OULUN AMMATTIKORKEAKOULU

Samuli Jylhä

MOBIILIPELIOHJELMOINTI JA KÄYTTÖLIITTYMÄN TOTEUTUS UNITYLLÄ

MOBIILIPELIOHJELMOINTI JA KÄYTTÖLIITTYMÄN TOTEUTUS UNITYLLÄ

Samuli Jylhä
Opinnäytetyö
Kevät 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma, ohjelmistokehitys

---

Tekijä: Samuli Jylhä
Opinnäytetyön nimi: Mobiilipeliohjelmointi ja käyttöliittymän toteutus Unityllä
Työn ohjaaja: Kari Laitinen
Työn valmistumislukukausi ja -vuosi: Kevät 2018 Sivumäärä: 48

Opinnäytetyön tavoitteena oli ohjelmoida mobiilipeli käyttäen Unity-pelimoottoria. Lisäksi työssä tarkasteltiin käyttöliittymien ominaisuuksia ja luotiin pelille käyttöliittymä käyttäen Unityn valmiita työkaluja. Työssä myös käydään läpi Unityn tärkeimpiä ominaisuuksia peliohjelmointia varten. Työ tehtiin tarkoituksena laajentaa omaa osaamista peli- ja käyttöliittymäohjelmoinnissa.

Lähes kaikki pelin ominaisuudet tehtiin Unity Editorin valmiilla työkaluilla. Ohjelmoimiseen käytettiin Unityn yhteydessä Visual Studiota. Pelin grafiikat luotiin käyttäen selaimessa toimivaa piirto- ja animointityökalua nimeltä Piskel.

Lopputuloksena saatiin tehtyä mobiilipeli, joka toimii sulavasti ja ilman isompia ongelmia. Pienellä loppuviimeistelyllä peli olisi jopa tarpeeksi pitkälle kehitetty julkista jakelua varten. Unityn dokumentaatioon ja käyttöliittymien teoriaan perehtyminen oli erittäin kehittävää. Näiden asioiden parempi ymmärtäminen auttaa varmasti tulevissa projekteissa.

Asiasanat: Unity-pelimoottori, mobiilipelit, käyttöliittymät

ABSTRACT

Oulu University of Applied Sciences
Degree program of information technology, software development

---

Author: Samuli Jylhä
Title of thesis: Mobile game programming and user interface development with Unity
Supervisor: Kari Laitinen
Term and year when the thesis was submitted: Spring 2018 Number of pages: 48

The goal of this thesis was to develop a mobile game with the Unity Engine. In addition, general features of user interfaces were studied, and a user interface was created for the game by using the default features of Unity. Some of the key features of Unity are reviewed in this thesis. The thesis was done with the aim to expand skills in game and user interface development.

Almost all the features of the game were made with the default features of Unity Editor. Visual Studio was used for programming. The game's graphics were made with Piskel, which is a free online editor for animated sprites.

As a result, a mobile game was successfully developed, and it works smoothly and without any significant bugs. With a little finishing, the game would be suitable for public distribution. Studying Unity's documentation and theory of user interfaces will certainly be helpful in future projects.

Keywords: Unity Engine, mobile games, user interface

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön alkuperäisenä tavoitteena oli luoda kaksiulotteinen mobiilipeli Android-alustalle käyttäen Unity-pelimoottoria. Toisena tavoitteena oli perehtyä käyttöliittymiin ja niiden kehittämiseen Unityllä. Työn ohjelmoinnit tehtiin C#-kielellä käyttäen Visual Studiota Unityn yhteydessä. Pelin grafiikat tehtiin selainpohjaisella piirto- ja animointityökalulla nimeltä Piskel. Opinnäytetyö toteutettiin oman osaamisen kehittämiseksi.

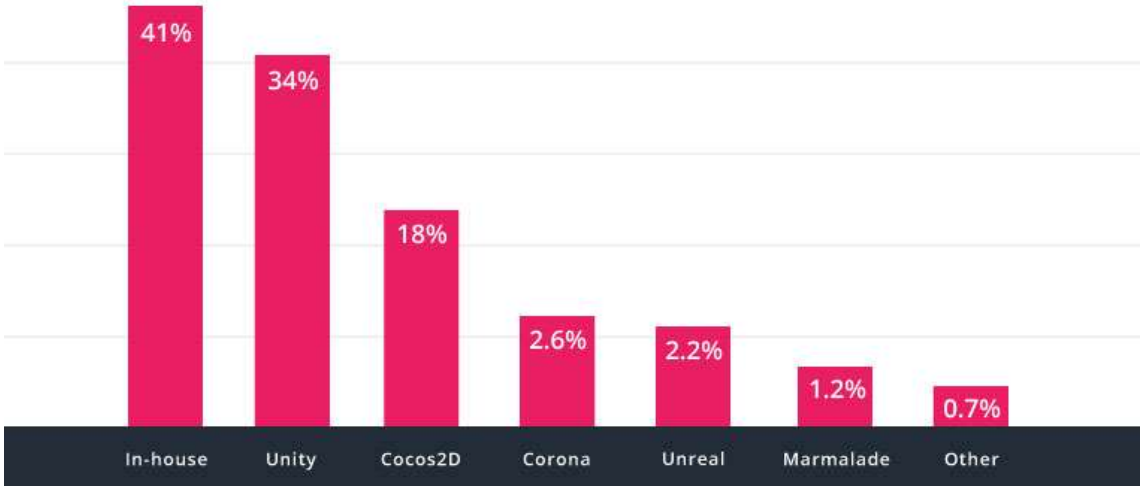
Työ aloitetaan tutustumalla Unityn ominaisuuksiin, jotka ovat avainasemassa niin tässä kuin monessa muussakin peliprojektissa. Näiden ominaisuuksien esittelyn jälkeen työssä esitetään selostusluontoisesti kehitetty peli sekä välillä perehdytään tarkemmin eri ominaisuuksien toteuttamistapaan.

Työssä luodaan myös katsaus hyvien käyttöliittymien ominaisuuksiin sekä käydään läpi toteutetun käyttöliittymän suunnitteluprosessi. Tämän lisäksi työssä tarkastellaan Unityn käyttöliittymätyökaluja sekä käyttöliittymän toteuttamista niitä käyttäen.

2 UNITY-PELIMOOTTORI

2.1 Unityn valinta pelimoottoriksi

Unity on maailman käytetyin niin sanottu kolmannen osapuolen pelinkehitysohjelmisto, ja sen suosio kasvaa jatkuvasti. Se soveltuu erittäin hyvin mobiilipelin kehittämiseen. Vuonna 2016 noin kolmasosa suosituimmista ilmaisista mobiilipeleistä oli tehty Unityllä (kuva 1). (1.) Myös tunnettu suomalainen mobiilipeliyritys Rovio käyttää Unityä osassa peleistään. Näitä pelejä ovat esimerkiksi Arngy Birds 2, Angry Birds Epic, Angry Birds Evolution ja Bad Piggies.



KUVA 1. 1 000 suosituimmasta mobiilipelistä 34 % on tehty Unityllä (1)

Unitystä on tarjolla neljä eri versiota: personal, plus, pro, ja enterprise. Personal on käyttäjälle ilmainen ja loput maksullisia. Suurimpana rajoitteena ilmaisversion käytölle on yrityksen vuositulot: sitä saa käyttää ainoastaan, jos yrityksen vuositulot ovat alle 100 000 $ vuodessa. Plus-versiota käyttävän yrityksen vuositulorajana on 200 000 $. Pro- ja enterprise-version käyttäjillä vuositulorajoja ei ole. Personal sopii hyvin opiskelijoiden ja harrastajien käyttöön, ja tämän vuoksi se on hyvä työkalu opinnäytetyön tekemiseen.

Unityssä pelien ohjelmoimiseen voidaan käyttää kolmea eri kieltä: C#, UnityScript ja Boo. Vuoden 2017 kesällä julkaistun version 2017.1 jälkeen ainoastaan C#-kielen tukemista jatketaan. (2.) C#-kielen hyvä tuntemus vaikutti Unityn valintaan pelimoottoriksi.

2.2 Unityn opinnäytetyössä käytettyjä ominaisuuksia

Unityssä on muutamia ominaisuuksia, jotka ovat käytössä lähes jokaisessa projektissa. Näiden ominaisuuksien ymmärtäminen on kohdan 3 toteutuksen ymmärtämisen kannalta oleellista.

2.2.1 Scene

Scenet ovat tiedostoja, jotka sisältävät kaikki pelissä esiintyvät asiat, kuten ympäristöt ja valikot. Yleensä yksi scene sisältää yhden tason pelistä. Koko peli olisi mahdollista rakentaa vain yhden scenen sisälle, mutta sen jakaminen pienempiin osiin helpottaa pelin kehittämistä. Pelien jakaminen pienempiin osiin on myös suorituskyvyn kannalta parempi vaihtoehto, koska vain aktiivisen scenen tiedot ladataan käytettävän laitteen muistiin.

2.2.2 MonoBehaviour

MonoBehaviour on perusluokka, josta kaikki Unityn skriptit periytyvät. Tärkeimpänä huomioitava on luokassa olevat Awake-, Start- ja Update-funktiot.

Awake-funktio ajetaan aina, kun skriptin sisältävä peliobjekti luodaan, vaikka kyseinen skripti ei olisi aktiivisena. Start-funktio ajetaan, kun skripti asetetaan aktiiviseksi. Awake-funktio siis ajetaan aina ennen Start-funktiota, joten sitä voidaan käyttää niiden asioiden määrittelemiseen, joita Start-funktiossa käytetään.

Update-funktio ajetaan jokaisen framen kohdalla, aina kun skripti on aktiivinen. Frame tarkoittaa videopeleissä ja videoissa yhtä näytölle piirrettävää kuvaa. Updaten toiminta alkaa siis heti Start-funktion jälkeen. Updatea käytetään asioihin, jotka tarvitsevat jatkuvaa muutosta, kuten peliobjektien liikuttamiseen. On tärkeää huomioida, että kuvataajuus (engl. frames per second) vaikuttaa siihen, kuinka usein Updatea kutsutaan. Unity asettaa muuttujaan Time.deltaTime viimeisimmän framen suorittamiseen käytetyn ajan, ja sitä voidaan käyttää kertoimena Updaten sisällä. Tällä menetelmällä muutokset tapahtuvat aina samassa määrin, riippumatta kuvataajuudesta.

2.2.3 GameObject

GameObject eli peliobjekti on tärkein käsite Unity Editorissa. Jokainen scenessä oleva asia – oli se sitten hahmo, kuva, kamera tai mikä tahansa – on peliobjekti. Ne eivät kuitenkaan itsessään tee paljon mitään; niille täytyy antaa komponentteja, jotka määrittelevät niiden käyttötarkoituksen. Esimerkiksi valona toimivaan peliobjektiin lisätään Light-komponentti tai kuvana toimivaan peliobjektiin Image-komponentti.

Unityssä on paljon valmiita käytettäviä komponentteja, tai niitä voi luoda myös itse. Pelinkehittäjien kirjoittamat skriptit lisätäänkin yleensä suoraan peliobjektiin, ja tämä myös mahdollistaa skriptien käsittelemisen suoraan editorin kautta. Ainoa komponentti, jonka jokainen peliobjekti vaatii toimiakseen, on Transform.

2.2.4 Transform

Jokaisessa scenen sisältämässä peliobjektissa on Transform-komponentti, eikä sitä voi poistaa. Sitä käytetään peliobjektin sijainnin, kierron sekä skaalan (engl. postition, rotation, scale) säilyttämiseen ja muuttamiseen. Nämä arvot mitataan suhteessa sen parent-peliobjektiin, eli esimerkiksi child-peliobjektin sijainti tarkoittaa sen etäisyyttä sen parentista. Jos peliobjektilla ei ole parent-peliobjektia, sen arvot mitataan pelimaailmassa. Unityn dokumentaatiossa (3, linkit Working in Unity -> Crating Gameplay -> GameObjects -> Transform) asia selitetään näin:

> "The position, rotation and scale values of a Transform are measured relative to the Transform's parent. If the Transform has no parent, the properties are measured in world space."

Mikäli parent-peliobjektin Transformin arvoja muutetaan, muutokset tulevat myös kaikille childeille. Tämä mahdollistaa peliobjektien käsittelyn ryhminä. Esimerkiksi koko pelin käyttöliittymä voidaan rakentaa yhden peliobjektin alle, ja mikäli kaikkien käyttöliittymäkomponenttien skaalausta halutaan muuttaa, voidaan muutos tehdä ainoastaan tähän yhteen peliobjektiin.

2.2.5 Rigidbody

Rigidbody on komponentti, joka mahdollistaa fysiikoiden vaikutuksen peliobjektiin. Kun Rigidbody on lisätty peliobjektiin, se reagoi painovoimaan ja törmäyksiin (engl. collision) muiden peliobjektien kanssa. Törmäyksien toimimiseksi peliobjektissa tulee olla Rigidbodyn lisäksi Collider-komponentti.

Fysiikoiden vaikutuksella liikutettavan peliobjektin liikkeet ovat hyvin realistisia. Kun Rigidbody on lisätty peliobjektiin, sitä ei tulisi liikuttaa Transform-komponentin arvoja muuttamalla. Realistisien tuloksien kannalta on parempi antaa fysiikkamoottorin hoitaa peliobjektin liikkeet. Peliobjektille voidaan antaa erilaisia voimia, joiden perusteella fysiikkamoottori hoitaa tulosten laskemisen. Voimat voivat tulla ympäristöstä tai skriptistä.

