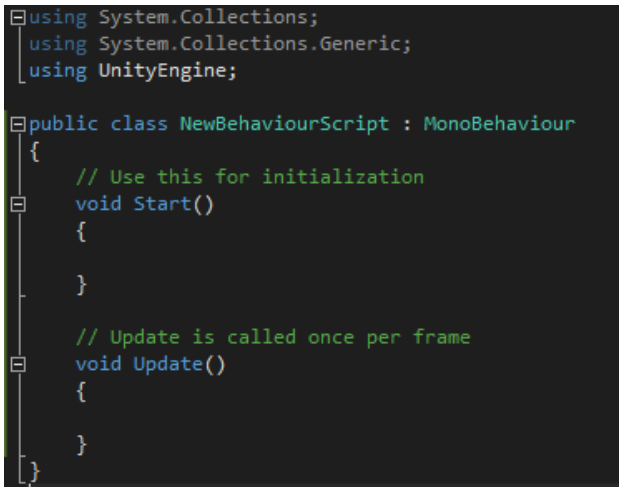
Pelimuotoja on alustavasti kaksi, Star challenge ja Random challenge. Star challenge-kentät aukeavat sitä mukaa kun edellinen kenttä on pelattu läpi. Random challenge-muodossa ensimmäinen kenttä avautuu pelattavaksi kun on suoritettu 5 Star challenge-kenttää. Joka kentän läpäisyn jälkeen näytetään ruudulla tulokset ja ne tallentuvat pelidatatiedostoon, joka on tallennettu iOS-laitteeseen. Joka kentän jälkeen on mahdollisuus katsoa mainoksia, joiden katsomisesta palkinnoksi saa lisää vinkkejä eli (Hint) pelipoletteja. Pelipolettien avulla pystytään selvittämään seuraavia kenttiä. Peli vaikeutuu mitä pidemmälle päästään.

5.1.1 Unityn funktioiden suoritusjärjestys

Unityssä C#-skripti edustaa käytöskomponentteja eli Behaviour components. Unityssä ei ole varsinaista Main loop-funktiota, jossa ohjelma lähtee liikkeelle, vaan Unityssä on erilaisia suoritusjärjestysfunktioita /10/. Yleisimmät niistä ovat Awake, Start ja Update.

Suoritusjärjestys toimii seuraavasti; Awake-funktio kutsutaan aina ennen Start-funktiota. Start-funktio kutsutaan ainoastaan kerran ensimmäisessä framessa. Yleensä tätä funktiota käytetään muuttujien alustamiseen. Update-funktio kutsutaan kerran joka framessa. Kuvassa 10 nähdään oletuskoodin rakenne, kun luodaan C# skriptiä. Tässä projektissa tulen käyttämään seuraavia funktioita; Awake,

OnEnable, OnDisable, Start, Update, OnDrawGizmos, yield WaitForSeconds, yield StartCoroutine, OnDestroy, OnTriggerEnter ja OnTriggerExit.



Kuva 10. Unity C# oletusskripti.

5.1.2 Singleton-objektit

Yleensä Unity-pelit muodostetaan näkymien (scenes) avulla. Green Puzzle Cityssä on MainMenu, SelectLevel, ja erilaisia pelikenttiä. Joka kerta kun mennään toiseen näkymään, oletuksena Unity tuhoaa kaikki edellisen kentän objektit ja uudessa näkymässä funktioiden suoritusjärjestys alkaa alusta. Välillä kuitenkin tarvitaan objekteja, jotka eivät tuhoudu siirryttäessä näkymien välillä. Tällainen objekti voi olla esimerkiksi pelin tallennuskomponentti, joka on tallennuksen kannalta välttämätön. Tähän ratkaisuun on olemassa Singleton- menetelmä (Singleton Pattern. /2/). Tässä pelissä on tällä hetkellä kolme objektia, jotka eivät tuhoudu ollenkaan; EventManager, SaveLoadDataManager ja AudioManager. Kuvassa 10 nähdään miten toteutetaan SaveLoadDataManager-luokan sisällä Singleton.

```
private static SaveLoadDataManager instance = null;
public static SaveLoadDataManager Instance
{
    get
    {
        if (instance == null)
        {
            instance = FindObjectOfType(typeof(SaveLoadDataManager)) as SaveLoadDataManager;

            if (instance == null)
            {
                Debug.LogError("There needs to be one active SaveLoadDataManager script on a GameObject in your scene!");
            }
        }
        return instance;
    }
}

private void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(gameObject);
    }
    else
    {
        DestroyImmediate(gameObject);
    }
}
```

Kuva 11. Singleton objekti.

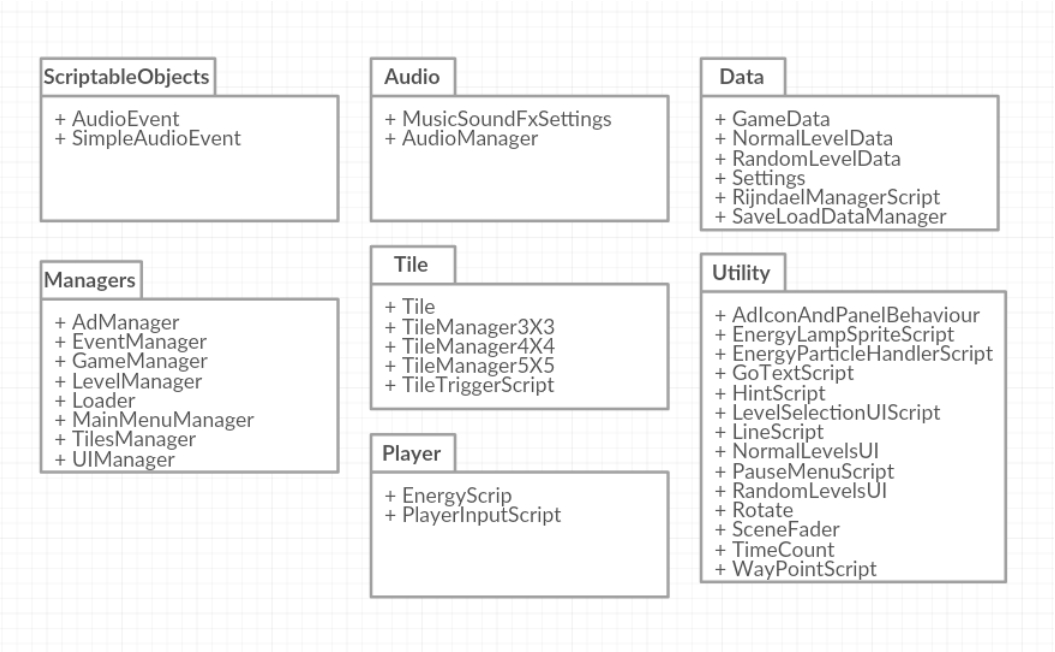
5.1.3 Pelimuodot

Green Puzzle Cityssä on tällä hetkellä kaksi pelimuotoa, Star challenge ja Random challenge. Star challengessa kaikki laatat on manuaalisesti asetettu. Random challengessa laatat asettuvat paikoilleen satunnaisesti joka kerta kun aloitat uuden pelin. Tämä tekee Random challenge-pelimuodosta hiukan haastavan. Molemmissa pelimuodoissa on Moves- ja Time-laskurit näkymän vasemmassa yläkulmassa. Moves kertoo kuinka monta siirtoa pelaaja voi tehdä, jotta kenttä saadaan pelattua läpi. Time, eli aika, kertoo sen kuinka kauan aikaa on kulunut kentän aloituksesta. Aikalaskuri pysähtyy kun kenttä saadaan ratkaistua.

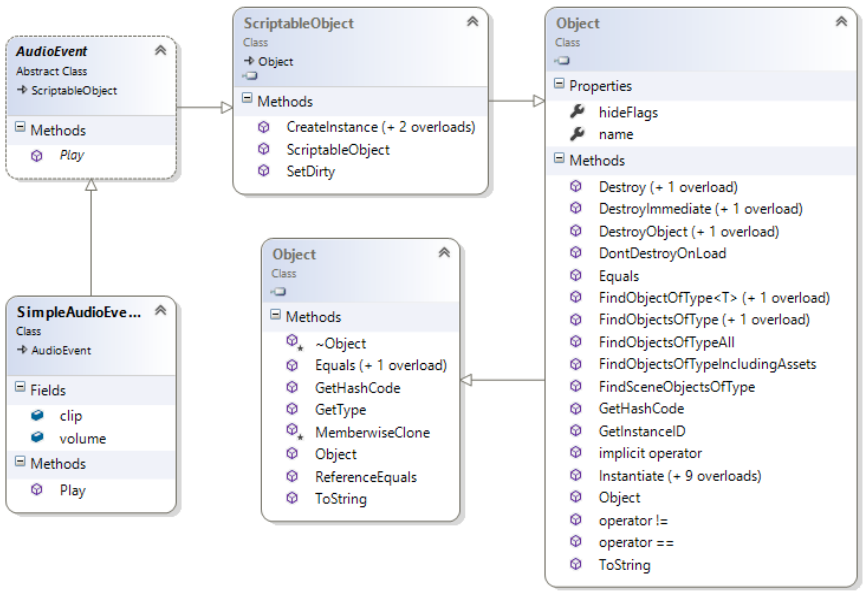
Kenttien suorituksen jälkeen Moves- ja Time-tiedot tallennetaan pelidataan. Tallennus tehdään tulevan GameCenter-lisäyksen vuoksi, jotta ennätysajat/-liikkeet voidaan siirtää top-listoille.

5.1.4 Luokkakaaviot

Green Puzzle Cityn toiminta muodostuu erilaisista skriptikomponenteista. Seuraavissa kuvissa nähdään UML-luokkakaaviot.

