



# Cocos2D-pelin käännös Unity-alustalle

Ville Kaikkonen

OPINNÄYTETYÖ  
Joulukuu 2021

Tietojenkäsittelyn tutkinto-ohjelma, Tradenomi  
Web-palvelut

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tietojenkäsittelyn tutkinto-ohjelma, Tradenomi  
Web-palvelut

TEKIJÄN SUKUNIMI, ETUNIMI: Kaikkonen, Ville  
Cocos2D -pelin käännös Unity -alustalle

Opinnäytetyö 30 sivua  
Joulukuu 2021

---

Opinnäytetyössä käännettiin Cocos2D-pelimootorilta mobiililaitteille suunniteltu videopeli Unity-pelimootorille. Tavoitteena oli selvittää, mitä muutoksia peliin, sen koodiin ja muihin resursseihin täytyy tehdä koodikielen ja alustan vaihtuessa. Pelissä muutoksia tehtiin animaatioihin ja pelin ominaisuuksien laajuuteen. C++ ohjelmointikielellä luotu koodi muutettiin C# kielelle ja Unity:n lisäosiksi. Samalla selvitettiin, millaisia ratkaisuja piti tehdä pelimootorien toimintojen eroavaisuuksista johtuvista puutteista Unity:lla.

Työ oli BeiZ Oy:n toimeksianto, ja sen tarkoituksena oli muuttaa peli pois vanhalta pelimootorilta, jonka kehitystyö on päättynyt. Näin peliä voidaan jatkokehittää myös uudemman sukupolven laitteille. Tämä on hyödyllistä laitteiden ruutukokojen muuttuessa ja uusien ominaisuuksien kehittyessä.

---

Asiasanat: cocos2d, unity, videopeli

## **ABSTRACT**

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Business Information Systems  
Web Services

KAIKKONEN, VILLE  
Converting a Cocos2D Game to Unity Platform

Bachelor's thesis 30 pages  
December 2021

---

A videogame running on Cocos2D game engine designed for mobile devices was converted to Unity game engine in this thesis. The goal was to find out what changes had to be made to the game itself, its code as well as other resources when the programming language and the platform change. The game's animations were adjusted, and gameplay features were extended. Code made with C++ programming language was converted into C# and added into Unity's plug-in extensions. Solutions into technical problems from changing to the Unity platform were investigated at the same time.

The product was commissioned by Beiz Oy, as the game was to be converted from an old game engine, which had its active development halted. This would allow the games to be developed onto the devices of the newer generations. This is especially useful, as new screen sizes and features become available due to their advancements in technology.

---

Key words: cocos2d, unity, video game

## SISÄLLYS

1	JOHDANTO .....	6
2	PELIMOOTTORIN VAIHTO .....	7
3	PELIN OMINAISUUKSIEN DOKUMENTAATIO .....	8
3.1	Pelin ominaisuuksien dokumentaatio .....	8
3.2	Pelin pelaaminen.....	8
3.3	Kuvien ja videoiden ottaminen .....	9
4	PELIN KÄÄNTÄMINEN.....	10
4.1	Tiedostojen käyttö käännöksessä .....	10
4.2	Taustakuvien käyttö käännöksessä .....	11
4.3	Koodin kääntäminen .....	12
4.4	Pelimoottorien eroavuudet .....	12
4.5	Objekteihin ja ulkoiseen koodiin viittaaminen .....	13
4.6	Editoriohjelma .....	14
4.7	Virtuaalikoordinaatit.....	15
4.8	Animaatiot .....	16
4.8.1	Animator -käyttöliittymä .....	17
4.8.2	Koordinaattipohjainen liike.....	18
4.9	Tiedon säilytys .....	20
4.9.1	Tiedon tallentaminen verkkoon.....	21
4.9.2	Pelin tietojen tallentaminen laitteelle.....	22
4.10	Kuva-arkit .....	22
5	OMIN AISUUKSIEN LISÄÄMINEN UNITY:LLE .....	24
5.1	Partikkelit UI -elementtinä .....	24
5.2	Kielivaihtoehdot.....	25
6	PELIN JATKOKEHITYS.....	26
6.1	Leikattujen ominaisuuksien lisääminen peliin.....	26
6.2	Vanhojen ominaisuuksien parantaminen .....	26
6.3	Pelin liittäminen sovellukseen .....	27
6.4	Pelin päivittäminen uusille teknologioille .....	27
7	POHDINTA .....	28
	LÄHTEET .....	30

**ERITYISSANASTO**

Elementti	Pelimoottorissa ilmenevä komponentti, joka voi olla kuva, objekti, painike tai muu osa peliä.
Mobiililaite	Älypuhelimet, -tabletit, ja vastaavanlaiset kannettavat laitteet
Näkymä	Jokin tietty osio pelistä, kuten päävalikko
Partikkeli	Pienikokoinen kuva suuremmassa erikoisefektissä
Pelimoottori	Videopeleille suunniteltu ohjelmistokehys
Plug-in	Lisäosa ohjelmalle
TAMK	Tampereen ammattikorkeakoulu
UI	“User Interface”, eli käyttöliittymä

## 1 JOHDANTO

BeiZ Oy on kehittänyt lapsille suunnattuja opetuspelejä mobiililaitteille aikaisemmin Cocos2D -pelimoottorilla. Tällä pelimoottorilla tehdyt ohjelmat on kirjoitettu C++ -ohjelmointikielellä. Kyseinen pelimoottori on ollut pitkään käytössä videopelien kehityksessä, mutta sitä ei enää aktiivisesti jatkokehitetä. Sen sijaan kyseisen pelimoottorin kehittäjät ovat siirtyneet Cocos2D-x -nimisen uuden pelimoottorin pariin (Chukong Technologies). BeiZ Oy:llä on myös koettu vaikeuksia löytää vanhalle pelimoottorille dokumentaatiota joillekin sen ominaisuuksista, joka on vaikeuttanut jatkokehitystä yhtiön ohjelmoijien vaihtuessa. Tästä syystä BeiZ Oy otti käyttöön Unity -pelimoottorin, jota kehitetään aktiivisesti ja jolle löytyy useita lisäosia. Unity toimii sekä Javascript- että C# -ohjelmointikielillä, joista päätettiin käyttää pääasiallisesti jälkimmäistä. Tämän myötä tuli selvittää, millaisia toimenpiteitä tuli ottaa pelimoottorin ja ohjelmointikielen vaihtuessa. Työssä on käännetty Unity-pelimoottorille BeiZ Oy:n peli "Lolan Matikkajuna".

## 2 PELIMOOTTORIN VAIHTO

Cocos2D -pelimoottori julkaistiin vuonna 2010 ja sai viimeisen päivityksensä vuonna 2017. Cocos2D:n oma sivusto ei ole tämän jälkeen nähtävästi saanut päivityksiä, jonka lisäksi suuri osa sivuston linkeistä eivät ole toiminnassa. Chukong Technologies on siirtänyt huomionsa sen kehittämisestä uudempaan Cocos2D-x -pelimoottoriin, jota päivitetään vakituisemmin. Myös Cocos2D-x:n dokumentaatio on kuitenkin vailla päivitystä vuoden 2019 jälkeen (Chukong Technologies 2019). Cocos2D on keskittynyt iOS-mallin Apple Incorporated -yhtiön laitteilla toimiville käyttöjärjestelmille, mutta Cocos2D-x on laajentunut myös muille alustoille.

BeiZ Oy:n halusi uuden pelimoottorin, jota päivitetään vakituisemmin omien peliensä jatkokehitystä ajatellen. Yhtiön sisällä todettiin, että Cocos2D:n dokumentaatio oli paikoin myös vajavaista, joka vaikeutti pelien uusimista Cocos2D-x -pelimoottorille. Cocos2D ei ole yhtä suosittu kuin Unity, jonka myötä sille on myös helpompi löytää kehittäjiä kuin Cocos2D:lle: suurin osa Cocos -pelimoottoreiden kehittäjistä ovat keskittyneet Cocos2D-x -pelimoottoriin. Unity on maailman suosituin pelimoottori, sillä noin puolet julkaistuista videopeleistä maailmassa pohjautuivat tähän pelimoottoriin vuonna 2018 (Unity CEO says half of all games are built on Unity 2018).