Joissakin tapauksissa voi olla hyödyllistä, että peliobjektilla on Rigidbody-komponentti, mutta liikkeitä ei tehdä fysiikkamoottorin avulla. Tällainen tapaus on esimerkiksi se, jos pelihahmoa halutaan liikuttaa ilman fysiikkaominaisuuksia, mutta sen halutaan reagoivan muiden peliobjektien colliderien kanssa. Tällaisessa tapauksessa colliderit on hyvä laittaa triggereiksi. Tällöin peliobjektit eivät vaikuta toistensa liikkeisiin, vaan ne ainoastaan havaitsevat kosketuksen. Tämänkaltaista liikettä kutsutaan kinemaattiseksi (engl. kinematic) liikkeeksi. Rigidbody-komponentissa on Is Kinematic -ominaisuus, joka poistaa peliobjektin kontrollin fysiikkamoottorilta ja mahdollistaa sen liikuttamisen kinemaattisesti.

Rigidbodyn muutokset tulisi tehdä FixedUpdate-funktiossa eikä Update-funktiossa, jossa suurin osa muista muutoksista tehdään. Syynä tälle on se, että Unity tekee fysiikkapäivitykset aina saman ajan välein, toisin kuin Update-funktiossa, jossa framejen kestot ovat vaihtelevia. Unityn dokumentaatiossa (3, linkit Scripting API -> Unity Engine -> Classes -> Rigidbody) asia selitetään näin:

> “Physics updates are carried out in measured time steps that don't coincide with the frame update. FixedUpdate is called immediately before each physics update and so any changes made there will be processed directly. “

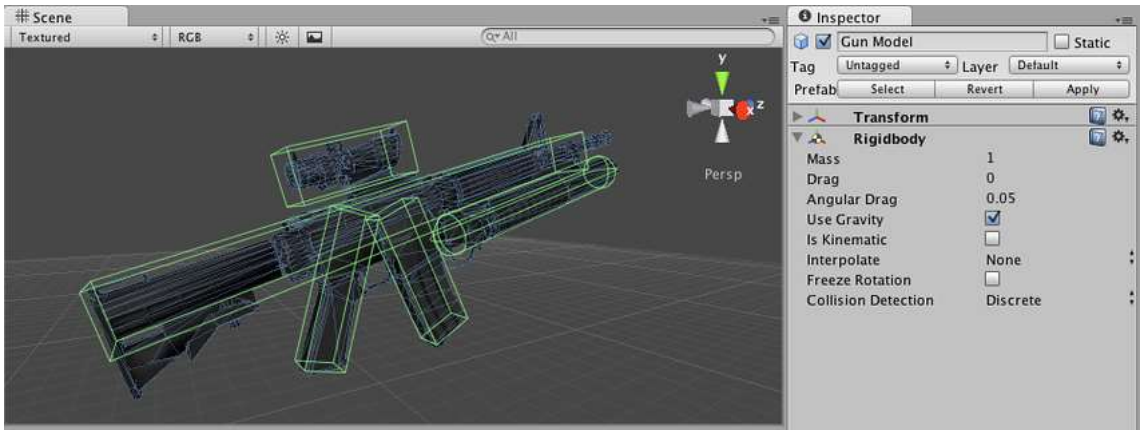
2.2.6 Collider

Collider on näkymätön komponentti, joka määrittää peliobjektin muodon törmäyksiä (engl. collision) varten. Sen ei tarvitse olla tarkalleen saman muotoinen peliobjektin mallin (engl. mesh tai sprite) kanssa, ja pelin suorituskyky laskee, jos liian tarkkoja collidereita käytetään paljon. Näitä tarkasti kappaleen muotoisia collidereita kutsutaan mesh collidereiksi.

Suorituskyvyn kannalta on parempi käyttää Unityn valmiita collider-tyyppejä. Näitä ovat kolmiulotteisissa peleissä Box Collider, Sphere Collider ja Capsule Collider, ja kaksiulotteisissa peleissä Box Collider 2D ja Circle Collider 2D. Näitä kutsutaan primitive collidereiksi.

Yksi peliobjekti voi sisältää monta collideria (kuva 2). Useita primitive collidereita käyttämällä voidaan päästä pelattavuuden kannalta tarpeeksi tarkkoihin arvioihin peliobjektien muodosta ja pitää samalla suorituskyky hyvänä.

Triggerit (engl. trigger) ovat collidereita, jotka eivät luo varsinaista törmäystä. Ne eivät käyttäydy kuin kiinteät esineet, vaan antavat toisten esineiden liikkua niiden lävitse. Triggereitä käytetään vain havaitsemaan, kun yksi collider tulee toisen colliderin alueelle.



KUVA 2. Aseen collider luotu monta primitive collideria käyttämällä. Vihreät viivat havainnollistavat collidereita. Ridigbody-komponentin asetukset näkyvissä Inspector-ikkunassa. (3, linkit Physics -> 3D Physics Reference -> Rigidbody.)

Törmäyksen tunnistaminen skriptissä tapahtuu OnCollisionEnter-funktiolla. Mikäli collider on asetettu triggeriksi, sen käsitteleminen tapahtuu OnTriggerEnter-funktiolla. Myös törmäyksien jatkumista ja päättymistä voidaan käsitellä vastaavasti OnCollisonStay- ja OnCollisionExit-funktiolla. Esimerkiksi peliobjektin kosketus triggeriin voidaan käsitellä esimerkin seuraavanlaisella funktiolla:

```
private void OnTriggerEnter2D(Collider2D collision)
{
     print("Collision begins");
}

private void OnTriggerExit2D(Collider2D collision)
{
     print("Collision ends");
}
```

2.2.7 Coroutine

Tavallinen funktio suoritetaan kokonaan yhden framen aikana. Tämä tarkoittaa, että sillä ei voi suorittaa ajan myötä tapahtuvia tapahtumasarjoja, kuten liikkeitä ja animaatioita. Tämän kaltaiset tilanteet on mahdollista suorittaa Update-funktion sisällä, mutta monesti kaikkia tehtäviä ei tarvitse suorittaa joka framen kohdalla, vaikka niitä suoritettaisiin säännöllisesti. Joka framen kohdalla suoritettavat tehtävät voivat helposti laskea suorituskykyä. Tämän vuoksi monesti on käytännöllisempää käyttää coroutine-funktiota näihin tehtäviin.

Coroutine on funktio, jolla on kyky keskeyttää suorittaminen, palauttaa kontrolli Unitylle ja jatkaa suoritusta myöhemmin siitä kohdasta, mihin se jäi. Oletusarvoisesti suorittamista jatketaan seuraavan framen aikana, mutta sille on myös mahdollistaa asettaa ajallinen viive, jonka jälkeen suorittaminen jatkuu. Viivettä käyttämällä voidaan vähentää joka framen kohdalla suoritettavia tehtäviä, ja tämä voi tuoda huomattavan parannuksen suorituskykyyn. Viive asetetaan käyttämällä funktiota WaitForSeconds.

Coroutinen tyypiksi tulee asettaa IEnumerator, ja se vaatii toimiakseen yield return -lauseen jossain sen sisällä. Tämä yield return on se kohta koodissa, missä funktion suorittaminen keskeytetään ja jatketaan. C#-kielessä coroutinen käynnistämiseen käytetään StartCoroutine-funktiota.

3 MOBIILIPELI SPACE GAME

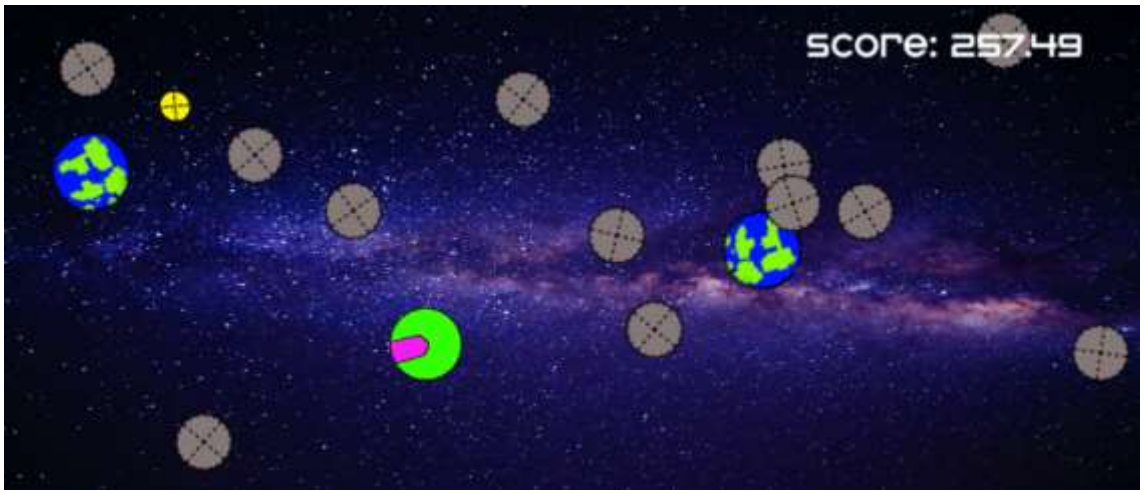
Opinnäytetyön yhteydessä kehitettiin mobiilipeli käyttäen Unity-pelimoottoria. Pelin projektinimeksi valittiin *Space Game*. Se on suunniteltu käytettäväksi mobiililaitteilla, mutta Unityn mahdollistaman monialustaisuuden ansiosta, sen käyttäminen on mahdollista myös esimerkiksi Windows-sovelluksena tai selaimessa. Pelin testaukset on tehty kahdella Android-laitteella sekä Unityn editorilla.

Lähes kaikki pelissä käytetyt ominaisuudet on tehty itse, tai ne ovat Unityn oletusarvoisesti tarjoamia ominaisuuksia. Pelin grafiikat on piirretty itse, joten ne ovat hyvin yksinkertaiset.

3.1 Pelin idea

Space Gamessa pelaaja lentää avaruudessa väistellen ympärillään lentäviä asteroideja ja pyrkii ampumaan asteroidien keskellä lentäviä planeettoja. Peli muistuttaakin paljon vuonna 1979 Atarin julkaisemaa kolikkopeliä *Asteroids.*

Tavoitteena pelissä on saavuttaa mahdollisimman suuri pistemäärä, ennen kuin pelaaja törmää asteroidiin, jolloin peli päättyy. Jokaisen planeetan tuhoaminen nostaa pelaajan pistemäärää ja kasvattaa pelihahmon kokoa. Suuremmalla pelihahmolla asteroidien välttteleminen on haastavampaa, joten pelaajan on kannattavaa pyrkiä pienentämään pelihahmoaan. Tämä tapahtuu keräämällä avaruudessa lentäviä tähtiä, jotka ovat pieniä keltaisia ympyröitä. Kuvassa 3 on havaittavissa kaikki mainitut peliobjektit. On myös huomioitavaa, että kuvassa pelihahmo on jo kasvanut paljon aloitustilannetta suuremmaksi.



KUVA 3. Pelinaikainen käyttöliittymä. Pelihahmo (vihreä ympyrä lähellä ruudun keskipistettä), planeetat (kaksi sinivihreää ympyrää), tähti (pieni keltainen ympyrä ruudun vasemmassa yläkulmassa) ja asteroidit (harmaat ympyrät).

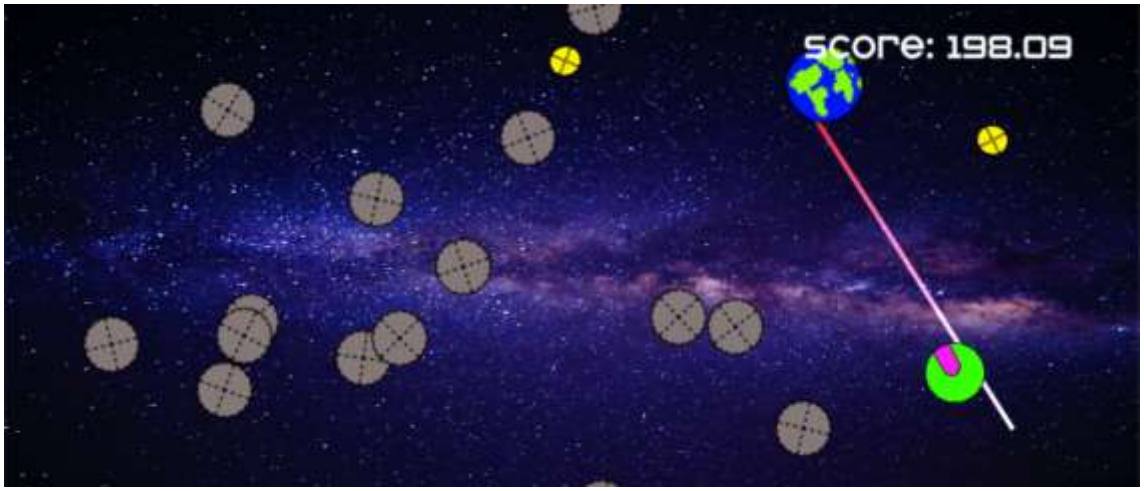
Mikäli pelaaja ampuu asteroidia, se hajoaa pienempiin osiin ja osat lähtevät leijailemaan satunnaisesti valittuihin suuntiin. Asteroidit on mahdollista hajottaa neljäsosiin ja kahdeksasosiin. Kahdeksasosan ampuminen tuhoaa palasen kokonaan. Asteroidien ampuminen ei lisää pelaajan pisteitä, ja pienet asteroidien osat täyttävät ruudun helposti, mikäli pelaaja ampuu niitä liikaa. Tämä vaikeuttaa pelitilannetta huomattavasti, koska pelaajalla on enemmän peliobjekteja näytöllä väisteltävänä.