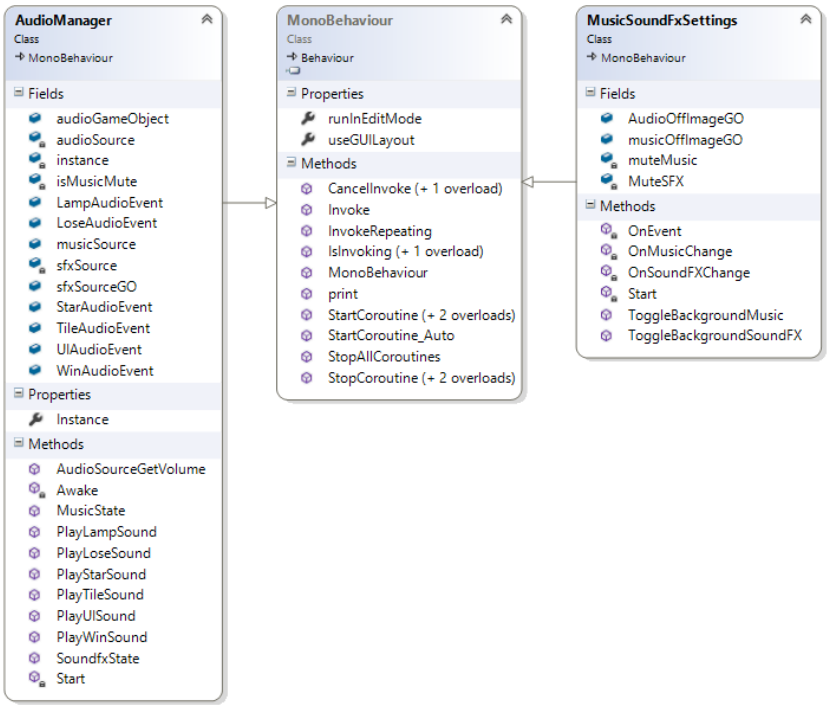


Kuva 12. Packages eli hakemistot, joissa kaikki luokat on luokiteltu



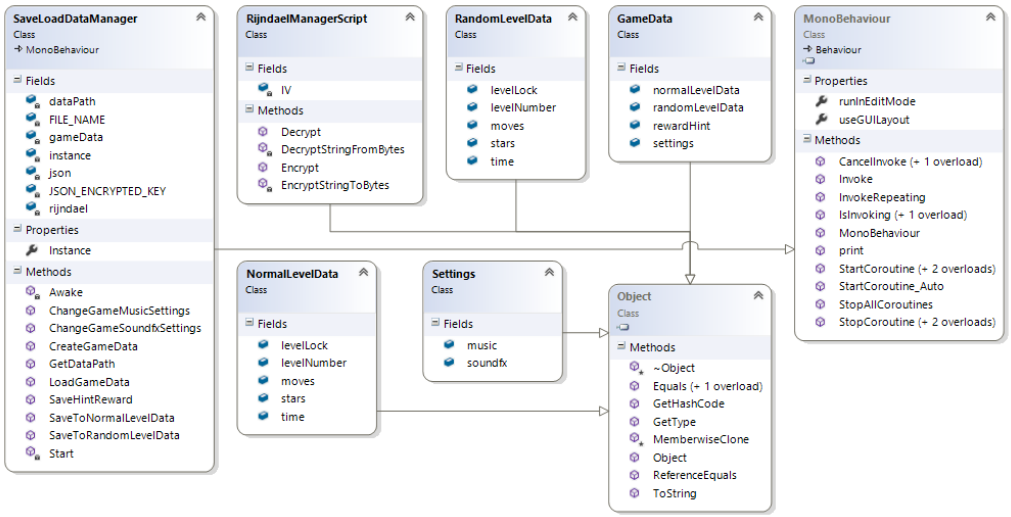
Kuva 13. ScriptableObject-luokat.

Kuvassa 13 nähdään SimpleAudioEvent.cs-luokan muuttujat, sekä funktio ja niiden toiminta AudioEvent.cs abstract-luokkaan.



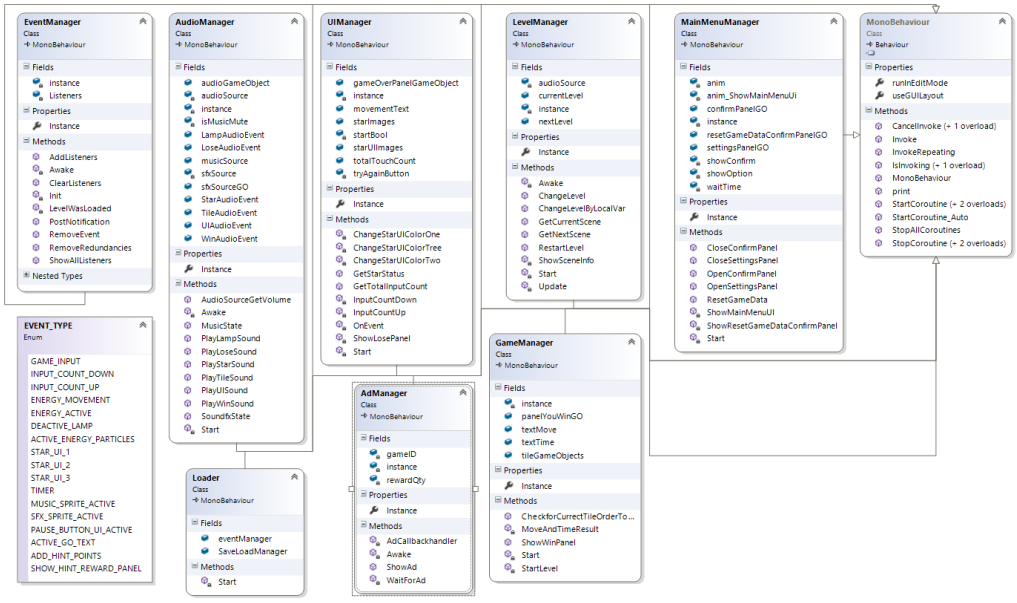
Kuva 14. Audioluokat.

Kuvassa 14 nähdään AudioManager.cs- sekä MusicSoundFxSettings.cs-luokkien muuttujat ja funktiot. Molemmat luokat perivät monoBehaviour-luokasta.



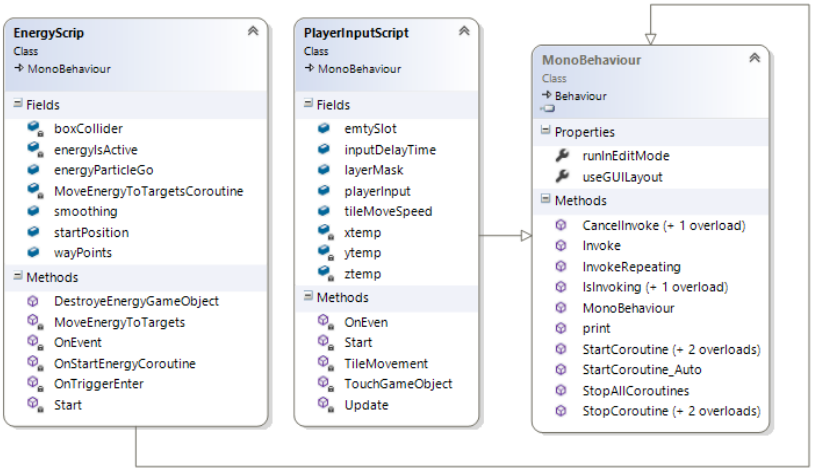
Kuva 15. Data-luokat.

Kuvassa 15 nähdään SaveLoadDataManager.cs ja RijndaelManagerScript.cs muuttujat ja funktiot. GameData.cs-luokassa on kolme listamuuttujaa seuraavista luokista; NormalLevelData.cs, RandomLevelData.cs ja Settings. Managerissa tallennus tapahtuu listoina JSON-tiedostoon.



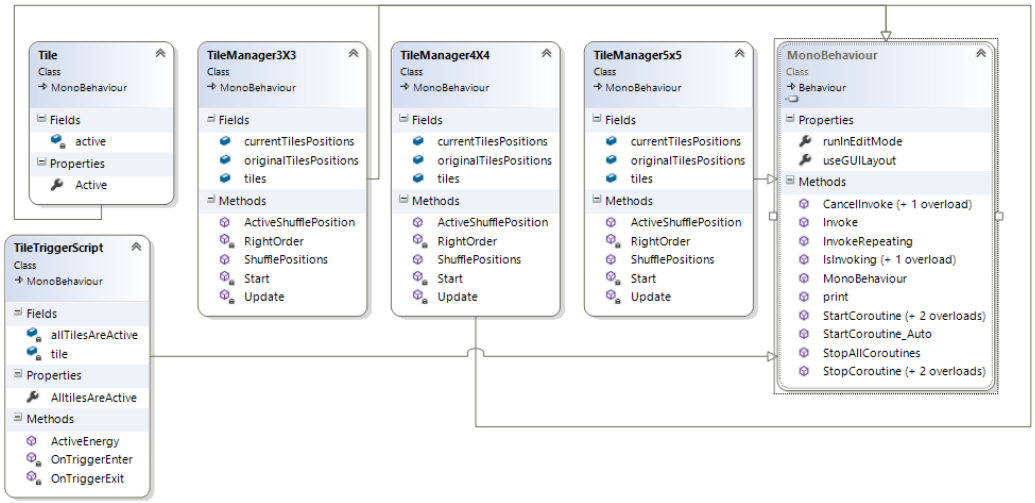
Kuva 16. Manager-luokat.

Kuvassa 16 nähdään Green Puzzle Cityn Manager-luokan muuttujat sekä funktiot. Kaikki Manager-luokat perivät MonoBehaviour-luokasta. Tärkein Manager-luokka on EventManager, sillä kaikki luokat keskustelevat toistensa kanssa tämän luokan kautta.



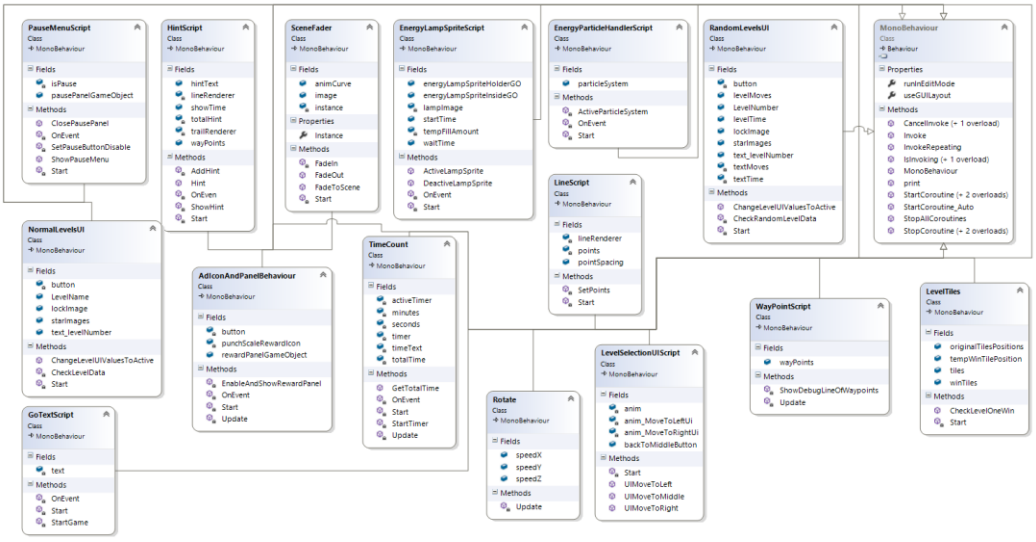
Kuva 17. Player-luokat.

