2D-pelin suunnittelu ja teko Unityllä

Mika Kulmala

Opinnäytetyö
Huhtikuu 2015
Tietotekniikan koulutusohjelma Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Mika Kulmala
2D-pelin suunnittelu ja teko Unityllä

Opinnäytetyö 36 sivua, joista liitteitä 8 sivua
Huhtikuu 2015

Työssä perehdytään 2D-pelin suunnitteluun ja kehitystyökaluihin, joita on hyvä käyttää 2D-peliä tehdessä.

Ensin perehdytään Unity ja Project Libre ohjelmistoihin, joita käytetään pelin teossa. Näistä ohjelmistoista on lyhyet kuvaukset ja selitykset, kuinka niitä käytetään sekä kuvat käyttöliittymistä.

Ohjelmistojen jälkeen perehdytään pelin suunnitteluun, joka sisältää mm. pelityypit, pelialusta ja projektisuunnitelma.

Pelisuunnittelun jälkeen perehdytään työssä pelin ohjelmointiin ja miten se voidaan toteuttaa peliä varten.

Ennen lopullista yhteenvetoa on pieni peliprojekti, jossa hyödynnetään tässä työssä esitettyjä tietoja.

Asiasanat: Unity, suunnittelu, pelaaminen

ABSTRACT

Tampere University of Applied Sciences
Degree programme in ICT Engineering
Software Engineering

Mika Kulmala:
Designing 2D-Game and Developing It with Unity

Bachelor's thesis 36 pages, appendices 8 pages
April 2015

This work goes through game designing, game development and tools, which are good to use when implementing a game.

First there are Unity and Project Libre programs, which are used in creating a game. There is a description and explanation how they are used and pictures of their user interfaces.

After the programs is game designing, which includes game types, platforms and project designing.

After the game designing is programming and how it could be implemented for the 2D-game.

Before the conclusion is a small game project, which uses the information in this thesis.

Key words: Unity, designing, development, gaming

SISÄLLYS

1 JOHDANTO 6
2 Kehitystyökalut 7
2.1 Unity 7
2.2 Project Libre 8
3 Pelisuunnittelu 10
3.1 Pelityypit, kohdeyleisö ja ulkoasu 10
3.2 Pelialusta ja ohjaus 10
3.3 Projektisuunnitelma 11
4 Pelin toteutus 12
4.1 Ohjelmointi 12
4.1.1 Ohjaus 13
4.1.2 Graafinen käyttöliittymä 15
4.2 Kenttäsuunnittelu 16
4.3 Spritet ja niiden käyttö 17
5 Peliprojektin toteutus 21
5.1 Pelaajan ohjelmointi 21
5.2 Kaivettavien palikoiden ohjelmointi 23
6 Yhteenveto 25
LÄHTEET 26
LIITTEET 28
Liite 1. Pelaajan fysiikan hallinta. 28
Liite 2. Pelaajan ohjaus 31
Liite 3. Palikka 32

LYHENTEET JA TERMIT

Frame	Kehys, joka on osa videota. Video koostuu useammasta kehyksestä, yleensä elokuvassa sekunti videota sisältää 24 kehystä.
GUI	Graafinen käyttöliittymä
Anime	Japanilainen animaatio
Tekstuuri	Kuva, jota käytetään kolmiulotteisten monikulmion tahojen kuvioimiseen
Mac OS X	Applen kehittämä käyttöjärjestelmä Macintosh-tietokoneisiin.
Linux	Käyttöjärjestelmä tietokoneille.
Android	Googlen kehittämä käyttöjärjestelmä puhelimille ja mobiililaitteille
Tizen	Linux-pohjainen mobiilikäyttöjärjestelmä
Windows Phone 8	Microsoftin kehittämä mobiilikäyttöjärjestelmä
Playstation 3	Sonyn pelikonsoli
Playstation 4	Sonyn pelikonsoli
Playstation Vita	Sonyn mobiilipelikonsoli
Xbox One	Microsoftin kehittämä pelikonsoli
Xbox 360	Microsoftin kehittämä pelikonsoli
Wii U	Nintendon kehittämä pelikonsoli

1 JOHDANTO

Tässä työssä perehdytään eri työvaiheisiin, jotka kuuluvat pelinkehitykseen. Luvussa 2 käydään läpi kehitystyökalut, joita tässä työssä käytetään. Luvussa 3 käydään tarkemmin läpi pelisuunnittelua, mm pelityypit, pelialusta ja projektisuunnitelma. Luvussa 4 käydään läpi pelin toteutusta, jossa käydään läpi ohjelmointi, kenttäsuunnittelu ja spritet. Luvussa 5 käydään läpi pienen peliprojektin toteutus, jossa hyödynnetään tässä työssä esitettyjä asioita.

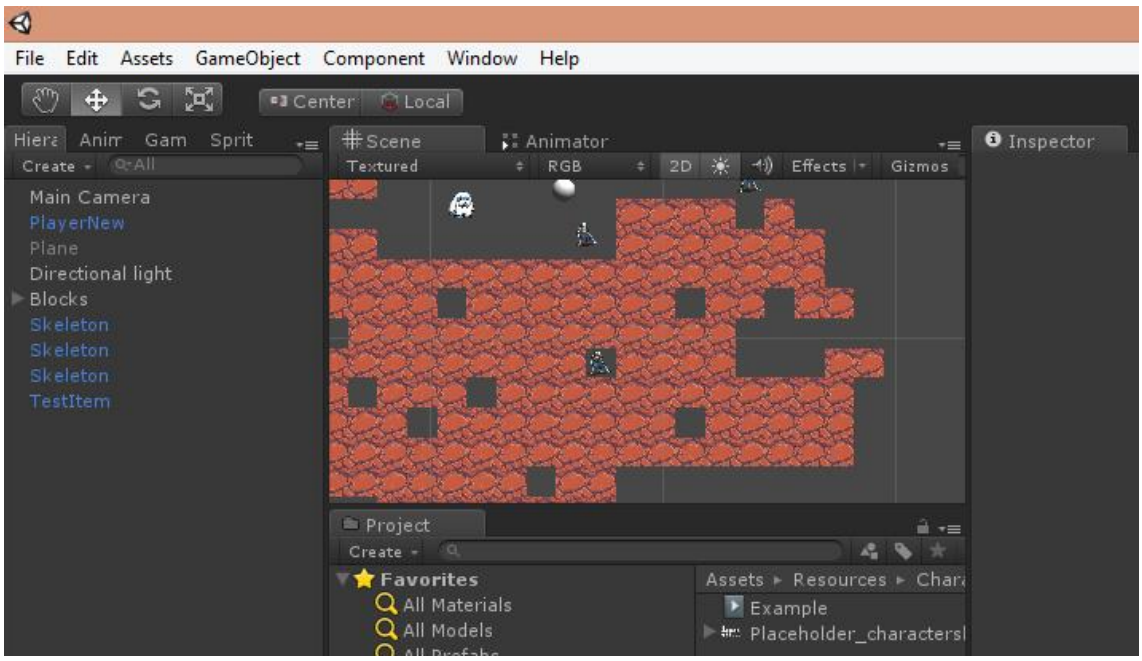
Yksi kaikkein vaikeimmista osista pelinteossa on suunnittelu. Voidaan sanoa että pieniä pelejä varten ei tarvitse tehdä suunnitelmia, vaan pitää keskittyä vain itse projektiin. Tämä on kuitenkin niin kaukana totuudesta kuin vaan voi olla (Gameindustry 2015). Suunnittelua pidetään monesti kiusana, mutta sitä voidaan sanoa välttämättömäksi kiusaksi, yleisesti ne projektit jotka on suunniteltu onnistuvat huomattavasti useammin, kuin ne joita ei ole suunniteltu.

2 Kehitystyökalut

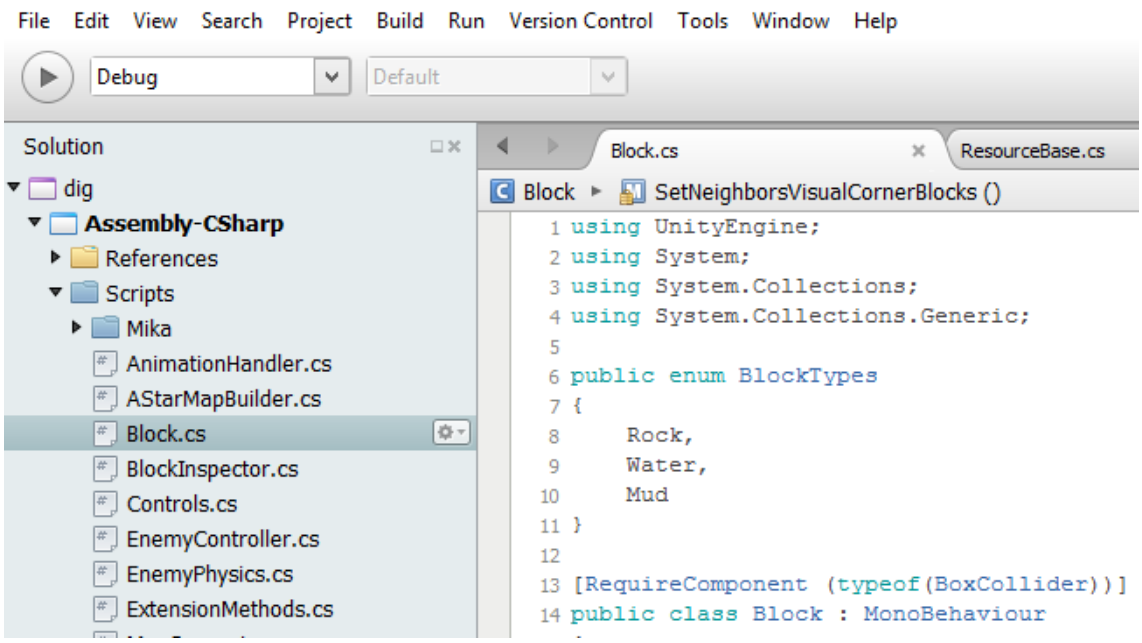
2.1 Unity

Unity Technologiesin kehittämä Unity on pelimoottori ja ohjelmointiympäristö. Unityn ensimmäinen versio julkaistiin 1.6.2005 ja viimeisin versio 5.0 julkaistiin 3.3.2015. Unity on yksi käytetyimmistä pelienkehitys ympäristöistä.
Unityn suureksi eduksi voidaan katsoa se, että sen avulla pelejä voidaan kehittää Windows, Mac OS X, Linux – alustoille, Android, Tizen, Windows Phone 8 – puhelinkäyttöliittymille, Playstation 3, Playstation 4, Playstation Vita, Xbox One,
Xbox 360 ja Wii U - pelikonsoleille