Oleellista pelin pisteytysmekaniikassa on kuitenkin se, että planeettojen tuhoamisesta saatava pistemäärä on harvoin sama. Pisteet määräytyvät sen perusteella, kuinka kaukana tuhottu planeetta oli pelaajasta sekä kuinka pitkän matkan pelaaja liikkui ennen ammuksen lähettämistä planeettaa kohti. Mahdollisimman suuripisteisten yksittäisten laukausten tavoitteleminen onkin oleellista, koska pelihahmo kasvaa jokaisen tuhotun planeetan jälkeen aina saman verran, joten pienipisteiset laukaukset vaikeuttavat peliä kasvattamalla pelihahmoa, mutta samaan aikaan kasvattavat pelaajan pistetiliä vain hieman. Lisäksi pelissä on tulostaulukko parhaista yksittäisistä laukauksista. Toinen tulostaulukko pelissä on korkeimmista yhteispistemääristä. Pelaajan on siis hyvä pitää molemmat mielessä pelin edetessä.

3.2 Tähtäys, ampuminen ja osumat

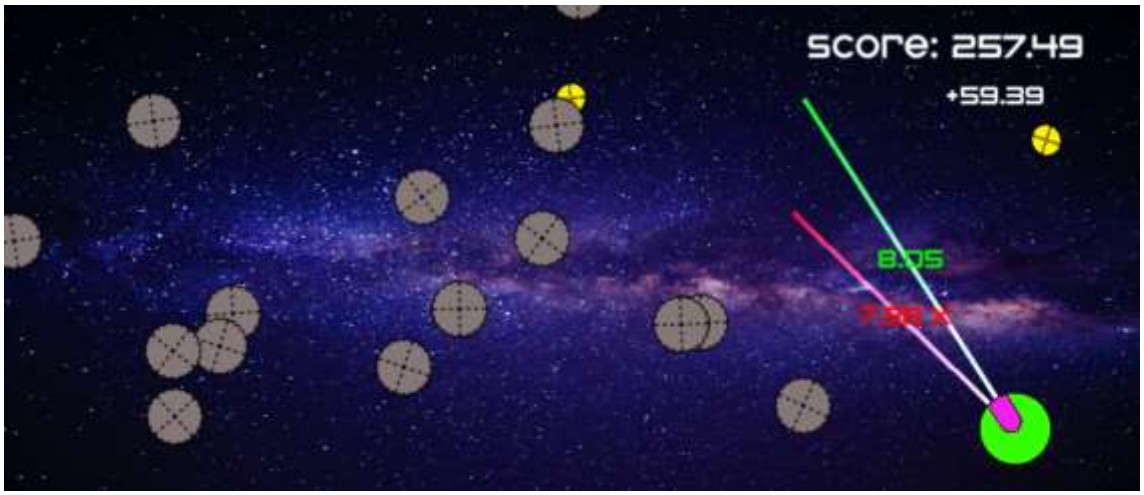
Pisteyttämisen ymmärtämiseksi on kuitenkin tärkeää ensin ymmärtää tähtääminen ja ampuminen pelissä. Tähtääminen tapahtuu vetämällä sormea kosketusnäytöllä, jolloin peli piirtää tähtäysviivan

näytölle. Kun sormi nostetaan pois näytöltä, tähtäysviiva jää piirretyksi odottamaan pelihahmon liikkumista. Painalluksen alussa pelihahmo lähtee liikkumaan kohti paikkaa, jossa sormi aloitti kosketuksen. Kun pelihahmo saavuttaa tämän paikan, se ampuu ammuksen tähtäysviivan suuntaisesti. Kuvassa 4 nähdään vaaleanpunainen tähtäysviiva piirrettynä näytölle. Pelihahmo on kuvassa jo lähellä tähtäyksen aloituspistettä, josta ampuminen tapahtuu. Tämä pää viivasta on vaaleampi kuin viivan toinen pää, jota kohti ammutaan.



KUVA 4. Pelaaja ampumassa kohti planeettaa. Vaaleanpunaisen tähtäysviivan vaaleampi pää näyttää pistettä, josta ammus lähetetään, ja tummempi pää suuntaa, johon se ammutaan.

Ammuksen osuessa planeettaan näytölle piirretään kaksi uutta viivaa. Ensimmäinen viivoista kuvaa pelaajan tähtäyksen aikana liikkumaa matkaa. Tätä matkaan kuvataan punaisella viivalla. Toinen viivoista kuvaa luodin lentorataa ja osuman etäisyyttä. Tätä kuvataan vihreällä viivalla. Lisäksi näytöllä ilmoitetaan nämä etäisyydet numeroilla. Yksittäisen planeetan tuhoamisesta saadut pisteet saadaan kertomalla nämä kaksi lukua keskenään. Liikuttu matka (punainen viiva) toimii kertoimena ammuksen kulkemalle matkalle (vihreä viiva). Kuvassa 5 on havaittavissa molemmat viivat sekä niiden pituudet numeroina. Kuvan tilanteessa kertoimeksi on saatu 7,38 ja osuman etäisyydeksi 8,05. Niiden tulo ilmoitetaan oikeassa yläkulmassa kokonaispistemäärän alapuolella.



KUVA 5. Kuva pelitilanteesta, jossa pelaaja on juuri tuhonnut planeetan. Näytölle piirretyt viivat havainnollistavat osuman pisteytystä. Uusi pistemäärä ilmoitetaan myös ruudun oikeassa yläkulmassa.

On huomioitavaa, että ampuminen voi tapahtua myös paikoiltaan, jos pelaaja jatkaa tähtäämistä, vaikka pelihahmo on jo saavuttanut laukauspisteen. Tällöin punaista viivaa ei piirretä ja kertoimeksi asetetaan 1. Tämä tuo peliin lisähaastetta ja vaatii pelaajalta hieman ennakointikykyä, jotta laukaukset osuisivat liikkuviin kohteisiin.

Osuman jälkeen viivat ja etäisyyttä havainnollistavat tekstit häivytetään näytöltä seuraavan parin sekunnin aikana. Lisäksi tekstit liikkuvat hitaasti ylöspäin näytöllä. Tämä lisää peliin elävyyttä. Häivyttäminen tehdään coroutine-funktiolla. Tarkempaa tietoa coroutinen toiminnasta kohdassa 2.2.7.

Alla esitellään tekstien häivyttämiseen käytetty funktio. Funktion parametrina annetaan aika float time, jolla määritellään aika, jonka verran häivytys kestää. Toisena parametrina annetaan Text text, joka viittaa siihen tekstikomponenttiin, joka funktiolla häivytetään. Häivyttäminen tapahtuu komponentin värin alpha-arvoa eli läpinäkyvyyttä muutamalla. Värin RBG-arvot pysyvät koko suorituksen ajan samana. Ennen häivyttämisen aloittamista funktio odottaa sekunnin käyttämällä yield return new WaitForSeconds(1) -funktiota. Tekstit näkyvät siis sekunnin ajan selvänä näytöllä, jotta pelaajalla on aikaa lukea ne. Tämän jälkeen häivyttäminen suoritetaan while-silmukan sisällä, kunnes värin alpha-arvo on nolla. Tekstien häivyttämiseen käytettävä funktio on seuraavanlainen:

```
IENumerator FadeText(float time, Text text)
{
      text.color = new Color(text.color.r, text.color.g,
```

```
            text.color.b, 1);
        yield return new WaitForSeconds(1);
        while (text.color.a > 0.0f)
        {
            text.color = new Color(text.color.r,
                text.color.g, text.color.b,
                text.color.a - (Time.deltaTime / time));
            yield return null;
        }
    }
```

Osumaa havainnollistavien viivojen häivyttäminen tapahtuu lähes samanlaisella funktiolla. Erona funktioissa on se, että viivojen häivyttämiseen käytetty funktio tekee häivyttämisen tekstikomponentin sijaan LineRenderer-komponenttiin, jota käyttäen viivat on piirretty. Lisäksi yhdessä LineRenderer-komponentissa on kaksi väriä: viivan aloituspään väri ja viivan loppupään väri. Molemmat näistä väreistä häivytetään saman while-silmukan sisällä.

3.3 Pelihahmon ja muiden peliobjektien liikkuminen

Pelihahmoa liikutetaan pelissä koskettamalla kosketusnäyttöä. Hahmo lähtee liikkumaan kohti kosketuspistettä heti kosketuksen alussa. Myös tähtäämiseen käytettävässä näytön pyyhkäisemisessä peli rekisteröi pyyhkäisyn aloituspisteen liikkumisen kohteeksi. Tietokoneella pelattaessa kontrollit toimivat samalla tavalla hiirellä tehtynä.

Unity tarjoaa monta erilaista tapaa peliobjektien liikuttamiseen. Sopivan tavan valintaan vaikuttavia asioita on monia, kuten minkä tyyppistä peliä ollaan tekemässä, minkälaisia fysiikkaominaisuuksia siinä käytetään ja tuleeko liikkeen olla tasaista vai vaihtelevaa.

Unity tarjoaa Rigidbody- ja Transform-komponentit peliobjektien liikuttamista varten. Suurimpana erona näiden kahden välillä on se, että Rigidbody ottaa huomioon fysiikoiden vaikutukset liikkeeseen. Se voi antaa fysiikkamoottorin hoitaa osan, tai kaikki, peliobjektiin vaikuttavista liikkeistä. Esimerkkinä tästä voidaan käyttää Rigidbody-komponenttiin ohjelmoitua painovoimaa: painovoima voi olla ainoa kappaleeseen vaikuttava voima, tai se voi vaikuttaa kappaleeseen muiden voimien lisäksi. Lisätietoa Transform- ja Rigidbody-komponenteista kodissa 2.2.4 ja 2.2.5.

Rigidbodyn käyttäminen mahdollistaa sulavamman liikkeen kuin Transformin käyttäminen. Esimerkkinä tästä voidaan käyttää seinään törmäämistä: kun Rigidbodyllä liikutettava peliobjekti törmää seinää, se pysähtyy jääden paikoilleen seinää vasten. Kun taas Transformilla liikutettava peliobjekti törmää seinään, se jää tärisemään seinää vasten, pyrkiessään jatkamaan matkaa seinän läpi.

Seuraavaksi käydään läpi erilaisia yleisimpiä liikkumistapoja.

1. Transform.positionin muokkaaminen lisäämällä uusi sijainti peliobjektin senhetkiseen sijaintiin voidaan tehdä seuraavasti:

```
transfrom.position += transform.forward
   * Time.deltaTime;
```

2. Transfrom.position muokkaaminen Transform.Translate-funktiolla. Toimii lähes samalla tavalla kuin kohta 1, mutta uuden sijainnin sijaan sille annetaan määrä, jonka se liikkuu:

```
transform.Translate(movement * Time.deltaTime);
```

3. Transform.position muokkaaminen Vector3.MoveTowards-funktiolla. Funktiota käytetään suorassa linjassa kahden pisteen välillä liikkumiseen seuraavasti:

```
transform.position = Vector3.MoveTowards
   (transform.position, target.position, speed);
```

4. Rigidbody.MovePosition on samantyylinen kuin kohta 1, mutta rigidbodyn fysiikkaominaisuuksien ansiosta liike on sulavampi ja ottaa huomioon muut liikkeeseen vaikuttavat fysiikat:

```
rb.MovePosition(transform.position
   + transform.forward * Time.deltaTime);
```

5. Rigidbody.AddForce liikuttaa peliobjektia lisäämällä siihen voimaa halutun vektorin suuntaisesti. Funktiolle voidaan myös asettaa parametrina force mode, jolla voidaan vaikuttaa halutun voiman tyyppiin (force, acceleration, impulse, velocity change). Funktiota voidaan käyttää seuraavalla tavalla:

```
rb.AddForce(transform.forward * thrust,
   ForceMode.Impulse);
```

6. Rigidbody.velocity vaikuttaa suoraan peliobjektin nopeuteen. Tätä tapaa ei suositella käytettäväksi kuin erityistapauksissa, koska se voi aiheuttaa odottamatonta ja epärealistista käytöstä. Tällainen erityistapaus on esimerkiksi hyppääminen, jossa nopeuden muutoksen tulee tapahtua välittömästi. Huomioitavaa on myös se, että kun peliobjektille asetetaan nopeus, sitä ei tarvitse asettaa joka framen kohdalla uudestaan. Nopeus voidaan asettaa seuraavasti:

```
rb.velocity = new Vector3(0, 10, 0);
```

Pelissä ei käytetä Unityn valmiita fysiikoita, joten liikkeet on tehty suoraan Transform-komponenttia muokkaamalla. Pelihahmon liikkuminen on tehty Vector2.MoveTowards-funktiolla. Kyseistä funktiota käytetään peliobjektin liikuttamiseen kahden pisteen välillä. Hahmoa liikutettaessa nämä kaksi pistettä ovat aina tiedossa – hahmon sijainti ensimmäisenä pisteenä ja sormen painalluksen sijainti toisena pisteenä – joten funktio sopii käyttötarkoitukseen erittäin hyvin.