### **3 PELIN OMINAISUUKSIEN DOKUMENTAATIO**

#### **3.1 Pelin ominaisuuksien dokumentaatio**

Videopeleillä on dokumentaatiota kuten muillakin koodeilla ja ohjelmilla. Dokumentaatio helpottaa jatkokehittämistä osoittamalla pelin ominaisuuksia ja toimintoja koodista sekä muista pelimoottorin komponenteista, joita jatkokehittäjä voi lukea ilman aikaisempaa kokemusta pelistä. Tämä on erityisen tärkeää, jos ketään alkuperäisistä kehittäjistä ei ole opastamassa jatkokehitystä omilla tiedoilla pelin kehityksestä. BeiZ Oy:n Lolan Matikkajunan alkuperäisiä kehittäjiä ei ollut työtä varten saatavilla, jonka myötä jatkokehittäjien täytyi turvautua alkuperäisen pelin dokumentaatioon.

Dokumentaatio ei aina kata pelin kaikkia ominaisuuksia, jolloin käännettävästä pelistä täytyy tehdä uutta havaintopohjaista dokumentaatiota. Seuraavaksi esitelen BeiZ Oy:lla käytettyjä metodeja Cocos2D -pelin ominaisuuksien dokumentointia varten, joita käytettiin myös tässä työssä.

#### **3.2 Pelin pelaaminen**

Pelin kaikkien kääntäjien on hyvä pelata käännettävä peliä kokonaisuudessaan saadakseen tuntuman sen tunnelmasta, funktioista sekä niiden ajoituksesta, äänimaailmasta ja yleisestä struktuurista. Tämä toimenpide helpottaa myös kääntäjien yhteydenpidossa pelin ominaisuuksista ja auttaa jatkokehittämisen suunnittelussa. Peliä pelattiin jatkuvasti uudelleen läpi pelin kehityksen aikana, silloin kun haluttiin selvittää tiettyjä yksityiskohtia pelistä, esimerkiksi nappulan sijainnista ruudulla tietyssä vaiheessa peliä. Tätä tehtiin, jos otetut videot ja kuvat eivät osoittaneet haluttua kohtausta, tai jos kyseessä olevaa ominaisuutta ei ilmennyt sillä kerralla.

Kun uudesta Unity-käännöksestä peliä saatiin pelattavissa oleva kehitysversio, myös sitä pelattiin läpi kehityksen. Tällä tavalla havaittiin vikoja ja muita ongelmia, joita sitten pystyi tarpeen tullen korjaamaan.



### **3.3 Kuvien ja videoiden ottaminen**

Koska elementtien asemia ja asentoa ei voida suoraan kopioida koordinaattien pohjalta Cocos2d-pelistä Unity-peliin, tulee pelistä ottaa kuvia. Kuvista voidaan nähdä, miten pelin elementtejä, kuten painikkeita ja käyttöliittymää, on aseteltu.

Pelilaitteelle on myös hyvä asentaa jonkinlainen ohjelma, jolla voidaan ottaa videoita pelistä animaatioiden liikkeiden ja tapahtumien välisten ajoitusten mittaamista varten. Ajoitus voidaan suurimman osan ajasta havaita suoraan pelin koodista, mutta toisiinsa nähden lomittain tapahtuvia animaatioita, peliefektejä, ja muita ominaisuuksia on hankala hahmottaa ilman visuaalista vertailua.

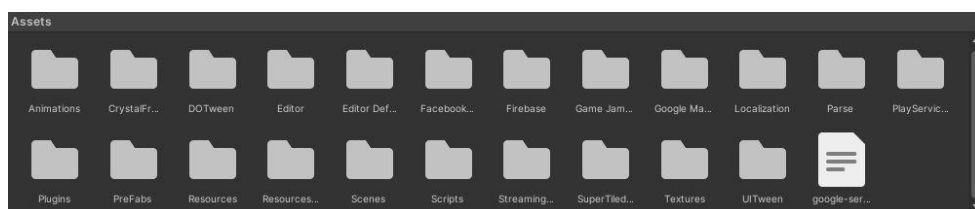
## 4 PELIN KÄÄNTÄMINEN

### 4.1 Tiedostojen käyttö käännöksessä

Unity-pelimootorilla toimii suurin osa yleisistä tiedostomuodoista, joita käytetään myös Cocos2D-pelimootorilla. Tämän myötä vanhoja tiedostoja ei tarvitse muuttaa toiseen muotoon vaan voidaan siirtää suoraan uudelle alustalle. Cocos Creator ja Unity -käyttöohjeista havaittiin, että molemmat pelimootorit tukevat samanlaisia kuva-, ääni-, ja videotiedostoja. Unity kykenee käyttämään laajempaa valikoimaa tiedostoformaatteja kuin Cocos2D, jolloin kääntämisessä Unity-pelimootorille ei tule suoria ristiriitoja tiedoston jälleenkäytössä.

Unity:lla suurin osa tiedostoista asetetaan "Assets"-tiedostokansioon, joka voidaan suomeksi kääntää sanaksi "Resurssit". Tiedostojen ei tarvitse tässä kansiossa olla missään erityisessä järjestyksessä tai lajiteltuna alakansioihin, sillä pelimoottori havaitsee ne sieltä jo valmiiksi. Lajittelu on kuitenkin suositeltavaa jatkokehityksen kannalta, jotta tiedostot löytyvät tulevaisuudessa kenelle tahansa.

BeiZ Oy:llä kansiot jaettiin pääasiallisesti tiedostomuodon ja sen tarkoituksen mukaisesti. Kuvatiedostot asetettiin esimerkiksi "Textures"-kansioon ja Unity:n lisäosat "Plugins"-kansioon. Kaikki tiedostot on nimetty englannin kielellä kansainvälisen ymmärrettävyyden saavuttamiseksi ja yleisten koodistandardien mukaisesti (kuva 1).



KUVA 1 Assets -kansio. Kuvassa esimerkki, kuinka kansion sisältöä on jaettu alakansioihin.

## 4.2 Taustakuvien käyttö käännöksessä

Koska markkinoille on ilmestynyt pelin alkuperäisen julkaisun jälkeen uusia mobiililaitteita, joissa on uudet näytön kokosuhteet, piti tämä ottaa huomioon peliä tehtäessä. Pelin valikkoa työstäessä selvisi taustakuvien olevan liian pieniä uusille, leveämmille ruuduille. Tällaisessa tilanteessa oli kaksi vaihtoehtoa, joiden mukaan toimia.

Ensimmäinen ratkaisu olisi ollut lisätä tummat reunukset kuville, jotka tulee mahdollisimman vähän esille leveämmillä ruuduilla. Tämä on suosittu ratkaisu vanhojen pelien uusimisessa nykyajan laitteille, jotta saadaan vanha kuvasuhde keinotekoisesti samana. Tekniikasta käytetään englanninkielistä ilmaisua "letterboxing", joka on lainattu elokuvatuotannon sanastosta (Merriam-Webster 2021). Unity:lla tekniikka voidaan suorittaa asettamalla näkymän kamera -komponentin taustaväri mustaksi, jolloin kaikki kameran näkemät tyhjät alueet väritetään mustaksi. Jos mikään näkymässä oleva elementti ulottuu taustakuvan ulkopuolelle, se kuitenkin näkyy mustaa taustaa vasten. Tämän takia on suositeltavampaa asettaa mustaksi värjätyt kuvaelementit heti taustakuvan ulkopuolelle niin, että ulos ulottuvat elementit peittyvät niiden alle, eikä niiden kanssa voi olla vuorovaikutuksessa.