Pelin ohjelmointi Unityllä toimii käyttämällä drag&drop-työkaluja, joilla voidaan määritellä pelin osien riippuvuudet toisiinsa. Unityn käyttöliittymä koostuu: yläpalkissa olevista valikoista, joissa voidaan mm. vaihtaa työstettyjen kenttien välilä, luoda uusia kenttiä, lisätä komponentteja, luoda uusia komponentteja ja avata uusia ikkunoita Unityyn. Kehittäjälle tärkeimmät ikkunat ovat Scene, Game, Hierarchy, Project ja Inspector -ikkunat. Kuvassa 1. on Unityn käyttöliittymä ja siitä nähdään miten se koostuu mm. Scene, Inspector ja Hierarchy ikkunoista. Scene-ikkunasta nähdään pelin raaka versio ja sinne voidaan raahata erilaisia objekteja mm. valoja, kameroita, kiviä ja pelattavia hahmoja. Scene-ikkunassa tehdään käytännössä kaikki kenttäsuunnitteluun liittyvät asiat: raahataan peliin liittyvät asiat paikoilleen ja katsotaan eri kuvakulmista että peli näyttää halutun mukaiselta. Game-ikkunassa nähdään miltä peli näyttää, tätä ikkunaa käytetään pääasiassa pelin testaukseen. Hierarchy ikkunasta nähdään kenttään raahatut esineet ja oliot. Project-ikkunasta nähdään käytännössä kansiorakenne missä on mm. äänet, kuvat, videoklipit ja esineet ovat. Kuvassa 2. on Unityn Monodevelop-ohjelmointiympäristö, jonka vasemmassa reunassa on lista tehdyistä ohjelmakoodeista ja oikealla valitun ohjelmakoodin koodi, sekä mahdollisuus muokata sitä.



Kuva 1. Unity



Kuva 2. MonoDevelop ohjelmointiympäristö

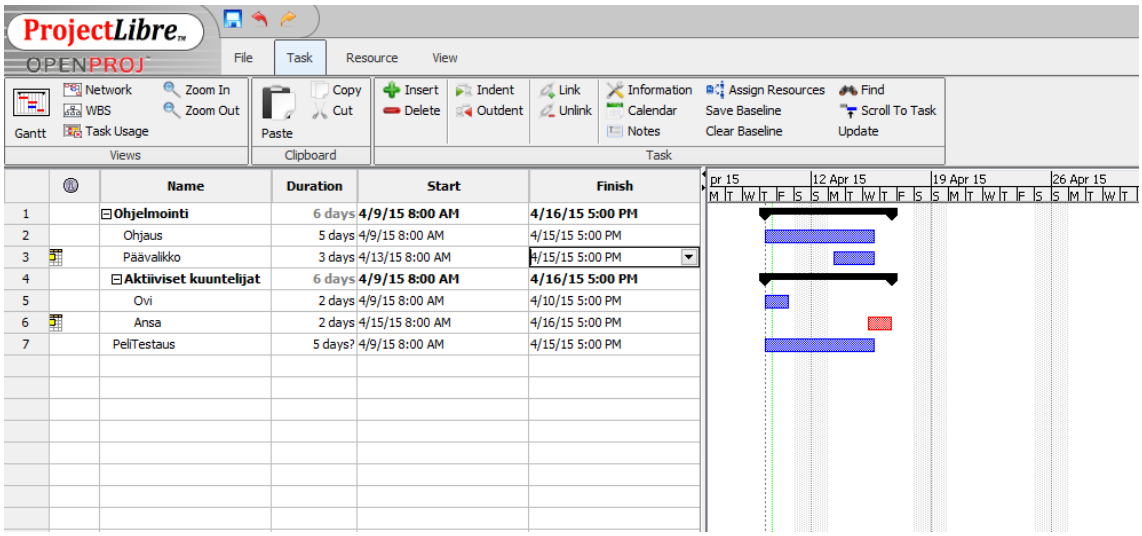
2.2 Project Libre

Project Libre on avoimeen lähdekoodiin perustuva projektinhallintaohjelma. Project Libre on tehty korvaamaan Microsoft Project ja tämän vuoksi sen käyttöliittymä on myös hyvin samankaltainen Microsoft Projectin kanssa. Project Libre on täysin yhteensopiva Microsoft Project 2010 kanssa.

Aina kun projekti on suunniteltu, on tärkeää että on projektisuunnitelma ja vaikka sen tekemiseen menee aikaa, niin hyvä projektisuunnitelma tulee säästämään todella paljon aikaa ja yleensä rahaa (Projectsmart 2014).

Peliprojektin tekeminen kannattaa aina aloittaa hyvällä suunnitelmalla. Kannattaa kerätä ainakin seuraavat asiat ylös: Mitä toimintoja pelissä on, hajoittaa toiminnot paloihin, josta näkee mistä osista ne koostuvat, määritellä tehtävät oikeille henkilöille, määritellä toimintojen välillä olevat riippuvuudet, aikataulu ja tarkistuspisteet. (GameProject)

Kuvassa 3. on tehty Project Librellä Gantt-kaavio. Kaavioon on määritelty projektin kesto, sekä jokaisen projektin osan kesto, aloitus ja lopetus.



Kuva 3. Project Libre käyttöliittymä.

3 Pelisuunnittelu

Pelisuunnittelu on tärkeä osa pelin onnistumisessa, työssä on aina hyvä tietää mitä pitää tehdä ja miten se kannattaa tehdä, ennen kuin aloitat sen tekemisen (WhyGameDesign). Ilman hyvää pohjatyötä peli ei luultavasti koskaan valmistu tai pelinteko vie liian paljon aikaa, joka vaikuttaa kustannuksiin ja pelin kysyntä on saattanut laskea kannattamattomaksi. Hyvällä suunnittelulla siis minimoidaan kustannukset, saadaan päämäärä pelille, jotta tiedetään yleensä aina mikä on seuraava etappi peliä tehtäessä. Pelisuunnittelussa on tärkeää ottaa huomioon ainakin nämä asiat: kohdeyleisö, pelialusta, ulkoasu ja pelityyppi. Pelisuunnittelussa on hyvä käyttää hyväksi mm. projektinhallintatyökaluja.

3.1 Pelityypit, kohdeyleisö ja ulkoasu

Pelejä on monenlaisia ja tyypit voivat olla esimerkiksi seuraavanlaisia tai niiden yhdistelmiä: Toimintapeli, seikkailupeli, ajopeli, simulaatiopeli, strategiapeli tai roolipeli.

Pelityypin valinta vaikuttaa mm. pelin kysyntään ja mihin maihin sitä kannattaa erityisesti markkinoida. Etelä-Koreassa pelataan paljon ja siellä erityisen suosittuja pelityyppejä ovat roolipelit ja strategiapelit (Gaming addiction 2015). Mikäli haluat, että peli kehitetään Etelä-Korealaisille, luonnollinen valinta on siis toinen näistä pelityypeistä.

Pelityypin ja kohdeyleisön valinnan jälkeen täytyy miettiä pelin ulkoasua. Ulkoasu vaikuttaa suuresti pelin myyntiin ja huonokin peli voi myydä hyvin kun se näyttää hyvältä. Ulkoasua valittaessa pitää taas muistaa kohdeyleisö, eli pitävätkö he esim. realistisen näköisestä grafiikasta, animesta vai länsimaalaisesta animaatiosta.

3.2 Pelialusta ja ohjaus

Pelialusta vaikuttaa pelin ohjaukseen paljon, esim. tabletilla ja puhelimella ei ole fyysisiä näppäimiä, kun taas pelikonsoleilla on niihin tarkoituksella tehdyt ohjaimet.

Pelialustat eroavat toisistaan myös tehossa, sillä eri alustoilla on erilaiset prosessorit, eri määrä muistia ja fyysisten näyttöjen koko voi erota erittäin paljon, esim. matkapuhelimen näyttö voi olla kokoa 4 tuumaa, kun taas pelikonsoleilla pelataan paljon suuremmilta näytöiltä.

Pelialustaa valittaessa on hyvä huomioida minkälaista peliä on tekemässä. Esimerkiksi Applen matkapuhelimien virallinen sovelluskauppa App Store kieltää mm. uskontoja pilkkaavan materiaalin heidän kaupassaan, joten mikäli haluat tehdä sellaisen pelin iPhonelle, et voi myydä sitä App Storessa (AppStore).

Unity pystyy kääntämään pelin useille eri alustoille, mutta se ei ole aivan suoraviivaista, esimerkiksi juuri ohjauksessa pitää osata ottaa huomioon se, että tabletissa ei ole näppäimiä, vaan sille pitää määrittää ohjaus liikkeillä ja kosketuksilla näppäimien sijaan. Mikäli aiot tehdä saman pelin usealle alustalle, kannattaa peli suunnitella siten että se toimii kaikissa laitteessa. Tässä on hyvä huomioida suorituskyvyltään kaikkein heikoin, sekä se laite jossa on vähiten ohjausmahdollisuuksia.