Muiden peliobjektien – eli asteroidien, planeetoiden, tähtien ja ammuksen – liikuttaminen tapahtuu eri tyylillä. Myös niillä on kohdepiste, jota kohti ne liikkuvat, mutta kohteen saavuttamisen hetkellä ne eivät pysähdy, vaan jatkavat samaan suuntaan samalla nopeudella. Tähän tarkoitukseen sopiva tapa on Transform.Translate-funktio; funktiolle voidaan antaa suunta, jota kohti peliobjekti liikkuu, ja objekti pitää liikkeensä vielä alkuperäisen kohteen ohitettuaan. Alla nähdään suuntavektorin laskeminen käyttäen peliobjektin senhetkistä sijaintia ja parametrina annettua kohteen sijaintia. Esimerkiksi ampumistilanteessa kohdepisteeksi asetetaan tähtäysviivan loppupää. Suunta saadaan vähentämällä peliobjektin sijainti kohteen sijainnista seuraavasti:

```
void GetDirection(Vector3 target)
{
```

```
        direction = target – transform.position;
        direction.Normalize();
    }
```

Vektorin laskemisen jälkeen käytetään Vector3-luokan Normalize-funktiota. Tämä funktio säilyttää vektorin alkuperäisen suunnan, mutta asettaa vektorin suuruudeksi 1. Tämä mahdollistaa myöhemmin peliobjektia liikutettaessa ymmärrettävän nopeuden laskemisen. Ilman Normalize-funktiota nopeudet voisivat olla arvaamattomia.

Kun suuntavektori on laskettu, peliobjektin liikkuminen voidaan tehdä Update-funktion sisällä. Funktiolle annetaan parametrina laskettu suuntavektori direction kerrottuna ajan muutoksella Time.deltaTime ja peliobjektin halutulla liikkumisnopeudella moveSpeed (tarkempaa tietoa Update-funktiosta ja ajan muutoksesta kohdassa 2.2.2). Liikkuminen voidaan tehdä seuraavasti:

```
void Update()
{
    transform.Translate(direction * Time.deltaTime
        * moveSpeed, Space.World);
}
```

Viimeisenä parametrina annettu space määrittää, onko peliobjektin liikkeet suhteellisia pelimaailman koordinaatteihin vai kappaleen omiin koordinaatteihin ja kiertoon (engl. rotation). Unityn dokumentaatiossa (3, linkit Scripting API -> UnityEngine -> Enumerations -> Space.) asia selitetään näin:

> ”The coordinate space in which to operate.
>
> Use Space.world to transform a GameObject using Unity’s world coordinates, ignoring the GameObject’s rotation state. Use Space.self to transform a GameObject using its own coordinates and consider its rotations.”

3.4 Asteroidien, planeetoiden ja tähtien luominen

Asteroidit, planeetat ja tähdet luodaan satunnaisesti valittuihin paikkoihin näytön alueen ulkopuolella, josta ne lähtevät etenemään kohti satunnaisesti valittuja paikkoja näytön sisäpuolella. Mikäli peliobjektit poistuvat näytön alueelta, ne poistetaan pelistä automaattisesti.

Jotta pelin haastavuus olisi sopiva, tulee tiettyjä peliobjekteja luoda sopivan ajan välein. Asteroidien luominen on kaikista tiheintä, jotta peli ei olisi liian helppo. Asteroideja luodaankin pelissä yksi sekunnissa, joka on huomattavasti nopeammin kuin muita peliobjekteja: planeettojen luomistiheys on yksi seitsemässä sekunnissa ja tähtien yksi 15 sekunnissa. Peliobjektien luominen tapahtuu coroutine-funktiolla, jonka voi laittaa odottamaan peliobjektien luomisen välissä halutun ajan verran (lisätieto coroutinesta kohdassa 2.2.7).

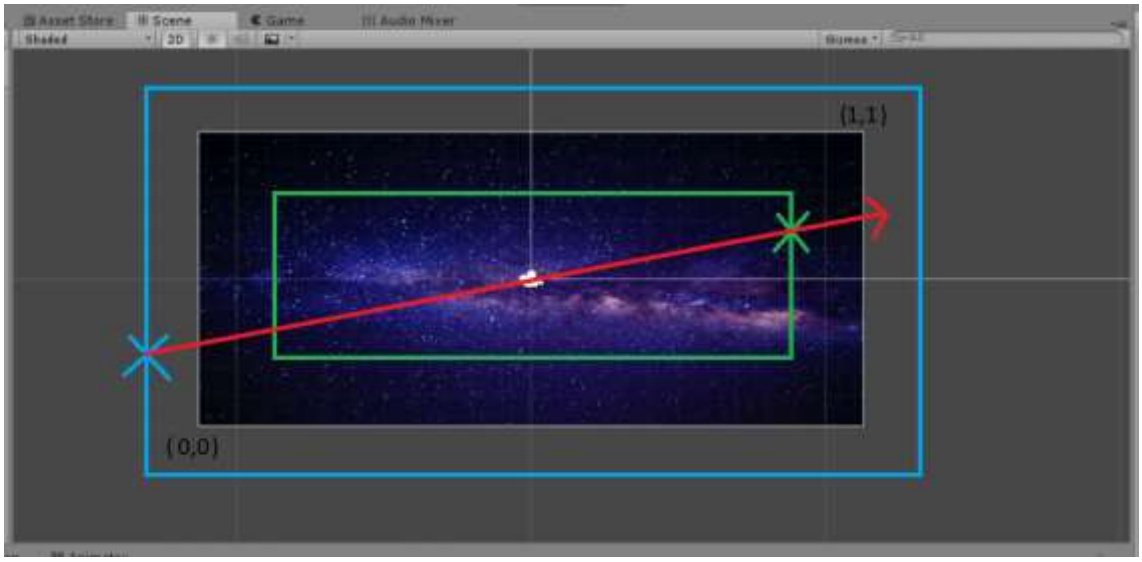
Toinen tärkeä pelin haastavuuteen vaikuttava tekijä on peliobjektien nopeus. Nopeus asetetaan peliobjekteille niiden luomisen jälkeen, mutta se ei ole aina sama. Peli asettaa jokaiselle luodulle peliobjektille satunnaisen nopeuden väliltä 0,5–3. Vertailun vuoksi pelihahmon liikkumisnopeus on 5. Muita objekteja nopeamman hahmon kontrollointi mahdollistaa asteroidien välttelemisen, mutta nopeimpiin objekteihin verrattuna ero ei ole liian suuri, jotta peli kävisi liian helpoksi.

Käyttämällä Unityn Camera-luokan funktiota ViewportToWorldPoint voidaan käsitellä pelimaailman koordinaatteja suhteellisesti kameraan nähden, ja tämä helpottaa niiden käsittelemistä huomattavasti. Muunnoksen jälkeen, näytön koordinaatit ovat välillä (0, 0) ja (1, 1), joista ensimmäinen mainittu on näytön vasen alakulma ja jälkimmäinen sen oikea yläkulma. Unityn dokumentaatiossa (3, linkit Scripting API -> UnityEngine -> Classes -> Camera -> ViewportToWorldPoint.) asia selitetään näin:

> "Transforms position from viewport space into world space.
>
> Viewport space is normalized and relative to the camera. The bottom-left of the viewport is (0,0); the top-right is (1,1)."

Kuvassa 6 on havainnollistettu näytön alue, peliobjektien luomisalue, niiden kohdealue, sekä esimerkki yhden peliobjektin liikkeestä. Taustakuvan täyttämä alue kuvaa pelaajan laitteen näyttöä.

Kuvaan on myös merkitty näytön kulmat (0, 0) ja (1, 1). Näytön ulkopuolella olevalla sinisellä suorakulmiolla kuvataan peliobjektien luomisaluetta. Peliobjekti voidaan siis luoda mihin tahansa kohtaan tämän suorakulmion reunoja. Näytön sisäpuolella oleva vihreä suorakulmio kuvaa peliobjektien kohdealuetta, josta luotu peliobjekti ottaa satunnaisesti valitun pisteen kohteekseen, jota kohti se lähtee liikkumaan. Punainen nuoli toimii esimerkkinä yhden peliobjektin mahdollisesta lentoradasta.



KUVA 6. Peliobjektien luomisalue (sininen), kohdealue (vihreä) ja esimerkki peliobjektin lentoradasta (punainen nuoli).

Luomisalueen etäisyys näytön reunasta on 10 % näytön leveydestä. Näytön kulmien ollessa pisteissä (0, 0) ja (1, 1), 10 %:n etäisyys saadaan helposti käyttämällä koordinaatteja (-0,1; -0,1) ja (1,1; 1,1). Kohdealueen etäisyys näytön reunasta on 20% näytön leveydestä. Kohdealue on näytön sisäpuolelle, joten alueen reunat tulevat välille (0,2; 0,2) ja (0,8; 0,8).

3.5 Datan tallentaminen

Pelissä tallennetaan tietoja sekä pelaajan laitteeseen paikallisesti että pilvipalveluun. Paikalliseen tallentamiseen käytetään Unityn PlayerPrefs-luokkaa ja pilvipalveluna käytetään palvelua nimeltä Dreamlo.

3.5.1 PlayerPrefs

Unity tarjoaa PlayerPrefs-luokan, jolla voi tallentaa halutut tiedot laiteeseen pelisessioiden välillä. Tämän luokan käyttömahdollisuudet ovat kuitenkin hyvin rajalliset, koska sitä käyttämällä on mahdollista tallentaa ainoastaan int-, float- ja string-tyyppisiä muuttujia. Ongelman ratkaisemiseksi wiki.unity3d.com-sivustolta löytyy Eric Hainesin kirjoittama luokka PlayerPrefsX, jota käyttämällä on mahdollista tallentaa esimerkiksi taulukoita, vektoreita ja värejä (4).

PlayerPrefs-luokan käyttäminen on yksinkertaista. Jokaiselle tallennettavalle muuttujalle annetaan oma avain (engl. key), jota käyttämällä muuttujaa voidaan lukea ja päivittää. Muuttujan voi myös poistaa poistamalla avain johon sen on tallennettu. Muuttujien tallentaminen tapahtuu set-funktioilla, ja niiden lukeminen tapahtuu get-funktioilla. Esimerkiksi nimimerkin “Esko” tallentaminen username-avaimeen tapahtuu seuraavasti:

```
PlayerPrefs.SetString("username", ”Esko”);
```

Saman username-avaimen lukeminen tapahtuu seuraavasti:

```
string username = PlayerPrefs.GetString("username");
```

Pelissä pidetään tilastoja pelaajan suoriutumisesta. Nämä tiedot tallennetaan pelaajan laitteeseen PlayerPrefs-luokkaa käyttämällä. Tilastoitavia tietoja ovat

- paras kokonaispistemäärä
- pelattujen pelien määrä
- pisimmän pelin kesto
- pelien yhteenlaskettu kesto
- tuhottujen planeettojen määrä
- paras yksittäinen laukaus
- kaikkien laukausten keskiarvo
- paras laukauksen etäisyys
- paras laukauksen kerroin
- kerättyjen tähtien määrä.

Näiden tietojen lisäksi tallennettavana ovat myös pelaajan käyttämät asetukset. Asetuksissa määritetään pelaajan käyttäjänimi sekä pelinaikainen laitteen kierron lukitus. Kaikki tallennukset, paitsi laukausten keskiarvo, on tehty käyttämällä PlayerPrefs-luokkaa. Laukausten keskiarvon laskemiseen tarvitaan lista laukauksista, joiden perusteella keskiarvo lasketaan. Tämä lista on taulukkona, jonka tallentamiseen tarvitaan luokkaa PlayerPrefsX.

3.5.2 Dreamlo

Pelissä on kaksi tulostaulukkoa parhaita tuloksia varten. Toiseen taulukkoon tallennetaan korkeimmat kokonaispistemäärät ja toiseen korkeimmat yksittäiset laukaukset. Tulostaulukot on tehty käyttämällä ilmaista palvelua nimeltä Dreamlo, jonka tekijä ja ylläpitäjä on amerikkalainen ohjelmoija Carmine Guidan. Dreamlon idea on tarjota pelikehittäjille helppokäyttöinen ja nopea tapa luoda tulostaulukot peleihin, ilman tarvetta omille palvelimille ja tietokannoille (5, linkki faq).

Dreamlo hoitaa kaiken kommunikoinnin palvelimen kanssa käyttämällä http:n get-metodia (5, linkki developer). Käyttäjälle Dreamlo antaa uniikin verkko-osoitteen, jota käyttämällä tulostaulukkoa voidaan muokata tai lukea. Muokkaamiseen tarkoitettu osoite sisältää pitkän merkkijonon, jota kutsutaan private codeksi. Toinen palvelun tarjoama merkkijono on public code, jota käytetään tulostaulukkojen lukemista varten. On siis tärkeää, että private code pysyy ainoastaan pelin kehittäjien tiedossa, etteivät ulkopuoliset käyttäjät pääse muokkaamaan taulukoita.