Toinen vaihtoehto, johon työssä lopulta päädyttiin, oli laajentaa kuvaa kuvanmuokkausohjelmalla. Tässä tapauksessa käytetty muokkausohjelma oli Adobe Photoshop, mutta käytännössä mikä tahansa tiedostoon yhteensopiva ohjelma olisi myös toiminut. Taustakuvaa laajennettiin kopioimalla sen reunoille toistuvia elementtejä muista pelin taustalla näkyvistä kuvatiedostoista ja sen myötä laajennettiin maisemaa.

Joitain taustoja haluttiin jatkokehityksen yhteydessä uudistaa kokonaan, jolloin graafikko loi uuden version taustakuvasta. Tällä tavalla vanhasta taustakuvasta saatiin nykyaikaisempi, eikä vanhaa kuvaa siten tarvinnut venyttää uusiin mittoihin tai sovittaa eri ruudunkokoja varten. Kokonaan uuden taustakuvan luominen kuitenkin vie enemmän aikaa kuin vanhan tiedoston käyttäminen.

### 4.3 Koodin kääntäminen

Pelimoottorit käyttävät eri ohjelmointikieliä: Cocos2D käyttää C++ (Chukong Technologies 2020) ja Unity C# ja Javascript-kieliä (Unity Technologies 2019). Osa vanhasta C++ -koodista voidaan kuitenkin muuttaa plug-in -lisäosiksi Unity:n versiolle pelistä. "Plug-in" tarkoittaa Unity:n yhteydessä lisäosaa, joka laajentaa pelimoottorin omaa koodia antaen sille uusia ominaisuuksia. Työssä käännettiin alkuperäisen pelin koodista plug-in, jolla peli kykeni havaitsemaan muita BeiZ Oy:n pelejä yhtiön Apple Incorporated -laitteilla. Tämä suoritettiin luomalla uusi ".mm" -tiedosto, johon haluttu koodi kopioidaan ja yksinkertaisesti viedään Unity -editorissa 'Asset' -kansioon, josta pelimoottori löytää sen tarvittaessa. On suositeltavaa tehdä plug-in kansio, josta ne löytyvät myös jatkokehittäjille helposti. Kun kyseistä lisäosaa tarvitaan, se voidaan kutsua sen omalla nimellä (kuva 2).

```
[DllImport("__Internal")] private static extern bool canOpenUrl(string url);  
private static bool _canOpenUrl(string url)  
{  
    return canOpenUrl(url);  
}
```

KUVA 2. Plug-in. Kuvassa on rivejä koodista, joissa viitataan malliesimerkkinä "canOpenUrl" plug-in -tiedostoon.

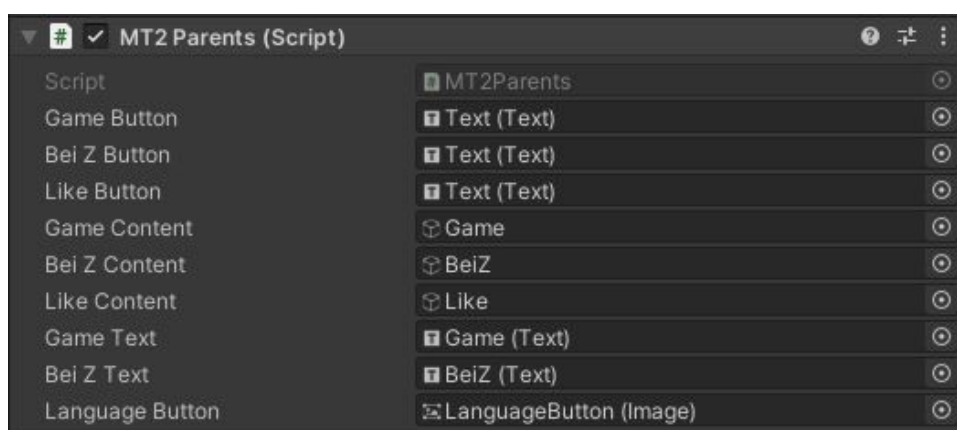
### 4.4 Pelimoottorien eroavuudet

Unity- ja Cocos2D-pelimoottorit eroavat toisistaan ohjelmointikieleltään. Cocos2D toimii kokonaan C++ pohjalta (Chukong Technologies 2020), kun taas Unity:n pelit toimivat joko C# tai Javascript-kielillä (Unity Technologies 2021). Monista samankaltaisuuksistaan huolimatta pelimoottoreilla on eri lähestymistavat pelin kehitykseen, jotka täytyy ottaa huomioon pelin käännöksessä. Kaikkia pelin koodeja, elementtejä ja muita komponentteja ei voida suoraan kääntää alustalta toiselle, vaan useassa tapauksessa niitä täytyy soveltaa uuden alustan rajoitusten sekä mahdollisuuksien mukaisesti.

Unity tarjoaa laajan valikoiman mahdollisia uudistuksia Cocos2D:sta käännettäville peleille jatkokehittämisen suhteen. Animaatioita on helpompi ja sujuvampi tehdä Unity:n animaattorilla, joka mahdollistaa nopeiden animaatioiden teon käännökseen. Unity:lla on myös laajempi valikoima erikoisefektejä, kuten partikkeleita, joita voidaan hyödyntää visuaalisien ominaisuuksien parantamisessa.

#### 4.5 Objekteihin ja ulkoiseen koodiin viittaaminen

Sekä Cocos2D:ssä että Unity:ssä voidaan viitata koodin ulkopuolisiin resursseihin, oli ne sitten tiedostoja pelin resurssikansioissa tai muita objekteja, jotka ovat sillä hetkellä aktiivisina pelissä. Unity kuitenkin suosii editorissaan toimivaan “serialization” eli sarjoittamisjärjestelmää. Jokainen resurssi pelissä saa oman sarjanumeronsa automaattisesti, joita käyttäen Unity kykenee viittaamaan koodista toiseen ilman pitkiä ja monimutkaisia tiedonhakukomentoja. Kun koodiin määritetään julkinen muuttuja, voidaan sitä muokata mistä tahansa kyseisen koodin ulkopuolelta. Myös tämä muuttuja saa oman sarjanumeronsa, jolloin siihen voidaan tarpeen tullen määrittää oma arvonsa editorissa koskematta itse koodiin. Koodille yksityiseksi määritetyille muuttujille voidaan myös asettaa erikseen sarjanumerointi asettamalla sen luomisen alkuun koodissa ilmaisu “[SerializeField]”, jolloin se on käsiteltävissä editorissa (Unity Technologies 2021) Kuvassa 3 näkyy yhden koodikomponentin sarjoitetut muuttujat.



KUVA 3. Sarjoitetut muuttujat. Kuvassa C#-koodiin perustuva komponentti, jossa useampi sarjoitettu muuttuja.

Sarjoitettuihin muuttujiin voidaan valita mistä tahansa osasta resursseja tai saman näkymän elementtejä, jotka sisältävät tarvittavan muuttujan. Jos muuttuja on esimerkiksi tekstiä, voidaan siihen viedä elementti, joka koostuu tekstistä tai sisältää tekstiä. Kun haluttu elementti on viety sarjoitetun muuttujan paikalle, voidaan sitä muokata sitten koodissa. Siihen ei tarvitse tehdä silloin erillistä viitasta tai tiedoston hakua.