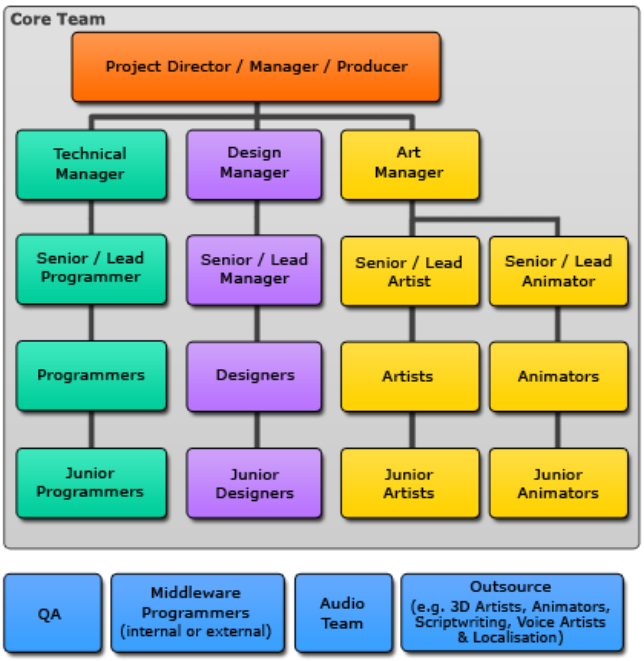
3.3 Projektisuunnitelma

Projektisuunnitelma on tärkeä osa pelintekemistä, on tärkeää dokumentoida selkeästi mitä asioita peliin tulee ja kuinka kauan aikaa on varattu mihinkin osa-alueeseen. Hyvässä projektisuunnitelmassa on käyty ainakin seuraavat asiat läpi: budjetti, resurssit, riskit, aikataulu, laatu, tehtävät ja vastuualueet (GoodProjectPlan).

Projektia tehdessä asiat tulevat varmasti muuttumaan alkuperäisestä suunnitelmasta, esim. ohjaus voi tuntua kömpelöltä, pelin ulkoasu ei tunnu sopivan pelin tarinaan tms. Aina kun päätetään tehdä jokin asia erilailla kuin alkuperäisessä suunnitelmassa on määritelty, on tärkeää päivittää projektisuunnitelma, jotta se pysyy ajantasalla.

4 Pelin toteutus

Pelin toteutukseen tarvitaan yleensä seuraavia asioita: suunnitelma, laitteisto, ohjelmisto ja tiimi joka toteuttaa pelin. Tiimin on hyvä koostua projektipäälliköstä, ohjelmoijista, suunnittelijoista, graafikoista ja animaattoreista. Yksi henkilö voi olla vastuussa useammasta alueesta ja työtä voi usein myös ulkoistaa. Kuvassa 4. on esimerkki minkälainen pelin kehitystiimi voi olla. Tiimiä johtaa tuottaja, joka vastaa koko pelin teosta. Tuottajan suoria alaisia ovat, tekninen johtaja, suunnittelu johtaja ja taidepuolen johtaja.



Kuva 4. Game Development Team (Blitzgamestudios).

4.1 Ohjelmointi

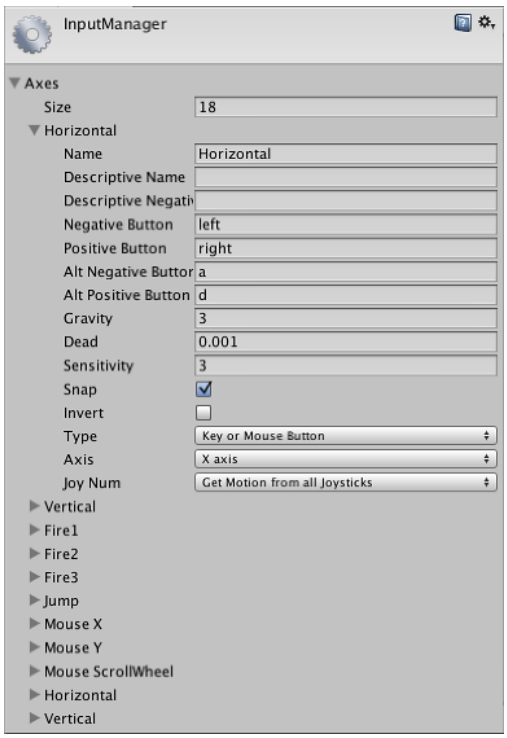
Peliä varten tarvitsee yleensä ohjelmoida seuraavat asiat: graafinen käyttöliittymä, ohjaus ja aktiivisia kuuntelijoita hyödyntävät asiat kuten ovet ja ansat. Peliohjelmointi on miellyttävää sillä yleensä esim. kuuntelijat eivät ole riippuvaisia ohjauksesta tai käyttöliittymästä jne. joten mikäli jokin scripti tarvitsee kirjoittaa uusiksi tai poistaa kokonaan, se ei tarkoita että kaikkia muitakin scriptejä tarvitsee kirjoittaa uusiksi.

Ohjelmointi kannattaa yleensä aloittaa liikkumisesta, jotta pelihahmoa voidaan liikutella eri puolilla kenttää ja testata sen avulla helposti mm. aktiivisia kuuntelijoita hyödyntäviä asioita.

4.1.1 Ohjaus

Ohjaus on tärkeä osa peliä, hyvällä ohjausmekaniikalla saadaan pelaamisesta miellyttävää ja haastavaa. Hyvän ohjauksen tunnistaa mm. siitä, että sitä on mukava ja helppo käyttää, se on tarkka eli käyttäytyy aina samalla tavalla. Ohjaus vaikuttaa suuresti pelikokemukseen, miltä sinusta tuntuu kun ohjaat peliä, miten keskustelet aktiivisten kuuntelijoiden kanssa ja ovatko ensimmäiset minuutit pelissä hauskoja vai eivät (Digitaltutors).

Unityssä ohjauksen tekeminen on helppoa ja esimerkiksi liikkumisen tekeminen onnistuu nopeasti. Unityssä on myös Input Manager, jonka avulla voidaan määritellä mm. ohjaukseen vaikuttavat napit. Kuvassa 5. on Input Manager, johon on määritelty mm. akselit, niiden napit, vaihtoehtoiset napit ja herkkyys.



Kuva 5. Input Manager (Unity)

Kuvassa 6. näkyy kuinka liikkuminen onnistuu todella yksinkertaisesti. Tämä scripti asetetaan peliobjektille, jonka halutaan liikkuvan ja tämän jälkeen peliobjekti liikkuu aina,

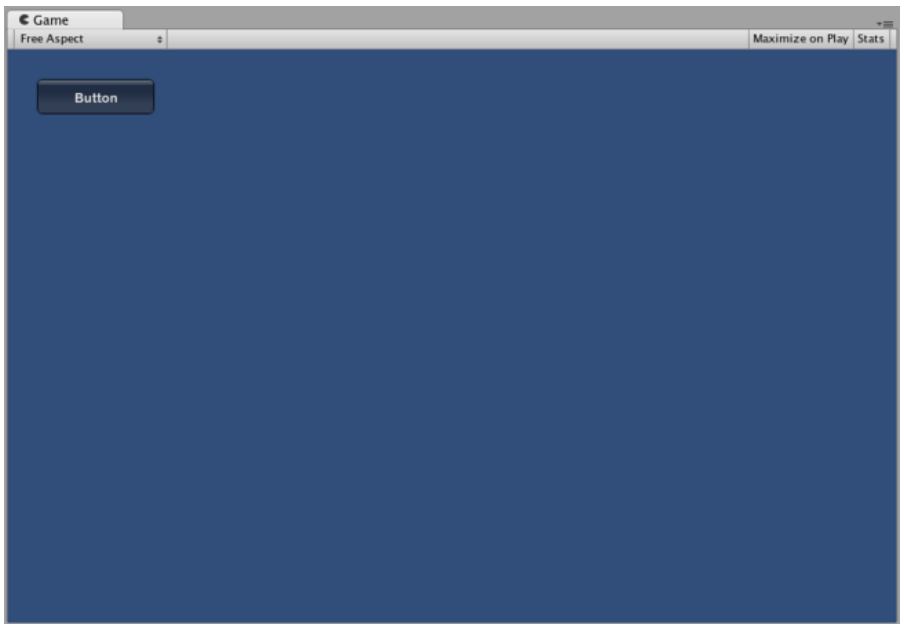
kun painetaan A- tai D-nappia. Ohjaus voidaan myös tehdä näytöllä olevilla napeilla, kuten kuvan 7. ohjelmakoodi luo kuvan 8. kaltaisen napin.

```
public class Liikkuminen : MonoBehaviour {
    public float nopeus = 1f
    void Update()
    {
    // Jos painetaan nappia D liikutaan oikealle nopeudella nopeus * ajan muutos
        if(Input.GetKey(KeyCode.D))
            transform.position += new Vector2(speed * Time.deltaTime, 0.0f);
        if(Input.GetKey(KeyCode.A))
            transform.position -= new Vector2(speed * Time.deltaTime, 0.0f);
    }

    }
```

Kuva 6. Liikkuminen vasemmalle ja oikealle

```
public Pelaaja pelaaja;

 void OnGUI () {
   if (GUI.Button (new Rect (25, 25, 100, 30), "Button")) {
       if (pelaaja)
       pelaaja.transform.position += new Vector2(speed * Time.deltaTime, 0.0f)
   }
}
```

Kuva 7. Napin scripti



Kuva 8. nappi jolla voi liikuttaa pelaajaa (Unity)

Lista osasta komennoista joita voidaan käyttää käyttää mm. pelaajan liikuttamiseen:

GUI.Button	- Palauttaa totta, kun näytöllä olevaa nappia on painettu.
GUI.RepeatButton	- Palauttaa totta niin kauan kun näytöllä oleva nappi on pohjassa
Input.GetKey	- Palauttaa totta niin kauan kun nappi on pohjassa
Input.GetKeyUp	- Palauttaa totta kun nappi palautuu ylös
Input.GetKeyDown	- Palauttaa totta kun nappi menee pohjaan
Input.GetButton	- Palauttaa totta kun Input managerissa määriteltyä olevaa nappia on painettu
Input.GetMouseButton	- Palauttaa totta kun hiiren nappia on painettu
Input.GetTouch	- Palauttaa totta kun kosketetaan

4.1.2 Graafinen käyttöliittymä

Hyvän käyttöliittymän tehtävä on tarjota tärkeää tietoa selkeästi ja nopeasti, jonka jälkeen poistua käyttäjän tieltä kun tieto on nähty (Gamedevelopment). Graafisessa käyttöliittymässä on asiat mitkä yleensä näkyvät pelaajalle koko ajan, sekä erilaiset valikot. Pelaajan energia, pisteet ja nimi voivat esimerkiksi näkyä ruudulla koko ajan, kun taas esimerkiksi, lopeta, tallennalla tai jatka pelaamista sisältä valikko voidaan asettaa pelaajalle näkyviin aina, kun hän painaa valittua nappia.