Dreamlo antaa valmiit esimerkkikoodit sen käyttämisestä, joten sen käyttämisen pitäisi olla kaikille helppoa ja mutkatonta. Esimerkiksi uuden käyttäjän lisäämiseksi tarvitaan private coden lisäksi ainoastaan käyttäjän nimi sekä pistemäärä, jotka lisätään verkko-osoitteeseen seuraavasti:

http://dreamlo.com/lb/privatecode/add/username/100

missä

privatecode on tulostaulukon uniikki private code,
username on käyttäjännimi,
100 on pistemäärä.

Mikäli samalla käyttäjänimellä pelaava pelaaja saavuttaa korkeamman pistemäärän kuin edellisellä kerralla, Dreamlo päivittää tuloksen automaattisesti ja jättää ainoastaan korkeamman tuloksen tulostaulukkoon.

Tulostaulukon muokkaaminen onnistuisi siis esimerkiksi vain selainta käyttämällä. Unityä käytettäessä verkko-osoitteiden käyttäminen onnistuu helposti käyttämällä siihen tarkoitettua luokkaa WWW.

4 KÄYTTÖLIITTYMÄT

Käyttöliittymällä tarkoitetaan käytettävän ohjelman tai laitteen osaa, jonka avulla käyttäjä käyttää tuotetta. Siihen kuuluu informaatio, jota käyttäjä saa ohjelmalta, ja keinot, joilla käyttäjä kontrolloi sitä. (6.)

Peleissä osa käyttöliittymästä on ne asiat, jotka pelaaja näkee. Tällä tarkoitetaan graafista käyttöliittymää eli asioita, jotka piirretään näytölle. Näitä ovat yleensä peleissä valikot, kuvat, napit ja muut vastaavat. Toinen osa käyttöliittymästä on se, miten pelaaja peliä hallitsee. Se voi tapahtua esimerkiksi näppäimistöllä, hiirellä, kosketusnäytöllä tai peliohjaimella. (7.)

4.1 Hyvän käyttöliittymän ominaisuudet

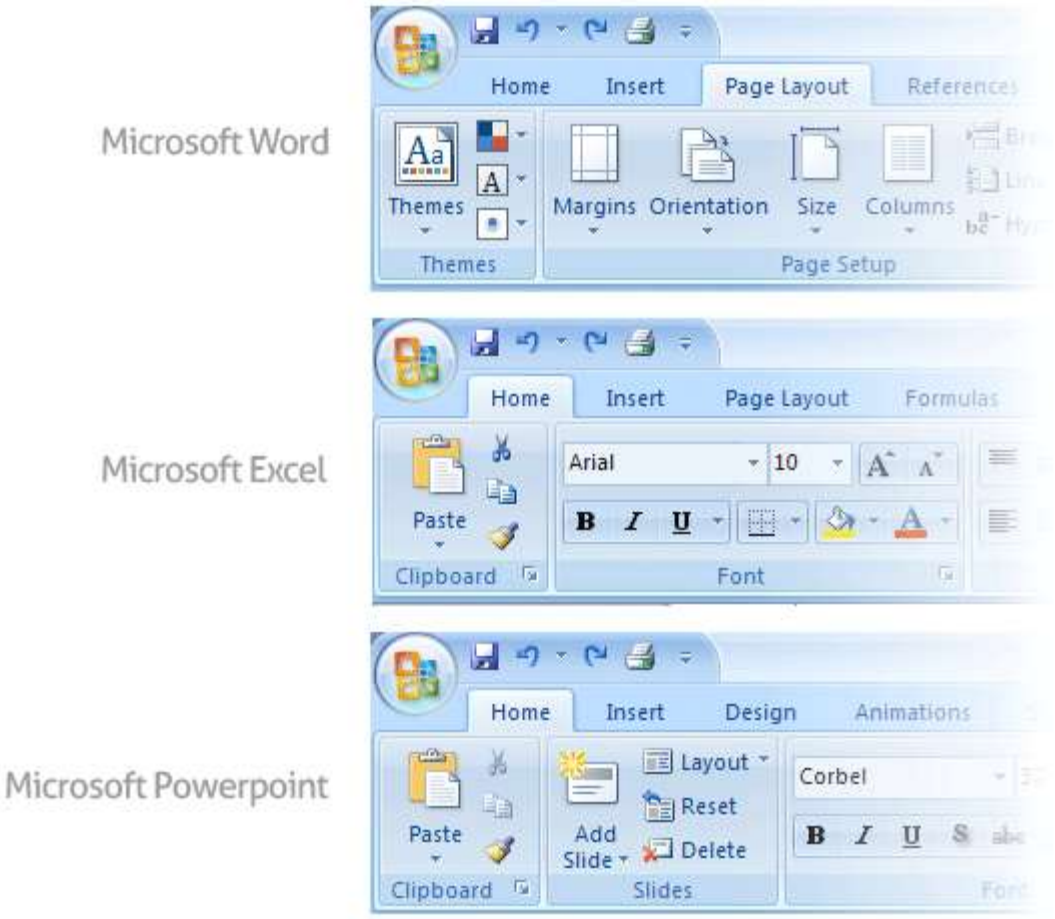
Hyvän käyttöliittymän tärkein ominaisuus on, että se on tarpeeksi selkeä. Jos käyttäjä ei ymmärrä kuinka käyttöliittymä toimii, koko ohjelma muuttuu hyödyttömäksi. Käyttöliittymän tulisi tehdä kaikki asiat selväksi sekä visuaalisen hierarkian että sisällön kannalta, jotta sen käytössä ei olisi yhtään epäselvyyksiä. (8; 9; 10.)

Käyttöliittymän tulisi tarjota tarvittava tieto tarpeeksi tiivistettynä. Selkeyden kannalta on hyvä lisätä selventäviä määritelmiä ja selityksiä, mutta tämä johtaa helposti siihen, että näytöllä on liikaa asioita huomioitavana. Silloin käyttäjän on vaikea löytää asioita, joita hän etsii, ja häneltä kuluu turhaa aikaa selityksien lukemiseen. Toimivan käyttöliittymän kannalta onkin oleellista löytää tasapaino selvennyksien ja huomioitavien asioiden määrän välillä. (8.)

Käyttäjä voi ymmärtää käyttöliittymän toiminnan, vaikka hän käyttäisi sitä ensimmäistä kertaa. Tämä on mahdollista, kun tietyt elementit käyttöliittymässä ovat jo entuudestaan tuttuja. Käyttäjä on luultavasti käyttänyt aikaisemmin muita käyttöliittymiä, joista hänelle on jäänyt mieleen tietyt ominaisuudet. Kun nämä samat ominaisuudet löytyvät uudesta käyttöliittymästä, käyttöliittymä tuntuu tutulta ja johdonmukaiselta. Täysin uudenlaisen layoutin kehittäminen voi tuntua hyvältä ja jännittävältä idealta, mutta tämän kaltaisissa asioissa on yleensä parempi käyttää tuttua ja turvallista lähestymistapaa. (8; 9.)

Hyvä käyttöliittymä toimii nopeasti ja sulavasti, ja mikäli ohjelmassa on pitkiä latausaikoja, ne eivät vaikutta käyttöliittymän nopeuteen, vaan ne hoidetaan taustalla. Jos jokin asia vaatii käyttöliittymän käytön keskeyttämistä tai rajoittamista, käyttäjän tulisi saada jonkinlaista palautetta tapahtuvista asioista. Esimerkki tällaisesta palautteesta on latausikoni, josta käyttäjä voi päätellä, että ohjelma lataa, eikä ole jumittunut. Toinen yleisesti käytetty tapa on latauspalkki, josta käyttäjä saa myös jonkinlaisen arvion latauksen etenemisestä. Hitaasti toimiva käyttöliittymä voi antaa tunteen huonosti toimivasta ohjelmasta, vaikka todellisuudessa se toimisi muuten erinomaisesti. (8; 9; 10.)

Johdonmukainen käyttöliittymä sallii käyttäjien tunnistaa käyttökuviot. Tämä tarkoittaa, että samanlaiset asiat pysyvät vastaavanlaisina eri puolilla käyttöliittymää. Esimerkiksi monta erillistä sivua sisältävässä käyttöliittymässä palautuspainike on joka sivulla samassa paikassa ja samannäköinen. Samankaltaisuuden avulla käyttäjä pystyy ennakoimaan eri käyttöliittymäelementtien toimintaperiaatteet. Esimerkkinä kuvassa 7 nähdään kolmen Microsoft Office -ohjelmistoon kuuluvien ohjelmien työkalupalkit. Näiden ohjelmien käyttöliittymät ovat keskenään hyvin samankaltaiset. Kun käyttäjä oppii käyttämään yhtä ohjelmaa, seuraavan ohjelman opettelu tuntuu helpolta ja johdonmukaiselta. (8.)



KUVA 7. Microsoft Officen eri ohjelmien käyttöliittymät ovat keskenään samankaltaisia (8)

Estetiikka ei ole toimivan käyttöliittymän kannalta välttämätöntä, ja siisti ulkonäkö ei korvaa huonoa suunnittelua. Kuitenkin hyvään ulkonäköön panostaminen kannattaa. Houkuttelevan käyttöliittymän käyttäminen on miellyttävää, ja käyttäjät palaavat mielellään ohjelman pariin. On kuitenkin tärkeää huomioida, että käyttäjien kohderyhmä vaikuttaa siihen, minkälainen on hyvännäköinen käyttöliittymä. Käyttöliittymän ulkonäkö tulisikin suunnitella haluttua kohderyhmää varten. (8; 9)

Yleensä ohjelmissa on muutama ominaisuus, joita käytetään eniten. On tärkeää tiedostaa mitä nämä ominaisuudet ovat, ja mitä käyttäjät yrittävät niillä saavuttaa. Hyvä käyttöliittymä mahdollistaa niiden nopean ja tehokkaan käytön, esimerkiksi korostamalla niitä ja tarjoamalla niihin pikakuvakkeet. Muutama tärkeä ja nopeasti käytettävä ominaisuus on yleensä parempi kuin pitkä lista vähemmän hyödyllisiä ominaisuuksia. (8.)

Käyttäjät tekevät helposti virheitä ohjelmia käyttäessään. Käyttäjä voi esimerkiksi poistaa jonkin tiedoston tahtomattaan. Huonolla käyttöliittymällä pienikin virhe voi olla vakava ja tehdyt muutokset

peruuttamattomia. Hyvä käyttöliittymä sen sijaan antaa käyttäjälle mahdollisuuden perua vahingot. Se voi esimerkiksi antaa käyttäjälle mahdollisuuden perua tiedoston poistamisen, tai kysyä varmistusta aina tiedostoja poistettaessa. (8.)

4.2 Space Gamen käyttöliittymän suunnittelu

Space Gamen käyttöliittymää suunniteltaessa käytettiin apuna Paladin Studiosin julkaisemaa opasta *The 8-step Guide To Interface Design for iPhone Games (11).*

4.2.1 Kohdelaitteet

Käyttöliittymää tehtäessä on tärkeää ottaa huomioon laitteet, joille sitä ollaan tekemässä. Mobiililaitteiden ja tietokoneiden käyttöliittymät eroavat toisistaan, ja mobiililaitteissakin on suuria eroavaisuuksia. Esimerkiksi mobiililaitteiden näyttöjen resoluutiot vaihtelevat paljon.

Space Game on suunniteltu käytettäväksi Android-mobiililaitteilla. Android-laitteiden resoluutioissa on paljon eroavaisuuksia, joten yhdelle tietylle resoluutiolle tehtävä käyttöliittymä ei toimi. Space Gamen käyttöliittymä on suunniteltu skaalautumaan automaattisesti käytettävän laitteen resoluution mukaan. Lisää tästä kohdassa 4.3.1.