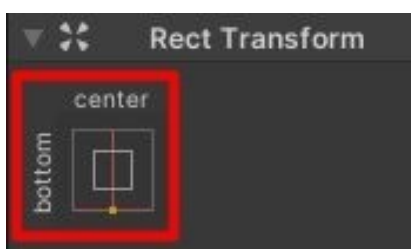
## 4.6 Editoriohjelma

Cocos2D-pelimootorilla ei ole omaa editoriohjelmaa, vaan kyseinen ominaisuus on tullut vasta uudemmalla Cocos2D-x -pelimootorilla (Chukong Technologies 2020). Editorilla tarkoitetaan ohjelmaa, jossa voidaan muokata kehitettävän pelin ominaisuuksia monipuolisilla työkaluilla, jotka eivät rajoitu pelkästään koodiin. Tämä tarkoittaa siis sitä, että pelit Cocos2D-pelimootorilla tehdään puhtaasti koodaamalla. Cocos2D:lle on olemassa editoriohjelmia, kuten SpriteBuilder, mutta ne täytyy asentaa ja ladata erikseen pelimootorin lisäksi (Trengrrove 2015). Unity:n yksi huomattavimpia ominaisuuksia on sen editoriohjelma, joka auttaa pelin elementtien asettelussa ja vähentää vaadittavan koodin määrää pelille. Editorin ansiosta ei tarvita niin tarkkaa koordinaattikokoelmaa kuin Cocos2D:llä, vaan tarvittavat elementit voidaan asettaa editorin ruudulla. Kyseisille elementeille voidaan erikseen määrittää editorissa myös omat kooditiedostonsa, jotka antavat niille uusia ominaisuuksia. Näistä tiedostoista käytetään nimitystä ”script”, eli käsikirjoitus (Unity Technologies 2021). Tämä kaikki vähentää tarvittua aikaa ohjelmoinnille helpottaen pelin jatkokehittämistä helppolukuisuudellaan.

Haasteena huomattiin editorittomasta pelimootorista kääntämisessä koordinaattien ja muiden koodin ominaisuuksien paikoittainen vaikealukuisuus, sillä ne eivät kääntyneet suoraan editoripohjaiselle Unity:lle. Ongelmaa alettiin ohittaa tekemällä suuresta osasta taustalla toimivista elementeistä ruudun koon mukaan skaalautuvia. Tämä siis tarkoittaa elementtien kasvavan ja kutistuvan koossa ruudun kokosuhteiden mukaan, jolloin peli näyttää ja toimii suunnilleen samalla tavalla kaikilla laitteilla.

## 4.7 Virtuaalikoordinaatit

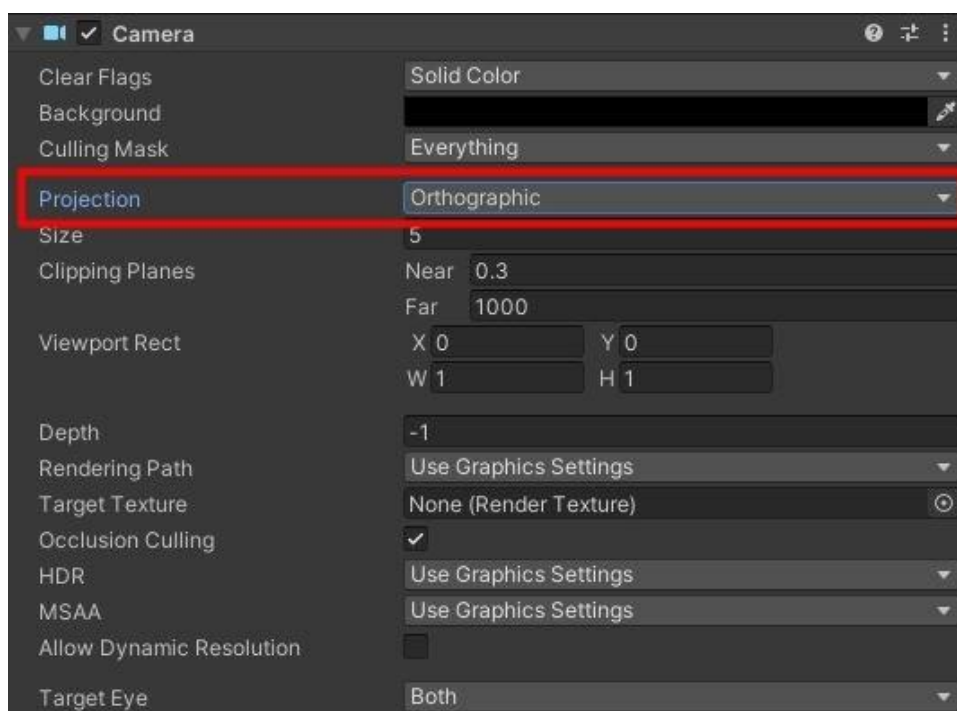
Cocos2D-pelimoottorissa ruudulla esiintyvät elementit asetellaan koordinaateilla. Cocos2D katsoo koordinaattien nollapisteen alkavan ruudun vasemmasta yläkulmasta, kun taas Unity aloittaa vasemmasta alakulmasta. Cocos2D:ssa elementeille voidaan kuitenkin asettaa “ankkuri”, joka määrittää uuden sijainnin kyseisen elementin nollapisteelle. Tämä on erityisen hyödyllistä, kun halutaan asettaa ruudulla olevia elementtejä ruudun kulmiin näyttökoosta huolimatta. Muuten elementtien sijainti määräytyisi yhden ainoan kulman pohjalta ilman erillistä ohjelmointia ja elementtien sijainnit olisivat aina samalla paikalla ruudun koon kasvaessa tai kutistuessa. Ilman tätä ominaisuutta osa elementeistä voijoutua ruudun ulkopuolelle tai muiden elementtien teille. Unity:lla “ankkuri”-ominaisuus on vain UI -elementeistä, jonka myötä käännöksessä vuorovaikutettavat elementit päätettiin laittaa kyseiseen muotoon. UI on lyhenne sanoista “User Interface” eli käyttöliittymä. Unity:n UI-elementit ovat kuvia, nappuloita ja muita elementtejä, jotka on suunniteltu laitettavaksi pelinäköymän päälle kuvastamaan tietoa pelaajalle. Niitä voidaan käyttää myös moniin muihinkin tarkoituksiin. Ankkuriominaisuutta säädetään Unity:n käyttöohjeen mukaan UI -elementistä löytyvällä Rect Transform-komponentilla, josta säädetään myös elementin kokosuhteita ja sijaintia (kuva 4).



KUVA 4. Rect Transform. UI-elementin kuvasuhteita säättävä “Rect Transform”-komponentti Unity:n editoriohjelmassa. Kuvassa on korostettuna elementin “ankkurin” säädin, joka on asetettuna ruudun alareunan keskiosan kohdalle.

Virtuaalikoordinaateissa tulee ottaa huomioon myös Unity:n kamera-komponentti. Unity:n käyttöohjeiden mukaan kyseinen komponentti määrittää, mitä ruudulla näkyy pelaajalle. Kameralla projektion perusasetuksena on perspektiivinäkymä, joka on hyödyllinen kun halutaan tehdä kolmiulotteisia efekte-

jä. Perspektiivissä esineet muuttavat kokoansa suhteessa kameraan, mutta kun pelin halutaan pysyvän samoissa kokosuhteissa kaikilla laitteilla, on parempi laittaa asetus ortografiselle näkymälle. Ortografisessa näkymässä kaikki elementit piirretään ruudulle samoilla kokosuhteilla, eli yksi senttimetri näyttää aina yhden senttimetrin pituiselta kameran etäisyydestä huolimatta. Koska Cocos2D toimii ortografisella näkymällä, laitettiin työssäkin kyseinen näkymä päälle. Se pitää muistaa asettaa joka näkymässä erikseen, jos kamerakomponenttia tai sen asetuksia ei suoraan kopioida uuteen näkymään. Kuva 5 havainnollistaa, missä osassa kamerakomponenttia tämä asetus määritetään.



KUVA 5. Kamerakomponentti. Kuvassa on kamerakomponentin ominaisuuksista näkymä, jossa kameran projektion asetus on korostettu.