Graafinen käyttöliittymä kannattaa suunnittella siten että tärkeät tiedot ovat helposti nähtävissä, sopivan kokoisia ja sopivassa paikassa, jotta ne eivät haittaa pelaamista. Pelaajan energia voi olla tärkeä tieto pelaajalle, mutta se voi häiritä pelaajan pelaamista jos se täyttää puolet peliruudusta. Graafista käyttöliittymää kannattaa siis suunnitella ja testailla erilaisia variaatioita ennen kuin valitsee mikä on sopiva.

Unityssä graafisen käyttöliittymän teko on tehty helpoksi, voidaan esimerkiksi vain luoda tyhjä peliobjekti nimeltä pelaajaEnergia, joka asetetaan kameran lapsi objektiksi, eli tämän objektin paikka muuttuu aina kameran mukana. Tälle objektille tehdään scripti nimeltä pelaajaEnergia jossa se hakee aina pelaajan peliobjektista pelaajan nykyisen energian ja maksimi energian. Näitä tietoja käyttäen scripti pitää pelaajaEnergian sisällä olevan tekstin muodossa ”pelaajanNykyinenEnergia / pelaajanMaxEnergia”, eli esimerkiksi 5 / 10.

```
public class pelaajaEnergia: MonoBehaviour
{
    public Pelaaja pelaaja1;
    void OnGUI()
    {
        //Tekstin väri on musta
        GUI.color = Color.Black

        if(pelaaja != null)
        GUI.Label(new Rect (80,550,260,90), pelaaja1.nykEnergia.ToString()) +
        " / " + pelaaja1.maxEnergia.ToString());
    }
}
```

Kuva 9. Pelaajan energia

OnGUI:ta kutsutaan automaattisesti monta kertaa framen aikana, jolloin tieto käyttöliittymässä pysyy lähes koko ajan ajantasalla. GUI.Label luo graafisen käyttöliittymäelementin, jonka muoto on suorakulmio ja paikka näytöllä 80 pikseliä oikealle ruudun vasemmasta reunasta ja 550 pikseliä alaspäin ruudun yläkulmasta.

Esimerkki GUIelementtejä, joita voi luoda Unityssä.

GUILabel	-	Luo teksti- tai tekstuurilapun ruudulle
GUIButton	-	Luo napin jolle voi määrittää tekstin, tekstuurin ja funktion
GUIDragwindow	-	Luo GUI elementin jota voi liikuttaa

4.2 Kenttäsuunnittelu

Kenttäsuunnittelu on tärkeä apu tarinan luomiseen ja saamaan pelaajat koukuttumaan peliin. Huonolla kenttäsuunnittelulla voidaan pilata koko peli, vaikka peli on muuten erittäin näyttävä ja hyvä (Digitaltutors).
Kentät opettavat pelaajan pelaamaan ja nauttimaan pelistä. Pelin edetessä kentissä tulee vastaan yleensä pieniä muutoksia ja pulmia, joista selviämällä voidaan käyttää pelihahmon taitoja mitä ei ole aikaisemmin käytetty tai kentässä tarvitsee työntää laatikko painokytkimelle, jotta pelaaja voi edetä.

Jotta peli on miellyttävä, täytyy pelin vaikeusaste tehdä sopivaksi. Tässä on tärkeää muistaa minkä ikäisille peli on suunnattu. Liian helppoa peliä eivät pelaajat jaksa pelata pelaamisen takia, vaan tarinan ja ulkoasun vuoksi. Mikäli peli taas on liian vaikea, eivät

pelaajat pääse pelissä eteenpäin ja jonkin ajan jälkeen lopettavat pelin pelaamisen. On siis tärkeää löytää hyvä tasapaino pelin vaikeusasteeseen.

Pelaajalle on tärkeää tietää minkä takia hän on tietyssä kentässä. Tarinallisessa pelissä jos pelaaja esim. matkaa Amazonian viidakkoon, voi siihen sopivia teemoja olla lentokenttä ja sademetsä.

Kentän teossa on hyvä pitää mielessä teema, vaikeusaste ja uusien asioiden opetus. Kenttää tehtäessä on kannattavaa tehdä kenttä ensin pelattavaksi, sillä se helpottaa kentän testausta. Pelattavaan kenttään on jälkikäteen helppo lisätä erilaisia asioita jotka vaikuttavat peliin, kuten ansoja.

Ensimmäisessä kentässä pelaajalle on hyvä opettaa pelin perusmekaniikat ja vaikeusaste on hyvä pitää matalalla, jotta pelaaja tottuu pelin ohjaukseen ja ymmärtää miten peliä pelataan. Esimerkiksi Super Mario Bros pelissä ensimmäisessä kentässä käydään hyvin läpi pelin perusmekaniikat, liikkuminen, hyppääminen, vihollisten tuhoaminen, kuoleminen, kolikoiden ja pelaajaa voimistavien esineiden toiminta.

Pelin edetessä on hyvä opettaa lisää uusia asioita mitä pelissä voidaan tehdä. Tällä tavoin peli ei tunnu toistavan itseään.

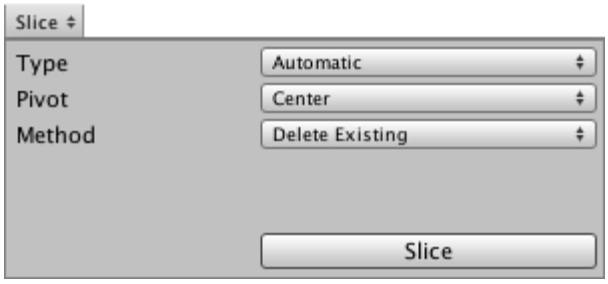
4.3 Spritet ja niiden käyttö

Sprite on bittikarttagrafiikkaa, joka on suunniteltu olemaan osa suurempaa kokonaisuutta. Se voi olla staattinen tai animoitu kuva (Digitaltutors). Kuvassa 10. on esimerkki sprite sheet pelin Spelunky hahmosta. Spritejä käytetään siksi, että niiden avulla voidaan säästää muistia, jolloin tilaa säästyy mm. efekteille ja musiikille. Muistin säästö tapahtuu siten että monta spriteä laitetaan samaan tekstuuriin.

Kuva 10. Sprite sheet

Kuvssa 11. on Unityn oma sprite editor, jolla voidaan leikata tekstuurista spritet. Mikäli spritet ovat samankokoisia, voidaan käyttää sprite editorin automatic slicing toimintoa, joka leikkaa tekstuurit automaattisesti kuten kuvassa 12.

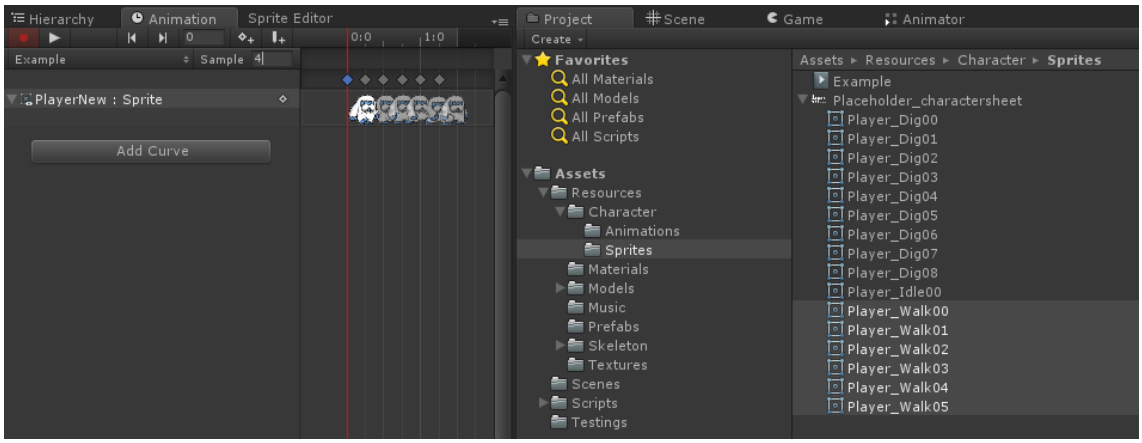
Kuva 11. Sprite Editor Unityssä



Kuva 12. Automatic slicing sprite editorissa (Unity)