4.2.2 Mobiililaitteen käyttöasento

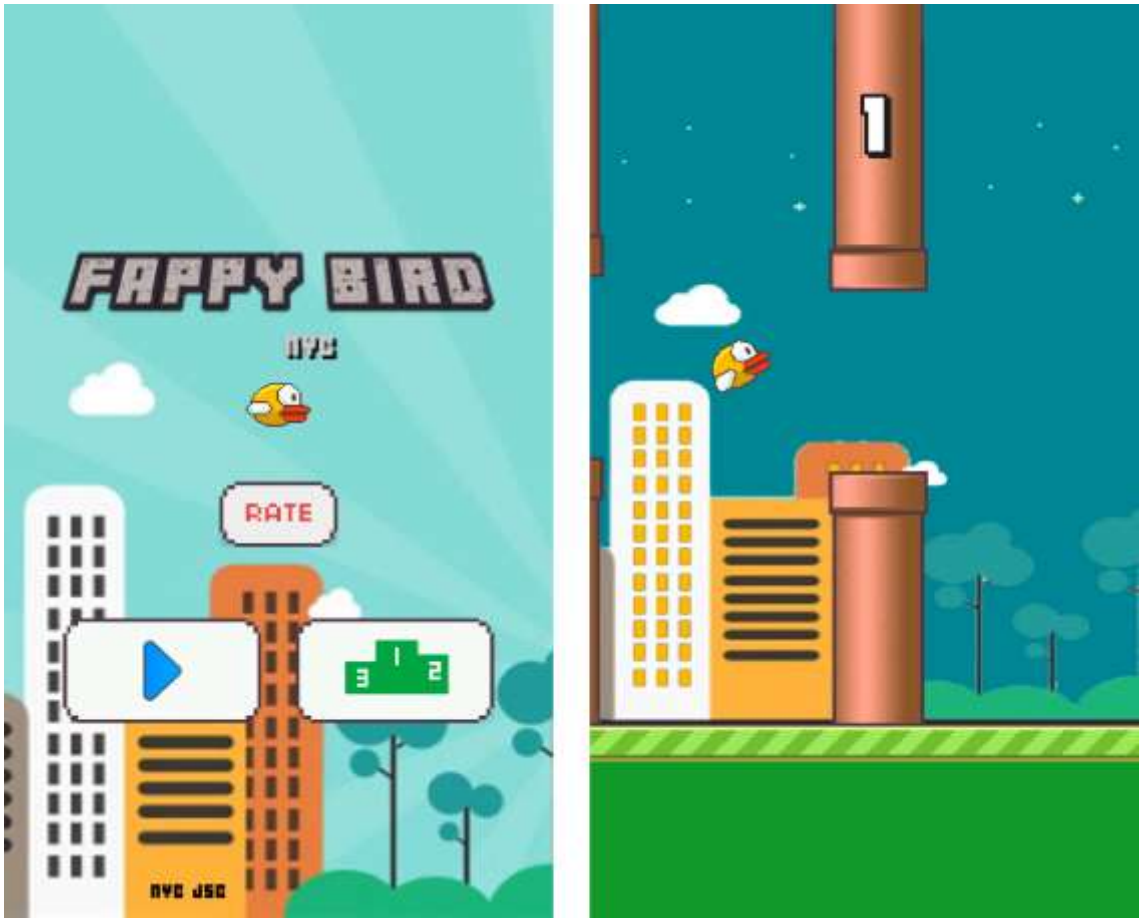
Mobiililaitteilla käytettävien sovellusten suunnittelussa on huomioitava, että laitetta on mahdollista käsitellä eri asennoissa. Suunnitteluvaiheessa onkin tärkeä päättää, missä asennossa laitetta käytetään, ja käyttöliittymä tulee suunnitella sen mukaisesti.

Space Gamen valikot on suunniteltu käytettäväksi vaaka-asennossa. Pelinaikainen kierto toimii kuitenkin molemmissa asennoissa. Pelin aikana laitetta kääntäessä peli ei saa kuitenkaan häiriintyä, eli peliobjektit eivät saa vaihtaa paikkaa näytöllä. Tämän vuoksi asennon vaihtaminen vaikuttaa ainoastaan tiettyihin käyttöliittymäelementteihin. Pelinaikainen kierto on myös mahdollista lukita pelin asetuksista. Lisää laitteen käyttöasennon toteutuksesta kohdassa 4.3.5.

4.2.3 Vaikutteiden etsiminen

Käyttöliittymää suunniteltaessa on hyvä katsoa vaikutteita jo olemassa olevista peleistä. Muiden suunnitelmien kopioimista tulisi välttää, mutta samantyyliset pelit toimivat hyvinä inspiraation lähteinä.

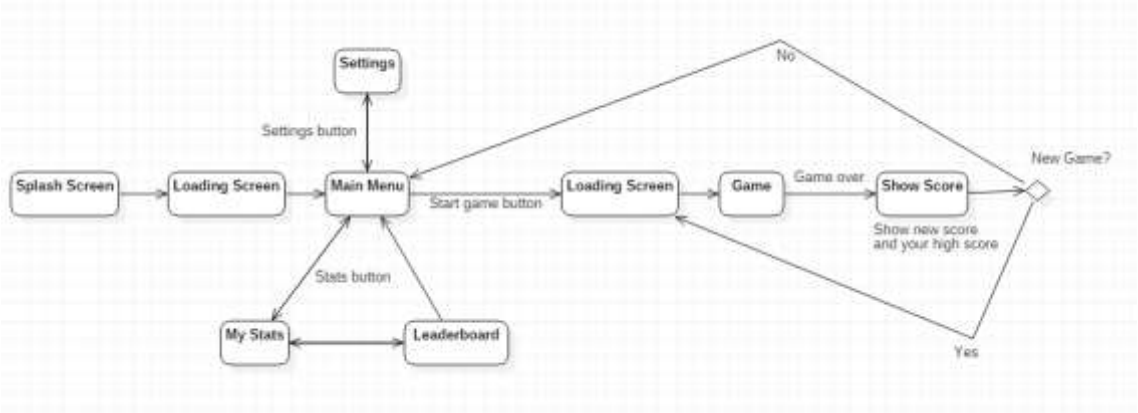
Space Game on yksinkertainen peli, eikä siinä ole paljoa ominaisuuksia. Tämän vuoksi vaikutteita haettiin myös yksinkertaisista peleistä. Kuvassa 8 on kuvakaappaukset Fappy Birdin päävalikosta sekä pelinäkymästä. Valikossa on vain pelin otsikko ja muutama nappi, ja kaikki turhat asiat on jätetty pois. Pelinäkymässäkin on ainoastaan yksi pisteitä näyttävä käyttöliittymäelementti, ja kaikki muu tila on varattu itse pelille.



KUVA 8. Fappy Birdin valikko- ja pelinäkymä.

4.2.4 Kaavio näkymistä ja ominaisuuksista

Suunnitteluvaiheessa on hyvä tehdä lista eri näkymistä ja ominaisuuksista, joita käyttöliittymä tarvitsee toimiakseen. Seuraavaksi tulisi miettiä, kuinka eri ominaisuudet liittyvät toisiinsa ja kuinka eri näkymien välillä liikutaan. Näistä asioista onkin hyvä piirtää kaavioita, joista näkee selvästi eri vaiheet ja toiminnot. Kaavioista tulisi käydä ilmi, mitä informaatiota ja painikkeita missäkin näkymässä on, ja miten käyttäjä liikkuu näkymien välillä. Kuvassa 9 näkyy koko Space Gamen toiminnallisuutta kuvaava kaavio.



KUVA 9. Suunnitelma Space Gamen näkymistä ja niiden välillä liikkumisesta.

4.2.5 Piirretty suunnitelma

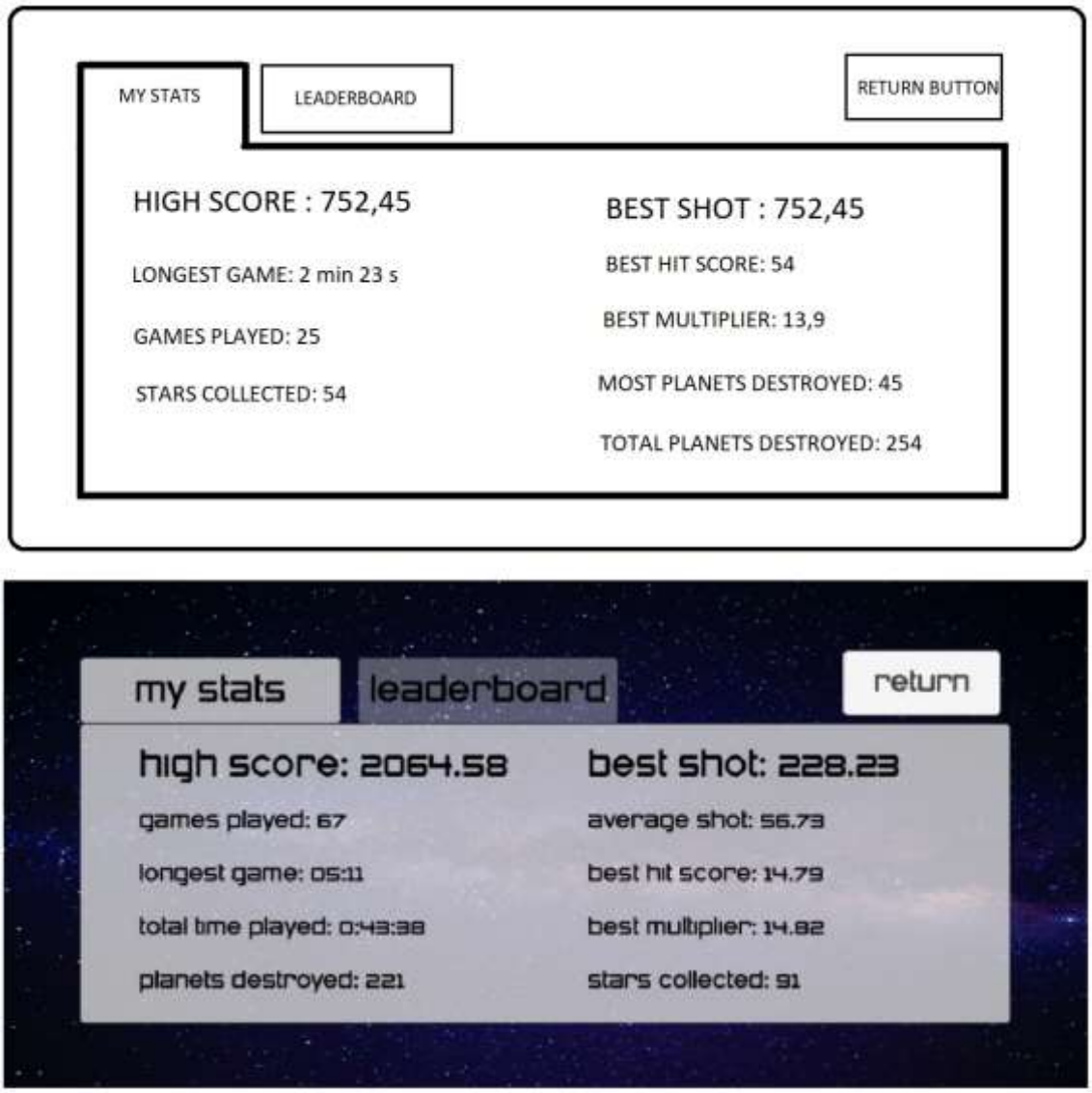
Ennen varsinaisen käyttöliittymän tekemistä siitä on hyvä piirtää jonkinlainen suunnitelma, josta näkyy käyttöliittymäelementtien sijoittelut ja toiminnallisuudet. Tässä vaiheessa tulisi päättää, mitä informaatiota näkymässä on ja mitä käyttäjä voi tehdä sillä.

Mobiililaitteille suunniteltaessa piirretty suunnitelma on hyvä laittaa kuvana laitteen näytölle. Tietokoneen näyttö on yleensä suurempi kuin mobiililaitteen näyttö, ja suunnitteleminen voi olla haastavaa eri kokoiselle näytölle. Tällä menetelmättä on hyvä varmistaa, että suunnitelma skaalaa hyvin myös eri kokoisella näytöllä.

KUVA 10. Piirretty suunnitelma tulostaulukosta

4.2.6 Lopputulos

Space Gamen käyttöliittymän lopullinen toteutus on hyvin lähellä alkuperäistä suunnitelmaa, eikä sitä alettu ensimmäisen toteutuksen jälkeen enää muuttamaan. Kuvassa 11 on verrattavana yhden näkymän piirretty suunnitelma ja lopullinen toteutus. Kaikki käyttöliittymän osat – fonttia lukuun ottamatta – on tehty Unityn omilla työkaluilla ja kuvilla.



KUVA 11. Pelaajan omat tulokset näyttävän ikkunan piirretty suunnitelma ja lopullinen toteutus

4.3 Käyttöliittymän tekeminen Unityllä

4.3.1 Canvas ja skaalaus

Canvas on peliobjekti, jonka sisälle tulisi sijoittaa kaikki käyttöliittymäelementit. Se määrittelee, miten käyttöliittymäelementit piirretään näytölle. Ne voidaan piirtää suhteellisesti näyttöön, kameraan tai pelimaailmaan nähden.