Kuvassa 17 on EnergyScript.cs ja PlayerInputScript.cs muuttujat sekä funktiot. Molemmat luokat perivät MonoBehaviour-luokasta. EnergiaScript-luokka hoitaa energian kulkua tuuliturbiinista tai aurinkopaneelista taloihin. PlayerInputScript-luokka hoitaa kosketukset mobiilin ruudusta peliobjekteihin.



Kuva 18. Tile-luokat.

Kuvassa 18 esitetään pelilaatan toimintaluokat ja niiden muuttujat sekä funktiot. TileManager3x3.cs-, TileManager4x4.cs- ja TileManager5x5.cs-luokat hoitavat laattojen satunnaissekoituksen.



Kuva 19. Utility-luokat.

Kuvassa 19 nähdään kaikki lisäluokat, niiden muuttujat ja funktiot.

5.1.5 PlayerInputScript.cs

Luokka PlayerInputScript.cs käsittelee pelaajan kosketusinputeja ja niiden toiminnan seurausta laattojen liikkeisiin. Green Puzzle Cityssä kosketus laattoihin toimii seuraavasti; update-funktion sisällä tarkistin onko kosketuksia ruutuun vain yksi. Mikäli näin on, niin kutsutaan seuraava funktio nimeltä TouchGameObject (**Kuva 20**). Tämä funktio piirtää debuggaus-viivan kamerasta pelikenttään ja tarkistaa osutaanko nyt oikeaan layermaskiin. Jos tämä pitää paikkansa, kutsutaan TileMovement-funktio. TileMovement-funktio hoitaa laattojen siirtämisen. Kuvassa 21 nähdään TileMovement-funktio.

```
#region Touching GameObjects
private void TouchGameObject()
{
    if(Input.touchCount > 0)
    {
        int inputs = Input.touchCount;

        for(int i = 0; i < inputs; i++)
        {
            Touch touch = Input.GetTouch(i);
            Ray ray = Camera.main.ScreenPointToRay(touch.position);
            RaycastHit hit;

            if(Physics.Raycast(ray, out hit, 1000, layerMask))
            {
                if(touch.phase == TouchPhase.Began)
                {
                    #if UNITY_EDITOR
                    Debug.Log("Touch: " + hit.collider.name + " Layermask " + hit.collider.gameObject.layer);
                    Debug.DrawLine(transform.position, hit.point, Color.red, 0.5f);
                    Debug.Log(Vector3.Distance(hit.transform.position, emtySlot.position));
                    #endif
                    StartCoroutine(TileMovement(hit));
                }
                if(touch.phase == TouchPhase.Ended)
                {
                    //TODO: Send ended event
                }
                if(touch.phase == TouchPhase.Stationary || touch.phase == TouchPhase.Moved)
                {
                    //TODO: Send Stationary and moved event

                }
                if(touch.phase == TouchPhase.Canceled)
                {
                    //TODO: Send canceled event

                }
            }
        }
    }
}
#endregion
```

Kuva 20. TouchGameObject-funktio.

```
IEnumerator TileMovement(RaycastHit hit)
{
    if (Vector3.Distance(hit.transform.position, emtySlot.position) == 2)
    {
        EventManager.Instance.PostNotification(EVENT_TYPE.INPUT_COUNT_DOWN, this, null);

        xtemp = hit.transform.position.x;
        ytemp = hit.transform.position.y;
        ztemp = hit.transform.position.z;

        AudioManager.Instance.PlayTileSound();
        iTween.PunchScale(hit.transform.gameObject, new Vector3(0.2f, 0.2f, 0.2f), 1.0f);
        iTween.MoveTo(hit.transform.gameObject, emtySlot.position, tileMoveSpeed);

        emtySlot.transform.position = new Vector3(xtemp, ytemp, ztemp);

        yield return new WaitForSeconds(inputDelayTime);;
    }
}
```

Kuva 21. TileMovement-funktio.

5.1.6 EventManagerit

EventManagerilla /9/ on erittäin tärkeä rooli koko pelin kehityksessä. Koska kehitän pelin mobiililaitteille, tulee karsia kaikki turhat seikat pois, jotka hidastavat pelin suorituskykyä. Update-funktiossa yleensä pyöritetään kaikki pelilogiikat, kuten if- ja switch-lausunnot. Mitä enemmän siellä on kyselykoodeja, sitä hitaammaksi peli tulee. EventManagerin avulla vältetään turhien kyselyjen ja lauseiden kirjoittamista Update-funktion sisälle. Eventit kutsutaan vain ja ainoastaan silloin kun niitä tarvitaan, tai jos on jotain muutoksia tullut tiettyihin objekteihin. Tämä on yksi tapa nopeuttaa pelin suorituskykyä. Kuvassa 22 nähdään erilaisia eventejä, jotka ovat käytössä pelissä.

```
public enum EVENT_TYPE
{
    //GAME_START,
    //GAME_END,
    GAME_INPUT,
    INPUT_COUNT_DOWN,
    INPUT_COUNT_UP,
    ENERGY_MOVEMENT,
    ENERGY_ACTIVE,
    DEACTIVE_LAMP,
    ACTIVE_ENERGY_PARTICLES,
    //FADE_IN,
    //FADE_OUT,
    STAR_UI_1,
    STAR_UI_2,
    STAR_UI_3,
    TIMER,
    MUSIC_SPRITE_ACTIVE,
    SFX_SPRITE_ACTIVE,
    PAUSE_BUTTON_UI_ACTIVE,
    ACTIVE_GO_TEXT,
};
```

Kuva 22. Pelieventit

```
private void Start()
{
    if(EventManager.Instance != null)
    {
        EventManager.Instance.AddListeners(EVENT_TYPE.GAME_INPUT, OnEven);
    }
}
```

Kuva 23. Eventin lisäys AddListener-funktioon.

```
EventManager.Instance.PostNotification(EVENT_TYPE.GAME_INPUT, this, true);
```

Kuva 24. Eventin kutsu PostNotification-funktioon.

PostNotification funktion kutsun jälkeen OnEvent-funktio suoriutuu kuvan 25 mukaisesti. Tässä tapauksessa playerInput-muuttuja, joka on boolean (eli true or false), saa Params-muuttujasta uuden arvon.

```
private void OnEven(EVENT_TYPE Event_Type, Component sender, object Params)
{
    switch (Event_Type)
    {
        case EVENT_TYPE.GAME_INPUT : playerInput = (bool)Params;
            break;
        default:
            print("Flowing Font defualt state");
            break;
    }
}
```

Kuva 25. OnEvent-funktio.

```
public bool playerInput = false;

private void Start()
{
    if(EventManager.Instance != null)
    {
        EventManager.Instance.AddListeners(EVENT_TYPE.GAME_INPUT, OnEven);
    }
}

private void OnEven(EVENT_TYPE Event_Type, Component sender, object Params)
{
    switch (Event_Type)
    {
        case EVENT_TYPE.GAME_INPUT : playerInput = (bool)Params;
            break;
        default:
            print("PlayerInputScript-GAME_INPUT defualt state");
            break;
    }
}
```

Kuva 26. GAME_INPUT-eventin lisäys ja kutsu.

5.1.7 ScriptableObject

ScriptableObject /8 on luokka, jossa voi säilyttää dataa. Tätä dataa voidaan käyttää kun halutaan luoda objekteja, joiden ei tarvitse olla kiinni peliscenessä. Hyvä puoli tässä on se, että tietoja pääsee muokkamaan mistä tahansa sekä se, että editorissa voidaan tallentaa dataa editoinnin ja ajon aikana. Green City Puzzlessa käytän ScriptableObjectia kaikkiin ääniefekteihin.

ScriptableObjectin luominen on melko yksinkertaista. Tässä esimerkissä halusin toistaa ääniefektejä. AudioEvent.cs-skriptissä luodaan abstract-luokka ja abstract

function-prototyyppi (**Kuva 27**). SimpleAudioEvent.cs avulla luodaan ScriptableObject ja määritellään funktio (**Kuva 28**).

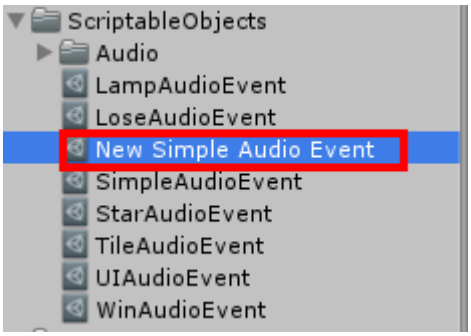
```
using UnityEngine;

public abstract class AudioEvent : ScriptableObject
{
    public abstract void Play(AudioSource source);
}
```

Kuva 27. ScriptableObjectin Audioevent abstrac-luokka.

```
using UnityEngine;

[CreateAssetMenu(menuName = "Audio Events/Simple")]
public class SimpleAudioEvent : AudioEvent
{
    public AudioClip clip;
    public float volume = 1.0f;

    public override void Play(AudioSource source)
    {
        source.clip = clip;
        source.volume = volume;
        source.Play();
    }
}
```

Kuva 28. ScriptableObjectin SimpleAudioEvent-luokka.

SimpleAudioEvent-luokan luonnin jälkeen Unity-editoriin ilmestyy Assets/Create/Audio Eventsin alle “Simple”-niminen ScriptableObject. Klikkaamalla sitä, Unity luo Project-kansioon kohdan “New Simple Audio Event”. Tälle annetaan nimi ja täytetään sille tiedot, eli Clip (äänitiedosto) sekä Volume (äänen voimakkuus). Kuvassa 29 nähdään esimerkki kaikista ScriptableObjecteista, joita käytin pelissä.



Kuva 29. Uuden AudioEvent ScriptableObjectin luominen.

ScriptableObjectin käyttäminen tapahtuu seuraavasti; luodaan esimerkiksi AudioManager.cs luokan sisälle SimpleAudioEvent-referenssi (**Kuva 30**).

```
public SimpleAudioEvent StarAudioEvent;
```

Kuva 30. SimpleAudioEvent-luokan referenssi.

