OULUN AMMATTIKORKEAKOULU

Vesa Väisänen

VERTAILU KAKSIULOTTEISEN PELIN KEHITTÄMISESTÄ UNITY- JA UNREAL ENGINE -PELIMOOTTOREILLA

VERTAILU KAKSIULOTTEISEN PELIN KEHITTÄMISESTÄ UNITY- JA UNREAL ENGINE -PELIMOOTTOREILLA

Vesa Väisänen
Opinnäytetyö
Kevät 2016
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Vesa Väisänen
Opinnäytetyön nimi: Vertailu kaksiulotteisen pelin kehitämisestä Unity- ja Unreal Engine -pelimoottoreilla
Työn ohjaaja: Veikko Tapaninen
Työn valmistumislukukausi ja -vuosi: Kevät 2016 Sivumäärä: 48

Työn tavoiteena oli tutkia Unity- ja Unreal Engine -pelimoottoreiden eroavaisuuksia kaksiulotteisen pelin kehityksessä. Työn aihe on itse keksitty, toimeksiantajaa opinnäytetyössä ei ole. Opinnäytetyössä tehtiin eri pelimoottoreilla kaksi identtistä peliä PC-alustalle ja vertailtiin pelimoottoreiden ominaisuuksia. Pelimoottoreiden vertailusta on hyötyä esimerkiksi aloittelevalle peliohjelmoijalle, joka haluaa tietää minkä pelimoottorin opetteluun kannattaa keskittyä kaksiulotteisessa pelinkehityksessä.

Pelimoottorin vertailussa keskitytään pelimoottorin käyttöön, ei niinkään teknisiin ominaisuuksiin. Vertailtavia ominaisuuksia ovat yhteensopivuus eri käyttöjärjestelmien kanssa, dokumentointi, käyttöliittymän käytettävyys, pelimoottorin lisenssin hinta, ohjelmointiominaisuudet, spritejen eli kuvien käsittely, törmäysten käsittely, animointi ja yhteensopivuus Subversion-versionhallintasovelluksen kanssa.

Tekijällä on taustallaan opiskeluaikana käydyt Unity-pelinkehityskurssit ja vuoden työkokemus Unity-pelinkehityksestä, mutta Unreal Enginen käytöstä ei ole kokemuksia. Unity-pelin toiminnot tehtiin C#-skripteillä ja Unreal Enginen toiminnot tehtiin Blueprint-kaavioita käyttäen.

Kummatkin pelimoottorit soveltuvat hyvin kaksiulotteisen pelin kehitykseen ja molempien pelimoottoreiden kehittäjät tukevat kaksiulotteista pelinkehitystä. Pelien valmistuttua tutkittiin pelimoottoreiden tärkeimpiä käytettävyyteen liittyviä ominaisuuksia kyseisten pelien tekemisessä, lisäksi tutkittiin muiden näkemyksiä pelimoottoreiden eroista. Pelejä ei ole julkaistu missään, sillä niiden kehitys jäi demo-tasolle.

Avainsanat: Unity, Unreal Engine, 2D, pelinkehitys, pelimoottori

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Vesa Väisänen
Title of thesis: Comparison of 2D game development using Unity- and Unreal Engine - game engines
Supervisor(s): Lecturer Veikko Tapaninen
Term and year when the thesis was submitted: Spring 2016 Number of pages: 48

Goal of this bachelor thesis is to study the differences between Unity and Unreal Engine in 2D game development. Two identical games were PC developed for this purpose with different game engines and then the game engines' features were compared. Comparing of game engines is helpful for beginner game programmers who want to know which game engine they should be learning when focusing in 2D game development. Subject of the thesis was self created and it was not commissioned by anyone.

The comparison of game engines focused on the use of the game engine, not on technical features. Compared features included compatibility with different operation systems, documentation, usability of the user interface, license price of the game engine, programming features, sprite handling, collision handling, animation, and compatibility with Subversion source control software.

Author has background studies in Unity game development and one year work experience in Unity game development but no previous studies or experience in Unreal Engine game development. Features of the Unity game were made with C# scripts and features of the Unreal Engine game were made with Blueprint-graphs.

Both game engines were suitable for 2D game development and developers of both game engines support 2D game development. After finishing the games their most important usability features of the game engines were studied in the development of the games. In addition studies were done on other people concerning their views of the differences between the game engines. Finished games were not released anywhere since their developed ended in demo-level.

Keywords: Unity, Unreal Engine, 2D, game development, game engine

SISÄLLYS

1 JOHDANTO

Työssä vertaillaan kahden eri pelimoottorin ominaisuuksia pelinkehityksessä. Pelimoottori on pelinkehityksessä käytettävä ohjelmistokokonaisuus, joka kääntää kehitetyn pelin ohjelmaksi hoitaen esimerkiksi grafiikan piirtämisen, syötteiden lukemisen, hahmojen liikuttamisen ja fysiikan mallinnuksen (Ohjelmointiputka. 2007). Opinnäytetyön aihe on itse keksitty, opinnäytetyöllä ei ole toimeksiantajaa. Pelimoottoreiden vertailusta on hyötyä esimerkiksi aloittelevalle peliohjelmoijalle, joka haluaa tietää, kumman pelimoottorin käytön opetteluun kannattaa ryhtyä, jos haluaa keskittyä kaksiulotteisen pelin kehitykseen. Peli tulee olemaan ainoastaan PC-alustalle tehty kokeilu. Sitä ei tulla julkaisemaan pelejä myyvässä kaupassa eikä muulla internet-sivulla.

Kaksiulotteisella pelillä tarkoitetaan platformer-tyyppistä peliä, jossa grafiikat on tehty pikseleillä tai kuvilla (spriteillä), jolloin grafiikat koostuvat yhdestä polygonista. Kolmiulotteisten pelien grafiikat koostuvat useista polygoneista. Koska kolmiulotteisen pelin grafiikat koostuvat useasta polygonista, kolmiulotteisen grafiikan suorittaminen vaatii tehokkaamman järjestelmän. (Quora.)

Pelit ovat olleet kaksiulotteisia videopelien kehityksen alusta alkaen 1950-1970-luvulla. Kolmiulotteisia pelejä alettiin kehittämään 1990-luvun puolivälissä, kun julkaistiin kolmiulotteista grafiikkaa suorittavat pelikonsolit kuten Sega Saturn, Sony Playstation ja Nintendo 64. PC-tietokonealustalle kolmiulotteiset grafiikat tulivat vasta 1990-luvun lopulla, koska PC-tietokoneista puuttuivat tarvittavat laitteistot kolmiulotteisen grafiikan suorittamiseen. Kaksiulotteisten pelien kehitys jatkui vielä 2000-luvulla Nintendo Gameboy -taskupelikonsoleilla. (Overmars 2012.) Kaksiulotteista grafiikkaa kuitenkin käytetään edelleen esimerkiksi mobiili- ja tietokonepeleissä.

Opinnäytetyössä tehdään kaksi identtistä peliä Unity-pelimoottorilla ja Unreal Engine -pelimoottorilla. Peli on saanut vaikutteita esimerkiksi Mega Man -pelisarjan peleistä. Mega Man -sarjan ensimmäinen peli julkaistiin Nintendo Entertainment Systemille vuonna 1987, pelit ovat tyyliltään sivusta päin kuvattuja toimintapelejä (IGN.).

Pelin alussa on alkuvalikko, jonka näppäimiä painamalla voidaan aloittaa peli tai sulkea peli. Alkuvalikon jälkeen itse pelikenttä alkaa, pelissä liikutellaan pelihahmoa, ammutaan vastaan

tulevia vihollisia ja lopuksi päästään maaliin, johon pelaajan osuttua pelissä palataan takaisin alkuvalikkoon. Pelialueessa on myös vaarallinen alusta, johon osumalla kenttä alkaa alusta.

2 UNITY-PELIMOOTTORIN KOKONAISUUS

2.1 Historia

Pelimoottorin kehityksen aloittivat tanskalaiset David Helgason, Joachim Ante ja Nicholas Francis, kun heidän ensimmäisestä pelistään GooBallista ei tullut kaupallista menestystä. Unityä kehittäväksi yhtiöksi perustettiin vuonna 2004 Unity Technologies. Unity julkaistiin vuonna 2005 ja pelimoottorissa tuettiin ainoastaan Applen OS X -käyttöjärjestelmää. Myöhemmin pelimoottorissa alettiin tukemaan myös internet-selaimia, mobiililaitteita, Microsoftin Windows-käyttöjärjestelmää, Nintendo Wiitä ja muita pelikonsoleja. (Seraphina 2013.)

Vuonna 2009 Unitystä julkaistiin ensimmäinen ilmainen kehitysversio (Unity Technologies. 2009). Vuonna 2010 pelimoottori sai Android-käyttöjärjestelmän tuen. 2013 Unityssä alettiin tukemaan Windows Phone -puhelimia sekä BlackBerry-käyttöjärjestelmää. Kaksiulotteisten pelien virallinen tuki lisättiin myös pelimoottoriin. (Seraphina 2013.)

2.2 Ominaisuudet

Unity on laaja-alainen pelimoottori, jolla voidaan kehittää peli alusta loppuun. Unityssä grafiikan rajapintoina käytetään alustasta riippuen Direct3D:tä (Windows), OpenGL:ää (Windows, Macintosh, Linux) tai OpenGL ES:ää (Android, iOS). Fysiikan mallinnuksiin Unityssä käytetään Nvidia PhysX:ää. (Seraphina 2013.)

Unityn graafisena editorina käytetään Unity Editoria, jolla voidaan tarkastella pelikehitystä useista eri näkymistä, joista yleisimmät ovat pelin kehitysselain (Scene), komponenttien tutkija (Inspector), peli (Game), animaatiot (Animator, Animation) ja projektin gameobjektit listaava hierarkia (Hierarchy).

Unityssä pelit muodostetaan Scene-kentistä, joihin on sijoiteltu Gameobject-luokkia. Gameobjectiin sisällytetään toimintaa muokkaavia komponentteja: komponentteja ovat esimerkiksi Gameobjetin paikan ja koon määrittävä Transform, fysiikkakomponettti Rigidbody, animaatiota suorittava Animator sekä skriptit. Gameobjektit voivat olla myös Prefab-tiedostoja, joita voidaan liittää esimerkiksi skriptin public-tyyppisiin olioihin ilman, että ne sijaitsevat Scenessä.

Ohjelmoinnissa yleisin käytetty ohjelmointiympäristö on MonoDevelop, mutta ohjelmointia voidaan suorittaa myös Microsoftin kehittämällä Visual Studiolla ja Notepad++:lla. MonoDevelop asentuu tietokoneelle Unityn asennuksen yhteydessä ja Unity itse tukee MonoDevelopin kehitystä. Ohjelmointikielinä käytetään C#:a ja JavaScriptiä.

3 UNITY-PELIN KEHITYS

3.1 Projektin luonti

Projekti aloitetaan luomalla ja painamalla menu-valikossa New Project...-painiketta. Tämän jälkeen projektille annetaan nimi ja määritetään sijainti, minne projekti tallennetaan. Tässä valikossa määritetään, onko peli kaksiulotteinen (2D) vai kolmiulotteinen (3D) sekä käytetäänkö projektissa valmiita Asset-paketteja. Koska pelistä tehdään kaksiulotteinen, valitaan 2D. Mitään Asset-paketteja ei käytetä projektissa. Lopuksi painetaan ”Create Project”-painiketta.

Tämän jälkeen avautunut Scene tallennetaan nimellä ”Menu” ja Scene lisätään buildauslistaan painamalla menu-valikosta File ja valitaan Build Settings. Build Settings -ikkunassa painetaan Add Open Scenes -näppäintä. Pelissä tarvitaan toista Game-nimistä Sceneä: tämä Scene luodaan painamalla menu-valikosta New Scene, tämän jälkeen se tallennetaan ”Game”-nimellä ja se lisätään Build Menuun.

3.2 Valikot

Projektin luonnin jälkeen tehdään valikot. Valikkojen pohjarunko tehdään lisäämällä projektiin Canvas-komponentin sisältävä gameobject. Canvas-gameobject luodaan painamalla menu-valikon GameObject-näppäintä ja valitaan UI-sarakkeessa sijaitseva Canvas.