## 4.8 Animaatiot

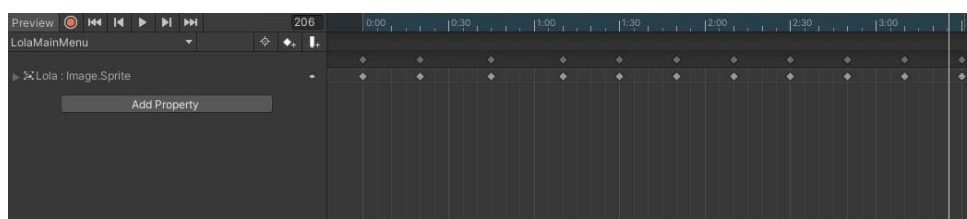
Cocos2D:ssa animaatioita tehdään pääosin klassisesti kuvien nopean tahdin vaihtamisella, luoden liikkeen illuusion. Elementtejä voidaan myös liikuttaa ruudulla koordinaatteja muuttamalla täydentäen animaatiota. Unity käyttää manuaalinsa mukaan "animator" -käyttöliittymää luodakseen animaatioita eri elementeille. Käyttöliittymässä voidaan tehdä edellä mainittu klassisen mallin ani-



maatio tai liikkeisiin ja muodon muuttamiseen perustuvia animaatioita. Animaation tekemiseen Unity:ssä ei tarvita koodia, vaan se voidaan tehdä puhtaasti elementtien kokosuhteiden ja koordinaattien manipuloinnilla editorin sisällä.

#### 4.8.1 Animator -käyttöliittymä

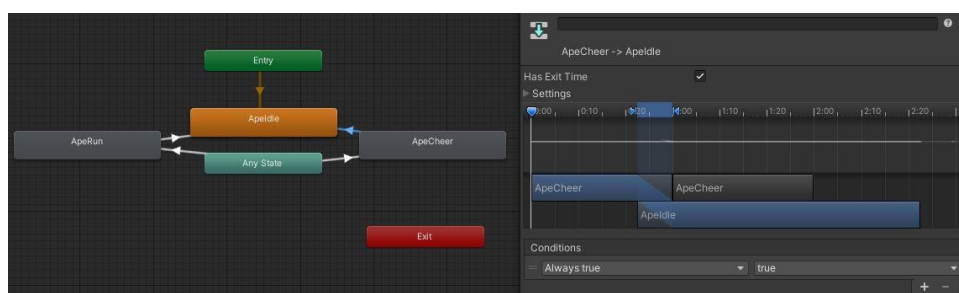
Animator-käyttöliittymässä on ”tallennus” -painike, jonka päälle asetettaessa voidaan elementille asettaa tiettyjä ”välietappeja” ajassa. Kun kyseisen välietapin kohdalla elementti asetetaan tiettyihin parametreihin koordinaattien, koon, ja sille asetetun kuvan suhteen, animaattori tekee animaation välietappien välille. Elementit muuttavat olemustansa välietapilta toiselle, kunnes animaatio päättyy. Välietapit näkyvät kuviossa 6 valkoisina nelikulmioina aikajanalla.



KUVA 6. Animator. Vasemmassa yläkulmassa on esillä tallennus, kelaus ja toistopainikkeet animaatiolle. Oikea puoli esittää animaation aikajanaa sekunneissa, jossa pienet nelikulmiot esittävät animaation välietappeja aikajanalla ja valkoinen viiva sillä hetkellä näkyvissä olevaa kohtaa aikajanasta.

Animaation sulavuutta voi säätää elementin omassa animator-komponentissa, johon animaatiot tallentuvat ja jossa animaatiot voidaan asettaa käyttäjän määrittämään järjestykseen. Komponentilla on oma käyttöliittymänsä, johon kuuluu animaatioiden hierarkiaa kuvaava näkymä sekä tarkkailunäkymä. Hierarlianäkymässä nähdään animaatioiden järjestys ja elinkaari. Näkymässä olevat palikat esittävät eri tilaa, joissa elementti voi olla. Sisääntulo (entry) on aloituspiste, josta elementti aina aloittaa sen latautuessa peliin. Poistuminen (exit) vie takaisin sisääntuloon, mutta sitä ei tarvitse käyttää. Loput tilat ovat täysin kehittäjän määriteltävissä, mukaan lukien tilojen siirtymät yhdestä tilasta toiseen, joita edustetaan valkoisilla nuolilla.

Jokaiselle tilalle voidaan asettaa animaatio, jota elementti alkaa suorittamaan siihen tilaan saapuessa. Tilojen välistä siirtymistä hallinnoidaan erilaisilla kehittäjän määrittämällä arvoilla, jotka voivat olla yksinkertaisia numeroarvoja, boolean ”tosi tai epätosi” muuttujia, tai Unity:n oma tapahtuma-arvo. Kun koodissa lähetetään animator-komponentille näissä määritetyissä arvoissa tapahtuva muutos, siirtyminen tilasta toiseen tapahtuu. Siirtymisiä voidaan myös erikseen muokata miten nopeasti animaatio vaihtuu aikaisemmasta seuraavaan tarkkailunäkymässä. Unity käyttää tästä siirtymäajasta nimitystä ”Exit Time” eli poistumisaika. Poistumisajassa animaatiot sulautetaan toisiinsa, jotta se näyttää sulavalta. Poistumisaika voidaan myös laittaa pois päältä, jolloin animaatio siirtyy ilman sulautusta tilasta toiseen, mutta voi saada animaation liikkeistä riippuen nopeita, syöksyviä liikkeitä. Tämä johtuu elementin nopeasta siirtymisestä koordinaateista toiseen, eikä tämä ole suositeltavaa, jos animaatioita ei ole suunniteltu erikseen toimimaan siltä pohjalta. Tämä saavutetaan asettamalla animaation alku- ja loppukoordinaatit olemaan tismalleen samanlaiset. Kuvassa 7 esitellään animaatioiden hierarkiaa ja poistumisaikaa kuvaavien paneelien ulkonäköä.



KUVA 7. Käyttöliittymä. Vasemmalla hierarkianäkymä ja oikealla tarkastelunäkymä, jossa näkyvissä siirtymisen poistumisaikaa määrittävät liikusäätimet. Siirtyminen on asetettu tapahtumaan aina animaation loppuun.

#### 4.8.2 Koordinaattipohjainen liike

Jos interaktiivisen elementin tarvitsee liikkua pelissä, on suositeltavaa silloin muuttaa liikkuvan elementin koordinaatteja. Animator käsittelee hyvin animaatioiden visuaalista puolta, mutta todellista liikettä varten on suositeltavaa käyttää koodin kautta elementtien Transform- ja Rect Transform-komponentteja, jotka

säätävät elementtien sijaintia pelissä. Animator ei varsinaisesti liikuta elementtejä muuta kuin väliaikaisesti suhteessa alkuperäisiin koordinaatteihinsa. Kun animaatio päättyy, palaavat elementin osat takaisin aloituspisteisiinsä, jos niiden sijaintia ei vielä erikseen muuteta koodissa.

Kahden eri koordinaatin välillä tapahtuvaa sulavaa liikettä saadaan aikaiseksi Lerp-metodilla. Lerp on lyhenne lineaarisesta interpoloinnista (englanniksi Linear Interpolation) (Dillet 2018). Lerp -metodia voidaan käyttää Unity:ssä Mathf-ominaisuudella, jossa Lerp on sen sisällä olevana ominaisuutena. Vector2-ominaisuudella, joka edustaa 2D-koordinaatteja (leveys ja korkeus), voidaan muuttaa Lerp -ominaisuuden antamat arvot yhteensopiviksi koordinaateiksi. Tarvittava liike saavutetaan antamalla koodissa elementille uudet koordinaatit halutulla ajalla Vector2-ominaisuutena, jonka elementin Transform, tai Rect Transform UI -elementin tapauksessa, tulkitsee elementin uusiksi koordinaateiksi. Elementin koordinaatin muuttumisarvo, jota Unity:n manuaali edustaa muuttujalla "t", voidaan koodissa muokata itse asetetulla kertoimella nopeuttaen tai hidastaen koordinaattien välisen liikkeen nopeutta. Lerp-metodilla ei ole liikkeelle loppua, vaan se käytännössä vain hidastaa elementin liikettä pikkuhiljaa äärettömän pieneksi. Tämän takia koodissa pitää tämä ottaa huomioon ja lopettaa liike, kun se saavuttaa halutun nopeuden ja etäisyyden lopulliseen koordinaattiinsa. Tällä tekniikalla luotiin usea liikkuva objekti työssä (kuva 8).

```
void Update()
{
    if (!goingUp) _cloudRT.anchoredPosition = new Vector2(_cloudRT.anchoredPosition.x, Mathf.Lerp(_upperPos, _lowerPos, _t));
    else _cloudRT.anchoredPosition = new Vector2(_cloudRT.anchoredPosition.x, Mathf.Lerp(_lowerPos, _upperPos, _t));
    _t += _speed * Time.deltaTime;

    if (_t > 1)
    {
        _goingUp = !_goingUp;
        _t = 0;
    }
}
```