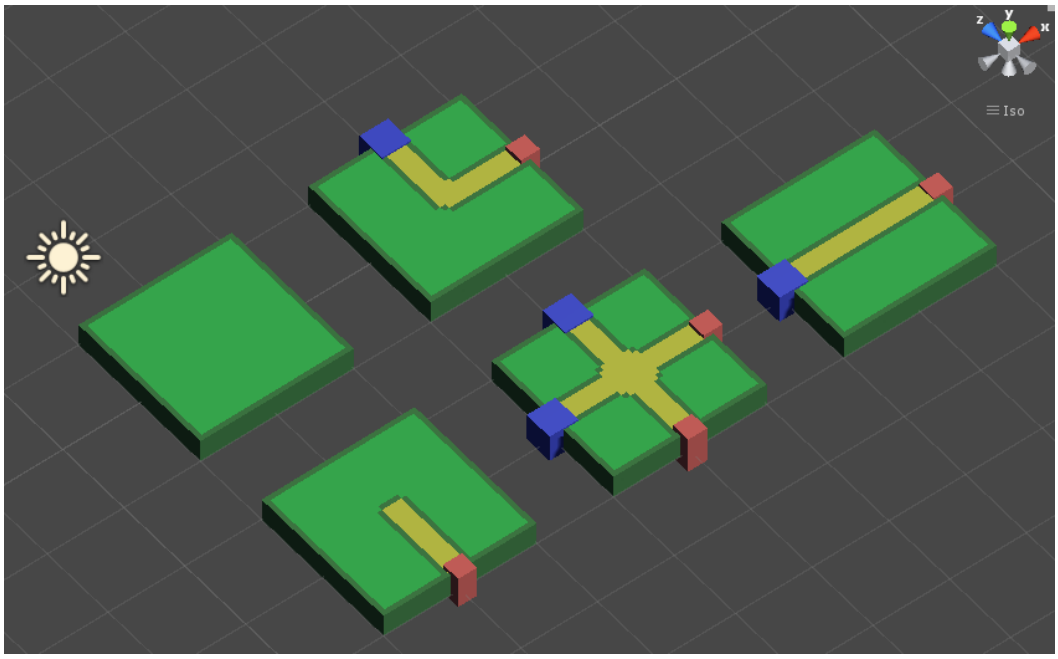
Ääniefektin toistamiseksi kutsutaan seuraava metodi (**Kuva 31**).

```
public void PlayStarSound()
{
    StarAudioEvent.Play(sfxSource);
}
```

Kuva 31. Ääniefektin toistaminen.

5.2 Kenttien suunnittelu

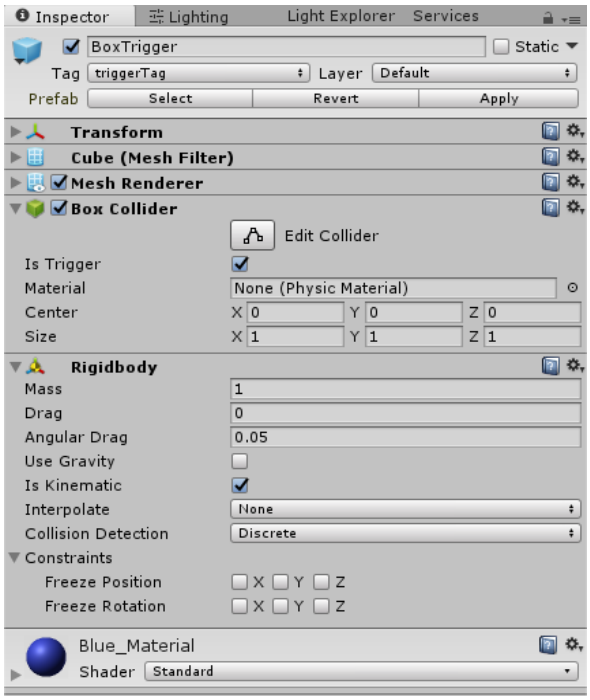
Kenttiä halusin suunnitella siten, että muutamalla pelilaatalla voisi rakentaa erikokoisia ja eri tyyppisiä kenttiä helposti. Laattoja on yhteensä viisi kappaletta ja kaikki ovat erilaisia. Kuvassa 32 nähdään kaikki laatat, joita pelissä käytetään.



Kuva 32. Kaikki pelilaatat.

Laatoissa on kiinni kaksi pientä kuutiota, punainen ja sininen kuutio. Kuutiot edustavat kytkimiä. Kytkimien avulla laatat tunnistavat toisensa ja siten aktivoivat energian.

Sinisessä kuutiossa on kaksi tärkeää komponenttia, jotta kytkimet toimisivat oikein. Tähän tarvittiin Box Collider-komponentti, jossa ”is trigger”-kohta on aktiivinen, Rigidbody-komponentti, jossa ”use gravity” ei ole aktiivinen ja ”is kinematic” on aktiivinen. Molemmat ovat fysiikkaan liittyviä komponentteja. Sen lisäksi tarvitaan Tag. Tässä Tag-tunnistus on nimeltään ”triggerTag”. Kuvassa 33 nähdään esimerkki komponentista.



Kuva 33. Sininen kytkin.

Punainen kuutio edustaa kytkimen skriptauksen osuutta nimeltä TileTriggerScript.cs. Jotta kytkimet voisivat tunnistaa toisensa, tarvitaan tässäkin peliobjektissa Box Collider. Koodissa käytetään OnTriggerEnter-funktiota ja kytkimet aktivoituvat Tagin perusteella, jos peliobjektin collider Tag on sama kuin ”triggerTag”, niin funktiot suoritetaan (**Kuva 34**).

```
private void OnTriggerEnter(Collider other)
{
    int i = 0;
    if (other.gameObject.CompareTag("triggerTag"))
    {
        if(other.GetComponentInParent<Tile>() != null)
        {
            other.GetComponentInParent<Tile>().Active = true;
            //Debug.Log("Name of the parent: " + other.gameObject.GetInstanceID().ToString());
            //Debug.Log("Name of the parent: " + other.gameObject.name + " : " + (i+1));
        }
        else
        {
            Debug.Log("Tile in Parent IS NULL!");
        }

        i++;

        allTilesAreActive = GameManager.Instance.CheckforCurrectTileOrderToWin();

        #if UNITY_EDITOR
        Debug.Log("All tiles are: " + allTilesAreActive);
        #endif

        int totalTouchCount = UIManager.Instance.GetTotalInputCount();

        //if all times are active and total move count are more/equal then 0
        if (allTilesAreActive && totalTouchCount >= 0)
        {
            ActiveEnergy();
        }
    }
}

public void ActiveEnergy()
{
    //active Energy movement
    EventManager.Instance.PostNotification(EVENT_TYPE.ENERGY_MOVEMENT, this, true);
    //active all energy particlesystems effects in scene
    EventManager.Instance.PostNotification(EVENT_TYPE.ACTIVE_ENERGY_PARTICLES, this, true);
    //disable input
    EventManager.Instance.PostNotification(EVENT_TYPE.GAME_INPUT, this, false);
    //Stop Time Count
    EventManager.Instance.PostNotification(EVENT_TYPE.TIMER, this, false);
}
```

Kuva 34. Punaisen kytkimen OnTriggerEnter-funktio.

Kaikissa laatoissa, jotka yhdistetään toisiinsa, täytyy olla Tile.cs skripti. Kuvassa 35 nähdään koodi.

```
using System;
using UnityEngine;

public class Tile : MonoBehaviour
{
    [SerializeField]
    private bool active = false;

    public bool Active
    {
        get { return active; }
        set { active = value; }
    }
}
```

Kuva 35. Tile.cs skripti.

Tile.cs skriptin tehtävä on antaa tietoa GameManager.cs:lle siitä, ovatko kaikki laatat aktiivisia. Jos ovat, niin energia pääsee kulkemaan esimerkiksi aurinkopaneeleista taloihin ja tällä tavalla kenttä saadaan läpäistyä. Kuvassa 36 nähdään GameManager.cs-luokassa funktion CheckforCurrentTileOrderToWin-koodi.

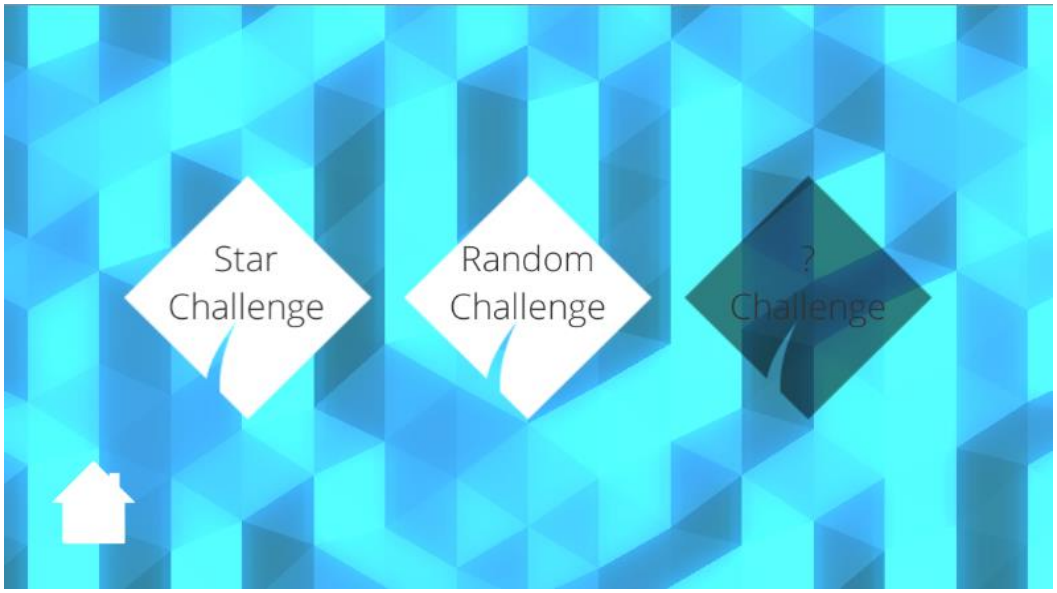
```
public bool CheckforCurrectTileOrderToWin()
{
    for (int i = 0; i < tileGameObjects.Length; i++)
    {
        if ((tileGameObjects[i].GetComponent<Tile>().Active).Equals(false))
        {
            return false;
        }
    }
    return true;
}
```

Kuva 36. CheckforCurrentTileOrderToWin-funktio.