Valikossa tulee olemaan kaksi näppäintä. Toisesta näppäimestä peli alkaa ja toisesta näppäimestä peli suljetaan. Näppäin luodaan painamalla hiiren oikealla näppäimellä Hierarchy-näkymässä sijaitsevaa Canvas-gameobjectia ja valitaan UI-sarakkeessa sijaitseva Button. Näppäimiä tehdään kaksi kappaletta. Unity luo näppäimet päällekkäin, joten ne on suositeltavaa siirtää erilleen toisistaan. Näppäimiä voi siirrellä valitsemalla ne Scene-ikkunassa ja liikuttelemalla näppäimiä tai muuttamalla Inspector-ikkunassa sijaitsevan Rect Transform -komponentissa olevia Pos X- ja Pos Y -arvoja. Näppäimet voidaan nimetä painamalla Hierarchy-näkymässä sijaitsevaa Button-gameobjectin pudotusvalikkoa, jolloin Button-gameobjectista paljastuu lapsigameobject ”Text”. Textissä on Text-komponentti, jonka Text-alueeseen kirjoitetaan haluttu teksti, aloitusnäppäimeen kirjoitetaan ”Start Game” ja lopetusnäppäimeen ”Quit”. Tämän jälkeen projektin Game-valikossa valikon pitäisi näyttää kuvan 1 mukaiselta.

KUVA 1. Valikon näppäimet

3.3 Valikon näppäimien toiminta

Näppäimien toiminnassa käytetään skriptiä, joka sijaitsee scenessä sijaitsevassa gameobjectissa. Sceneen luodaan gameobject nimeltä Scenemanager ja tälle luodaan C#-skripti valitsemalla Hierarchyn Scenemanager-gameobject ja painamalla Inspectorissa sijaitsevaa Add Component -näppäintä. Valitaan New Script ja tehdään siitä C#-scripti nimeltä SceneChange. Tämän jälkeen avataan MonoDevelop-ohjelmointiympäristö kaksiosklikkaamalla SceneChange-skriptiä.

MonoDevelopissa scriptistä voidaan poistaa Start()- ja Update()-funktiot, sillä niitä ei tässä scriptissä tarvita. Scriptiin tehdään kaksi funktiota (lähdekoodi 1).

```
public void ChangeScene () {
    Application.LoadLevel(0);
}

public void Quit(){
    Application.Quit ();
}
```

LÄHDEKOODI 1. Valikon näppäimien toiminta

ChangeScene()-funktiossa aloitetaan peli avaamalla Build Settings -valikossa sijaitseva Scene-järjestysnumero. Menu-Scenen järjestysnumero on 0 ja Game-Scenen 1. Quit()-funktiossa suljetaan ohjelma.

Jotta näppäimet toimisivat, niille tulee laittaa yhteys SceneChange-skriptissä oleviin funktioihin. Kun Start Game-näppäin valitaan Hierarchyssä, Inspectorissa on On Click() -osio, johon voidaan lisätä näppäintä klikatessa tapahtumia toimintoja. Toiminto lisätään painamalla osion +-näppäintä ja osioon ilmestyy tyhjä toiminto. Painetaan valintamenua, jossa lukee ”None (Object)” ja valitaan Scenessä sijaitseva Scenemanager-gameobject. Nyt painetaan No Function -valintamenua ja valitaan SceneChange-scriptissä sijaitseva ChangeScene-funktio. Quit-näppäimen toiminta tehdään samalla tavalla kuin Start Game -näppäimessä, mutta No Function -valintamenussa valitaan SceneChange-skriptin Quit()-funktio. Menu-Scene on tämän jälkeen valmis ja voidaan ryhtyä tekemään Game-Sceneä.

3.4 Pelialueen luonti

Game-sceneen luodaan alustoja, joiden päällä pelin hahmot liikkuvat. Ne ovat yksinkertaisia harmaita suorakaiteita, joilla on törmäyksen mahdollistava collider-komponentti. Luodaan gameobject, jolle lisätään Sprite Renderer- ja Box Collider 2D -komponentit. Sprite Rendereriin lisätään Paint-ohjelmalla tehty harmaa sprite, joka tekee alustasta näkyvän. Tämän jälkeen voidaan muokata alustan kokoa Transform-komponentin Scale-arvoja. Box Colliderista tehdään saman kokoinen kuin alusta muokkaamalla Size-arvoja.

Pelissä on punainen pohja, johon osumalla pelaaja joutuu aloittamaan pelin alusta. Pohja tehdään samalla tavalla kuin harmaa alusta, mutta väri tulee olemaan punainen; se on koko kentän levyinen ja siihen lisätään skripti. Pohjaan luodaan Death-skripti ja skriptistä poistetaan Start()- ja Update()-funktiot. Skriptiin lisätään yksi funktio lähdekoodi 2:een.

```
void OnCollisionEnter2D(Collision2D col)
  {
    if (col.gameObject.transform.name == "Player") {
      Application.LoadLevel(Application.loadedLevelName);
    }
  }
```

LÄHDEKOODI 2. Pelaajan törmäys

Funktio on nimettävä OnCollisionEnter2D-nimiseksi jotta Unity ymmärtäisi sen olevan törmäystä tutkiva funktio, ja siinä on lopussa ”2D”, jotta funktio toimisi kaksiulotteisessa pelissä, kolmiulotteisessa pelissä sitä ei ole. Funktioon tulee Collision2D-arvo col. Funktiossa on if-lause, jossa tutkitaan, onko pohjaan törmännyt Player-niminen gameobject. Jos Player on törmännyt niin peli avaa parhaillaan päällä olevan scenen uudestaan eli peli alkaa alusta.

3.5 Hahmon luonti ja fysiikat

Peliin luodaan Player-niminen gameobject, jolle lisätään Sprite Renderer-, Rigidbody 2D- ja Box Collider 2D -komponentit. Pelihahmon Sprite Rendererissä käytetään pelihahmon monia spritejä, joita käytetään animoinnissa, pelihahmon Sprite Rendereriin laitetaan ensin sprite, jossa se on paikallaan. Rigidbody 2D-komponentti antaa pelihahmolle fysiikat eli pelihahmo putoaa alaspäin painovoiman lakien mukaisesti. Box Collider 2D taas mahdollistaa törmäyksen alustojen ja pohjien kanssa. Jos pelihahmo putoaa ja törmää pelissä oleviin harmaisiin alustoihin, se jää paikalleen. Mikäli pelihahmo törmää punaiseen pohjaan, peli alkaa alusta. Rigidbody 2D -komponentissa kannattaa valita Freeze Rotation Z -valinta päälle, koska muuten jos pelihahmo on harmaan alustan reunalla, se voi alkaa kaatumaan Z-akselissa.

Pelihahmoon luodaan Char_movement-niminen skripti, jossa ohjataan pelaajaa ja animaatiota. Hyppääminen tapahtuu tekemällä skriptiin kohta, jossa lisätään Rigidbodyyn kohdistuvaa voimaa (lähdekoodi 3). Koodinpätkä lisätään Update()-funktioon.

```
if (Input.GetKeyDown(KeyCode.Space) && (rb.velocity.magnitude <= 0.01f))
    {
       rb.AddForce(Vector2.up * 400);

    }
```

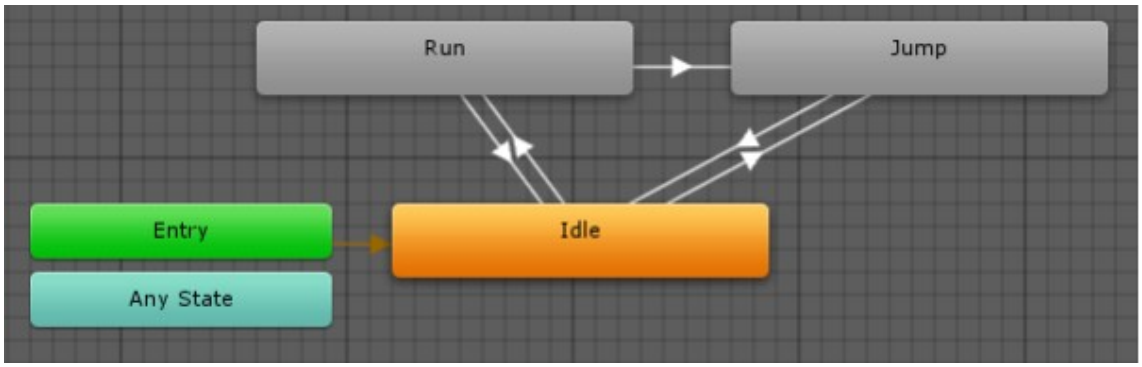
LÄHDEKOODI 3. Hyppy

If-lauseessa tutkitaan, onko välilyöntiä painettu ja onko rigidbodyssä pystysuuntaista voimaa. Tämän jälkeen rigidbodyyn lisätään ylöspäin kohdistuvaa voimaa.

3.6 Hahmon animointi

Gameobjectin spritejä voidaan vaihdella ja liikutella halutulla tavalla lisäämällä siihen animaatioita. Pelaaja-gameobjectiin lisätään Animator-komponentti ja Assets-valikoimaan lisätään Player-niminen Animator Controller -tiedosto, Player-niminen Animator Controller viedään Animator-komponenttiin. Assets-valikoimaan luodaan myös kolme Animationia, joiden nimet ovat Idle, Jump ja Run. Tämän jälkeen avataan Player-Animator Controller Animator-ikkunassa.

Animator-ikkunan kaavioon lisätään Entry-tilan lisäksi kolme tilaa: Idle, Jump ja Run, tila luodaan painamalla hiiren toista näppäintä ja valitaan Create State ja Empty. Näiden kolmen tilan väliin laitetaan yhteensä viisi siirtymävaihdetta: meno-paluu Idlestä Jumpiin, paluu Runista Jumpiin ja meno-paluu Idlestä Runiin. Siirtymä tehdään painamalla hiiren toista näppäintä tilan kohdalla ja valitaan ”Make Transition”, jonka jälkeen lisätään se haluttuun tilaan. Siirtymien lisäyksen jälkeen kaavio näyttää kuvan 2 mukaiselta.



KUVA 2. Pelaajan animaatioiden kaavio

Jotta animaatio toimisi, animaatioiden tilojen siirtymien pitäisi tapahtua, kun joku parametri muuttuu. Animaatioon lisätään kaksi boolean-tyyppistä parametria: Run ja Jump. Run-parametri on tosi silloin kun pelihahmo liikkuu ja Jump on tosi silloin kun pelihahmo on ilmassa. Siirtymiin lisätään parametrit alla olevan taulukon 1 mukaisesti. Jokaiseen tilaan tulee lisätä animaatio. Se lisätään Animator-näkymässä valitsemalla tila ja siirtämällä haluttu animaatio Assets-valikoimasta animaation tilaan.

TAULUKKO 1. Siirtymien parametrien tulostaulu

Animaatioiden tilojen siirtymä	**Parametrin tila**
Idle → Jump	Jump = tosi
Idle ← Jump	Jump = epätosi
Run → Jump	Jump = tosi
Idle → Run	Run = tosi
Idle ← Run	Run = epätosi

Jokaisessa animaation tilassa tulee olla jokin kuva tai kuvasarja. Jotta tiloja voidaan muokata, tulee Player-gameobject olla valittuna Hierarchystä. Tämän jälkeen muokkaus Animation-näkymässä on mahdollista. Animation-näkymässä valitaan muokattava animaatio ja painetaan ”Add Property”-näppäintä, josta valitaan Sprite Rendererin kohdasta Sprite. Nyt voidaan lisätä kuva tai kuvasarja aikajanalle.

Pelihahmo-gameobjectiin liitetyllä skriptillä ohjataan myös animaattoria muuttelemalla Animator Controlleriin lisättyjä parametrejä. Animaatioiden ohjauksessa ensin tehdään olio Animator-komponentista (lähdekoodi 4).

```
Animator anim;
anim = GetComponent<Animator> ();
```
LÄHDEKOODI 4. Animator-olion luonti

Tämän sen jälkeen muutellaan Animatorin parametrejä. Esimerkiksi Jump-parametriä muutellaan hyppäystilanteessa tutkimalla, onko rigidbodyssä pystysuuntaista voimaa ja sitten muokataan boolean-tyyppistä Jump-parametriä (lähdekoodi 5).

```
if (rb.velocity.magnitude >= 0.01f) {
        anim.SetBool ("Jump", true);
    } else {
        anim.SetBool("Jump", false);
    }
}
```
LÄHDEKOODI 5. Jump-parametrin käsittely

3.7 Hahmon liikkuminen

Hahmoa liikutellaan Char_movement-skriptin Update()-funktiossa. Vasemmalle päin liikuttaessa tutkitaan, onko vasenta nuolinäppäintä painettu ja onko myös oikea nuolinäppäin painettuna. If-lauseessa siirretään gameobjectin transform-komponentin paikkaa X-akselin suunnassa ja tarvittaessa käännetään pelihahmon muuttamalla Transformin X-suuntaista skaalaa. Myös Animator-komponentin Run-parametri säädetään todeksi, jotta pelihahmo näyttää juoksevalta. Kun nuolinäppäintä ei enää paineta, Run-parametri palautetaan epätodeksi (lähdekoodi 6).

```
if(Input.GetKey(KeyCode.LeftArrow) && !Input.GetKey(KeyCode.RightArrow))
    {
        transform.localScale = new Vector2(-2, transform.localScale.y);
        transform.position = new Vector2(transform.position.x - speed, transform.position.y);
        if(anim.GetBool("Run") == false)
            anim.SetBool("Run", true);
    }
else {
        anim.SetBool("Run", false);
    }
```

LÄHDEKOODI 6. Hahmon liikkuminen ja Run-parametrin käsittely.