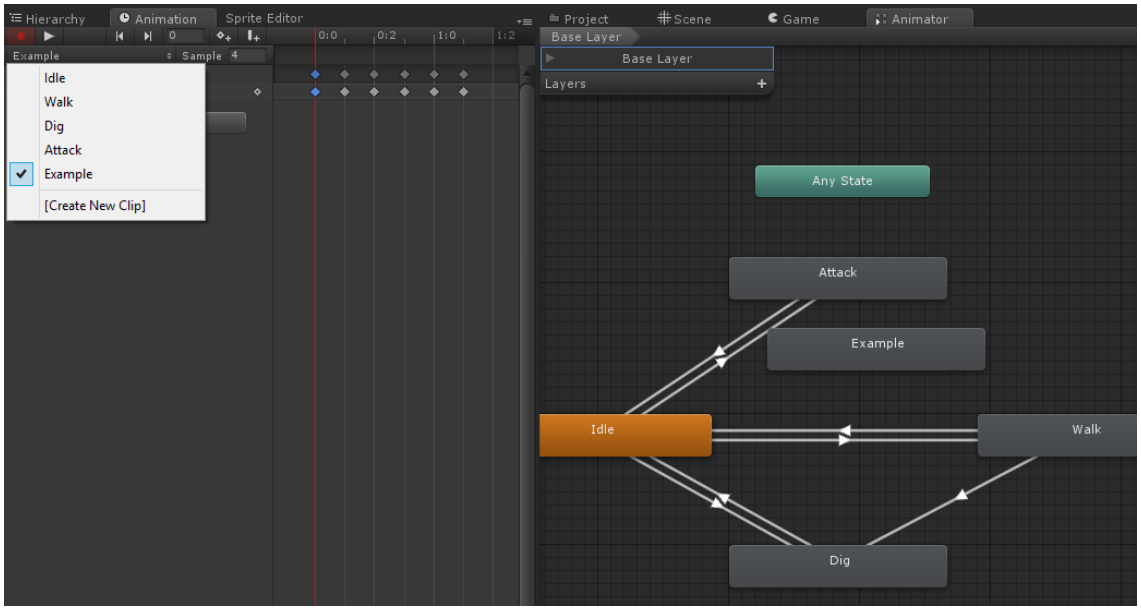
Sprite editorissa leikattujen spritejen avulla voidaan luoda helposti animaatioita unityyn. Kuvassa 13. luodaan animaatio siten, että avataan Animation ikkuna Unityssä, painetaan’Create new clip’, jonka jälkeen raahataan sprite editorilla luomasi halutut

spritet animaatioon. Animation ruudussa voidaan asettaa kuvien vaihtumisvälin, eli kuinka usein kuva vaihtuu sekunnin aikana. Tämän jälkeen voidaan mennä unityn Animator ikkunaan, jossa nähdään luodut animaatiot, ja voidaan määritellä missä tilanteissa animaatiot vaihtuvat.



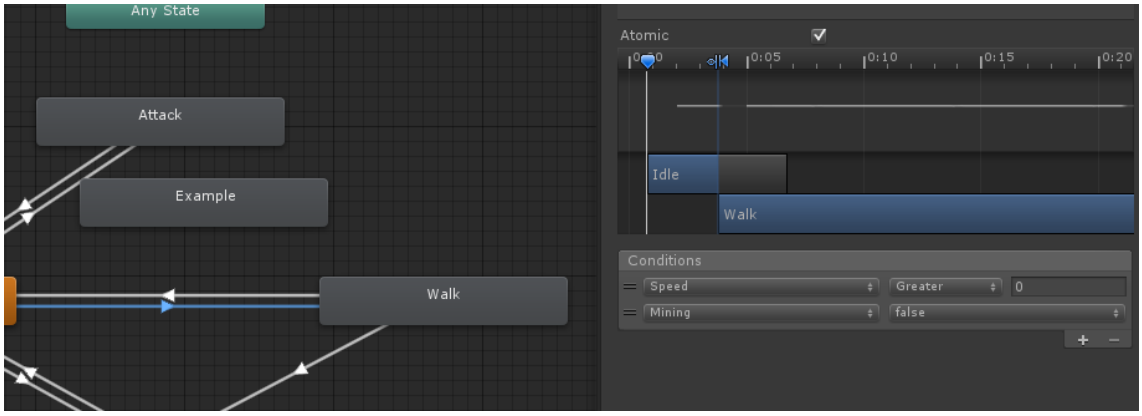
Kuva 13. Animaation luonti Unityssä.

Kuvassa 14. nähdään, että keltainen idle on hahmon perusanimaatio, eli animaatio jota hahmo alkaa käyttämään pelin alussa. Idleen on linkitetty Attack, Walk ja Dig Animaatiot. Animaatiot vaihtuvat niille määriteltyjen ehtojen avulla esim. Idle vaihtuu Walk animaatioksi kun pelaaja liikkuu.



Kuva 14. Unityn Animator näkymä

Kuvassa 15. näkee kuinka Idle muuttuu kävely animaatioksi, kun nopeus on suurempi kuin 0 ja kun Mining on epätosi.



Kuva 15. Idle animaation vaihtuminen Walk animaatioksi

Animaatio pitää määritellä myös scripteissä, eli Speed pitää linkittää muuttujaan, jotta animaation vaihtuminen toimii oikein.

Kuvasta 16. nähdään kuinka Mining on linkitetty playerPhysics.mining muuttujaan, ja animation controller tarkastelee tätä ja muita siihen linkitettyjä muuttujia, jotta se tietää tarvitseeko animaatioita muuttaa.

```
anim.SetFloat("Speed", Mathf.Abs(Input.GetAxis("Horizontal")));
anim.SetBool ("Mining", playerPhysics.mining);
anim.SetBool ("Attacking", fighting);
```

Kuva 16. Animaatioiden linkitys ohjelmakoodissa.

5 Peliprojektin toteutus

Käyttäen edellä esiteltyjä asioita pelin toteutuksesta, tehtiin pienimuotoinen 2D-peli, jossa pelaajan tehtävänä oli kaivaa mineraaleja maasta, samalla varoen erilaisia vaaroja.

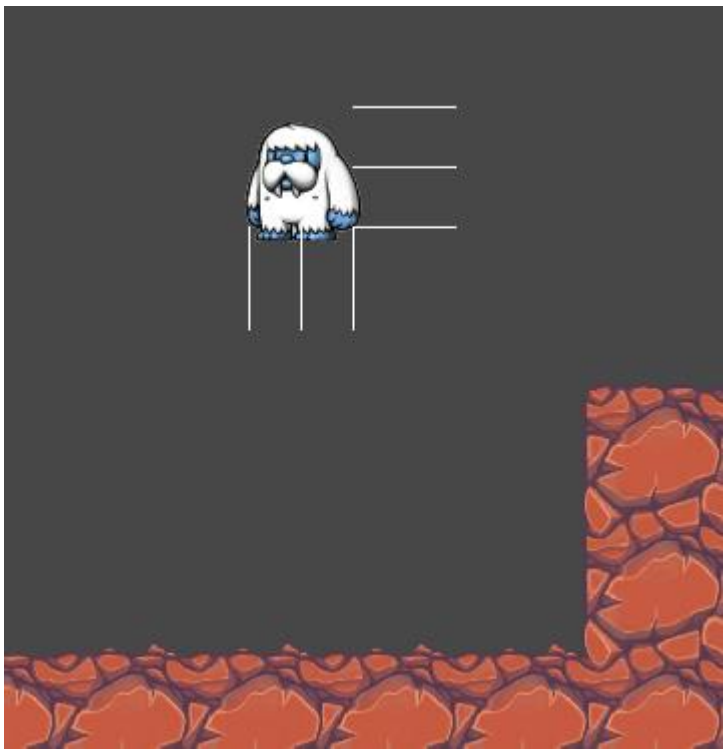
Aluksi tehtiin alustava suunnitelma, jossa määriteltiin mitä peliin halutaan tulevan. Peliin haluttiin kaivettavia maapaloja, maapalojen alta ilmestyviä aarteita ja liikutettava pelihahmo. Peliin tarvitsemat grafiikat saatiin ystävältäni Emma Valkilta ja käyttämällä spritesheettiä Spelunkystä (Spelunky). Kuvassa 17 on valmis peliprojekti.

Kuva 17. Peliprojekti

5.1 Pelaajan ohjelmointi

Pelaajan haluttiin pystyvän liikkumaan, hyppäämään ja kaivamaan. Pelaajaa varten ohjelmoitiin aluksi erittäin yksinkertaiset tavat liikkua, siten ettei se välittänyt ollenkaan vastaantulevista esteistä tai painovoimasta. Yksinkertaisen liikkumisen jälkeen kehitettiin liikkumista paremmaksi, siten että luotiin pelaajaan vaikuttava painovoima, hyppääminen ja törmäyksen tunnistus.

Pelaaja tunnistaa törmäyksen lähettämällä säteitä siihen suuntaan, mihin hän liikkuu. Säteeseen osuessa objektiin jonka läpi pelaaja ei voi mennä, peli pysäyttää pelaajan.