5.3 Näkymät - Scenes

Unity scenes edustaa kenttänäkymiä. Näkymiä Green Puzzle Cityssä ovat; Main-Menu, SelectLevel, Credits, HowToPlay, Star Challenge Level_1 – Level_10 ja Random Challenge Random_Level_1 – Random_Level_3. Seuraavissa kuvissa nähdään pelin näkymät.

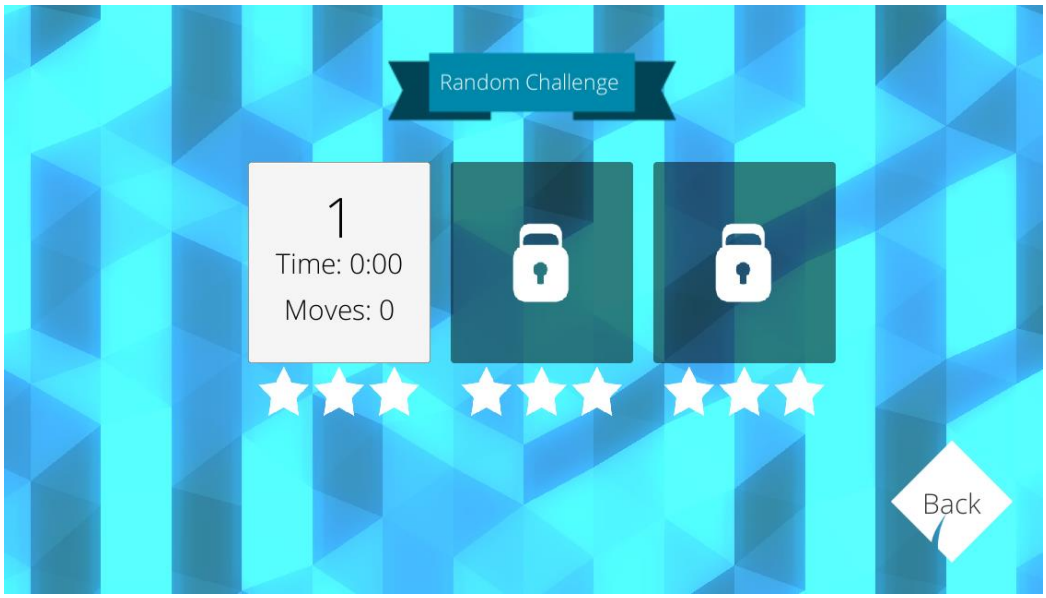
Kuva 37. MainMenu-näkymä.



Kuva 38. SelectLevels-näkymä.

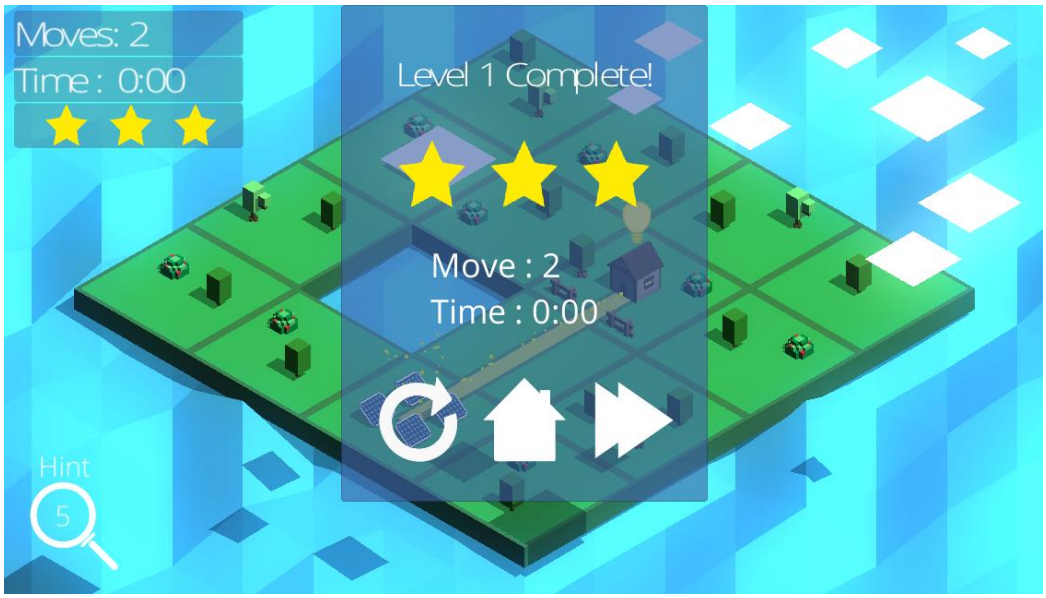
Kuva 39. Star Challenge Levels-näkymä.

Kuva 40. Star Challenge Level - 1.



Kuva 41. Random Challenge-näkymä.

Kuva 42. Random Challenge Level – 1.



Kuva 43. Kenttä 1 suoritettu.

Kuva 44. Kenttä 1 epäonnistui.

5.4 Pelidatan tallennus

5.4.1 JsonUtility

JsonUtility /5/ on hyödyllinen funktio silloin, kun käytetään JSON-tiedostoon tallennusta. JsonUtilityn staattiset funktiot ovat FromJson, FromJsonOverwrite ja ToJson. Tässä pelissä tarvitaan ainoastaan ToJson- ja FromJson-funktioita.

ToJson-funktion avulla voidaan muuttaa objektit JSON-tiedostoon. Luokassa NormalLevelData on erilaisia muuttujia, jotka tullaan tallentamaan JSON-tiedostoon. Jotta luokka voidaan tallentaa JSON-muotoon, niin luokka täytyy merkata ”Serializable”:ksi (**Kuva 45**).

```
[Serializable]
public class NormalLevelData
{
    public string levelNumber;
    public bool levelLock = false;
    public string time;
    public int moves;
    public bool[] stars = { false, false, false };
}
```

Kuva 45. NormalLevelData-luokka.

Tallennus tapahtuu seuraavasti (**Kuva 46**.).

```
//object to json
json = JsonUtility.ToJson(gameData, true);

//encrypt data
byte[] encryptedData = rijndael.Encrypt(json, JSON_ENCRYPTED_KEY);

//save data to file
File.WriteAllBytes(dataPath, encryptedData);
```

Kuva 46. Pelidatan tallennus tiedostoon.

FromJson-funktion avulla voidaan luoda JSON-muoto objektiksi. Eli se data, joka on tallennettu JSON-muotoon, voidaan tuoda takaisin objektimuotoon, tehdä muutokset siihen ja tallentaa se takaisin JSON-muotoon.

Muuttaminen JSON-muodosta objektiksi tapahtuu seuraavasti. ”If”-lauseessa kysytään onko tiedostopolku jo olemassa. Jos on, niin siirretään tiedostossa olevat bytet uuteen byte-taulukkoon. Dekryptataan kaikki bytet, muutetaan ne string-muotoon ja data muutetaan objektiksi. Tässä tilanteessa meidän objekti on GameData. Lopuksi lähetetään data eteenpäin.

```
if (File.Exists(dataPath))
{
    //read all bytes to array
    byte[] decryptData = File.ReadAllBytes(dataPath);

    //Decrypt Byte to string
    string jsonFromFile = rijndael.Decrypt(decryptData, JSON_ENCRYPTED_KEY);

    //json data to object
    GameData data = JsonUtility.FromJson<GameData>(jsonFromFile);
    return data;
}
else
{
    Debug.Log("Cannot load path " + dataPath + "\n" + " File name: " + FILE_NAME);
    return null;
}
```

Kuva 47. Muutetaan JSON-data objektiksi.

5.4.2 AES kryptaus (RijndaelManager)

Koska tallennan kaikki pelidatat JSON-tiedostoon, pelissä on se vaara, että pelaajat voivat avata JSON-tiedoston, joka on tekstitiedosto, ja tehdä muutoksia siihen. Tämän takia kryptataan pelidata .dat-tiedostoon, jotta näin ei kävisi.

Datan kryptaukseen käytetään AES (Advanced Encryption Standard /3/, alkuperäiseltä nimeltä Rijndael). .NET Frameworkista löytyy ”namespace”-nimellä System.Security.Cryptography, joka sisältää RijndaelManaged-luokan /4/. Microsoftin sivuilta löytyi tähän aiheeseen liittyvät ohjeet ja esimerkkikoodi, joka on erittäin sopiva tämän ongelman ratkaisuun. Sivuilta löytyi myös ohjeet kuinka käytetään

tätä luokka. Käytän tätä esimerkkikoodia Green Puzzle Cityssä ja muokkaan sitä tarpeen mukaan.

Encrypt-funktio vastaanottaa kaksi muuttujaa, ”original” ja ”key”. Molemmat ovat string-tyyppisiä muuttujia. Original-muuttuja on JSON-data, joka muutetaan ja kryptataan EncryptStringToByte-funktiolla. Key-muuttuja antaa oikeudet muuttaa tiedostoa. Ilman oikeaa key-muuttuja-arvoa, tallennus ei onnistu. Kuvassa 48 nähdään keyn string-muuttuja.

```
private readonly string JSON_ENCRYPTED_KEY = "AdsX2#s15d4aS6?d4/(¤%&/()fd'325s"; //32 chars
```

Kuva 48. Key-muuttuja.

Kuvassa 49 nähdään Encrypt-funktio.

```
public byte[] Encrypt(string original, string key) // key must be 32chars
{
    byte[] encrypted = null;

    try
    {
        byte[] iv = Encoding.ASCII.GetBytes(IV);
        byte[] keyBytes = Encoding.ASCII.GetBytes(key);

        using (RijndaelManaged myRijndael = new RijndaelManaged())
        {
            myRijndael.Key = keyBytes;
            myRijndael.IV = iv;

            encrypted = EncryptStringToBytes(original, myRijndael.Key, myRijndael.IV);
        }
    }
    catch (Exception e)
    {
        Debug.LogFormat("Error: {0}", e.Message);
    }
    return encrypted;
}
```

Kuva 49. Encrypt-funktio.