Hahmon liikkuminen oikealle tehdään samalla tavalla kuin vasemmalle liikuttaessa. Transformin x-akselin suuntainen skaala ja liikkuminen x-akselin suunnassa on vastakkainen.

3.8 Kamera

Hierarchyssä sijaisevaan Main Camera -gameobjectiin luodaan skripti, jossa seurataan Player-gameobjectin liikkumista. Luodaan FolliwingCamera-skripti ja lisätään alla olevat rivit skriptiin (lähdekoodi 7).

```
GameObject player;
void Start () {
    player = GameObject.Find ("Player").gameObject;
}
void Update () {
    transform.localPosition = player.transform.position + new Vector3 (0, 1, -10);
    }
```

LÄHDEKOODI 7. Pelaajaa seuraava kamera

Skriptissä ensin luodaan Player-gameobjectista player-olio ja muutetaan Main Cameran Transform-komponentin paikka samaksi kuin Player-gameobjectilla. Paikkaan lisätään myös yksi float-arvo Y-akselissa, jotta pelaaja näkisi paremmin, mitä on Player-gameobjectin yläpuolella, ja lisätään kymmenen float-arvoa Z-akselissa, jotta kamera olisi oikealla paikalla.

3.9 Luodin luonti ja toiminta

Pelihahmo ampuu pelissä luoteja. Jos pelihahmo ampuu luodilla vihollista, vihollinen tuhoutuu. Luoti tehdään luomalla Hierarchyyn Bullet-niminen gameobject ja listätään siihen Sprite Renderer-, Circle Collider 2D -komponentit ja Bullet-niminen skripti. Bullet-skriptiin lisätään koodia (lähdekoodi 8).

```
void Update () {
    if(facingRight == true)
    transform.position = new Vector2(transform.position.x + speed, transform.position.y);
    else
    transform.position = new Vector2(transform.position.x + -speed, transform.position.y);
}
```

LÄHDEKOODI 8. Luodin suunnan määritys

FacingRight on boolean-tyyppinen muuttuja, jonka pelaaja tai vihollinen antaa luodille ampumistilanteessa. Se määrittää, lentääkö luoti oikealle vai vasemmalle. Luodin Transform-komponentin x-akselin paikkaan lisätään float-tyyppisen muuttujan ”speed” arvoa (lähdekoodi 9).

```
public GameObject go;
void OnTriggerEnter2D(Collider2D col)
{
    if (col.gameObject.transform.name == "Border") {
        Destroy (gameObject);
    }
        if ((col.transform.name == "Player") && (go.transform.name == "Enemy")) {
            Application.LoadLevel (Application.loadedLevelName);
        } else if ((col.transform.parent.name == "Enemy") && (go.transform.name == "Player")) {
            col.transform.gameObject.SetActive (false);
            Destroy (gameObject);
        }
    }
```

LÄHDEKOODI 9. Pelaajan ja vihollisen törmäyksen käsittely

Luodin ampumistilanteessa määritetään ampuja Gameobject-tyyppiseen go-olioon. Jos luoti törmää Border-nimiseen harmaaseen alustaan, luoti-gameobject tuhoutuu. Jos go-olio on pelaaja ja luoti törmää viholliseen, vihollinen piiloitetaan gameobjetin SetActive()-funktiolla sekä luoti-gameobject tuhotaan.

Pelaaja ampuu luodin painamalla Ctrl-näppäintä. Luodin ampuminen tehdään pelaajan Char_movement-skriptin Update()-funktioon (lähdekoodi 10).

```
bool facingRight = true;
void Update () {

    if (Input.GetKeyDown(KeyCode.LeftControl) || Input.GetKeyDown(KeyCode.RightControl)){

        bullet.GetComponent<Bullet>().go = gameObject;
        bullet.GetComponent<Bullet>().facingRight = facingRight;
        Instantiate (bullet, transform.position + new Vector3(0.2063f, 0f, 0f), transform.rotation);

    }
}
```

LÄHDEKOODI 10. Luodin ampuminen

Luoti ammutaan, jos Control-näppäintä on painettu, ja skripti syöttää luodista tehdyn bullet-olion go-muuttujaan ampujan olevan pelihahmo. Skripti myös syöttää boolean-tyyppiseen facingRight-muuttujaan saman arvon kuin Char_movement-skriptin facingRight-muuttujassa, joka tutkii kumpaan suuntaan pelaaja katsoo.

3.10 Vihollisten luonti

Viholliset ovat alustojen päällä sijaitsevia gameobjecteja, jotka ovat nimetty ”Enemy”. Ne tuhoutuvat, kun pelaajan ampuma luoti osuu niihin. Niillä on Sprite Renderer-komponentti spriteä varten ja Box Collider2D-komponentti törmäyksen rekisteröintiä varten. Vihollinen tuhoaminen on tehty luodin toimintaa kuvaavassa skriptissä. Valmis vihollinen on kuvassa 3.

KUVA 3. Vihollinen

3.11 Äänet

Scenen Hierarchyssa sirjaitsevaan Main Camera -gameobjectiin luodaan AudioController-niminen skripti, jolla ohjataan ääniä. Skripti pitää laittaa Camera-komponentin sisältävään gameobjectiin. Main Camera -gameobjectiin voidaan laittaa myös yksi Audio Source -komponentti, johon laitetaan pelin musiikki. Musiikki lisätään painamalla AudioClip-kohdan valikkoa ja valitsemalla haluttu musiikkitiedosto.

Pelin äänien lisäämistä varten projektiin kannattaa luoda Assets-kansioon Resources-kansio, johon tarvittavat äänet liitetään. Resources-kansiossa olevat äänet voidaan helposti lisätä skriptissä Main Camera -gameobjectiin luotaviin Audio Source -komponentteihin. Esimerkki AudioController-skriptin koodista, jossa käsitellään pelaajan ja vihollisen ampumisessa syntyvää ääntä (lähdekoodi 11):

```
AudioSource gunSource;
AudioClip gunClip;
void Start(){
gunSource = gameObject.AddComponent<AudioSource> ();
gunClip = Resources.Load("gun") as AudioClip;
gunSource.clip = gunClip;
}
   public void PlayGun(){
      gunSource.Play ();
}
```

LÄHDEKOODI 11. Ampumisäänen käsittely

AudioController-skriptissä ensin luodaan AudioSource- ja AudioClip-oliot, liitetään gunSource-olio Main Camera -gameobjectiin lisättävään Audio Source -komponenttiin ja liitetään gunClip-olioon Resources-kansiossa sijaiseva äänitiedosto. Yhdistetään gunSourcen clip-muuttujaan gunClip-olio. Luodaan PlayGun()-funktio, jossa suoritetaan ääni. Ääntä tulee käyttää Char_movement-skriptissä (lähdekoodi 12).

```
Audiocontroller ac;
void Start(){
ac = GameObject.Find ("Main Camera").GetComponent<Audiocontroller> ();
}
...
ac.PlayGun ();
```

LÄHDEKOODI 12. Ampumisäänen suorittaminen

Char_movement-skriptissä luodaan Audiocontroller-olio ja liitetään siihen Main Camera-gameobjectissa sijaitseva Audiocontroller-skripti. Ampumisääni suoritetaan Audiocontroller-skriptin PlayGun()-funktiolla.

3.12 Kentän läpäisy

Jotta peli voidaan läpäistä, luodaan peliin vielä Goal-gameobject. Gameobjectiin lisätään Sprite Renderer, johon lisätään maalia muistuttava sprite, BoxCollider2D-komponentti IsTrigger-ominaisuudella törmäystä varten ja ”Goal”-niminen skripti, jossa törmäys tunnistetaan. Goal-skriptissä on varsin yksinkertainen törmäyksen tunnistaminen (lähdekoodi 13).

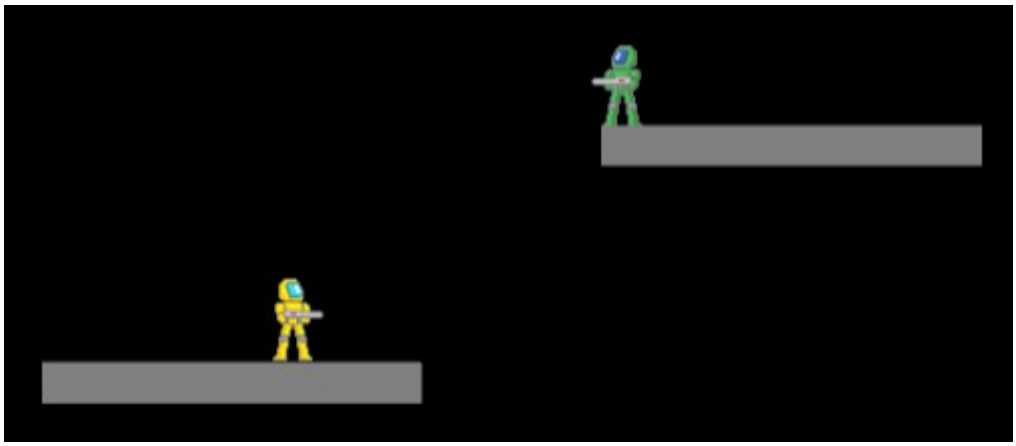
```
void OnTriggerEnter2D(Collider2D col)
{
    if (col.gameObject.name == "Player") {

        Application.LoadLevel(1);

    }
}
```

LÄHDEKOODI 13. Kentän läpäisy

Collider-tapahtumassa tunnistetaan pelaaja ja avataan build-valikossa oleva päävalikko-scene. Peli on nyt valmis pelattavaksi (kuva 4).



KUVA 4. Valmis peli

4 UNREAL ENGINE -PELIMOOTTORIN KOKONAISUUS

4.1 Historia

Unreal Engine on Epic Gamesin luoma ja ylläpitämä pelimoottori, jota on kehitetty vuodesta 1998 alkaen. Unreal Engine on tukenut alun perin PC-pelejä, mutta myöhemmin versiossa 2.0 se on alkanut tukemaan Playstation 2-, Xbox- ja Gamecube-pelikonsoleja. Xbox 360- ja Playstation 3 -pelikonsoleiden ilmestyessä Unreal Enginestä tuli suurimpien pelejä julkaisevien ja kehittävien yhtiöiden työkalu. Versiossa 3 Unreal Engine on alkanut tukemaan matkapuhelimissa Applen iOS-käyttöjärjestelmää. (IGN 2010.)

Nykyään Unreal Engine tukee kaikkia yleisimpiä tietokonekäyttöjärjestelmiä, pelikonsoleita ja mobiilikäyttöjärjestelmiä. Unreal Engine tukee myös virtuaalitodellisuuslaitteita. (Epic Games. 2016a.)

4.2 Ominaisuudet

Grafiikoiden renderöintiin Unreal Engine käyttää DirectX 11:tä (Epic Games. 2016b). Fysikoiden mallinnukseen Unreal Engine käyttää Nvidia PhysX:ää (Epic Games. 2016c). Unreal Enginen graafisena editorina käytetään Unreal Editoria, jolla voidaan tarkastella projektia useista eri näkymistä, joista yleisimmät ovat pelin kehitysselain, työkalunäkymä (Modes), sisältönäkymä (Content Browser) ja projektin luokat listaava lista (World Outliner). Blueprint-ikkunassa yleisimmät näkymät ovat komponenttilista (Components), blueprintin luokat, skriptit ja muuttujat listaava lista (My Blueprint), kehitysselaimet (ViewPort, Construction Script, Event Graph), ominaisuusnäkymä (Details) ja kääntäjän ulostulo (Compiler Results).