Pelissä canvas on asetettu piirrettäväksi suhteellisesti näyttöön nähden (Screen Space - Overlay). Lisäksi sen sisältämä komponentti Canvas Scaler asettaa elementit skaalautumaan näytön koon

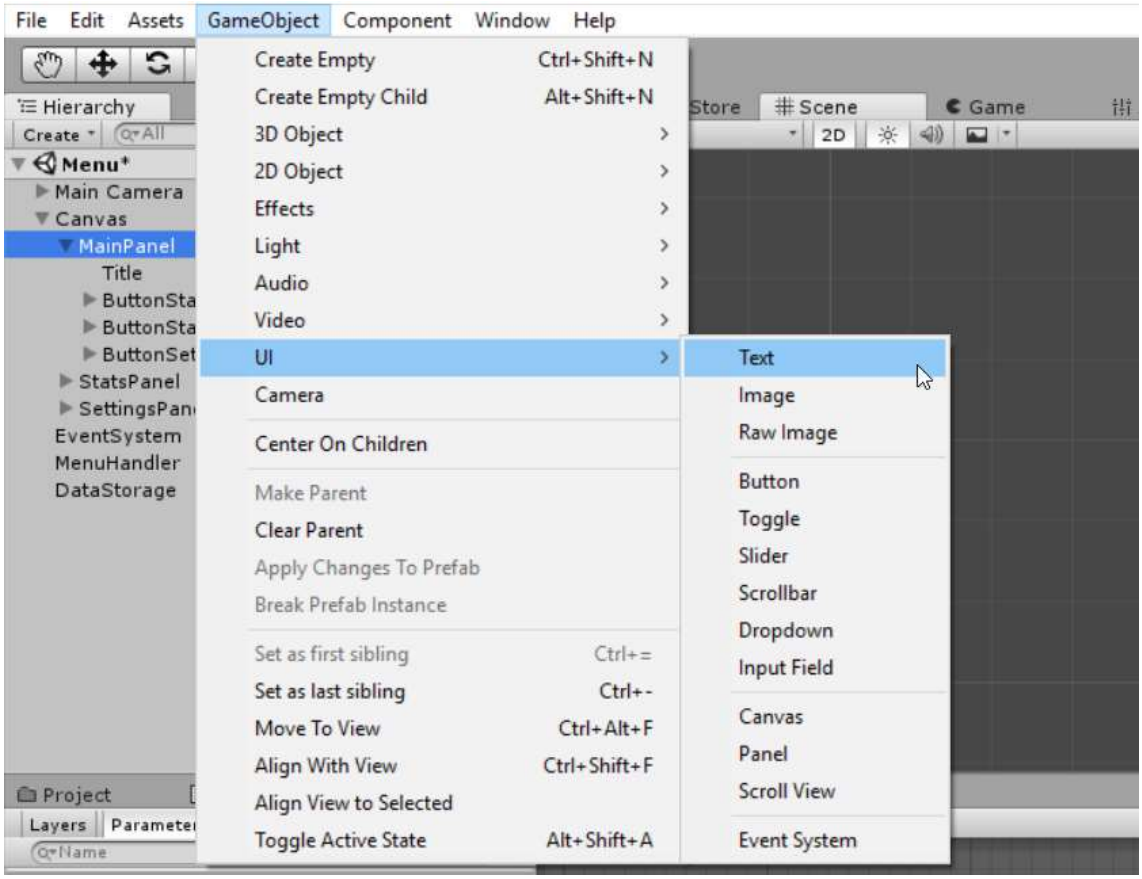
mukaan (Scale With Screen Size). Tämä tarkoittaa, että niiden sijainti ja skaalaus pysyy samana eri kokoisilla näytöillä.

Kuvassa 12 nähdään canvas ja sen komponentit Inspector-ikkunassa, sekä canvaksen sisälle rakennettu käyttöliittymäelementtien hierarkia Hierarchy-ikkunassa. Elementit piirretään siinä järjestyksessä, kuin ne ovat hierarkiassa: ylimmäinen elementti piirretään ensimmäiseksi ja alimmainen viimeiseksi.

KUVA 12. Canvas, sen komponentit, ja käyttöliittymäelementtien hierarkia.

4.3.2 Käyttöliittymäelementit

Unity tarjoaa valmiita käyttöliittymäelementtejä käyttöliittymien tekemiseen. Ne ovat canvaksen sisälle asetettavia peliobjekteja, joissa on niiden toiminnan määrittäviä komponentteja. Tällaisia elementtejä ovat esimerkiksi tekstit, napit ja kuvat. Kuvassa 13 on lista Unityn valmiista elementeistä sekä esimerkki tekstiobjektin lisäämisestä canvakseen.



KUVA 13. Unityn käyttöliittymäelementit ja tekstin lisääminen päävalikkoon

Kuvassa 14 nähdään start game -nappi valittuna scenessä. Kuvan oikeassa laidassa on Inspector-ikkunassa napin toiminnan määrittävät komponentit. Inspector-ikkunan alalaidassa on määriteltynä, mitä nappi tekee, kun sitä painetaan (On Click). Siihen on asetettu suoritettavaksi MenuHandler-luokan funktio StartGame. StartGame-funktion toiminta on nähtävissä kohdan 4.3.3 koodiesimerkissä.

KUVA 14. Start game -nappi scenessä, sekä sen komponentit ja toiminnan määrittävä funktio valittuna.

4.3.3 Latausnäyttö ja scenen vaihto

Latausnäyttö (engl. loading screen) on videopeleissä yleisesti käytettävä asia. Se on yleensä kuva, joka näyttää käyttäjälle, että ohjelma lataa tarvittavia tiedostoja. Latausnäytössä on monesti myös latauspalkki, josta käyttäjä saa arvion latauksen kestosta.

Unityssä latausnäyttöä on hyvä käyttää scenen vaihdon yhteydessä. Sceneä vaihdettaessa, kaikki uuden scenen tiedot ladataan muistiin, ja isoissa sceneissä tämä voi viedä pitkänkin ajan. Vaihto on hyvä tehdä coroutine-funktiolla, joka lataa uuden scenen asynkronisesti (tarkempaa tietoa coroutinesta kohdassa 2.2.7.) Scenen vaihtaminen ja latausnäytön aktivoiminen latauksen ajaksi voidaan tehdä seuraavasti:

```
void StartGame()
{
      loadingScreen.SetActive(true);
      StartCoroutine(LoadNewScene("Game"));
}

IEnumerator LoadNewScene(string sceneName)
{
      AsyncOperation async =
            SceneManager.LoadSceneAsync(sceneName);
```

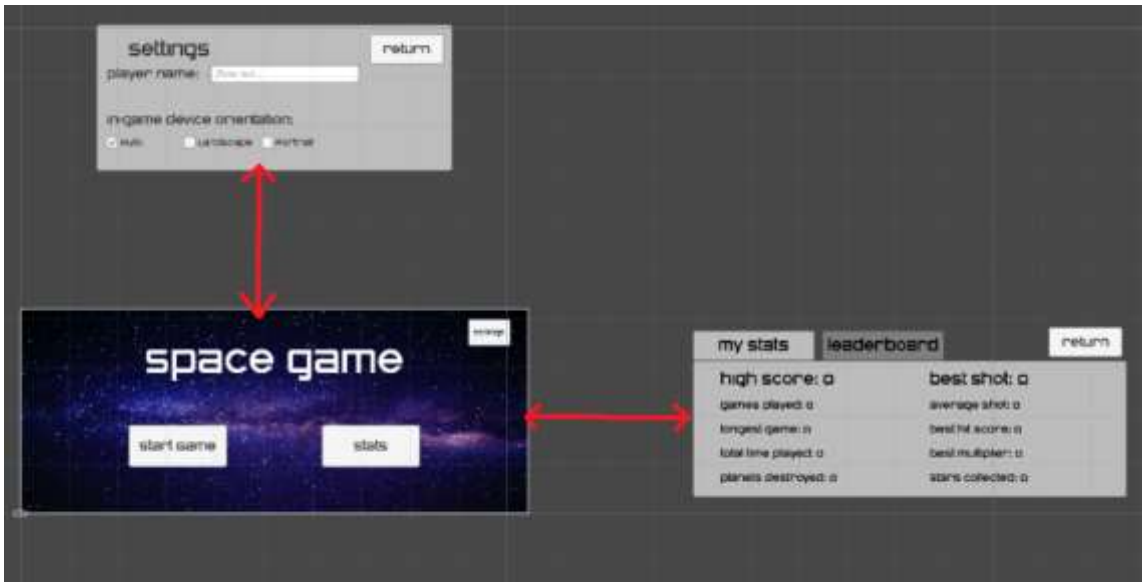
```
        while (!async.isDone)
        {
              yield return null;
        }
    }
```

Tässä esimerkissä loadingScreen-peliobjekti sisältää koko näytön täyttävän kuvan, joka asetetaan vaihdon alussa aktiiviseksi. Tätä peliobjektia ei tarvitse asettaa epäaktiiviseksi, koska uudessa scenessä sitä enää ole, ja vanhan scenen peliobjektit poistetaan automaattisesti vaihdon jälkeen.

4.3.4 Animaatiot

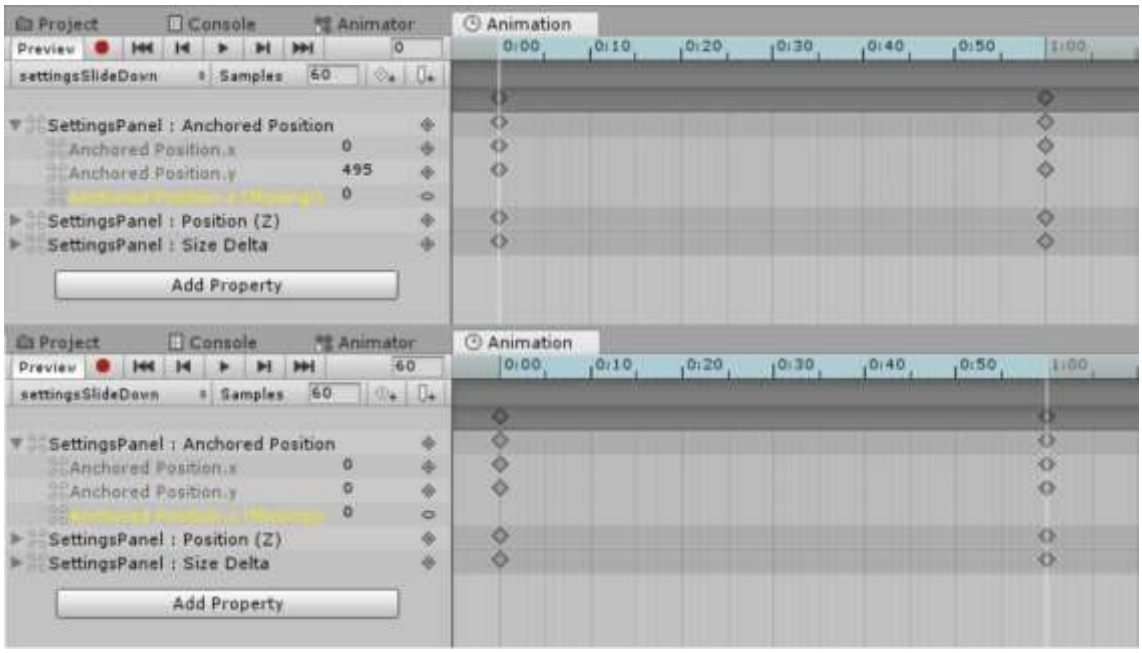
Käyttöliittymäanimaatiot voivat tehdä selvän eron hyvän ja huonon näköisen sovelluksen välillä. Käyttäjät eivät yleensä kiinnitä niihin paljon huomiota, mutta oikein toteutettuina, ne voivat tarjota hyödyllistä informaatiota ja saada aikaa miellyttävän käyttökokemuksen. (12.)

Pelin käyttöliittymän animaatiot on tehty Unityn omilla animaatiotyökaluilla. Valikoissa näkymien välillä liikkumisessa on yksinkertaiset liukumisanimaatiot. Asetukset-ikkunaan mentäessä päävalikko liukuu alas ja asetukset ylhäältä sen tilalle. Tulostaulukoihin mentäessä päävalikko liukuu vasemmalle ja tulostaulukot oikealta sen tilalle. Takaisin päävalikkoon palatessa liukumiset tapahtuvat vastaavasti eri suuntiin. Nämä liikkeet on havainnollistettu kuvassa 15.



KUVA 15. Näkymien välillä siirryttäessä valikot vaihtavat paikkaa liukumalla

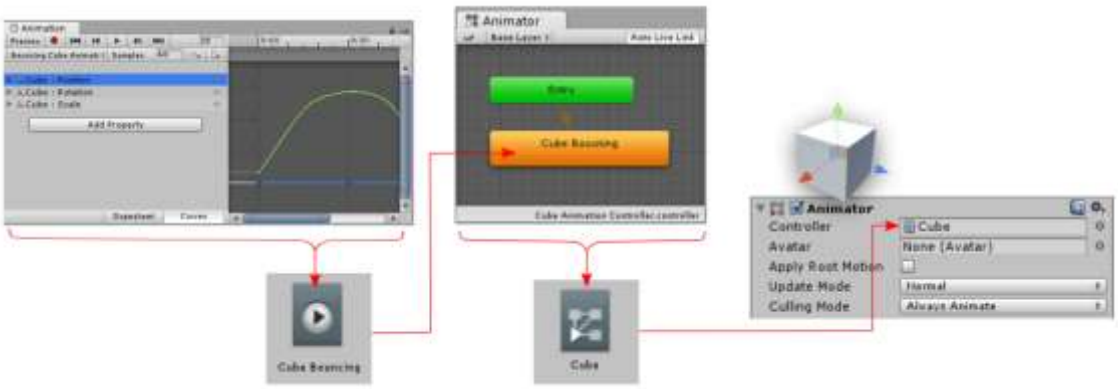
Jokaiselle animaatiolle tehdään oma Animation Clip -tiedosto. Näitä tiedostoja käsitellään Animation-ikkunassa. Kuvassa 16 nähdään asetukset-ikkunan alaspäin liu'uttamiseen käytettävä animaatio. Kuvan ylemmässä osassa on animaation alkupää, jossa ikkunan sijainti y-akselilla on 495. Alemmassa osassa nähdään animaation loppupää, jossa on sijoitettu uusi sijainti halutun ajan päähän. Tämä uusi sijainti on 0, joten animoitu ikkuna liukuu animaation aikana 495 yksikköä alaspäin.



KUVA 16. Unityn animaatiotyökalu, jossa aktiivisena settingsSlideDown-animaatio.