Decrypt-funktio on melkein samanlainen kuin Encrypt-funktio. Ainoa ero on, että tässä Decrypt-funktiossa, funktio vastaanottaa ”soup”-nimisen bytetaulukon ja

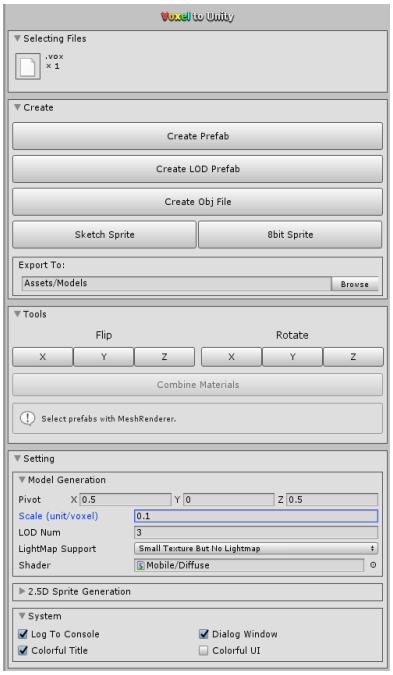
string-muuttujan nimeltä ”key”. Tässä funktiossa muutetaan soup-bytetaulukko string-muotoon ja lisätään se outstring-muuttujaan. Muutettu outstring-muuttuja palautetaan takaisin (**Kuva 50**).

```
public string Decrypt(byte[] soup, string key) //key must be 32chars
{
    string outString = "";

    try
    {
        byte[] iv = Encoding.ASCII.GetBytes(IV); //16
        byte[] keyBytes = Encoding.ASCII.GetBytes(key);

        using (RijndaelManaged myRijndael = new RijndaelManaged())
        {

            myRijndael.Key = keyBytes;
            myRijndael.IV = iv;

            outString = DecryptStringFromBytes(soup, myRijndael.Key, myRijndael.IV);
        }
    }
    catch (Exception e)
    {
        Debug.LogFormat("Error: {0}", e.Message);
    }

    return outString;
}
```

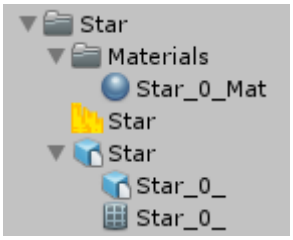
Kuva 50. Decrypt-funktio.

5.5 Voxel To Unity

Voxel To Unityn avulla muutetaan kaikki vox-päätteiset tiedostot obj-tiedostoksi. Editorin yläpalkista Tools/Voxel To Unity, saadaan työkalu auki (**Kuva 51**). Sen jälkeen tuodaan vox-tiedostot Unity projektin sisälle, ja valitaan tiedosto, joka halutaan muuttaa obj-tiedostoksi. Oletuksena mallin koko on aika iso. Muutin asetuksia siten, että scale (unit/voxel) arvo on 0.1, niin sain sopivan kokoisen mallin. Loin uuden objektin klikkaamalla Create Obj file (**Kuva 52**). Voxel To Unity luo meille kuvanmukaisen hakemiston, tässä tapauksessa meillä Star-hakemisto ja sen alihakemistot. Tällä tavalla sain optimoitua obj-mallin, joka on sopiva mobiilipelille.



Kuva 51. Voxel To Unity-editori.



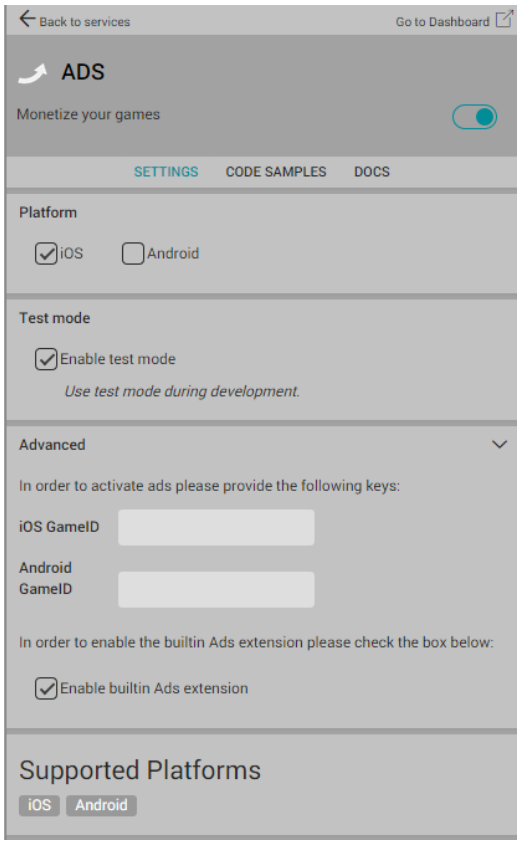
Kuva 52. Voxel To Unityn luoma hakemisto.

5.6 Pelimusiikki ja ääniefektit

Unity Asset Store on erittäin hyvä materiaalilähde, jos aika tai taito ei riitä johonkin pelin osaan. Minulla on hyvin vähän kokemusta musiikin tai ääniefektien tekemisestä, joten olen ladannut ne Unity Asset Storesta (Unity Asset Store /6/.).

5.7 UnityAds

UnityAds-integrointi tapahtuu seuraavasti. Unity editorissa Windows/Servises alta löytyy ikkuna, jossa aktivoidaan kaikki Unityn palvelut. UnityAdsin kohdalla aktivoidaan UnityAds services, tämän jälkeen Unity asentaa taustalla tarvittavat komponentit. Platformin eli alustan kohdalla valitaan iOS. Otetaan käyttöön ”Enable test mode” ja advanced-kohdassa syötetään iOS GameID, jonka Unity itse luo sinulle (**Kuva 53**). Ikkunan oikeassa yläkulmassa on Go to Dashboard-painike, jolla päästään Unity Servicen webappiin. Listalta valitaan projektin nimi ja siitä siirrytään Platform-eli alustavalikkoon, tähän kohtaan valitsin Apple App Store. Tämän jälkeen voidaan valita halutaanko käyttää pelissä normaaleja videomainoksia, jotka voi ohittaa 5 sekunnin jälkeen, vai palkintovideoita, joita ei voi ohittaa, mutta mainoksen jälkeen pelaaja palkitaan. Tähän pelin valitsin palkintovideot, koska pelissä voi ansaita vihjepelimerkkejä (Hint) ja niiden avulla voidaan saada vinkkejä kuinka ratkaistaan kenttiä. /11/



Kuva 53. Unity Ads-asetukset.

Pelissä on kaksi tapaa ansaita vihjeitä. Ensimmäinen vaihtoehto on, että päävalikosta, asetuksen alta, löytyy painike, josta voi saada yhden Hint-pelimerkin katsomalla mainosvideon. Toinen tapa on kun kenttä on suoritettu, ilmestyy sen jälkeen Earn Hint-painike ruudun oikeaan alakulmaan ja painamalla sitä, aukeaa mainosvideo, jonka jälkeen saa yhden Hint-pelimerkin.

5.7.1 UnityAds-kooditoteutus

Koodin Awake-funktiossa alustetaan Advertisement-luokka syöttämällä string- ja boolean-muuttujat. String-muuttuja on gameId ja boolean on true, jos peli on testausvaiheessa. ShowAd-funktiossa on ShowOption-objekti, jolla pyydetään video-

mainos. Mainos pyydetään seuraavasti; options.resultCallBack- ja if-lauseissa odotetaan kunnes mainosvideo on nähty loppuun. AdCallbackHandler-funktiossa case ”ShowResult.Finish” tulee aktiiviseksi kun mainosvideo on katsottu loppuun. Tämän jälkeen palkitaan pelaaja yhdellä Hint-pelimerkillä. Kuvassa 54 Unity Ads-koodin toteutus.

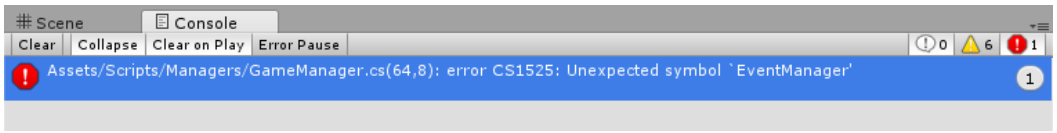
```
using UnityEngine;
using System.Collections;
using UnityEngine.Advertisements;

public class AdManager : MonoBehaviour
{
    [SerializeField]
    string gameID = [redacted];
    [SerializeField]
    int rewardQty = 1;

    private static AdManager instance = null;
    public static AdManager Instance...

    void Awake()
    {
        Advertisement.Initialize(gameID, true);
    }

    public void ShowAd(string zone = "")
    {

#if UNITY_EDITOR
        StartCoroutine(WaitForAd());
#endif

        if (string.Equals(zone, ""))
            zone = null;

        ShowOptions options = new ShowOptions();
        options.resultCallback = AdCallbackhandler;

        if (Advertisement.IsReady(zone))
            Advertisement.Show(zone, options);
    }

    void AdCallbackhandler(ShowResult result)
    {
        switch (result)
        {
            case ShowResult.Finished:
            #if UNITY_EDITOR
                Debug.Log("Video completed. User rewarded " + rewardQty + " credits.");
            #endif
                //Show reward panel
                EventManager.Instance.PostNotification(EVENT_TYPE.SHOW_HINT_REWARD_PANEL, this, null);

                //Add 1 Hint reward to totalHint Count
                EventManager.Instance.PostNotification(EVENT_TYPE.ADD_HINT_POINTS, this, rewardQty);

                break;
            case ShowResult.Skipped:
                Debug.LogWarning("Video was skipped.");
                break;
            case ShowResult.Failed:
                Debug.LogError("Video failed to show.");
                break;
        }
    }
```

Kuva 54. UnityAds-koodin toteutus.

6 TESTAUS

6.1 Koodin testaus editorissa

Koodin testaus onnistui parhaiten käyttämällä Unity-editoria. Jos koodissa oli esimerkiksi syntaksivirheitä (**Kuva 55**), Console-ikkunaan ilmestyy virheilmoitus. Tässä tapauksessa ongelma on polun Assets/Scripts/Managers-hakemiston alla, skriptissä nimeltä GameManager.cs, koodirivien 8 – 64 välillä. Kaksoisklikkaamalla virheilmoitusta, avautuu Visual Studio-editori, ja kursori menee siihen kohtaan, jossa virhe ilmenee. Tällä tavalla on helppo ratkaista pieniä ongelmia. Jos koodissa on isompia ongelmia, joihin ei helposti löydy ratkaisua, niin siinä tapauksessa voidaan käyttää Visual Studion omaa debuggaus-työkalua.