Unreal Enginessä pelit muodostuvat Level-kentistä, joihin on sijoiteltu blueprint-luokkia. Blueprint luokan yleisimmät tyypit on näyttelijä (Actor) sekä pelinappula (Pawn). Blueprint sisältää toimintaa muokkaavia komponentteja, joita ovat esimerkiksi blueprintin paikan ja koon määrittävä Transform, kaksiulotteisen pelinappulan kontrollointikomponentti CharacterMovement, spriten käsittelyssä käytettävä Paper Sprite, skriptit ja Event Graph-kaaviot.

Unreal Engine -pelin ohjelmoinnissa käytetään ohjelmointiympäristönä Microsoft Visual Studiota, ohjelmointikielenä käytetään C++:aa. Ohjelmoinnin sijaan pelin ominaisuudet voidaan tehdä myös blueprint-kaavioilla.

5 UNREAL ENGINE-PELIN KEHITYS

5.1 Projektin luonti

Unreal Engine -projekti luodaan painamalla Unreal Editor -ikkunassa New Project -välilehteä. New Project -näkymässä valitaan Blueprint-välilehti ja valitaan 2D Side Scroller -asetukset. Kun 2D Side Scroller -asetukset on valittuna, projektiin luodaan hyödyllisiä kaksiulotteisen pelin kehityksessä käytettäviä luokkia ja blueprint-kaavioita. Näiden ansiosta ei tarvitse tehdä koko peliä alusta asti. Tämän jälkeen voidaan määritellä, minne projekti sijoitetaan kiintolevylle ja minkä niminen projekti on. Kun kaikki on valmista, painetaan Create Project -näppäintä.

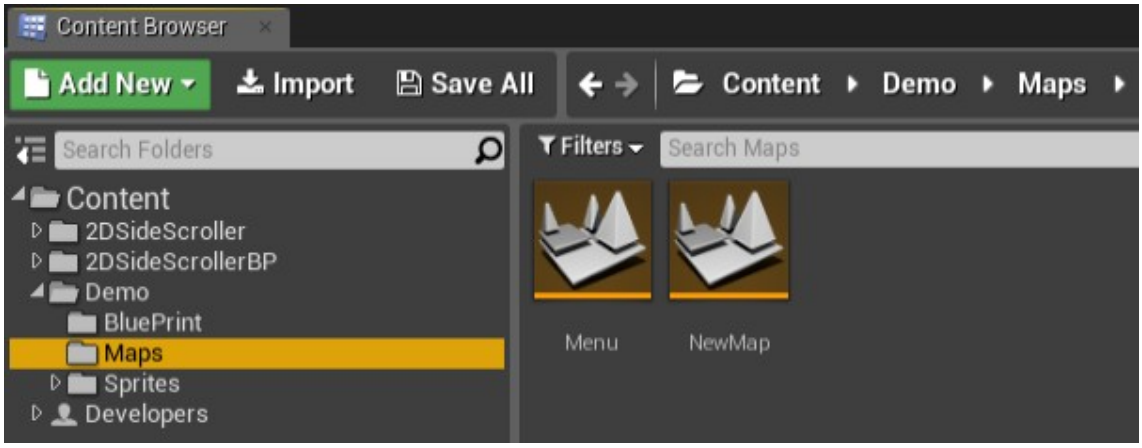
Unreal Engine luo projektiin esimerkkikentän, jossa pelaaja voi liikkua pelihahmolla. Peliin on tässä vaiheessa luotu pelihahmon liikkumiseen vaadittavat luokat. Pelihahmolla on esimerkki-spriteillä tehdyt animaatiot, joita voidaan käyttää pohjana myöhemmin pelihahmon animoinnissa. Pelihahmolta kuitenkin puuttuu hyppyanimaatio.

Ensin luodaan kopio pelaajahahmon 2DSideScrollerCharacter-BluePrintistä. Kehittäessä muokataan koko ajan luotua kopiota ja alkuperäistä blueprintiä voidaan pitää varalla, jos jokin kehityksessä menee pieleen. Ruudun alaosassa olevassa Content Browserissa tehdään uusi kansio. Painetaan kerran Content-otsikkoa, jolloin kansio tehdään projektin juureen. Kansio luodaan painamalla hiiren oikeaa näppäintä ja valitaan New Folder. Kansio nimetään esimerkiksi ”Demo”:ksi. Mennään sisälle Demo-kansioon ja luodaan uusi kansio nimeltä ”BluePrint”. Mennään takaisin Content-kansioon ja mennään 2DSideScrollerBP ja BluePrints -kansioon, kopioidaan 2DSideScrollerCharacter-blueprint ja liitetään se luotuun Demo- ja BluePrint -kansioon. Kopioitu 2DSideScrollerCharacter-BluePrint uudelleennimetään BP_Player-nimellä.

Nyt muutetaan peli käyttämään luotua BP_Player-blueprintiä. Painetaan näkymän keskiosan yläosissa olevaa Settings-näppäintä ja painetaan World Settings -näppäintä. Näkymän oikealle sivulle avautuu World Settings -valikko, josta muutetaan Game Mode -kohdasta Selected GameMode -pudotusvalikosta kohta Default Pawn Class. Default Pawn Classiin muutetaan pudotusvalikosta BP_Player-blueprint.

Luodaan Demo-kansioon BluePrint-kansion rinnalle uusi kansio, kansio nimetään Maps-nimellä ja Maps-kansioon lisätään kaikki pelin kentät. Luodaan ensin kenttä, jossa pelin varsinaiset tapahtumat tapahtuvat. Painetaan Menu-valikosta File ja painetaan New Level -näppäintä. New Level
-ruudussa painetaan Default-valintaa, jolloin luodaan uusi kenttä. Kenttä on hyvä tallentaa painamalla Menu-valikosta File ja valitsemalla Save As. Kenttä tallennetaan Maps-kansioon ja tallennetaan nimellä ”NewMap”. Luodaan toinen kenttä, josta tehdään alkuvalikko. Kenttä tehdään samalla tavalla kuin edellinen kenttä, mutta Default-asetuksen sijaan valitaan Empty Level. Kenttä nimetään ”Menu”. Content-kansio voidaan tehdä selkeämmän näköiseksi painamalla Search Folders -etsintätyökalun vieressä olevaa näppäintä.

Kenttä pitää ottaa käyttöön projektissa. Valitaan Menu-valikossa Edit ja painetaan Project Settings. Project Settings -ikkunassa valitaan vasemmalla reunalla olevasta valikosta Projectin kohta Maps & Modes. Tässä näkymässä muutetaan Default Maps -kohdossa kumpaakin kohtaan aikaisemmin tallennettu kenttä. Kuvassa 5 on luodut kentät Maps-kansiossa.



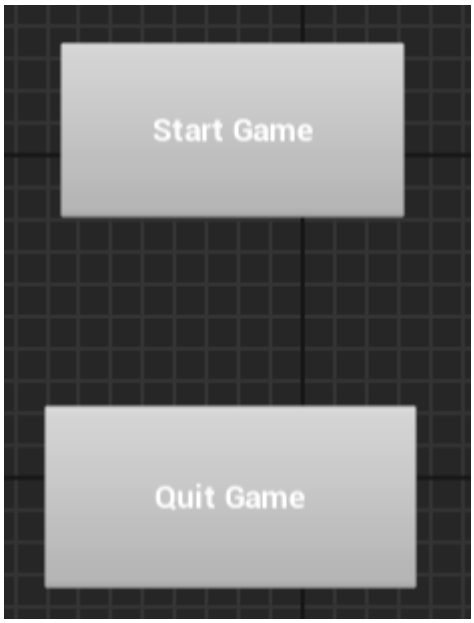
KUVA 5. Kentät Maps-kansiossa

Luodaan Content-kansioon uusi kansio nimeltä Sprites. Tähän kansioon sijoitetaan kaikki pelissä käytössä olevat spritet. Tähän kansioon sijoitetaan kaikki pelissä käytössä olevat spritet.

Luodaan resurssienhallinnassa projektin juureen kansio nimeltä ”Source”. Tähän kansioon sijoitetaan kaikki spritet, ennen kun ne siirretään projektiin. Liitetään Source-kansioon kaikki käytössä olevat spritet ja palataan takaisin Unreal Engineen. Content-menussa valitaan Sprites-kansio ja painetaan hiiren oikeata näppäintä, josta valitaan Import Asset -kohta. Valitaan kaikki Source-kansiossa olevat spritet ja painetaan Open.

5.2 Valikot

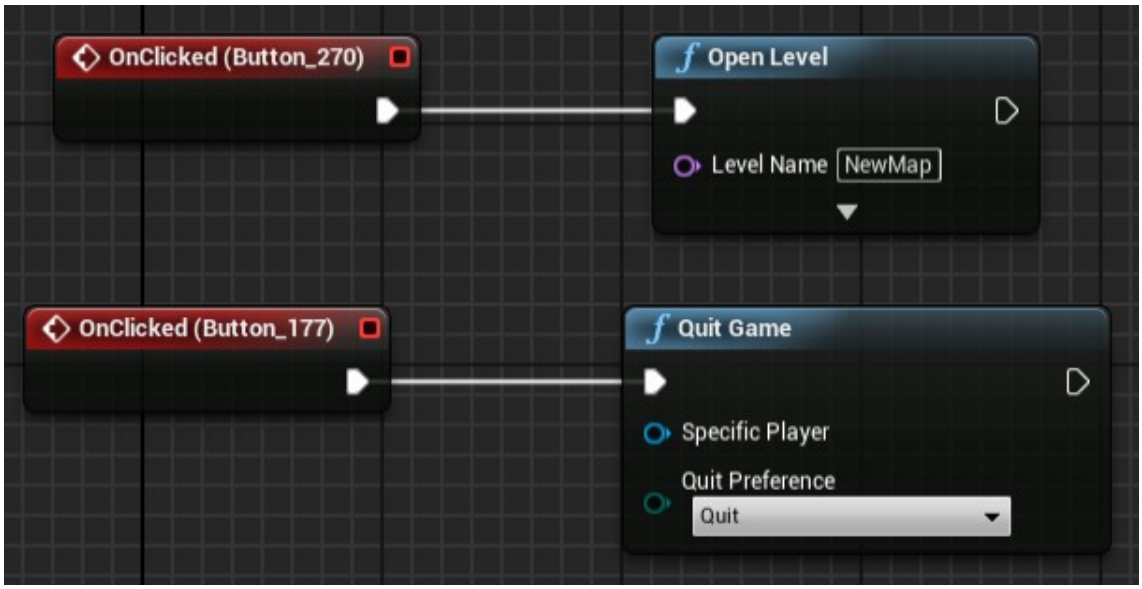
Avataan Maps-kansiosta Menu-kenttä. Blueprint-kansioon luodaan uusi menu blueprint, painetaan hiiren oikeata näppäintä ja valitaan Create Advanced Assetsista User Interface -valikko, josta valitaan Widget Blueprint, blueprint nimetään ”BP_Menu”. Tuplaklikataan blueprintiä, jolloin avautuu blueprint-ikkuna. Vasemmalla puolella olevasta Palette-näkymästä siirretään työskentelynäkymään kaksi Button-näppäintä ja kummankin näppäimen sisään yksi Text-tekstialue. Näppäimen kokoa suurennetaan ja ne asetellaan keskelle työskentelynäkymää. Valitaan näppäimen tekstialue ja muutetaan ikkunan oikealla puolella olevasta Details-näkymästä kummankin näppäimen tekstialueen Content-valikoiman olevaa Text-kohtaa. Näppäimet nimetään ”Start Game” ja ”Quit Game”. Kuvassa 6 on esitelty valikon näppäimet.



KUVA 6. Valikon näppäimet

5.3 Valikon näppäinten toiminta

Kun Start Game -näppäin on valittuna blueprint-ikkunassa, Details-näkymässä on Events-valikoima, jossa on OnClicked-event. Klikataan sitä, jolloin avautuu Event Graph -ikkuna, jossa voidaan tehdä näppäimen toiminnot. Lisätään toiminto klikkaamalla hiiren oikeaa näppäintä, jolloin avautuu lista, joista valitaan haluttu komento. Valitaan Open Level -komento, joka kytketään OnCliked-komentoon. Open Level -komennon Level Name -kohtaan kirjoitetaan ”NewMap”. Quit Game -näppäimessä on samanlainen OnClicked-event. Event Graph -näkymässä OnClicked-komentoon kytketään Quit Game -komento. Kuvassa 7 on esitelty valikoiden näppäimien toiminta Event Graphissa.