Peliobjektit tarvitsevat Animator-komponentin animaatioiden suorittamiseen. Animator-komponenttiin lisätään Animator Controller -tiedosto, johon lisätään halutut Animation Clipit (kuva 17). Animator Controllerilla määritellään, milloin haluttu Animation Clip suoritetaan. Tämän jälkeen niitä voidaan käsitellä skriptien kautta. Esimerkiksi asetuksien avaaminen on suoritettu seuraavasti:

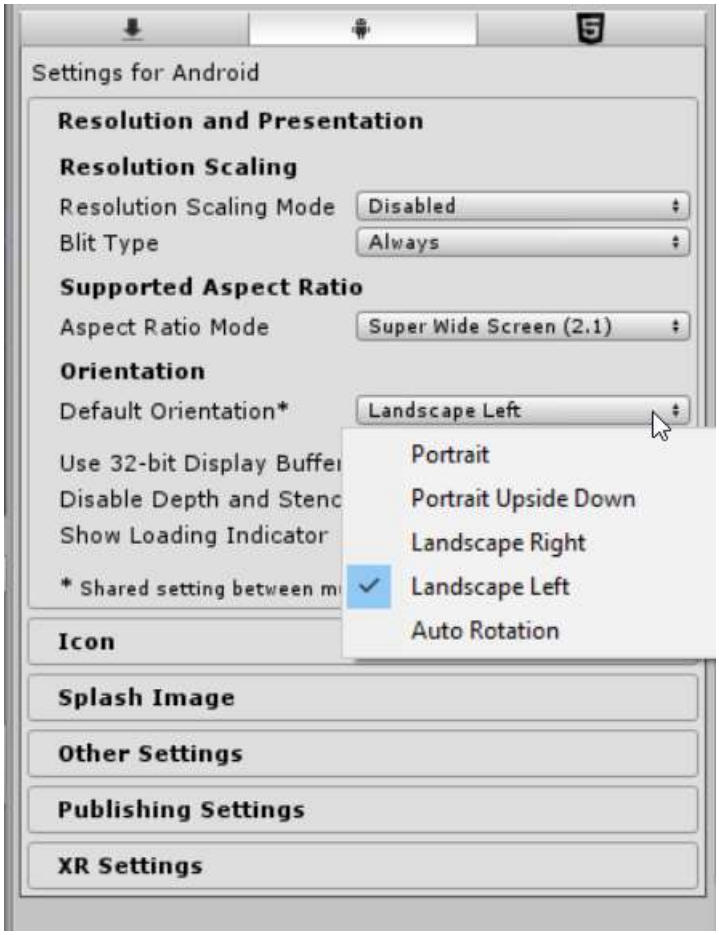
```
void OpenSettings()
{
      Animator mainAnim =
            mainPanel.GetComponent<Animator>();

      Animator settingsAnim =
                  settingsPanel.GetComponent<Animator>();

      settingsAnim.SetTrigger("slideDown");
      mainAnim.SetTrigger("slideDown");
}
```



KUVA 17. Vasemmalta oikealle: Animation Clip, Animator Controller ja Animator-komponentti. (3, linkit Animation -> Animation Clips -> Animation Window Guide -> Creating a New Animation Clip.)

4.3.5 Mobiililaitteen kierto

Unityn asetuksissa on mahdollista asettaa laitteen kierto lukituksi haluttuun asentoon (kuva 18). Space Gamessa valikoiden haluttu kierto on aina vaakatasossa, joten se on asetettu lukituksi vaaka-asentoon (Landscape Left). Käyttöasennon suunnittelusta lisää kohdassa 4.2.2.



KUVA 18. Asetukset Android-laitteelle, ja laitteen kierron valinta.

Pelin aikana laitetta on mahdollista pitää joko pysty- tai vaaka-asennossa. Unityn asetuksissa asento on aina lukittuna vaaka-asentoon, jonka ansiosta laitteen kääntäminen ei vaikuta peliobjekteihin ja häiritse peliä. Ainoastaan halutut käyttöliittymäelementit vaihtavat käännettäessä paikkaa ja kiertoa. Tämä saadaan aikaiseksi esimerkiksi seuraavalla skriptillä, joka sisältää laitteen asennon tarkistamisen sekä käyttöliittymäelementtien sijainnin ja kierron muuttamisen sen mukaan:

```
if(rotation == ”auto”)
{
      if (Input.deviceOrientation ==
            DeviceOrientation.Portrait)
      {
            //move score to another location
            scoreText.transform.position =
                  PortraitScore.position;
```

```
            //rotate score
            scoreText.transform.rotation =
                PortraitScore.rotation;
        }
        else
        {
            scoreText.transform.position =
                LandscapeScore.position;

            scoreText.transform.rotation =
                LandscapeScore.rotation;
        }
    }
```

Pelinaikaisen kierron asento on myös mahdollista lukita. Tämä on tehty valittavaksi Space Gamen asetuksiin. Kuvassa 19 nähdään asetukset-ikkuna, jossa kääntö on asetettu automaattiseksi. Myös yläpuolelle esitetyn koodiesimerkin tapauksessa tämä valinta on asetettu automaattiseksi.

KUVA 19. Asetukset-ikkuna, jossa valittavana laitteen kierron lukitus (in-game device orientation).

5 JATKOKEHITYSMAHDOLLISUUDET

Space Game on jo pitkälle kehitetty, mutta jotkin ominaisuudet vaatisivat vielä hiomista. Pienellä jatkokehityksellä pelistä saataisiin huomattavasti käyttäjäystävällisempi ja koukuttavampi. Viimeistelty peli voisi olla sopiva jopa julkiseen jakeluun.

5.1 Pelin nimi

Space Game, eli pelin tämänhetkinen nimi, on vain sen projektinimi. Nimi ei ole erityisen mieleenpainuva tai persoonallinen, joten se tulisi muuttaa persoonallisemmaksi.

5.2 Grafiikat

Kaikki pelin grafiikat on tehty itse ja ne tulisi tehdä uusiksi. Ulkopuolisen graafikon apu voisi visuaalisesti parantaa peliä huomattavasti. Myös käyttöliittymässä käytettyjen Unityn oletuskäyttöliittymäelementtien tilalle voisi luoda jotain visuaalisesti näyttävämpää.

5.3 Äänet

Videopelit ilman ääniä ovat huomattavasti vähemmän vakuuttavia. Tilanteeseen sopivat äänet lisäävät pelin uskottavuutta ja voivat tarjota tärkeää informaatiota. Musiikki pelissä voi luoda tiettyyn tilanteeseen tai toimintaan sopivan tunnelman. Musiikki voi myös tehdä pelistä mieleenpainuvan. (13.)

Space Gamen tämänhetkisessä toteutuksessa ei ole minkäänlaisia ääniä. Äänien lisäämien peliin on erittäin tärkeää pelin jatkokehityksen kannalta.

5.4 Tulostaulukon kehittäminen

Tämänhetkisissä tulostaulukoissa on näkyvillä vain viisi parasta tulosta. Tämä on hyvin rajoitettu määrä, ja peli voisi olla käyttäjien kannalta mielenkiintoisempi, mikäli suurempi tulosten määrä helpottaisi oman tuloksen saamista tulostaulukkoon. Tämänhetkisessä toteutuksessa käytössä oleva tulostaulukkopalvelu Dreamlo mahdollistaa 1 000 parhaan tuloksen käyttämisen, joten nykyisen tulostaulukon kehittäminen olisi mahdollista ja helppoa.

5.5 Sisäänkirjautuminen

Android-laitteilla käytettäviin peleihin on mahdollista integroida Googlen tarjoama Play Game Services -palvelu. Palvelun käyttäminen tarkoittaa sitä, että pelaajat voivat kirjautua sisään peliin omalla Google-tunnuksellaan. (14.)

Nykyisessä toteutuksessa ei ole käytössä minkäänlaista sisäänkirjautumista. Pelissä on ainoastaan asetuksiin lisättävä käyttäjänimi pelaajalle, joka tallennetaan käytettävään laitteeseen. Tämän vuoksi, saman käyttäjänimen omaavia pelaajia voi olla monta. Nykyisessä tulostaulukossa voi esiintyä yksi käyttäjänimi vain kerran, joten samalla käyttäjänimellä pelaavat pelaajat voivat vaikuttaa toistensa huipputuloksiin.

Jos peliin lisätään sisäänkirjautuminen Googlen kautta, jokaisen pelaajan käyttäjänimeksi tulee heidän Google-tunnuksensa. Jo pelkästään tämä parantaisi nykyisen tulostaulukon toiminnallisuutta. Google kuitenkin tarjoaa myös oman tulostaulukkonsa, jonka vuoksi Dreamloa ei enää tarvittaisi.

Pelattaessa Google-tunnuksella, myös muut pelaajan omat tiedot voitaisiin tallentaa käytettävän laitteen sijasta Googlen pilvipalveluun. Tämä mahdollistaisi pelaajan käyttäjätietojen käyttämisen monessa eri mobiililaitteessa. (14.)

5.6 Google Play -kauppa

Suurimpana tavoitteena voisi pitää pelin saamista julkiseen jakeluun. Android-sovelluksilla paras kanava tähän on Googlen Play-kauppa. Googlen vaatimuksiin ja suosituksiin Play-kaupassa julkaistavien sovelluksien ominaisuuksien kannalta ei opinnäytetyön yhteydessä perehdytty.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda mobiilipeli Android-alustalle ja perehtyä sen käyttöliittymään. Aihe oli mielenkiintoinen ja mielenkiinto pysyi yllä koko tekemisen ajan. Käytetyt työkalut olivat jo entuudestaan tuttuja, minkä vuoksi pelin kehitys sujui ilman suurempia ongelmia.

Vaikka Unity olikin jo ennestään tuttu pelinkehitysympäristö, sen osaaminen kehittyi huomattavasti. Yksi suuri syy tälle oli Unityn dokumentaation aktiivinen selaaminen, mikä lisäsi eri asioiden toiminnan ymmärtämistä. Ainoat täysin uudet asiat olivat animaatioiden tekeminen ja datan tallentaminen PlayerPrefs-luokalla. Näiden asioiden osaaminen on tulevien peliprojektien kannalta erittäin hyödyllistä.

Käyttöliittymien teorian tutkiminen auttaa jatkossa tunnistamaan käyttöliittymien tärkeitä ominaisuuksia. Yleensä käyttöliittymien suunnitteleminen ei ole yrityksessä insinöörien tehtävä, mutta omissa projekteissa suunnittelutaidoista on hyötyä.

Opinnäytetyön tavoitteet saavutettiin hyvin. Ainoa tavoite, jota ei vielä saavutettu, oli pelin saaminen Google Play -kauppaan yleiseen jakeluun. Peli vaatisi vielä loppuviimeistelyn sitä varten.

LÄHTEET

1. Company Facts. Unity Technologies. Saatavissa: https://unity3d.com/public-relations. Hakupäivä 17.4.2018.

2. Fine, Richard 2017. UnityScript's long ride off into the sunset. Unity Technologies. Saatavissa: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/. Hakupäivä 17.4.2018.

3. Unity Manual. 2018. Unity Technologies. Saatavissa: https://docs.unity3d.com/Manual/index.html. Hakupäivä 20.4.2018.

4. Haines, Eric 2014. ArrayPrefs2. Unify Community Wiki. Saatavissa: http://wiki.unity3d.com/index.php/ArrayPrefs2. Hakupäivä 19.4.2018.

5. Guida, Carmine. Dreamlo. Saatavissa: http://dreamlo.com/. Hakupäivä 19.4.2018.

6. User Interface. Every Interaction. Saatavissa: https://www.everyinteraction.com/definition/user-interface/. Hakupäivä 1.5.2018.

7. Sorensen, Alec 2016. What is the definition of a game interface? Quora. Saatavissa: https://www.quora.com/What-is-the-definition-of-a-game-interface. Hakupäivä 28.5.2018.

8. Fadeyev, Dmitry. 8 Characteristics Of Successful User Interface. Usability Post. Saatavissa: http://usabilitypost.com/2009/04/15/8-characteristics-of-successful-user-interfaces/. Hakupäivä 2.5.2018.

9. 5 aspects of a good user interface. 2014. Argon Design. Saatavissa: http://www.argondesign.com/news/2014/feb/5/5-aspects-good-user-interface/. Hakupäivä 2.5.2018.

10. Vukovic, Peter. 7 unbreakable laws of user interface design. 99design. Saatavissa: https://99designs.com/blog/tips/7-unbreakable-laws-of-user-interface-design/. Hakupäivä 2.5.2018.

11. Derk. The 8-step Guide To Interface Design for iPhone Games. Paladin Studios. Saatavissa: http://www.paladinstudios.com/2012/04/23/the-8-step-guide-to-interface-design-for-iphone-games/. Hakupäivä 4.5.2018.

12. Naji, Cassandra 2018. Prototyping mobile UI animations: Examples & free downloads. Justinmind. Saatavissa: https://www.justinmind.com/blog/prototyping-mobile-ui-animations-5-inspiring-examples/. Hakupäivä 8.5.2018.

13. The Importance of Sound. 2017. The Los Angeles Film School. Saatavissa: https://www.lafilm.edu/blog/the-importance-of-sound/. Hakupäivä 14.5.2018.

14. Google Play Game Services. Google. Saatavissa: https://developers.google.com/games/services/. Hakupäivä 14.5.2018.