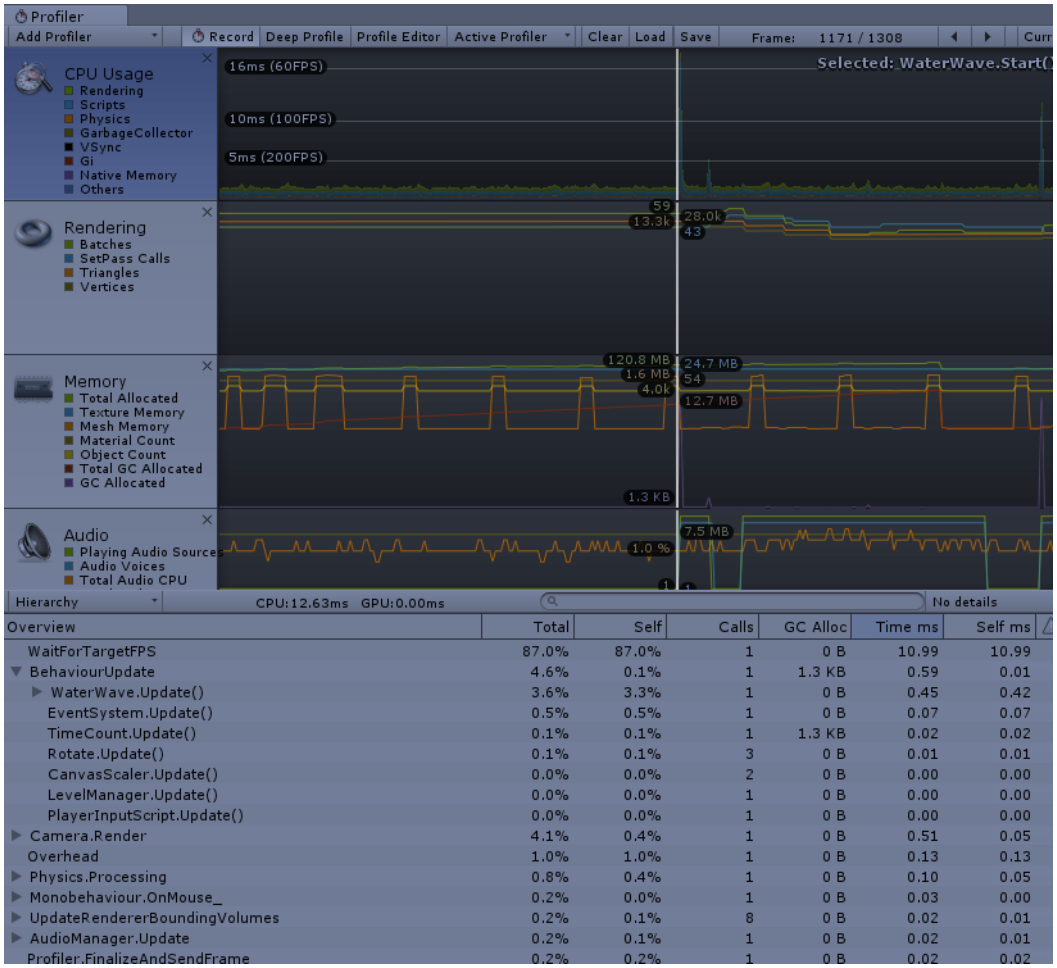


Kuva 55. Console-ikkunan virheilmoitus.

6.2 Unity Profiler

Unity Profilerin avulla voidaan optimoida pelin toiminta. Profiler antaa kokonaisuudessaan kuvauksen siitä kuinka kauan aikaa on mennyt tiettyyn pelitoimintaan sekä -alueisiin. Esimerkiksi kuinka kauan aikaa on mennyt prosentuaalisesti vaikka renderoimiseen, pelifysiikkaan, animointiin ja jopa pelilogiikkaan. Sillä voi myös analysoida GPU, CPU tai muistinsuorituskykyä.

Unity Profiler käynnistyy kun Profiler väli-ikkuna on auki (**Kuva 56**) ja äänitysnappi aktiivinen. Painamalla Play-painiketta Unity-editorin sisällä, Profiler alkaa tallentaa pelin suorituskykydataa ja se näkyy aikajanana. Painamalla Profilerin aikajanaa peli pysähtyy ja voidaan tutkia ja nähdä missä ongelma piilee. Yleensä jos Calls, GC alloc ja Time ms:ssa ilmenee isoja lukemia, niin silloin jokin vie liikaa resursseja.

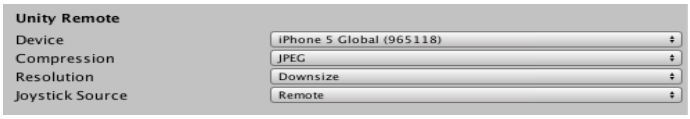


Kuva 56. Profiler aikajanat.

6.3 Unity Remote 5

Unity Remote on sovellus, joka on saatavilla sekä iOSlle että androidille. Sovelluksen tarkoitus on helpottaa pelin testausta mobiililaitteilla. Unity Remoten avulla pelejä ei tarvitse kääntää mobiilialustoille testattavaksi, vaan voidaan peilata editorin Game-ikkunaa suoraan Unity Remoten 5 ruudulle. Unity Remoten käyttämiseksi täytyy kytkeä laite johdolla tietokoneeseen.

Unity asetuksissa Edit /Project settings/Editor alta löytyy Editor asetukset, jossa on Device-valikko. Valikon alta ilmestyy laitteen nimi. Editorista painamalla Play päästään testaamaan peliä Unity Remoteen.

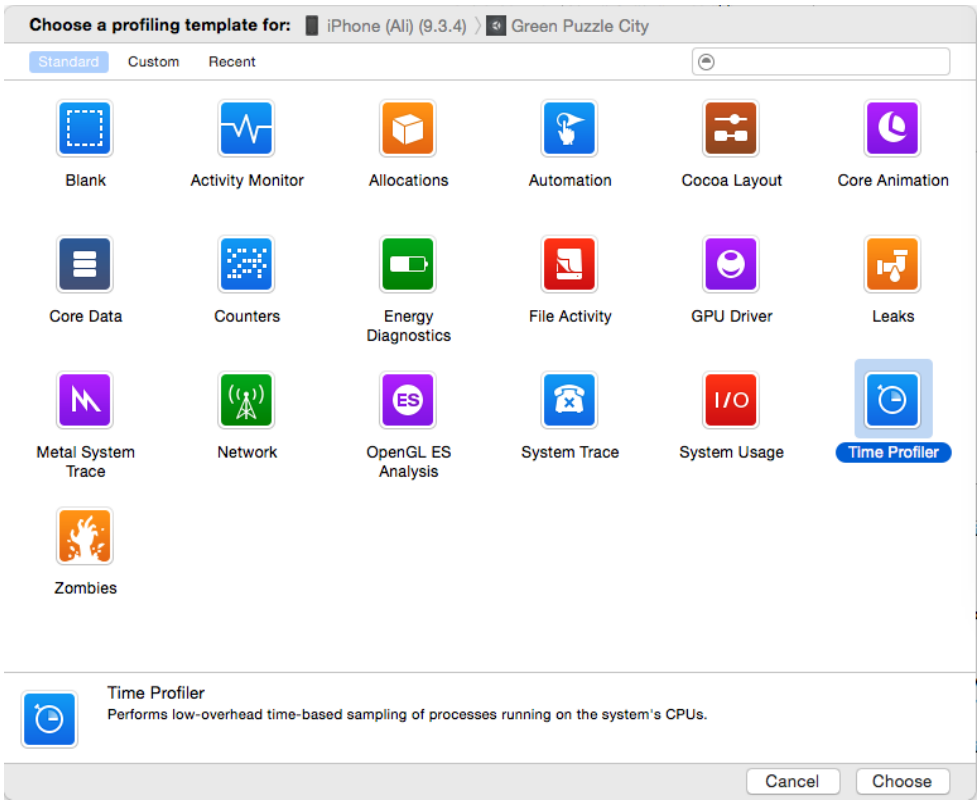


Kuva 57. Unity Remote 5:n Editor asetusvalikko.

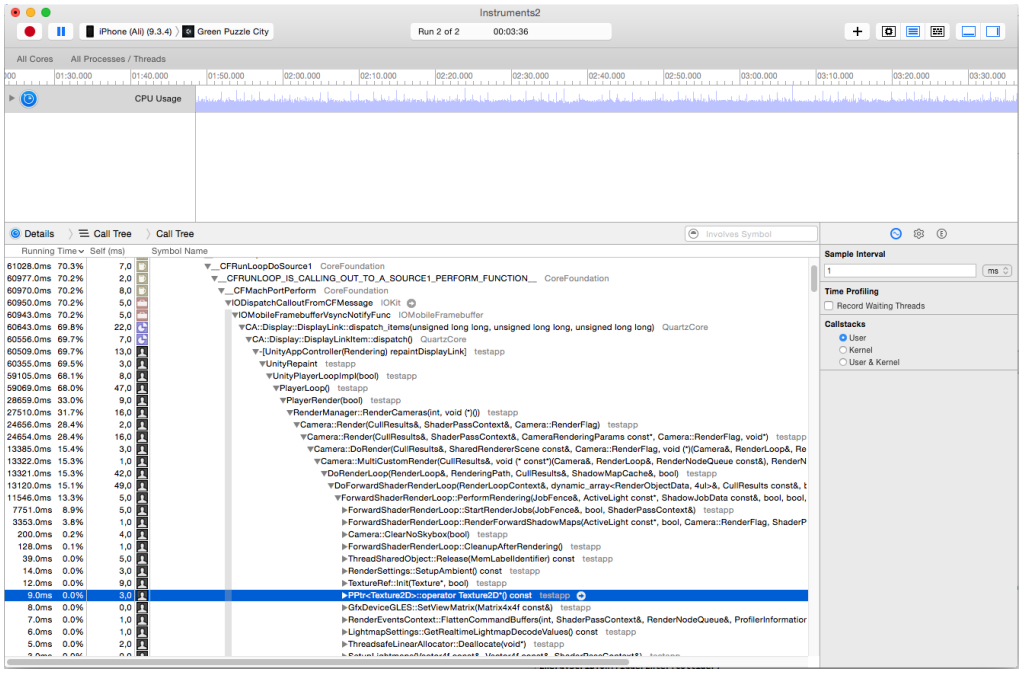
6.4 Xcode Instrument

On olemassa useita hyviä työkaluja, joilla voidaan analysoida pelin suorituskykyä. Mielestäni yksi parhaimmista iOS:lle on Instruments. Instruments on Xcoden suorituskyvyn analysointityökalu. Jotta voidaan käyttää tätä työkalua, niin täytyy ensin kääntää peli iOS-alustalle seuraavalla asetuksilla; Build asetusten alta löytyy Development Build- ja Script Debugging, nämä pitää olla käytössä. Kun ollaan käännetty ja rakennettu peli iOS-laitteeseen, niin voidaan Xcoden yläpalkin valikosta valita Product/Profile, josta päästään Instruments-editoriin (**Kuva 58**).