KUVA 7. Valikon näppäimien toiminta

Palataan takaisin Unreal Editor -ikkunaan. Ikkunan keskiosassa on Blueprints-näppäin, jota painetaan ja valitaan Open Level Blueprint, jolloin avautuu jälleen blueprint-ikkuna. Event Beginplay -eventiin kytketään Create Widget -komento, jonka Class-kohdaksi valitaan ”BP_Menu”. Lisätään Add to Viewport -komento ja Get Player Controller -komento, jonka Return Value -kohdasta lisätään Set Show Mouse Cursor -komento (kuva 8). Nyt valikko on valmis ja pelikenttä aukeaa Start Game -näppäintä painamalla (kuva 9).

KUVA 8. Valikon lisääminen peliin

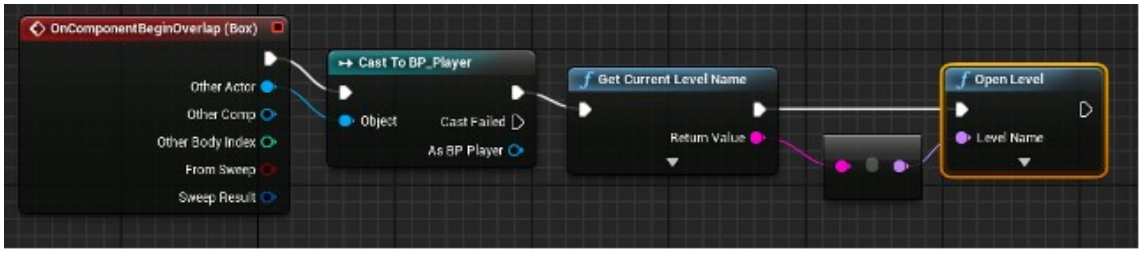
KUVA 9. Valmis valikko

5.4 Pelialueen luonti

Avataan Maps-kansiosta NewMap-kenttä. Ensin poistetaan alkuperäinen testialusta valitsemalla se ja painetaan näppäimistön Delete-näppäintä. Tehdään alustan kuvasta sprite valitsemalla kuva ja hiiren oikealla näppäimellä valitaan kohta Sprite Actions ja Create Sprite. Alustat voidaan nyt luoda peliin siirtämällä sprite pelialueelle pelihahmon alle ja muutetaan ikkunan oikealla puolella olevassa Details-näkymässä Scale-kohdan X- ja Y-arvoja. Alustalle luodaan automaattisesti törmäyksen rekisteröivä collider-komponentti.

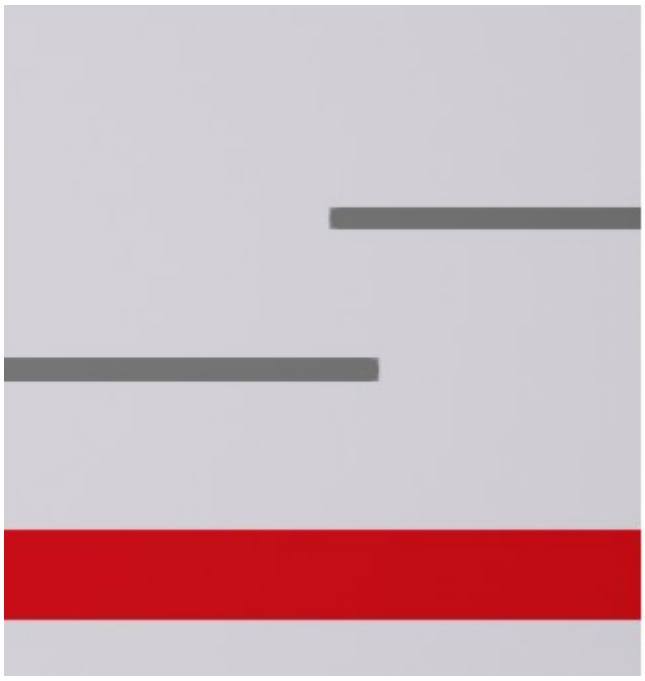
Vaarallisen punaisen alustan luomiseksi täytyy luoda uusi blueprint. Blueprint luodaan BluePrint-kansioon ja se tehdään painamalla hiiren oikeata näppäintä ja valitaan Create Basic Assetsista BluePrint Class. Uusi blueprint nimetään ”Death”. Tuplaklikkaamalla avautuu blueprintin muokkaamiseen käytettävä ikkuna. Lisätään vasemmalla ylhäällä olevasta Components-näkymästä Add Component -näppäimellä Paper Sprite -komponentti ja Box Collision -komponentti. Ikkunan oikealla puolella on Details-näkymä, jossa voi muutella komponentin ominaisuuksia. Paper Sprite -komponentin Details-näkymässä Source Sprite -kohdasta vaihdetaan käytettävä sprite. Box Collision -komponentin Details-näkymässä colliderin kokoa muutetaan Shape-kohdan X-, Y-, ja Z-arvoja muuttamalla. Details-näkymän alaosassa on Collision-kohta, jossa valitaan valinta Simulation Generates Hit Event. Alempana on Events-kohta, jossa klikataan kohtaa On Component Begin Overlap.

Nyt voidaan lisätä törmäykseen toimintaa Event Graph -välilehdessä. Törmäykseen lisätään Cast To BP_Player -komento, Get Current Level Name -komento ja Open Level -komento. Komennot liitetään yhteen kuvan mukaisesti (kuva 10).



KUVA 10. Pelaajan törmäys vaaralliseen alustaan

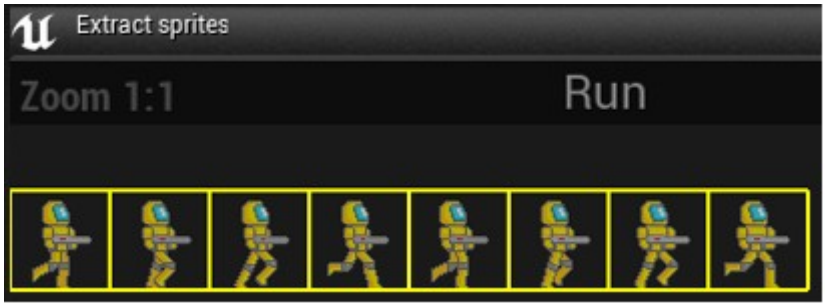
Testatessa pelin tulisi kuvan 11 kaltaiselta.



KUVA 11. Alustat

5.5 Hahmon luonti ja fysiikat

Koska pelissä käytetään retrohenkisiä pikselöityjä spritejä, ne pitää käsitellä Unreal Engineen sopiviksi. Se tehdään yksinkertaisesti painamalla hiiren oikealla näppäimellä Content-kansiossa olevaa spritesheetia ja valitaan Sprite Actions, josta vielä valitaan Apply Paper2D Texture Settings -valinta. Seuraavaksi erotellaan spritet spritesheetistä valitsemalla Sprite Actionsista Extract Sprites. Nyt avautuu ikkuna, jossa erottelu tapahtuu. Valitaan Sprite Extract Mode -pudotusvalikosta Grid ja muutetaan Cell Width -kohtaan oikea spriten leveys (kuva 12). Lopuksi painetaan Extract-näppäintä.



KUVA 12. Pelaaja-hahmon spritesheetin jakaminen erillisiin spriteihin

Hahmon fysiikat on valmiiksi tehty projektin luontivaiheessa, joten niitä ei tarvitse enää erikseen tehdä.

5.6 Hahmon animointi

Luodaan hahmon animaatiot, ensin luodaan animaation tila, jossa pelihahmo on paikallaan. Unreal Enginessä animaation tiloja kutsutaan flipbookeiksi. Flipbook luodaan valitsemalla animaatiossa käytetyt spritet Content-ikkunassa. Koska Idle-animaatiossa käytetään vain yhtä spriteä, valitaan vain yksi kuva. Sprite valitaan ja painetaan hiiren oikeaa näppäintä ja valitaan valikoimasta ”Create Flipbook”, jonka jälkeen Flipbook nimetään ”A_Idle”. Koska A_Idle-flipbook koostuu ainoastaan yhdestä spritestä, ei sitä tarvitse erityisemmin käsitellä toisin kuin useasta spritestä koostuvia flipbookeja.

Luodaan hyppy-flipbook. Koska sekin koostuu ainoastaan yhdestä spritestä, ei sitäkään tarvitse käsitellä. Hyppy-flipbook tehdään samalla tavalla kuin A_Idle-flipbook. Flipbook nimetään ”A_Jump”.

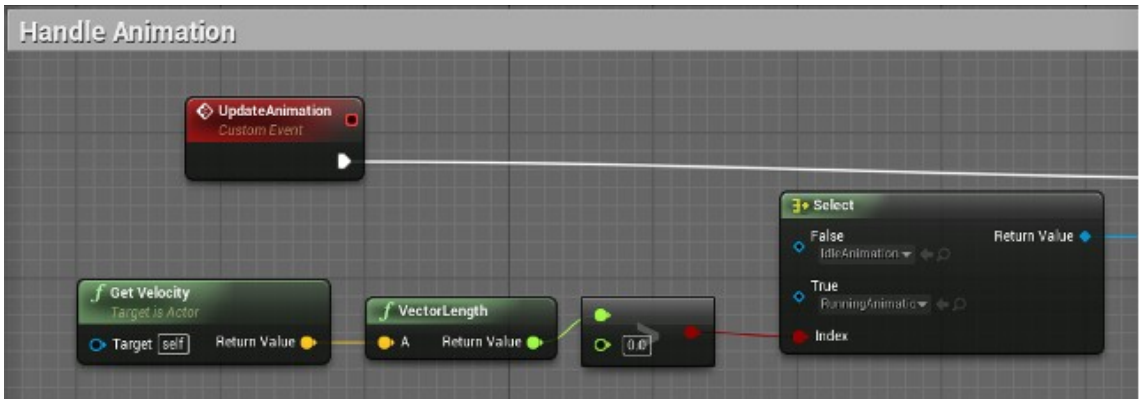
Luodaan juoksu-flipbook, valitaan Run-spritet ja tehdään niistä flipbook nimeltä ”A_Run”. Kaksoisklikataan luotua flipbookia, jolloin päästään flipbookin muokkaamisikkunaan. Flipbookissa spritet vaihtelevat liian nopeasti, joten flipbookin nopeutta tulee muuttaa. Muokataan ikkunan oikeassa reunassa olevan Sprite-valikon Frames Per Second -kohtaa ja muutetaan se 15:stä 10:een. Yksittäisten spritejen ajan pituutta flipbookissa voidaan muokata muuntelemalla ikkunan alaosassa olevia sprite-suorakaiteita, mutta sille ei nyt ole tarvetta.

Nyt muutetaan BP_Player-blueprintin flipbookit aikaisemmin luotuihin flipbookeihin. Kaksoisklikataan Content-kansiossa olevaa BP_Player-blueprintiä, jolloin avautuu blueprintin muokkaamisikkuna.

Ikkunassa valitaan Viewport-välilehti ja painetaan ikkunan vasemman puolen yläosassa olevasta Components-näkymästä ”Sprite (Inherited)”-kohtaa. Muokataan ikkunan oikealla puolella olevan Sprite-kohdan ”Source Flipbook”-kohtaa, niin että valitaan tiputusvalikosta A_Idle, jolloin vakioflipbookiksi valitaan Idle-flipbook.

Palataan takaisin Unity Editor -ikkunaan, jonka oikealla puolla olevasta World Settings-valikosta valitaan Game Mode-kohdasta GameMode Override -pudotusvalikko. Pudotusvalikosta valitaan 2DSideScrollerGameMode.

Palataan takaisin blueprint-ikkunaan, jossa mennään Event Graph -välilehteen. Tässä välilehdessä näytetään kaikki toimintalogiikka, jonka mukaan pelihahmo toimii. Event Graphissa on rajattuna Handle Animation -kohta, jossa muutellaan flipbookeja (kuva 13).