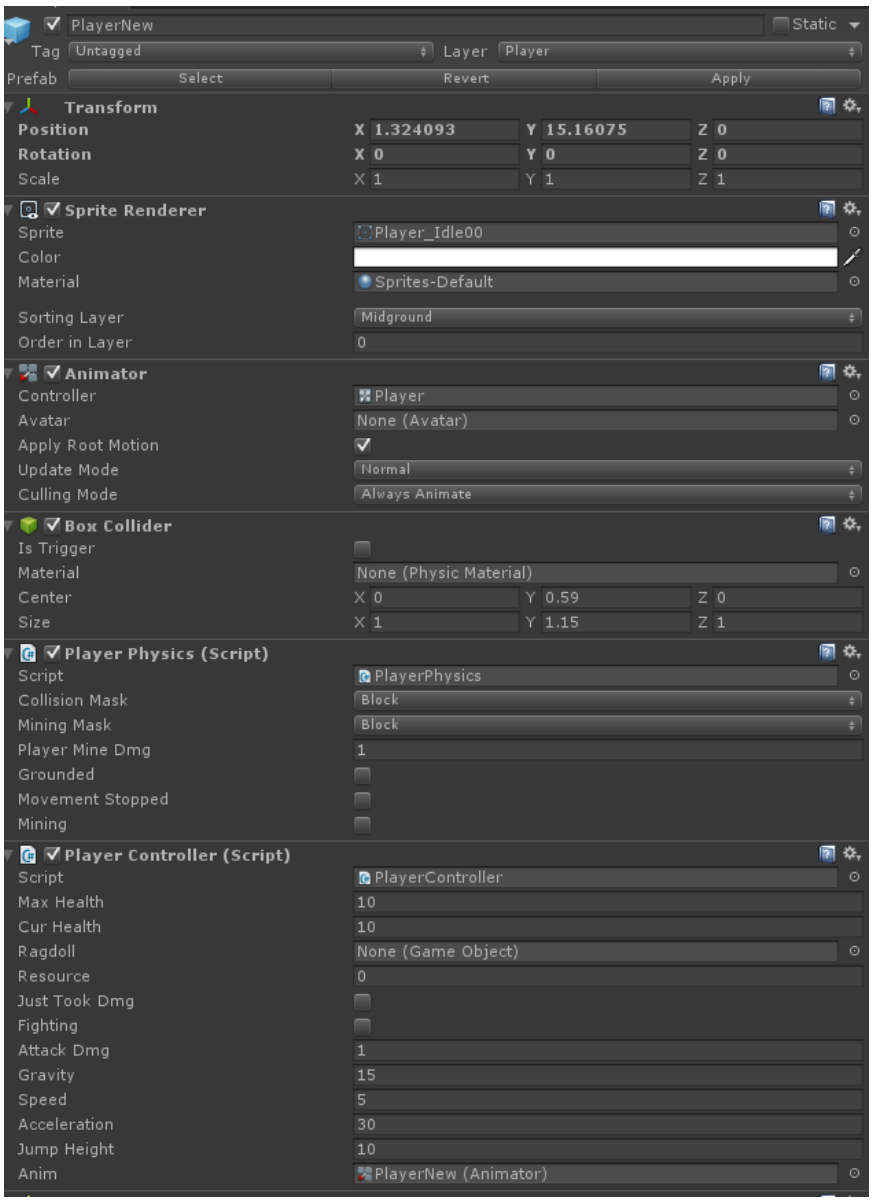


Kuva 18. Pelaaja lähettää tunnistussäteitä.

```
// tarkistetaan törmäykset ylä- ja alapuolelle
grounded = false;
for (int i = 0; i<collisionDivisionX; i ++) {
    float dir = Mathf.Sign(deltaY);
    float x = (p.x + c.x - s.x/2) + s.x/(collisionDivisionX-1) * i;
    float y = p.y + c.y + s.y/2 * dir;

    ray = new Ray(new Vector2(x,y), new Vector2(0,dir));

    //piirreellään viivat osumisen tarkkailua varten
    Debug.DrawRay(ray.origin,ray.direction);

    if (Physics.Raycast(ray,out hit,Mathf.Abs(deltaY) + skin,collisionMask)) {

        float dst = Vector3.Distance (ray.origin, hit.point);


        if (dst > skin) {
            deltaY = dst * dir - skin * dir;
        }
        else {
            deltaY = 0;
        }

        grounded = true;


        break;

    }
}
```

Kuva 19. Ohjelmakoodi pelaajan törmäyksen tunnistukseen ylä- ja alapuolella

Kuvassa 20. on pelaajan sisältämät komponentit, kuten animator, player physics ja player controller. Peliä pelatessa Unityssä voidaan seurata esim. player physicsiä ja tarkastella miten pelaajan arvot muuttuvat liikkuessa.

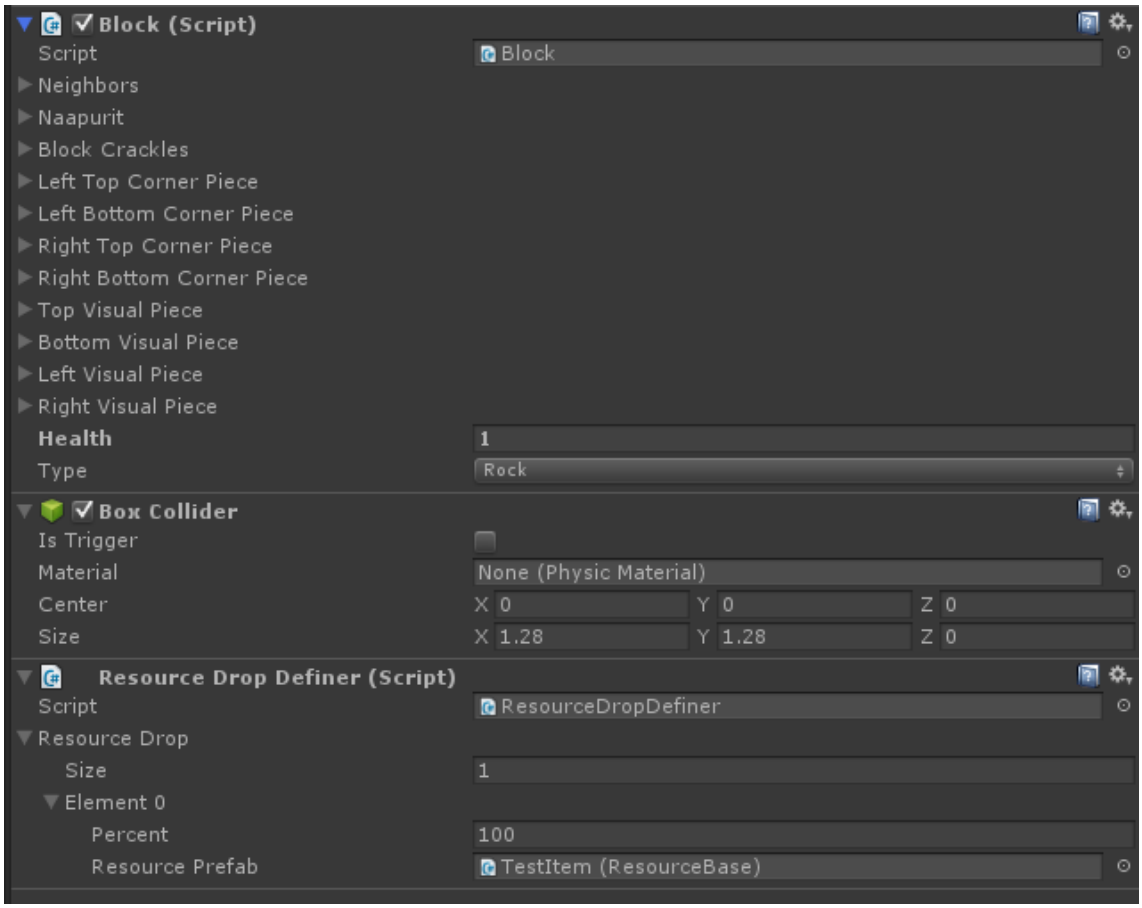


Kuva 20. Pelaajan sisällä olevat komponentit

5.2 Kaivettavien palikoiden ohjelmointi

Liikkumisen toteutuksen ohjelmoitiin pelaajalle mahdollisuus kaivaa palikoita. Palikoiden kaivaminen toimii siten että, pelaaja lähettää säteen siihen suuntaan kun hän yrittää kaivaa, ja mikäli säde osuu objektiin joka sisältää Block-komponentin pelaaja alkaa kaivamaan. Kaivuu toimii siten, että pelaaja soittaa kaivuu animaation ja animaation lopuksi pelaaja vähentää Block-komponentin omaavalta objektilta yhden energiapisteen. Silloin kun objektin energia vähenee nollaan, se käy läpi tuhoutumisfunktion, jossa mm. tarkastellaan jättääkö objekti aarteen jälkeensä ja lopuksi objekti tuhotaan.

Kuvassa 21. on palikan sisältämät komponentit, joissa mm. määritellään palikan energia, missä kohtaa se pysäyttää pelaajan sekä mitä tavaroita se voi luoda juuri ennen tuhoutumistaan.



Kuva 21. Block-komponentin omaava objekti.

```
//Use this to take dmg, inside this you should also check how it effects the blocks sprite.
public void TakeDmg(int x)
{
    health -= x;


    //If there are crackles
    if(BlockCrackles.Count > 0)
    if(StartHealth - WhenToChangeCrackle * CurrentCrackle >= health)
    {
        Debug.Log ("Pääsi ekaan kohtaan sisään ja vertailuarvot : " + (StartHealth - WhenToChangeCrackle * CurrentCrackle
        int index = CurrentCrackle-1;
        Debug.Log ("Index: " + index + " currentCrackle : " + CurrentCrackle);

        if(index >= 0 && index < BlockCrackles.Count)
            BlockCrackles[index].SetActive(true);

        index --;
        int indexHelper = index+1;
        if(index >= 0 && index <= BlockCrackles.Count && index+1 < BlockCrackles.Count)
            BlockCrackles[index].SetActive ( false);

        CurrentCrackle++;
    }

    if(health <= 0)
        Destroy ();
}
```

Kuva 22. Ohjelmakoodi kuinka Block käsittelee kaivuun.

6 Yhteenveto

Yksinkertaisen pelin tekeminen nykyaikaisilla työkaluilla on tehty erittäin helpoksi. Pelin hyvä suunnittelminen edesauttaa pelin tekoa huomattavasti sillä lähes kaikki asiat mitä pelin valmistamiseen tarvitaan, on ennalta määritelty ja suunniteltu. Hyvä suunnittelu tekee pelin tekemisestä järjestelmällistä, joka vähentää huomattavasti hetkiä jolloin ei tiedä mitä seuraavaksi pitää tehdä ja siten tehostaa pelin tekoprosessia.

Unityn käyttö 2D-pelin teossa on helppo omaksua ja miellyttävä käyttää. Unityyn löytyy paljon lähdemateriaalia, joiden avulla erilaisten pelien tekeminen onnistuu jouheasti. Unityllä on erittäin hyvät lähdesivut, josta on helppo löytää esimerkkejä ja tietoa asioiden käytöstä. Unityllä on myös hyvät keskustelupalsat, jossa kehittäjät keskustelevat keskenään pelien kehityksestä. Keskustelupalstoilla on helppo pyytää neuvoa ongelmatilanteisiin ja etsiä vastauksia ongelmiin joita tulee vastaan pelejä kehittäessä.

Unityn käyttö peliprojektissa onnistuu hyvin ja ongelmatilanteiden kohdalla vastaus ongelmiin löytyy nopeasti. Peliin on helppo kehittää uusia asioita ja niiden lisääminen itse peliin on tehty helpoksi ja sellaiseksi, ettei se yleensä vaadi aikaisempien asioiden muuttamista.