KUVA 8. Lerp-metodi. Kuvassa osa koodia, joka määrittää pelissä esiintyvän pilven liikettä kahden koordinaatin välillä. Pilven Rect Transform-komponentille on annettu nimitys "cloudRT" ja kahdelle koordinaatille "upperPos" ja "lowerPos".

## 4.9 Tiedon säilytys

Cocos2D voi säilyttää tietoa näkymien välillä, kunhan sitä ei ole luotu eksklusiivisesti tietyn näkymän sisään. Kun näkymästä siirrytään toiseen, pysyvät muut tiedot tallella. Unity on sidottu lujemmin näkymiinsä, eivätkä tiedot säily näkymästä toiseen. Esimerkiksi, jos asetusten näkymässä tehtäisiin muutoksia, tiedot voidaan tehdä muutoksena suoraan tarvittavaan tietoon Cocos2D:n koodissa ja voidaan käyttää toisessa näkymässä. Unity:lla tarvitaan erillinen koodi, joka voi säilyttää tämän tiedon näkymien välillä. Yleisimpiä ratkaisuja tähän on niin sanottu “game manager” eli pelin hallinnoitsija.

Pelin hallinnoitsija useimmiten sisältää asetusten tietoja, kuten äänenvoimakkuus, kirkkaus ja muita pelimekaanisesti tarpeellisia muuttujia, jotka täytyy säilyttää näkymien ja kohtauksien välillä. Pelin hallinnoitsijan elementistä, jossa koodi on kiinni, täytyy tehdä Unity:n objekti -tiedosto. Tämä varmistaa, että pelin hallinnoitsija säilyy joka näkymässä samanlaisena, eikä muutu niiden vaihtuessa. Objekti-tiedoston luominen tehdään editoriohjelmassa yksinkertaisesti “raahaamalla” elementtialueelta resurssipaneeliin kehittäjän haluamaan kansioon. Pelin hallinnoitsijan pitää tämän jälkeen vain tarkistaa näkymän käynnistyessä, onko saatavilla edellisen näkymän versiota olemassa ja kopioida se sitten päällekirjoittaessa.

Toinen tapa luoda pelin hallinnoitsija on käyttää Singleton-mallia. Kyseisessä mallissa peli tarkistaa näkymän vaihtuessa, onko pelin hallinnoitsijaa jo olemassa ja ottaa edellisen hallinnoitsijan ominaisuudet käyttöön sellaisen löytyessä. Työssä käytetty malli perustuu osittain tähän malliin, mutta on tehokkaampi tapa, sillä Singleton-malli saattaa aiheuttaa pelissä hidastuksia. Singleton-malli soveltuu paremmin pienimuotoisiin tilanteen muutoksiin pelissä. (Santana A.F. 2021) Kuvassa 9 esitellään työssä käytettyä mallia.

```

private static MathTrain2GameManager _instance;

public static MathTrain2GameManager Instance
{
    get
    {
        if (_instance == null)
        {
            GameObject managerObject = Instantiate(Resources.Load<GameObject>("MathTrain2GameManager"));
            _instance = managerObject.GetComponent<MathTrain2GameManager>();
        }

        return _instance;
    }
}

```

KUVA 9. Pelin hallinnoitsija. Kuvassa on osa koodia, jolla saadaan pelin hallinnoitsija tuomaan tietoja edellisestä näkymästä.

#### 4.9.1 Tiedon tallentaminen verkkoon

Joskus tietoa täytyy tallentaa myös pelaamiskertojen välissä. Alkuperäisessä Cocos2D-versiossa pelaajan eteneminen pelissä tallennettiin BeiZ Oy:n omalle palvelimelle, josta pystyttiin seuraamaan pelaajan edistystä pelissä. Tähän kuului se, kuinka paljon virheitä pelaaja teki pelin tehtävissä, missä tehtäväkategorioissa ja muuta lisätietoa. Tämä on erityisen hyödyllistä, kun halutaan opetuspelissä näyttää lapsipelaajan edistymistä pelin tehtävissä tämän vanhemmille, jota varten kyseinen systeemi kehitettiin.

Tallennussysteemi toimi JSON-tietoformaattilla, johon voidaan asettaa tietoa koodaajan asettamien kategorioiden mukaan, kuten pelaajan nimi, pelattu aika, onnistumisprosentti ja mitä ikinä tietoa tarvitaan talletettavaksi. Tiedon tallennusta varten käytettiin Unity:n manuaalista löytynyttä "List<IMultipartFormSection>"-listamuuttujaa. Kyseiseen listaan voidaan asettaa useita "MultipartFormDataSection"-muotoisia muuttujia. Nämä muuttujat ovat Unity:ssä yhteensopivia yleisten web-lomakkeiden kanssa, joita käytetään tiedon tallentamiseen PHP- ja Javascript-tekniikoilla. Jotta tallentaminen toimii, täytyy tietoa tallettava nettisivu ja palvelin valmistella sitä varten erikseen.

Kun tarvittavat tiedot ovat asetettuna muuttujiinsa, voidaan ne lähettää käyttäen Unity:n "UnityWebRequest"-ominaisuutta. Kyseisellä ominaisuudella kyetään tekemään internetin välityksellä yleisiä pyyntöjä, kuten "POST", "DELETE", ja

“GET”. Tiedon tallentamiseen käytetään “POST”-, ja sen hakemiseen “GET”-pyyntöjä.

#### 4.9.2 Pelin tietojen tallentaminen laitteelle

Yleisempi tapa tallettaa tiedostoja on tallettaa ne itse pelaamiseen käytetylle laitteelle. Unity:n “PlayerPrefs”-ominaisuus tallettaa manuaalin mukaan tekstiä numeropohjaisia muuttujia. Nämä tiedot voidaan asettaa yksinkertaisilla komennoilla, esimerkiksi “PlayerPrefs.SetFloat(MuuttujanNimi, 1.0)” asettaisi MuuttujanNimi -nimellä arvon pelin muistettavaksi. Tämä on yksinkertainen, mutta samaan aikaan hieman alkeellinen tapa tallettaa tietoa, sopien pääosin pelin asetuksien, kuten äänenvoimakkuuden, kirkkauden ja muiden pelikokemukseen liittyvien yksinkertaisten tietojen tallentamiseen. Nimi “PlayerPrefs” on lyhenne “Player Preferences”, eli pelaajan mieltymykset, joka viittaa nimen omaan käyttäjän asettamiin asetuksiin. Työssä ei kehitetty metodia monimutkaisten tietojen tallettamiselle pelilaitteelle.