KUVA 13. Handle Animation

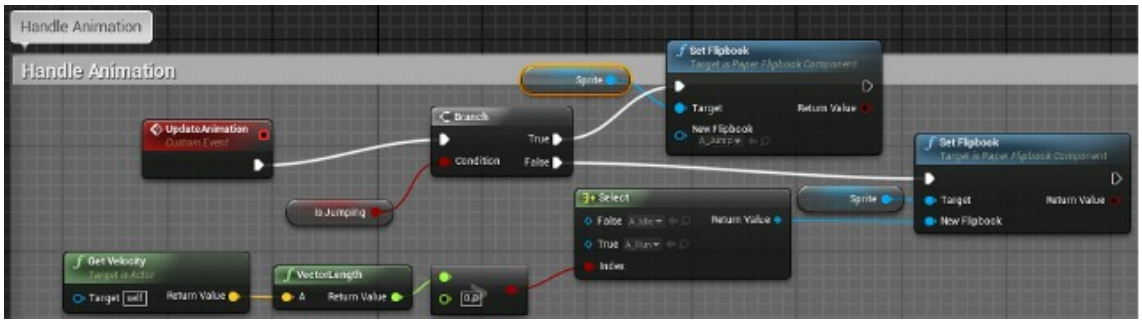
Handle Animationin Select-toiminnossa muutetaan hahmon animaatiota sen mukaan, liikkuuko pelihahmo. True-ehdolla muutetaan RunningAnimation-flipbook ja False-ehdolla IdleAnimation-flipbook. Nämä flipbookit ovat esimerkki-flipbookeja, jotka tulee vaihtaa aikaisemmin luotuihin flipbookeihin. False-ehto muutetaan A_Idle-flipbookiin ja True-ehto muutetaan A_Run-flipbookiin. Lopuksi tallennetaan.

Nyt voidaan testata pelihahmon toimintaa painamalla Play-näppäintä (kuva 14). Idle- ja Run-flipbookit toimivat, mutta Jump-flipbookia ei ole vielä lisätty peliin. Esc-näppäimellä päästään takaisin muokkaamaan blueprintiä.

KUVA 14. Pelihahmo kentässä

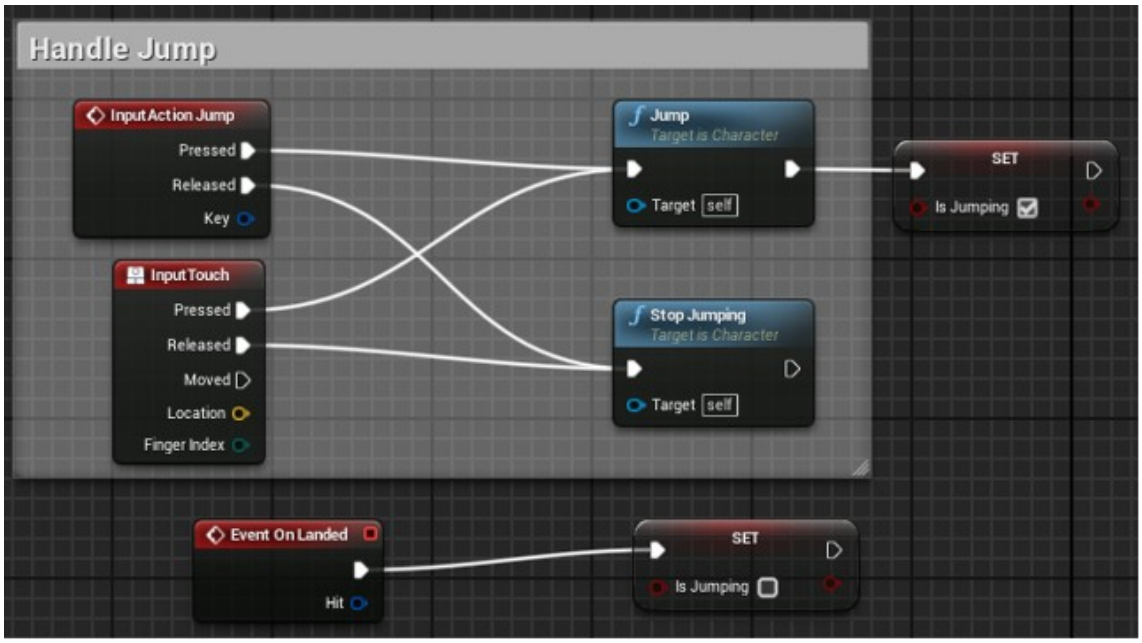
Muokataan pelihahmon collideria. Mennään blueprint-ikkunassa Viewport-välilehteen ja valitaan Components-näkymässä CapsuleComponent (Inherited). Ikkunan oikealla puolella olevasta Details-näkymästä voidaan muutella Shape-arvoja. Näitä muokkaamalla muutetaan collider oikean kokoiseksi. Nyt voidaan raahata Viewport-näkymässä spriteä oikeaan kohtaan collideria.

Laitetaan hyppyflipbook toimimaan oikein. Blueprint-ikkunan vasemmalla puolella olevassa My Blueprint -näkymässä luodaan uusi boolen tyyppinen muuttuja Variables-kohdasta, nimetään se ”Is_Jumping”. Nyt vedetään muuttuja Event Graph -välilehteen ja luodaan Get-tyyppinen komento. Luodaan Branch-komento ja Set Flipbook-komento. Set Flipbook-komentoon lisätään A_Jump-flipbook. Yhdistetään komennot Handle Animation-alueeseen Event Graphissa (Kuva 15).



KUVA 15. Hyppy-flipbookin muuttaminen animaatiossa

Handle Jump -alueeseen tulee vielä lisätä Is_Jumping -muuttujan Set-komento. Hyppy-flipbook toimii nyt niin, että flipbook muuttuu, mutta se ei palaudu hypyn loputtua. Lisätään Event Graphiin On Landed-event ja siihen liitetään Is_Jumping-muuttujan Set-komento (kuva 16). Nyt hyppyanimaatio toimii oikein.



KUVA 16. Hyppy-flipbookista poistuminen

5.7 Hahmon liikkuminen

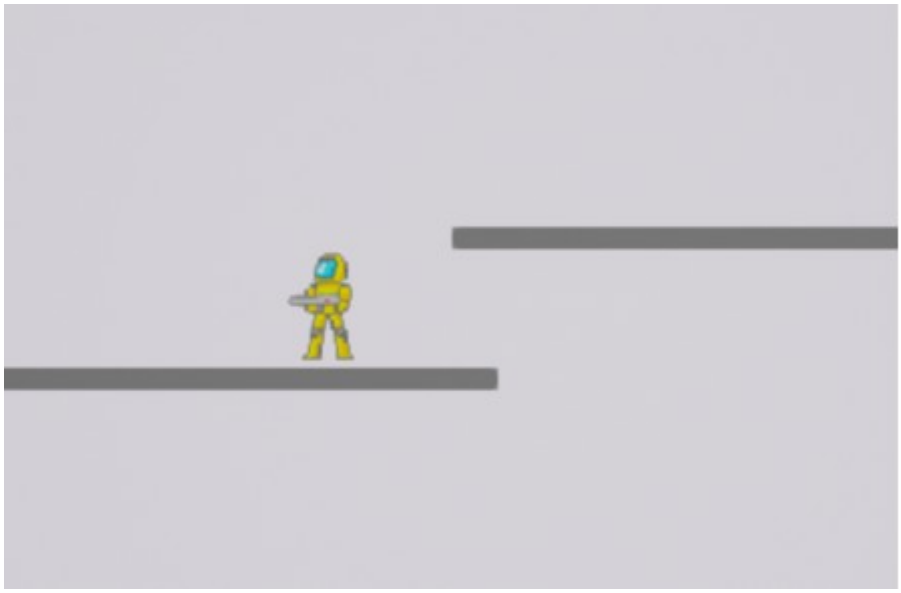
Nyt voidaan testata pelihahmon toimintaa ja animaatioita painamalla Play-näppäintä, pelihahmoa voidaan liikutella nuolinäppäimillä ja pelihahmolla voidaan hypätä välilyönti-näppäintä painamalla. Pelihahmo näyttää pieneltä, koska käytetyt spritet ovat pienikokoisia eikä kameraa ole vielä muokattu peliin sopivaksi.

Hahmon liikkuminen on valmiiksi tehty projektin luontivaiheessa, joten sitä ei tarvitse enää erikseen tehdä. Hahmo on kuitenkin aika nopealiikkeinen ja hypyn korkeus on todella suuri. Hahmon nopeutta voidaan säätää blueprint-ikkunassa, jossa valitaan Components-näkymästä CharacterMovement (Inherited). Nopeus muutetaan Details-näkymässä muuttamalla kohtasta Walking -> Max Walk Speed. Tämä kohta voidaan muuttaa 600:sta 400:aan. Hyppy muutetaan muuttamalla kohdasta Jumping / Fallings → Jump Z Velocity, joka float-arvo muutetaan 1000:sta 700:aan.

5.8 Kamera

Kameran resoluutio vaihdetaan sopivaksi blueprint-näkymässä. Valitaan Components-näkymässä SideViewCameraComponent ja muutetaan Details-näkymän Camera Settings

-kohdan Ortho Widthiä. Alkuperäinen leveys oli 2160, mutta sopiva leveys pelille on 1080. Kamera seuraa automaattisesti pelaajaa eikä sitä enää tarvitse muokata muuten (kuva 17).

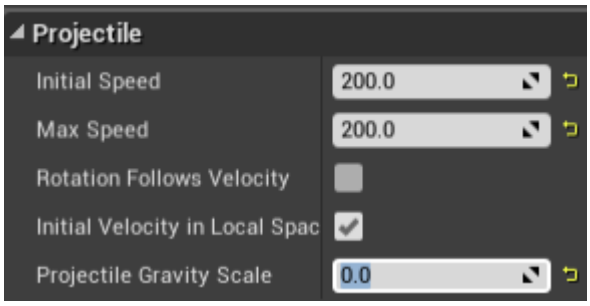


KUVA 17. Peli oikealla kameran resoluutiolla

5.9 Luodin luonti ja toiminta

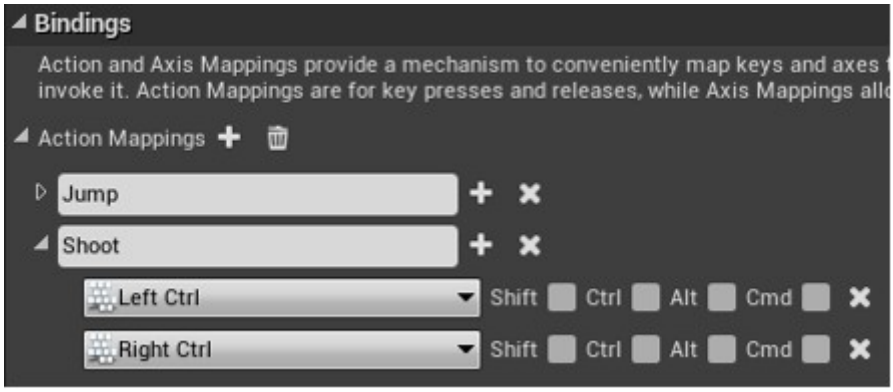
Luotia varten pitää luoda uusi blueprint. Se luodaan Content-kansion Blueprint-alikansioon. Blueprint tehdään painamalla hiiren oikeaa näppäintä ja valitaan Create Basic Assetsista Blueprint Class. Uusi blueprint nimetään ”BP_Shot”. Kaksoisklikataan blueprintiä, jolloin avautuu blueprint-ikkuna. Components-näkymässä lisätään Add Component -näppäimellä Paper Sprite -komponentti ja Projectile Movement -komponentti. Siirretään Paper Sprite -komponentti DefaultSceneRootiin, jotta saadaan asetettua komponentti komponenttihakemiston juureen.

Valitaan Projectile Movement -komponentti ja tutkitaan Details-näkymää. Projectilen Initial Speediä säätämällä voidaan muuttaa luodin nopeutta. Muutetaan se 200:aan. Max Speed -kohta muutetaan myös 200:n. Projectile Gravity Scale muutetaan nollaan, jotta luoti lentää suoraan (kuva 18). Paper Sprite -komponentin Details-näkymässä lisätään Source Sprite -kohdassa käytettävä luoti-sprite.