Pelin testaukseen Instrumentsissa tuli käytettyä paljon Time profileria, ja yleensä suurin osa ongelmista ilmeni StartUnityn, UnityLoadApplicationin ja PlayerLoopin välillä. Kuvassa 59 nähdään Instruments-aikajanalla.



Kuva 58. Instrument-editori.

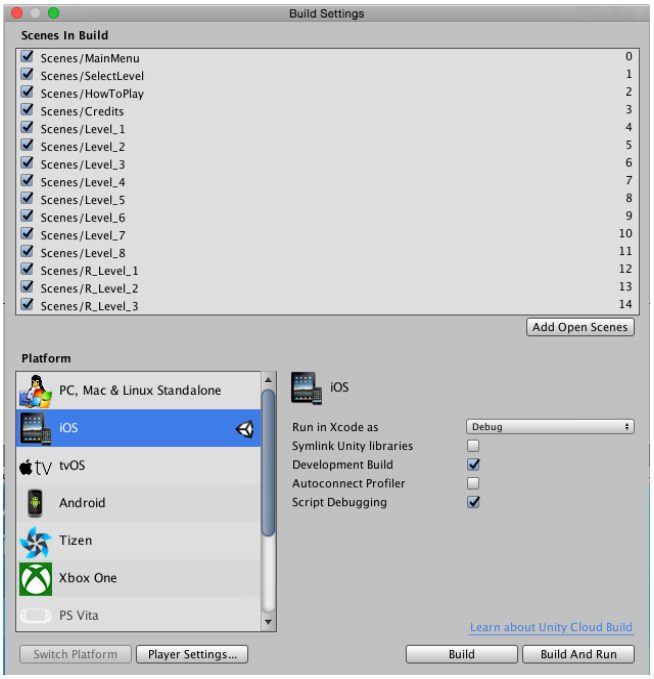


Kuva 59. Instrument-aikajana.

6.5 IOS-laite

Mobiililaitteella testaamista oli välttämätöntä tehdä useasti, sillä lopputulos ei aina ollut se miltä näytti editorissa tai Unity Remotessa. Käyttöliittymät eri laitteille (kuten iPhone ja iPad) ovat aina erinäköisiä tai -kokoisia. Green City Puzzlen testaukset tein iPhone 5 iOS versiolla 7.0 ylöspäin.

IOS-laitteeseen kääntämiseen ja rakentamiseen tarvitaan Mac-tietokone sekä Xcode- ja Apple developer-lisenssit. Kääntäminen tehdään seuraavasti; Unity editorista File/Build-asetukset painikkeesta avautuu Build Settings-ikkuna.

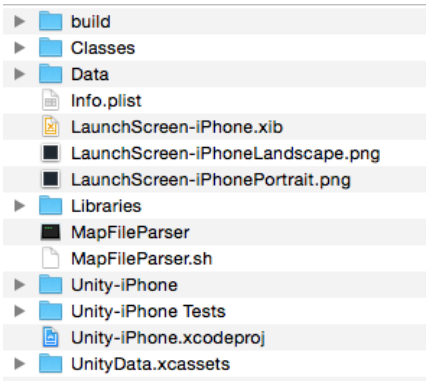


Kuva 60. Unityn Build Settings-ikkuna.

Platformin kohdalla valitaan iOS ja run in Xcode as ”Debug” valittuna. Valitaan myös päälle Development Build sekä Script Debugging. Build-asetuksien vasemmasta alareunasta saadaan Player Settings-ikkuna auki. Tässä vaiheessa täytyy antaa kaikki iOS:n liittyvät tiedot, ja nämä asetukset määrittävät sen mille laitteille, mikä versio, käännetäänkö testaus simulaattoriin vai itse laitteeseen ja paljon muuta asioita, jotka tulee ottaa huomioon ennen pelin kääntämistä. Yksi tärkein valikko

on Bundle Identifier, tämän avulla voidaan kääntää peli iOS-laitteeseen. Bundle Identifier voidaan luoda Applen Developer-verkkosivulta. Bundle Identifier näyttää seuraavalta: com.companyname.appname. Kun nämä kaikki asetukset ovat valmiita, niin peli on valmis kääntämiseen. Build asetuksien ikkunassa on ”Build”-painike jolla saadaan käännettyä peli. Kääntäminen kestää yleensä noin 5-10 minuuttia, pelin koosta riippuen.

Kun käännös on valmis, macOS finderista avautuu hakemisto, jossa on Xcode-projekti. Projekti avataan klikkaamalla Unity-iPhone.xcodeproj-tiedostoa (**Kuva 61**).



Kuva 61. Unityn luoma Xcode-projekti.

Xcode-editorin sisällä valitaan General - Identifyn kohdalle tiimin nimi, jonka Xcode vaati. Tarkistetaan Bundle Identifier, että se on oikein. Siirrytään Build-asetuksien puolelle ja kohtaan Code Signing. Koska tämä on testikäännös iOS:lle, niin meidän ei tarvitse muuta kuin vaihtaa PROVISIONING_PROFILE kohta automaattiseksi. Muuta tässä vaiheessa ei tarvitse tehdä. Sitten yläpalkista valitaan ”Product” ja ”Clean”, ja uudelleen ”Product Build for Testing”. Tämän pelin rakentaminen laitteeseen kesti noin 6-8 minuuttia, jonka jälkeen peli avautuu laitteeseen testattavaksi.

7 JATKOKEHITYS

Peli tuntuu hauskalta ja haastavalta kun pääsee pelin sisään ja kentissä eteenpäin. Pelissä on vielä paljon parannettavaa ja lisättävää, mutta joka tapauksessa tulen julkaisemaan tämän pelin App Storessa jossain vaiheessa tänä vuonna. Asioita, joita haluaisin vielä lisätä tähän peliin ennen julkaisua, on lisätä kenttien määrää ja optimoida peliä enemmän. Tällä hetkellä yhden kentän tekemiseen menee jonkin verran aikaa. Tämän ongelman ratkaisemiseksi minun täytyy kehittää työkalu.

Unityn sisään pystyy rakentamaan erilaisia työkaluja Unity editori skriptauksen avulla. Tällaiseen peliin, jossa on paljon kenttiä, olisi fiksua lähteä kehittämään karttaeditori-työkalu, joka helpottaisi jälkeenpäin kenttien suunnittelua sekä kehitystä.

IOS Game Center-integroinnin tulen toteuttamaan myöhemmin, koska tällä hetkellä en ole vielä päättänyt kuinka haluan pelaajien ennätyksien sekä saavutuksien näkyvän. Game Center-integrointi on hyvin helppoa, joten sen voi tehdä myöhemmässäkin vaiheessa.

8 YHTEENVETO

Tämän projektin tavoitteena oli luoda yksinkertainen, hauska ja visuaalisesti hieno iOS-mobiilipeli, jonka aiheena oli ”puhdas energia”. Kaikki odotukset tämän pelin kohdalla täyttyivät ja ehkä visuaalisesti hieman paremminkin kuin olin toivonut.

Pelituntuma onnistui hyvin, mutta aina on varaa parantaa tuntumaa mobiililaitteille. Heti projektin alussa tuli vastaan vaikeuksia, siitä miten kentän palikat linkitetään toisiinsa ja kuinka tarkistetaan voitto. Usean prototyypin jälkeen pääsin yksinkertaiseen ratkaisuun. Tämä ratkaisu tulee helpottamaan työtä jatkokehityksessä, mikäli haluan kehittää peliä pidemmälle. Vaikein osa tässä projektissa oli pelin tallentaminen iOS-laitteeseen. Pelidatan kryptauksesta selvää ottaminen oli haastavaa johtuen siitä, että tämä oli ensimmäinen kerta kun toteutin sen. Kuitenkin lopputulos oli hyvä ja tallennus onnistui.

C#-ohjelmointikielen osalta tuli opittua uusia asioita. Se mikä helpotti tämän projektin koodaamista, oli managerien ja eventien kanssa työskentely. Huomasin jälkeenpäin miten paljon eventit auttoivat kun pelistä tuli laajempi. Yleensä koodaaminen tuntuu sitä vaikeammalta mitä suurempi pelistä tulee, mutta tässä tapauksessa se oli yllättävän yksinkertaista.

3D-Voxelin kanssa työskentely oli hyvin helppoa ja hauskaa. Mallien tekemisessä ei ollut vaikeuksia ja yritin tehdä niistä mahdollisimman yksinkertaisia. Uskon, että mallien yksinkertaisuus teki pelistä visuaalisesti hienon.

Tällä hetkellä en ole suunnitellut tekeväni versioita muihin alustoihin, esimerkiksi Androidille. Jos peli saa hyvin latauksia, niin sitten voisin harkita pelin kääntämistä myös Android-alustalle, mutta ennen sitä peliin täytyy lisätä jonkin verran kenttiä ja parantaa pelin laatua.

Tämän projektin tekeminen vei paljon aikaa, mutta tuli opittua samalla myös paljon. Unity on hyvä työkalu ja tulen jatkossakin käyttämään sitä.

LÄHTEET

1. Unity, Viitattu, 04.04.2017
   https://unity3d.com/
2. Signleton, Viitattu 08.04.2017
   https://en.wikipedia.org/wiki/Singleton_pattern
3. Mikä on AES kryptaus , Viitattu 05.04.2017
   https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
4. Microsoftin RijndaelManaged luokka, Viitattu 05.04.2017
   https://msdn.microsoft.com/en-us/library/system.security.crypto-graphy.rijndaelmanaged(v=vs.110).aspx
5. JsonUtility, Viitattu 21.04.2017
   https://docs.unity3d.com/ScriptReference/JsonUtility.html
6. Unity Asset Store, Viitattu 11.04.2017
   https://www.assetstore.unity3d.com/en/
7. MagicaVoxel, Viitattu 14.04.2017
   https://ephtracy.github.io/
8. SciptableObject, Viitattu 22.04.2017
   https://docs.unity3d.com/ScriptReference/ScriptableObject.html
9. EventManager, Viitattu 19.04.2017
   http://wiki.unity3d.com/index.php?title=CSharpNotificationCenter
10. Execution Order of Event Functions, Viitattu 22.04.2017
    https://docs.unity3d.com/Manual/ExecutionOrder.html
11. Unity Ads, Viitattu 23.04.2017
    https://unityads.unity3d.com/help/monetization/getting-started