#### 4.10 Kuva-arkit

Kuva-arkit ovat kuvia, joka koostetaan useasta kuvasta yhteen kuvatiedostoon (kuva 10). Tämä helpottaa pelimoottorin latauksia rajoittamalla useamman kuvan latauksen yhteen tiedostoon. Sekä Cocos2D että Unity kykenevät rajaamaan nämä arkit pienemmiksi kuviksi. Erona Cocos2D kykenee kääntämään näitä kuvia Cocos2D Creator-manuaalin mukaan, samalla kun se piirtää ne ruudulle. Unity ei kykene tähän ilman että kuvaelementin rotaatiokoordinaatteja tarvitsisi muuttaa pelin koodissa. Tämä on erityisen ongelmallista, jos pelissä tarvitaan paljon kuvien vaihtelua keskenään. Jokaisen kuvan kääntäminen erikseen Unity-pelin sisällä on aikaa vievää, jonka myötä täytyy varmistaa kuva-arkkien kuvien olevan halutussa asennossa ennen niiden latausta tai tarpeen tullen erillisinä tiedostoina.



## 5 OMINAISUUKSIEN LISÄÄMINEN UNITY:LLE

Unity:ssa on useita valmiita ominaisuuksia, joita hyödyntää pelin kehityksessä. Siihen voidaan lisätä ominaisuuksia ulkoisista lähteistä, joita myydään erinäisissä verkkokaupoissa tai jaetaan ilmaiseksi erilaisilla blogeilla, foorumeilla, ja keskustelupalstoilla. Suurin osa näistä lisätyistä ominaisuuksista on luotu Unity:n valmiiksi olemassa olevista ominaisuuksista, usein yhden kooditiedoston sisällä. Jotkin lisäosat antavat itse pelimoottorille uusia ominaisuuksia plug-in – muodossa vaikuttaen pelin sijasta pelimoottoriin, johon sitten voidaan viitata koodissa.

### 5.1 Partikkelit UI-elementtinä

Cocos2D-versiossa peliä esiintyy yksinkertainen partikkeliefekti, joka kuvastaa ilotulituksia. Partikkeliefektit ovat visuaalisia erikoistehosteita, jotka koostuvat pienistä virtuaalisista partikkeleista. (Nemeth-Csoka 2020) Partikkelit ovat suurissa määrissä ilmeneviä kuvatiedostoja, jotka yhdessä esittävät jonkin sortin erikoisefektiä, kuten tulta ja räjähdysä. Unity:lla on oma valmis partikkeliefekti, mutta se ei näy UI -elementtien alta, vaan toimii samassa näkymässä kuin muut peruselementit. Koska käännös toimii pääosin UI-elementeillä, täytyi tehdä uusi ratkaisu partikkeliefektien näyttämiseksi.

Partikkelisysteemin sijasta tehtiin uusi koodi, joka luo UI-elementtien päälle "Image" eli kuvatyypin elementtejä, joille annettiin alkuperäisessä pelissä käytettyjen ilotulitteiden partikkeleiksi tarkoitettuja kuvatiedostoja. Cocos2D-versiossa kyseiset partikkelit liikkuvat yksinkertaisissa kaarissa "räjähdysen" keskipisteessä ilman muita erikoisefektejä. Kun Unityn versiolle annettiin partikkeleille liikeradat, lisättiin niille myös värin vaihtumis- ja kääntymisefektit. Tämä saavutettiin Unity:n coroutine-ominaisuudella. Coroutine on osa koodista, joka toimii erillisesti muusta koodista ja kykenee muun muassa mittaamaan aikaa ja milloin sen tulee esimerkiksi lopettamaan toimintansa. Jokaisella partikkelilla oli oma coroutine, joka mahdollisti sen liikkeen ja värin vaihtumisen suhteessa ajan kuluun.



## 5.2 Kielivaihtoehdot

Koska alkuperäinen Cocos2D–peli tehtiin useammalle kuin yhdelle kielelle, täytyi Unity:n versiolle luoda ominaisuus vaihtaa kieliä tarpeen tullen. Kielen vaihtaminen voidaan koodata peliin käsin, mutta se voi viedä paljon aikaa. Työtä varten BeiZ Oy oli ostanut Unity:n nettikaupan kautta plug-in -lisäosan, joka hoitaa suuren osan lokalisaatiosta automaattisesti käyttäen XML-tiedostoja oikeiden tekstien ja puhuttujen äänien asettamista peliin varten.

Kielen kääntyminen saadaan aikaiseksi tekemällä jokaista tekstiä ja puhuttua äänitiedostoa varten koodi, joka ensin tarkistaa asetuksista mikä kieli on päällä, jonka jälkeen se asettelee kieltä vastaavan tekstin tai äänen elementille. Tämä koodi tai tunniste tulee laittaa varsinaisen elementin tilalle. Koodissa sitten voidaan etsiä tunnisteiden mukaan siihen sopiva sen hetken kielen teksti tai jokin muu tiedosto. (Everything You Need to Know About Localization in Unity 2020.)

## 6 PELIN JATKOKEHITYS

### 6.1 Leikattujen ominaisuuksien lisääminen peliin

Alkuperäisistä Cocos2D-peleistä jäi muutama pelimekaniikka toteuttamatta oletettavasti aikarajoitusten takia, joista kuitenkin oli jo tehty ääni- ja kuvatiedostot. Unity -käännöksen yhteydessä ilmeni mahdollisuus tuoda nämä ominaisuudet käyttöön ilman sen suurempaa muutosta. Sopivia lisäominaisuuksia, joita voitiin tuoda esiin, olivat esimerkiksi pelimuodot, joihin löytyi kaikki tarvittavat resurssit valmiiksi. Myös erikoisefektit ja lisäanimaatiot toimivat tätä tarkoitusta varten, sillä ne eivät suoraan vaikuta pelin varsinaiseen toimintaan. Työhön toteutettiin parempi partikkelisysteemi, jossa käytetään suurempaa partikkelivariaatiota kuin alkuperäisessä. Alkuperäinen partikkelisysteemi käytti ainoastaan yhtä useasta olemassa olevasta partikkelimuodosta ja yhdellä värityksellä. Käännöksessä partikkelisysteemi käyttää kaikkia partikkeleita ja on laajentanut väriskaalaa huomattavasti.

### 6.2 Vanhojen ominaisuuksien parantaminen

Työssä havaittiin vanhan Cocos2D-pelin koodia ja muita tiedostoja tutkimalla, että osa pelin tehtävien logiikoista sallivat vain yhdenlaisen vaihtoehdon siitä, millainen haaste pelaajalle esitettiin. Esimerkkinä tästä on kertolaskutehtävässä, jossa tarkoituksena oli yhdistää kaksi numeroa, joiden yhteinen kerroin toteuttaisi pyydetyn numeron kertolaskussa mahdollisimman nopeasti. Kyseinen tehtävä pyysi aina luvuksi numeroa kahdeksan, mutta pelin kuvatiedostoista löytyi myös muille numeroille kuvakkeet tehtävää varten. Osana pelin jatkokehitystä kyseinen tehtävä uudistettiin tarjoamaan laajempaa valikoimaa mahdollisia numeroita, joita pelaajalta pyydetään.

Samankaltaisia pienempiä muutoksia tehtiin peliin muidenkin tehtävien koodissa. Pääasiallinen tarkoitus muutoksilla oli tuoda lisää sisältöä ja sitä myötä houkutella pelaajaa pelaamaan peliä uudelleen läpi. Saman tehtävän liiallinen toistaminen saattaisi tuttua ärsykkeenä kyllästyttää pelaajaa.

Pelin erikoisefekteihin lisättiin myös uusia ääniefektejä. Esimerkiksi aikaisemmin käsitelty ilotulitusefekti oli alun perin täysin äänetön. Peliä pyrittiin elävöittämään lisätyillä äänillä, jotka lisäävät ärsykeitä aikaisemmin varsin hiljaiseen tapahtumaan.