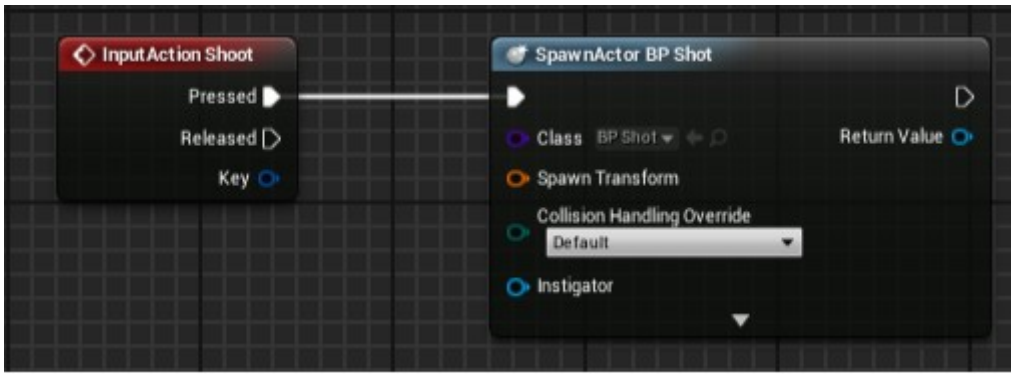
KUVA 18. Projectile Details-näkymässä

Luodaan uusi toiminto ampumiselle. Project Settings -kohdasta valitaan Engine-kohdan alta Input ja lisätään Action Mappings -kohtaan uusi toiminto, nimetään se Shoot ja lisätään toimintonäppäimiksi Left- ja Right Ctrl (kuva 19).



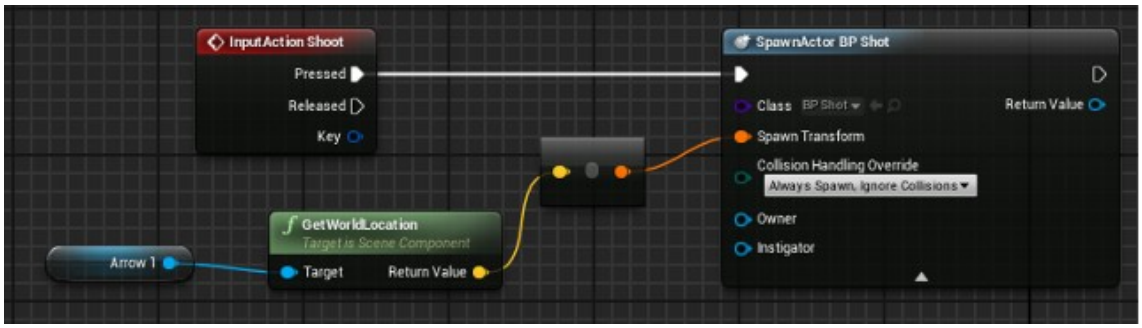
KUVA 19. Shoot-event ja Ctrl-näppäinten toiminta

Palataan takaisin BP_Player-blueprintiin. Event Graphiin lisätään aikaisemmin luotu ampumistoiminto Input Action Shoot. Lisätään Spawn Actor From Class -komento. Lisätään sen luokaksi BP_Shot (kuva 20).



KUVA 20. Luodin ampuminen

Valitaan BP_Playerin ViewPort-välilehti. Lisätään luodin lähtöpaikan määrittävä Arrow-komponentti AddComponent-näppäimellä. Komponentti nimetään se ”Arrow1”. Siirretään Arrow-komponentti pelihahmospriten piipun suulle. Palataan takaisin Event Graphiin ja lisätään Get World Rotation (Arrow1) (kuva 21).

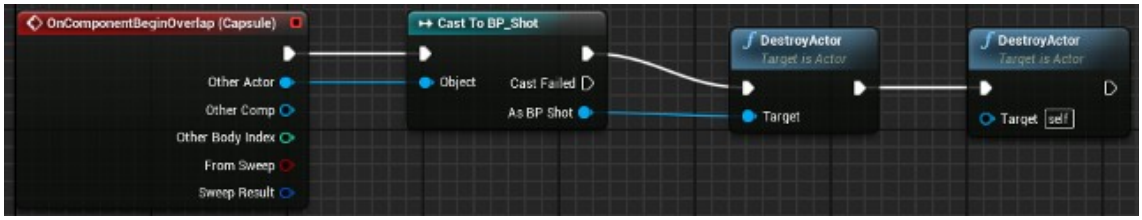


KUVA 21. Arrowin lisääminen Event Graphiin

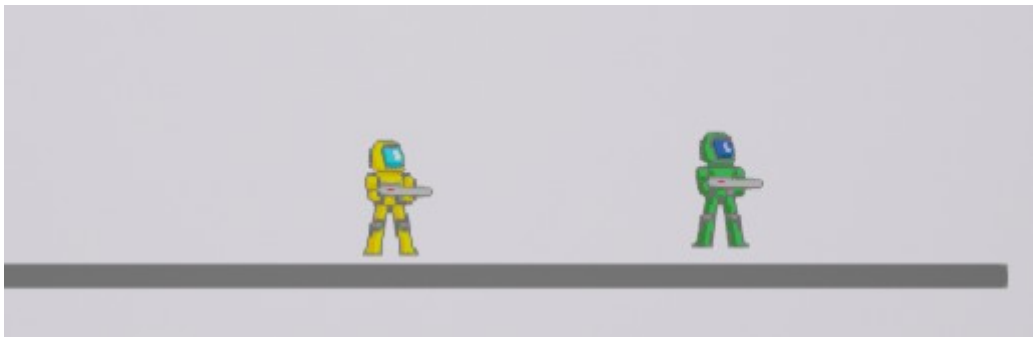
5.10 Vihollisten luonti

Luodaan uusi Actor-tyyppinen blueprint nimeltä ”BP_Enemy”, lisätään blueprintille komponentit Paper Sprite ja Capsule Collider, lisätään Paper Spriteen vihollisen sprite ja Capsule Colliderista muokataan sopivan kokoinen. Capsule Colliderin Detailsista valitaan Simulation Generates Hit Events ja luodaan On Component Begin Overlap -event.

Event Graphissa liitetään eventiin Cast To BP_Shot -komento ja kaksi Destroy Actor -komentoa. Toiseen Destroy Actoriin liitetään Cast To BP_Shot -komennon As BP Shot-viittaus, jotta luoti tuhoutuisi osuessa, toiseen Destroy Actoriin riittää Self-kohde, jotta vihollinen tuhoutuisi (kuva 22). Lopuksi vihollinen sijoitetaan pelialueelle ja testataan toiminta (kuva 23).



KUVA 22. Vihollisen tuhoaminen luodin osuessa

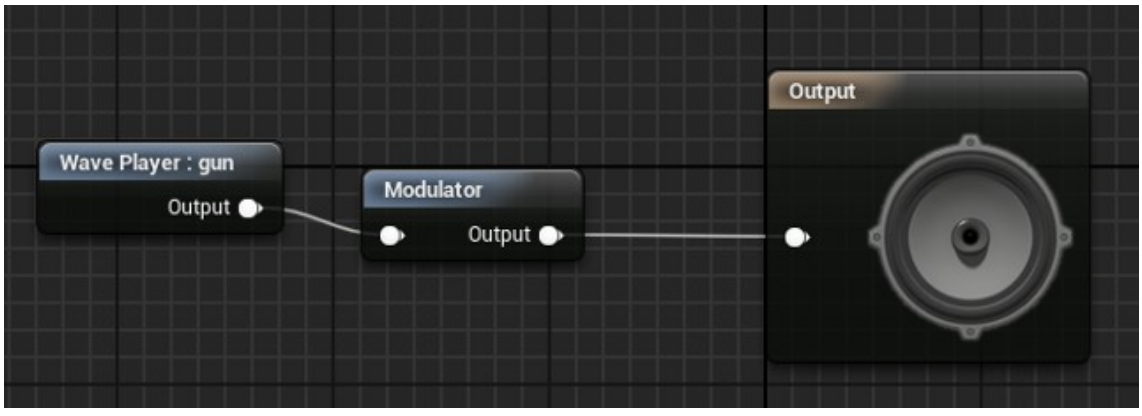


KUVA 23. Pelaaja ja vihollinen

5.11 Äänet

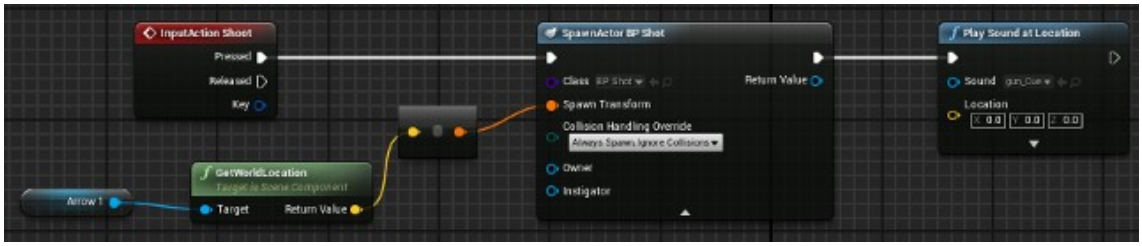
Luodaan peliin ääni, joka kuuluu, kun pelaaja ampuu luodin. Lisätään Content-kansioon ampumis-äänenä käytetty äänitiedosto ja klikataan ääntä hiiren oikealla näppäimellä ja painetaan Create Cue. Nimetään Cue ”gun_Cue”. Cue-tiedostot ovat Unreal Enginessä lähes samanlaisia kuin blueprintit, mutta niitä käytetään äänenkäsittelyssä.

Tuplaklikataan Cue-tiedostoa ja avataan Cue-ikkuna. Ikkunassa näkyy äänilähde-komento ja äänen ulostulo-komento. Näiden komentojen väliin lisätään Modulator-komento, jolla voidaan muokata äänenvoimakkuutta (kuva 24).



KUVA 24. Ampumisäänen Cue

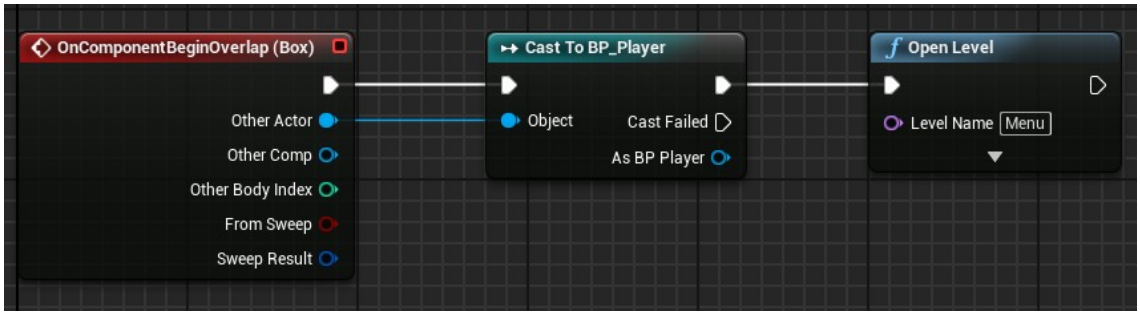
Lopuksi lisätään äänen suorittaminen BP_Player-blueprintiin. Avataan Event Graph ja lisätään Play Sound at Location -komento ampumiskomentoketjuun ja lisätään komennon Sound-kohtaan ”gun_Cue” (kuva 25).



KUVA 25. Äänen suorittaminen

5.12 Pelin läpäisy

Luodaan uusi Actor-tyyppinen blueprint ja nimetään se ”BP_Goal”. Blueprintille lisätään komponentit Paper Sprite ja Box Collision. Paper Spriteen lisätään maali-sprite ja Box Collisionista muokataan sopivan kokoinen. Kun blueprint on lisätty pelikenttään, sen kokoa tulee muokata sopivan kokoiseksi Details-näkymän Transform-valikossa. Box Collisionin Detailsissa lisätään On Component Begin OverLap -event. Event Graphissa blueprintille lisätään Cast To BP_Player-komento ja Open Level -komento, jonka Level Name -kohtaan kirjoitetaan ”Menu”-kentän nimi (kuva 26). Peli on nyt valmis pelattavaksi.



KUVA 26. Maalin toiminta

6 PELIMOOTTOREIDEN YLEINEN VERTAILU

Unityn ja Unreal Enginen eroista ei löydy kattavia tutkimuksia, mutta internetistä löytyy keskusteluja, joissa käsitellään pelimoottoreiden eroavaisuuksia.

6.1 Yhteensopivuus eri käyttöjärjestelmien kanssa

Kummallakin pelimoottorilla on tuetut julkaisut sekä Windowsille että Macintoshille. Pelimoottoreilla on myös ei-tuetut julkaisut Linuxille. Linux-julkaisu on julkaistu Unity-pelimoottorin blogissa toukokuussa 2015 ja se perustuu Windows-julkaisun versioon 5.1.0f3. (Bard 2015.)

Testasin Unityn toimintaa Linux Mint -käyttöjärjestelmällä. Unity oli helppo asentaa tietokoneelle, sillä riitti, että ladattiin .deb-asennustiedosto ja suoritettiin se. Vaikka Unity-peli on kehitetty Windows-julkaisun versiolla 5.3.3f1, peli toimi mainiosti pelimoottorin testausikkunassa, vaikka versio on vanhempi.