LÄHTEET

GamingAddiction.South Korea Cracks Down on Gaming Addiction. Luettu 15.3.2015
http://content.time.com/time/world/article/0,8599,1983234,00.html

StarcraftKorea. Why is Starcraft so popular in Korea. Luettu 15.3.2015
http://kotaku.com/5595262/why-is-starcraft-so-popular-in-korea

AppStore. App Store Review Guideline. Luettu 24.2.2015
https://developer.apple.com/app-store/review/guidelines/

IADT. Targeting Your Audience: 3 Important Tips for Designing Games. Luettu 20.3.2015
http://www.iadt.edu/student-life/iadt-buzz/may-2013/tips-for-designing-games-audience

Gamasutra. What exactly goes into porting a video game? Luettu 26.3.2015
http://www.gamasutra.com/view/news/222363/What_exactly_goes_into_porting_a_video_game_BlitWorks_explains.php

Wikihow. How to Design a Video Game. Luettu 2.2.2015
http://www.wikihow.com/Design-a-Video-Game

Susan M. Weinschenk 2011. 100 Things Every Designer Needs to Know about People. Berkeley: New Riders.

ProjectSmart. The Key Elements of a Good Project Plan. Luettu 20.2.2015
http://www.projectsmart.com/articles/the-key-elements-of-a-good-project-plan.php

Unity3D. Unity, Kuva napista. Luettu 2.3.2015
http://docs.unity3d.com/Manual/gui-Controls.html

Swingswingsubmarine, Spritesheet. Luettu 20.3.2015
http://www.swingswingsubmarine.com/2010/11/25/seasons-after-fall-spritesheet-animation/

Spritesheet pelistä Spelunky. Luettu 2.2.2015
http://www.spriters-resource.com/pc_computer/spelunky/sheet/56967/

Statista. US computer and video game sales by categorie.
http://www.statista.com/statistics/190184/us-computer-and-video-game-sales-by-categorie-2010/

Forbes. Summing Up Mobile Gaming In 2013 With A Single Word... Freemium. Ewan Spence. Luettu 1.4.2015
http://www.forbes.com/sites/ewanspence/2013/12/28/summig-up-mobile-gaming-in-2013-with-a-single-word-freemium/

BusinessInsider. 10 Ways Modern Technology Has Changed Video Games Forever In Just Five Years. Luettu 10.3.2015
http://www.businessinsider.com/10-ways-modern-technology-has-changed-video-games-forever-in-just-five-years-2011-11?op=1&IR=T

Digitaltutors. What Makes A Great Game The Key Elements Of Successful Games. Luettu 13.3.2015
http://blog.digitaltutors.com/what-makes-a-great-game-the-key-elements-of-successful-games/

Statista. Most Played PC Games. Luettu 17.3.2015
http://www.statista.com/statistics/251222/most-played-pc-games/

Insights. The Anatomy Of A Succesful Mobile Game. Luettu 10.3.2015
http://insights.wired.com/profiles/blogs/the-anatomy-of-a-successful-mobile-game?xg_source=msg_appr_blogpost

CIO. 9 Secrets to Project Management Success. Luettu 5.3.2015
http://www.cio.com/article/2599794/project-management/9-secrets-to-project-management-success.html

Gameindustry. How To Plan A Game Development Project. Luettu 1.3.2015
http://gameindustry.about.com/od/game-development/a/How-To-Plan-A-Game-Development-Project.htm

GameIndustry. Tips For Designing Games Audience. Luettu 10.4.2015
http://www.iadt.edu/student-life/iadt-buzz/may-2013/tips-for-designing-games-audience

MakeGames. Finishing a game. Luettu 10.4.2015
http://makegames.tumblr.com/post/1136623767/finishing-a-game

WhyDesign. Why Game Design Is Important. Luettu 10.4.2015
http://www.intelligent-artifice.com/2007/10/why-game-design-is-important.html

GameEngine. Choosing The Perfect Game Engine. Luettu 28.3.2015
http://www.gamasutra.com/blogs/MikoCharbonneau/20130222/187185/Choosing_the_Perfect_Game_Engine.php

Blitzgamestudios. Game Team Structure. Luettu 1.4.2015
http://www.blitzgamesstudios.com/media/gfx/pages/blitz-academy/game-dev/game_team_structure.png

Digitaltutors. Character Controls Camera 3cs Game Development. Luettu 22.3.2015
http://blog.digitaltutors.com/character-controls-camera-3cs-game-development/

Digitaltutors. Keeping Players Engaged Tips Great Game Level Design. Luettu 22.3.2015
http://blog.digitaltutors.com/keeping-players-engaged-tips-great-game-level-design/

GameDevelopment. Game UI By Example a Crash Course In The Good And The Bad. Luettu 15.3.2015
http://gamedevelopment.tutsplus.com/tutorials/game-ui-by-example-a-crash-course-in-the-good-and-the-bad--gamedev-3943

LIITTEET

Liite 1. Pelaajan fysiikan hallinta.

```
using UnityEngine;
using System.Collections;


[RequireComponent (typeof(BoxCollider))]
public class PlayerPhysics : MonoBehaviour {

    public LayerMask collisionMask;
    public LayerMask MiningMask;
    private BoxCollider collider;
    private Vector3 s;
    private Vector3 c;

    private Vector3 originalSize;
    private Vector3 originalCentre;
    private float colliderScale;

    private int collisionDivisionX = 3;
    private int collisionDivisionY = 3;
    private float skin = .005f;
    public int PlayerMineDmg = 1;

//  [HideInInspector]
    public bool grounded;
    public bool movementStopped;
    public bool mining = false;


    Ray ray;
    RaycastHit hit;

    void Start() {
        collider = GetComponent<BoxCollider>();
        colliderScale = transform.localScale.x;

        originalSize = collider.size;
        originalCentre = collider.center;
        SetCollider (originalSize, originalCentre);

    }

    public void Move(Vector2 moveAmount) {
        if(mining) return;
        float deltaY = moveAmount.y;
        float deltaX = moveAmount.x;
        Vector2 p = transform.position;

        // tarkistetaan törmäykset ylä- ja alapuolelle
        grounded = false;
        for (int i = 0; i<collisionDivisionX; i ++) {
            float dir = Mathf.Sign(deltaY);
            float x = (p.x + c.x - s.x/2) + s.x/(collisionDivisionX-1) * i;
            float y = p.y + c.y + s.y/2 * dir;

            ray = new Ray(new Vector2(x,y), new Vector2(0,dir));

            //piirreellään viivat osumisen tarkkailua varten
            Debug.DrawRay(ray.origin,ray.direction);

            if (Physics.Raycast(ray,out hit,Mathf.Abs(deltaY) + skin,collisionMask)) {

                float dst = Vector3.Distance (ray.origin, hit.point);


                if (dst > skin) {
                    deltaY = dst * dir - skin * dir;
                }
```

```
                }
                else {
                    deltaY = 0;
                }

                grounded = true;


                break;

            }
        }


        //Tarkistetaan  törmäykset sivuille.
        movementStopped = false;
        for (int i = 0; i<collisionDivisionY; i ++) {
            float dir = Mathf.Sign(deltaX);
            float x = p.x + c.x + s.x/2 * dir;
            float y = p.y + c.y -s.y/2 + s.y/(collisionDivisionY-1) * i;
            ray = new Ray(new Vector2(x,y), new Vector2(dir,0));

            //piirreellään viivat osumisen tarkkailua varten
            Debug.DrawRay(ray.origin,ray.direction);

            if (Physics.Raycast(ray,out hit,Mathf.Abs(deltaX) + skin,collisionMask)) {

                float dst = Vector3.Distance (ray.origin, hit.point);


                if (dst > skin) {
                    deltaX = dst * dir - skin * dir;
                }
                else {
                    deltaX = 0;
                }

                movementStopped = true;

                break;

            }
        }

        if (!grounded && !movementStopped) {
                        Vector3 playerDir = new Vector3 (deltaX, deltaY);
                        Vector3 o = new Vector3 (p.x + c.x + s.x / 2 * Mathf.Sign (deltaX), p.y + c.y + s.y / 2 * Mathf.Sign (deltaY));
                        ray = new Ray (o, playerDir.normalized);

                        if (Physics.Raycast (ray, Mathf.Sqrt (deltaX * deltaX + deltaY * deltaY), collisionMask)) {
                                grounded = true;
                                deltaY = 0;
                        }
                }

        Vector2 finalTransform = new Vector2(deltaX,deltaY);

        transform.Translate(finalTransform);
    }


    public void SetCollider(Vector3 size, Vector3 centre){
        collider.size = size;
        collider.center = centre;

        s = size * colliderScale;
        c = centre * colliderScale;
    }

    //Mining things below
    public IEnumerator MineDown ()
    {
        RaycastHit hit;



        Debug.Log("Kaivaa");
        if(mining || !Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, -1f, 0 ), out hit, 1f,MiningMask.value )) yield break;
        mining = true;
        yield return new WaitForSeconds(1f);
        if (Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, -1f, 0 ), out hit, 1f ))
        {
//          Debug.DrawLine ( transform.position, hit.point , Color.red);
//          Debug.Break ();
//          Debug.Log (hit.collider.name);
//          Debug.Log ("Kaivaa kohta 2");
            Block block = hit.transform.GetComponent<Block> ();
            if ( block != null )
            {
//              --block.health;
                block.TakeDmg(PlayerMineDmg);
                yield return new WaitForSeconds(0.05f);
            }

        }
        mining = false;
        yield return null;
     }

     public IEnumerator MineLeft()
     {
        RaycastHit hit;

        Debug.Log("Kaivaa");
        if(mining || !Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( -1, 0, 0 ), out hit, 1f,MiningMask.value )
           || !Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, -1f, 0 ), out hit, 1f,MiningMask.value )) yield break;
        mining = true;
        yield return new WaitForSeconds(1f);
        if ( Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( -1, 0, 0 ), out hit, 1f ))
        {
//          Debug.DrawLine ( transform.position + new Vector3(0,0.5f,0), hit.point , Color.red);
//          Debug.Break ();
//          Debug.Log (hit.collider.name);
//          Debug.Log ("Kaivaa kohta vasen,");
            Block block = hit.transform.GetComponent<Block> ();
            if ( block != null )
            {
                //              Debug.Log (block.health + " ennen vähennystä");
                block.TakeDmg(PlayerMineDmg);
                yield return new WaitForSeconds(0.05f);
            }

        }
        mining = false;
        yield return null;
    }
```

```
    }

    public IEnumerator MineRight()
    {
        RaycastHit hit;

        Debug.Log("Kaivaa");
        if(mining ||  !Physics.Raycast ( transform.position  + new Vector3(0,0.5f,0) , new Vector3 ( 1, 0, 0 ), out hit, 1f,MiningMask.value )
           || !Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, -1f, 0 ), out hit, 1f,MiningMask.value )) yield break;
        mining = true;
        yield return new WaitForSeconds(1f);
        if ( Physics.Raycast ( transform.position  + new Vector3(0,0.5f,0) , new Vector3 ( 1, 0, 0 ), out hit, 1f ))
        {
//          Debug.DrawLine ( transform.position + new Vector3(0,0.5f,0), hit.point , Color.red);
//          Debug.Break ();
//          Debug.Log (hit.collider.name);
//          Debug.Log ("Kaivaa kohta oikea");
            Block block = hit.transform.GetComponent<Block> ();
            if ( block != null )
            {
                //              Debug.Log (block.health + " ennen vähennystä");
                block.TakeDmg(PlayerMineDmg);
                yield return new WaitForSeconds(0.05f);
            }

        }
        mining = false;
        yield return null;
    }

    public IEnumerator MineUp()
    {
        RaycastHit hit;

        Debug.Log("Kaivaa");
        if( mining || !Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, 2f, 0 ), out hit, 1f,MiningMask.value)
           || !Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, -1f, 0 ), out hit, 1f,MiningMask.value ))
        {
            yield break;
//          Debug.DrawLine ( transform.position + new Vector3(0,0.5f,0), hit.point , Color.red);
//          Debug.Break();
        }
        mining = true;
        yield return new WaitForSeconds(1f);
        if ( Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, 2f, 0 ), out hit, 1f ))
        {
//          Debug.DrawLine ( transform.position + new Vector3(0,0.5f,0), hit.point , Color.red);
//          Debug.Break ();
//          Debug.Log (hit.collider.name);
//          Debug.Log ("Kaivaa kohta ylös");
            Block block = hit.transform.GetComponent<Block> ();
            if ( block != null )
            {
                //              Debug.Log (block.health + " ennen vähennystä");
                block.TakeDmg(PlayerMineDmg);
                yield return new WaitForSeconds(0.05f);
            }

        }
        mining = false;
        yield return null;
    }


}
```