### **6.3 Pelin liittäminen sovellukseen**

Huomattavin uudistus peliin on sen kuuluvuus sovellukseen, joka hallinnoi kaikkia siihen liitettyjä BeiZ Oy:n pelejä. Aikaisemmin BeiZ on julkaissut moninaisiin verkkokauppoihin ilmaisia kokeiluversioita kokonaisista peleistä. Tämä ei ole ollut täysin mahdollista Android -systeemin ”Play” -verkkokaupassa, joka tulkitsee kokeiluversiot eri nimellä oleviksi kopioiksi alkuperäisestä versiosta. Tämän myötä työstyön peliin kehitettiin yhteensopivuuksia BeiZ Oy:n muiden pelien kanssa. Lopullisessa versiossa kaikki pelit toimisivat samassa sovelluksessa, johon ladataan pelien tietoja erikseen niiden pelaamista varten. Moni kyseisistä peleistä käyttää yhteistä koodia välttääkseen liiallista tilan vientiä pelilaitteen muistissa.

### **6.4 Pelin päivittäminen uusille teknologioille**

Unity-pelimoottori on jatkuvassa jatkokehityksessä. Työn aikana pelimoottorille ilmestyi useampi päivitys laajentaen pelimoottorin ominaisuuksia ja kykenevyyksiä. Jatkokehitys pitää Unity:n ajan tasalla nykyaikaisten teknologioiden kanssa, mahdollistaen pelien julkaisun myös uusilla alustoilla. Tähän kuuluu myös uudet konsolisukupolvet, jolloin peli voidaan kääntää myös niille tarpeen vaatiessa.

## 7 POHDINTA

Työssä aikaansaatu Unity-käännös Cocos2D-versiosta on onnistuneesti samankaltainen alkuperäiseen nähden huomattavien parannusten kanssa. Peli käyttäytyy hyvin samalla tavalla, mutta uudessa versiossa on sulavammat animaatiot ja uusia erikoisefektejä, jotka vanhaan versioon vertailtaessa ovat hyvin sulavia. Jatkokehitys on ollut siis työssä onnistunut. Uudistuksista huolimatta peli testatessa näyttää ja tuntuu samanlaiselta kokemukselta. Ajoitukset ja äänitehosteet ovat ajastettu samalla tavalla kuin ennen, vaikka animaatioissa on havaittavissa eroja.

Työ perustui pääasiallisesti kokeiluun ja BeiZ Oy:n omiin käytäntöihin, jolloin siihen ei tehty ennakolta paljoa tutkimustyötä Cocos2D:n toimintojen läpikatsuksen lisäksi. Työtä tehdessä ei siten ollut varsinaista vertausmahdollisuutta muihin teoksiin, joilla mitata työn onnistumista, muuten kuin omien ja BeiZ Oy:n asettamien tavoitteiden mukaisesti. Ennen työn aloitusta tutkin kuitenkin, oliko joku muu tehnyt aikaisemmin pelimoottorien välistä käännöstä Cocos2D:n ja Unity:n välillä. Etsinnän tuloksena löytyi ainoastaan YouTube-videotoistopalvelusta video vuodelta 2015, koskien uudempaa Cocos2D-x -pelimoottoria, eikä soveltunut BeiZ Oy:n tarkoituksiin. Suuri osa peliä joudutaan uusimaan jatkokehitystä varten, eikä suora käännös ole edes kannattavaa, jos edes mahdollista.

Suhteutettuna Cocos2D:hen, Unity on hyvin helppokäyttöinen pelimoottori, vaatii ainoastaan osaamista joko C# tai Javascript-kielen kanssa. Loput pelimoottorin omista käytänteistä on nopeasti opeteltavissa joko kokeilemalla itse tai seuraamalla yksinkertaisia ohjeita joko Unity:n omilla verkkosivuilla tai toisten videopelien kehittäjien vinkkien tai neuvojen mukaan. Pelimoottorin editoriohjelma helpottaa pelin kehitystä huomattavasti ilman ylimääräisten työkalujen hankkimista, sillä kaikki pelin elementit saadaan haluttaessa ruudulle näkyville haluttuihin asemiin. Cocos2D vaatisi vastaavanlaisessa tilanteessa jonkinlaisen emulaattorin, jolla saavutettaisiin elementtien katselmoinnille mahdollisuus. Tämän lisäksi C# on ohjelmointikielenä erikseen suunniteltu helpommin ymmärrettäväksi, sillä suurin osa sen koodin komennoista suoritetaan englanninkielisillä

ilmaisuilla, joita ei ole lyhennetty. C++ -koodin käyttäminen ei vie niin paljoa laitteen muistia kuin C#, mutta siinä tulisi ottaa huomioon muistin käyttö manuaalisesti, joka tapahtuu C#-koodilla automaattisesti.

Työssä oli tarkoitukseni aluksi katsoa myös, millaista Cocos2D-pelin kehittäminen olisi, mutta Unity-version kehitystä varten olleessa koneessa ei ollut kyseisenlaisille työkaluille sitä varten enempää tilaa muistissa. Tämän koen työn lieväksi heikkoudeksi, sillä suuri osa Cocos2D:n ominaisuuksista tuli ilmi puhtaasti lukemalla muista lähteistä oman kokemuksen sijasta. Tämä ei lopulta haitannut pelin kehitystä Unity:lle, mutta hidasti sitä koodia tulkittaessa ja pelin komponentteja tarkastellessa. Tarkoituksena ei kuitenkaan ollut tehdä täydellistä kopiota alkuperäistä, vaan paranneltu versio, jonka myötä animaatioiden ja koodin samankaltaisuus eivät olleet työn päämäärä.

Peli perustuu pääosin käyttöliittymän kanssa kanssakäymiseen, jonka myötä uudessa versiossa päädyttiin käyttämään yksinomaan sille tarkoitettuja elementtejä.. Tavallisesti Unity-pelit käyttäisivät kyseisiä komponentteja kuvaaksensa tietoja pelaajalle pelitilanteesta käyttäen peruselementtejä kuvaaksensa varsinaista peliä. UI -elementit Unity:lla asettuvat aina peruselementtien päälle niiden asetteluun liittyvästä hierarkiasta huolimatta, jonka myötä peruselementtejä ei olisi kyetty edes käyttämään jälkikäteen. Kuitenkin UI-elementit muuten käyttäytyvät samalla tavalla kuin peruselementit, suurimpana erona niiden sijaintia ja muotoa hallinnoivat Transform ja Rect Transform-komponentit.

## LÄHTEET

Chukong Technologies. Cocos2D Creator manual. Käyttöohje. Julkaistu 23.6.2020. Luettu 5.6.2021. <https://docs.cocos.com/creator/manual/en/>

Chukong Technologies. Cocos2D Homepage. Cocos2D kotisivut. Julkaisupäivää ei ole merkitty sivustolle. Luettu 5.6.2021. <http://www.Cocos2D.org/>

Dillet, R. Unity CEO says half of all games are built on Unity. TechCrunch. Verkkoartikkeli. Julkaistu 5.9.2018. Luettu 22.11.2021. [tinyurl.com/r5p4r27f](https://tinyurl.com/r5p4r27f)

Everything You Need to Know About Localization in Unity. Medium. Verkkoartikkeli. Julkaistu 29.7.2020. Luettu 12.11.2021. <https://nosuchstudio.medium.com/localization-in-unity-3eda21c717db>

Merriam-Webster. Dictionary. Sanakirja. Päivitetty 2021. Luettu 23.11.2021. <https://www.merriam-webster.com/dictionary/letterboxed>

Nemeth-Csoka, R. What is a Particle Effect and Why Should You Use It? Smart Slider. Blogi. Julkaistu 7.5.2020. Luettu 12.11.2021. <https://smartslider3.com/blog/particle-effect/>

Santana, F.A. Implementing a game manager using Singleton pattern | Unity. Medium. Verkkoartikkeli. Julkaistu 27.7.2021. Luettu 12.11.2021. <https://medium.com/nerd-for-tech/implementing-a-game-manager-using-the-singleton-pattern-unity-eb614b9b1a74>

Trengrove, B. 2015. Cocos2D Game Development Essentials. Packt Publishing Ltd. Birmingham – Mumbai.

Unity Technologies. Unity manual. Käyttöohje. Julkaistu 2019. Luettu 5.6.2021. <https://docs.unity3d.com/2019.4/Documentation/Manual/>