Unreal Engine -julkaisun asentamiseen löytyy ohjeet Linux Mintin sivujen tutoriaalivalikoimasta. En kokeillut asentaa Unreal Engineä, koska sen laitteistovaatimukset ovat paljon kovemmat kuin Linux Mintiä käyttävä kannettava tietokoneeni on tehoiltaan. Pelimoottori vaatii 2,5 GHz:n prosessorin, 8 Gt RAM-muistia ja tehokkaan näytönohjaimen, joka ei ole kannettavan tietokoneen heikkotehoinen integroitu näytönohjain. (Hamid.)

Kummallakin pelimoottorilla voidaan julkaista tietokonepelejä Windowsille, Linuxille ja iOS:lle. Unity tukee myös uusimpia pelikonsoleja kuten XBOX One ja Playstation 4. (Unity Technologies. 2016a.)

Mobiilipelejä voidaan Unityllä ja Unreal Enginellä tehdä sekä Androidille että iOS:lle. Android-pelinkehitys vaatii, että tietokoneelle asennetaan ilmainen Android SDK itse pelimoottorin lisäksi. IOS-pelinkehitys vaatii, että pelinkehittäjä tai yritys on rekisteröitynyt maksulliseen Apple IOS Developer Programiin, kehittäjällä tulee myös olla Macintoshia käyttävä tietokone Xcode-kääntämisohjelman ajamiseen. (Unity Technologies. 2016b; Epic Games. 2016d.)

6.2 Dokumentointi

Unity-pelimoottorilla on kattava dokumentaatio Unityn internet-sivulla. Dokumentaatiosta löytyy jokainen Unityssä käytettävä funktio, josta on helppo etsiä tarvittavat funktiot. Youtubesta löytyy myös paljon Unity-pelinkehitykseen keskittyviä opastusvideoita.

Unreal Enginellä on kattava dokumentaatio nettisivuillaan. Youtubesta löytyy paljon opetusvideoita. Opinnäytetyön pelin tekeminen on opeteltu Youtubesta löytyvien videoiden avulla.

6.3 Käyttöliittymä ja yleinen käytettävyys

Unity on helppokäyttöinen pelimoottori kattavan dokumentaation ja helpon käyttöliittymän ansiosta. Unreal Enginen käyttöliittymä tuntui Unityn käyttöliittymään tottuneelta todella sekavalta. Toiminnot oli jaoteltu itse Unreal Editoriin ja Blueprint-ikkunaan. Näiden välillä asioiden hoitaminen oli mielestäni hankalaa.

6.4 Hinta

Unitystä on saatavilla ilmainen Personal Edition ja maksullinen Professional Edition. Professional Edition maksaa 75 Yhdysvaltain dollaria kuukaudessa. Professional Editioniiin kuuluu ilmaisen version ominaisuuksien lisäksi mm. pilvipalveluita, analytiikkatyökalut ja pelin suorituskyvyn mittaustyökalut. (Unity Technologies. 2016c)

Epic Games tarjoaa Unreal Enginen täydellisenä versiona ilmaiseksi opiskelijoille, elokuvantekijöille, arkkitehtuurien tekijöille ja voittoa tuottamattomille organisaatioille. Kun peliorganisaatio tekee tuloa yhdellä pelillään yli 3000 Yhdysvaltain dollaria, Epic Games veloittaa pelin tuloista 5 %. (Epic Games. 2016a)

Pitkällä tähtäimellä Unity on halvempi pelimoottori pelejä ammatikseen tekeville kehittäjille.

6.5 Ohjelmointi

Unityn skriptejä voidaan koodata kahdella eri kielellä: C#:lla ja Javascriptillä. Unreal Enginellä skriptaamiseen käytetään C++:aa, mutta Unreal Enginellä voi vaihtoehtoisesti tehdä monia operaatioita blueprint-kaavojen avulla. Koin Unityllä ohjelmointia helpommaksi. C#:lla tehdyt skriptit ovat mielestäni yksinkertaisempia kuin Unreal Enginen C++:lla tehdyt skriptit.

6.6 Spritejen käsittely

Spritejä käsitellään kummassakin pelimoottorissa lähes samalla tavalla. Spritesheetissä olevat spritet erotellaan toisistaan samalla tavalla ja samannimisiä näppäimiä painamalla. Jos oli aikaisemmin käyttänyt toista pelimoottoria, on spritejen käsitteleminen helppoa myös toisessa pelimoottorissa.

6.7 Törmäysten käsittely

Kummallakin törmäysten eli collisionien käsittely on tehty helpoksi. Unreal Enginellä piti luoda uusi event eri törmäyksien tiloista ja muistaa rastittaa Simulation Generates Hit Events -kohta, jotta törmäykset toimivat. Unityllä jokaisesta törmäystilasta luodaan skriptissä tietyn niminen funktio, johon kaikki törmäysten kontrollointi ohjelmoidaan.

6.8 Animointi

Kaksiulotteisten animaatioiden tekeminen spritesheetistä jaoteltuista spriteistä on tehty Unreal Enginessä helpommin. Flipbookin luominen onnistuu valitsemalla kaikki flipbookissa käytettävät spritet ja hiiren oikealla näppäimellä avautuvasta valikosta valitaan ”Create Flipbook”. Sen sijaan Unityn animoinnissa käytetyllä aikajanalla pystyy paremmin käsittelemään useasta yksittäisestä spritestä koostuvan hahmon animointia, sillä aikajanassa pystyy myös muuttamaan komponenttien ja skriptien muuttujia.

6.9 Äänien käsittely

Ääniä käsiteltiin kummassakin pelimoottorissa eri tavalla. Unityssä piti luoda skripti, jolla luotiin Audio Source -komponentit ja äänen suorittamisessa käytettävät public-tyyppiset funktiot, joita käytetään muissa skripteissä. Unreal Enginessä äänen suorittavaan blueprintiin lisättiin äänen suorittamiskomento, johon kytkettiin äänitiedoston Cue-komponentti. Unreal Enginessä äänien käsittely oli helpompaa.

6.10 Yhteensopivuus Subversion-versionhallinnan kanssa

Opinnäytetyössä tutkittiin Subversionin toimintaa lähettämällä peliprojektien tiedostot Rasberry Pi -pohjaiselle palvelimelle. Unityssä versionhallintapalvelimelle lähetetään ainoastaan projektikansion Assets- ja Project Settings -kansiot, koska muut kansiot ja tiedostot ovat pelimoottorissa automaattisesti luotavia (Unity Technologies. 2016d). Unreal Enginessä versionhallintapalvelimelle lähetetään ainoastaan projektikansion Binaries-, Config-, Content-, Source-kansiot ja projektikansion juuressa olevat tiedostot, muita kansioita ei lähetetä (Epic Games. 2016e). Ylimääräisten tiedostojen lähettämistä versionhallintapalvelimelle ei ole syytä tehdä, koska palvelimen kiintolevyt täyttyvät turhaan. Unity ja Unreal Engine toimivat kummatkin erinomaisesti Subversion-versionhallinnalla.

7 YHTEENVETO

Työssä kehitettiin kaksi peliä ja vertailtiin kehityksessä käytettyjen pelimoottoreiden kehitysominaisuuksia. Kummatkin pelimoottorit soveltuvat kaksiulotteisen pelin kehitykseen. Unity-pelinkehityksessä vaadittiin ohjelmointia pelin toimintojen tekemiseen ja siten se soveltuu lähinnä ohjelmointia osaaville henkilöille pelinkehitysalustaksi. Unreal Enginen blueprinttien ansiosta yksinkertaisten kaksiulotteisten pelien kehityksessä käytettävien operaatioiden tekeminen oli helppoa pienen perehtymisen jälkeen.

Tutkittuani muiden pelinkehittäjien kokemuksia eroavaisuuksista Unityn ja Unreal Enginen ominaisuuksista sain selville, että monet pitävät Unityä helpommin lähestyttävänä pelinkehitysalustana. Unityn ohjelmointiominaisuuksia kehuttiin paremmiksi sekä dokumentaatioiden ja kysymyspalstan laajuutta kehuttiin. Unreal Enginellä kehitetyt pelit on toisaalta kevyempiä ja siten Unreal Engine soveltuu paremmin komeilla grafiikoilla varustettujen pelien kehitykseen. (Quora. 2016)

Itse koen Unity-pelimoottorin kuitenkin miellyttävämmäksi kehitysalustaksi, sillä kyseisellä pelimoottorilla monimutkaisten ominaisuuksien tekeminen on yksinkertaisempaa ohjelmointiin tottuneelle ja käyttöliittymä on yksinkertaisempi. Yksinkertaisuudella uskon olevan merkitystä kehityksessä tulleiden virheiden etsimisen kannalta. Uskon, että tulen keskittymään tulevaisuudessa mieluummin Unity-pelinkehitykseen kuin Unreal Engine -pelinkehitykseen.

LÄHTEET

Bard, N. 2015. Unity Comes to Linux: Experimental Build Now Available. Unity Technoogies. Viitattu 28.4.2016, http://blogs.unity3d.com/2015/08/26/unity-comes-to-linux-experimental-build-now-available/

Epic Games 2016a. What is Unreal Engine 4?. Viitattu 29.4.2016, https://www.unrealengine.com/what-is-unreal-engine-4

Epic Games 2016b. Rendering and Graphics. Unreal Engine 4 Documentation. Viitattu 9.5.2016, https://docs.unrealengine.com/latest/INT/Engine/Rendering/

Epic Games 2016c. Physics Simulation. Unreal Engine 4 Documentation. Viitattu 9.5.2016, https://docs.unrealengine.com/latest/INT/Engine/Physics/

Epic Games, 2016d. Advanced iOS PC/Mac Workflow. Unreal Engine 4 Documentation. Viitattu 28.4.2016, https://docs.unrealengine.com/latest/INT/Platforms/iOS/GettingStarted/

Epic Games 2016e. Using SVN as Source Control. Unreal Engine 4 Documentation. Viitattu 15.5.2016, https://docs.unrealengine.com/latest/INT/Engine/Basics/SourceControl/SVN/

Hamid, J. Installing Unreal Engine in Linux Mint/Ubuntu. Community. Linux Mark Institute. Viitattu 28.4.2016, https://community.linuxmint.com/tutorial/view/1898

IGN. Mega Man Preview. Game Highlights. Viitattu 2.5.2016, http://www.ign.com/games/mega-man/nes-6000

IGN 2010. History of the Unreal Engine. Viitattu 28.4.2016, http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

Ohjelmointiputka 2007. Mikä ihmeen pelimoottori? Keskustelu. Viitattu 9.5.2016, http://www.ohjelmointiputka.net/keskustelu/15667-mik%C3%A4-ihmeen-pelimoottori/sivu-1

Overmars, M, 2012. Utrecht University. A Brief History of Computer Games. Viitattu 2.5.2016, http://www.cs.uu.nl/docs/vakken/b2go/literature/history_of_games.pdf

Quora. What is the main difference between 2D and 3D game development? Viitattu 9.5.2016, https://www.quora.com/What-is-the-main-difference-between-2D-and-3D-game-development

Quora 2016. What are the main pros and cons of Unity 3D and Unreal Engine?. Viitattu 28.4.2016, https://www.quora.com/What-are-the-main-pros-and-cons-of-Unity-3D-and-Unreal-Engine

Seraphina 2013. Wordpress. History of the Unity Engine [Freerunner 3D Animation Project]. Viitattu 9.5.2016, https://seraphinacorazza.wordpress.com/2013/02/14/history-of-the-unity-engine-freerunner-3d-animation-project/

Unity Technologies 2009. Unity Technologies Launches Version 2.6 of Its Platform and Makes Unity Freely Available. Viitattu 9.5.2016, http://web.archive.org/web/20130308073717/http:/unity3d.com/company/news/unity2.6-press.html

Unity Tecnologies 2016a. Multiplatform. Viitattu 28.4.2016, http://unity3d.com/unity/multiplatform

Unity Technologies, 2016b. Getting Started with iOS Development. Unity Manual. Viitattu 28.4.2016, http://docs.unity3d.com/Manual/iphone-GettingStarted.html

Unity Technologies 2016c. Get Unity. Viitattu 29.4.2016, https://unity3d.com/get-unity

Unity Technologies 2016d. Using External Version Control Systems with Unity. Unity Manual. Viitattu 15.5.2016, http://docs.unity3d.com/Manual/ExternalVersionControlSystemSupport.html