Liite 2. Pelaajan ohjaus

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(PlayerPhysics))]

public class PlayerController : Entity {

    public float gravity = 15;
    public float speed = 5;
    public float acceleration = 30;
    public float jumpHeight = 5;


    private float currentSpeed;
    private float targetSpeed;
    private Vector2 amountToMove;
    private PlayerPhysics playerPhysics;

    public Animator anim;
    private bool facingRight = false;

    void Start () {
        playerPhysics = GetComponent <PlayerPhysics> ();
        anim = gameObject.GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update () {

        if(!playerPhysics.mining)
        Flip();

        anim.SetFloat("Speed", Mathf.Abs(Input.GetAxis("Horizontal")));
        anim.SetBool ("Mining", playerPhysics.mining);
        anim.SetBool ("Attacking", fighting);

        Camera.main.transform.position = new Vector3 ( transform.position.x, transform.position.y, Camera.main.transform.position.z );

        if (playerPhysics.movementStopped)
        {
            targetSpeed = 0;
            currentSpeed = 0;
        }


        targetSpeed = Input.GetAxisRaw ("Horizontal") * speed;
        currentSpeed = IncrementTowards(currentSpeed, targetSpeed, acceleration);

        if (playerPhysics.grounded) {
            amountToMove.y = 0;

                if (Input.GetButtonDown ("Jump")) {
                    amountToMove.y = jumpHeight;
                }
            if(Input.GetKey (KeyCode.M))
            {
                if(Input.GetAxis("Vertical") < 0) {
                    StartCoroutine(playerPhysics.MineDown());
                }
                if(Input.GetAxis("Horizontal") < 0 && playerPhysics.movementStopped) {
                    StartCoroutine(playerPhysics.MineLeft());
                }
                if(Input.GetAxis("Horizontal") > 0 && playerPhysics.movementStopped) {
                    StartCoroutine(playerPhysics.MineRight());
                }
                RaycastHit hit;
                if(Input.GetAxis("Vertical") > 0 && Physics.Raycast ( transform.position + new Vector3(0,0.5f,0), new Vector3 ( 0, -1f, 0 ), out hit, 0.5f )){
                    StartCoroutine(playerPhysics.MineUp());
                }
            }
        }

        //Doublejump
        else if(Input.GetButtonDown("Jump") && playerPhysics.movementStopped)
        {
            amountToMove.y = jumpHeight;
        }

        amountToMove.x = currentSpeed;
        amountToMove.y -= gravity * Time.deltaTime;


        if(!playerPhysics.mining || Input.GetKey (KeyCode.M))
        playerPhysics.Move(amountToMove * Time.deltaTime);
    }



    private float  IncrementTowards(float n, float target, float a)
    {
        if (n == target) {
            return n;
        }
        else {
            float dir = Mathf.Sign(target - n);
            n +=a * Time.deltaTime * dir;
            return (dir == Mathf.Sign(target-n))?n: target;
        }
    }

    private void Flip()
    {
        //gets left
        if(!facingRight && Input.GetAxis("Horizontal") < 0)
        {
            facingRight = !facingRight;

            Vector3 thescale = transform.localScale;
            thescale.x *= -1;
            transform.localScale = thescale;
        }

        if(facingRight && Input.GetAxis("Horizontal") > 0)
        {
            facingRight = !facingRight;

            Vector3 thescale = transform.localScale;
            thescale.x *= -1;
            transform.localScale = thescale;
        }

    }

    public void AddResources(int ResourceAmount)
    {
        resource += ResourceAmount;
    }

```

Liite 3. Palikka

```
using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

public enum BlockTypes
{
    Rock,
    Water,
    Mud
}

[RequireComponent (typeof(BoxCollider))]
public class Block : MonoBehaviour
{

    public static List<Block> blocks;
    // x, 0, x
    // 3, x, 1
    // x, 2, x
    public Block[] neighbors = new Block[4];

    public List<GameObject> naapurit;

    public List<GameObject> BlockCrackles = new List<GameObject>();


    //These needs to be set to lists and needs to randomly get one piece from the list unless one is already activated
    public List<GameObject> LeftTopCornerPiece;
    public List<GameObject> LeftBottomCornerPiece;
    public List<GameObject> RightTopCornerPiece;
    public List<GameObject> RightBottomCornerPiece;

    public List<GameObject> TopVisualPiece;
    public List<GameObject> BottomVisualPiece;
    public List<GameObject> LeftVisualPiece;
    public List<GameObject> RightVisualPiece;

    public int health = 5;
    private int StartHealth = 5;
    public BlockTypes type = BlockTypes.Mud;
    private float WhenToChangeCrackle = 0f;
    private int CurrentCrackle = 1;
    private bool aboutToDie = false;


    void Awake ()
    {
        if(health <= BlockCrackles.Count)
        {
            health = BlockCrackles.Count+1;
        }

        if ( blocks == null )
            blocks = new List<Block> ();

        blocks.Add ( this );

        naapurit = new List<GameObject>(9);

        for(int x = 0;x<9;x++)
        {
            naapurit.Add(null);
        }
    }

    // Use this for initialization
    void Start ()
    {
        StartHealth = health;

        //calculates when to activate crackle gameobject.
        if(BlockCrackles.Count > 0)
            WhenToChangeCrackle = StartHealth / (BlockCrackles.Count+1);
        Debug.Log ("WhenToChange value: " + WhenToChangeCrackle);

```

```
        BoxCollider collider = GetComponent<BoxCollider>();
        collider.size = collider.size + new Vector3 ( 0, 0, 1 );

        GoTroughThisBlock();
    }



    //In here it destroys the block, instantiates destroy effects and checks neighbours and instantiates/activates/hides
    //Necessary block parts.
    public void Destroy()
    {
        aboutToDie = true;
//      Debug.Log ("Tarkistetaan naapurit");
        CheckNeighbors();

        SetVisualCornerBlocksOff();
//      GoTroughAllNeighbours();
        SetVisualSoftenersOnNeighbors();
        SetNeighborsVisualCornerBlocks();

//      Debug.Break();
//      SetVisualBlocks(joku palanen);
        if(this.GetComponent<ResourceDropDefiner>())
            SpawnResources();
        Destroy(this.gameObject);
    }


    //Use this to take dmg, inside this you should also check how it effects the blocks sprite.
    public void TakeDmg(int x)
    {
        health -= x;


        //If there are crackles
        if(BlockCrackles.Count > 0)
        if(StartHealth - WhenToChangeCrackle * CurrentCrackle >= health)
        {
            Debug.Log ("Pääsi ekaan kohtaan sisään ja vertailuarvot : " + (StartHealth - WhenToChangeCrackle * CurrentCrackle));
            int index = CurrentCrackle-1;
            Debug.Log ("Index: " + index + " currentCrackle : " + CurrentCrackle);

            if(index >= 0 && index < BlockCrackles.Count)
                BlockCrackles[index].SetActive(true);

            index --;
            int indexHelper = index+1;
            if(index >= 0 && index <= BlockCrackles.Count && index+1 < BlockCrackles.Count)
                BlockCrackles[index].SetActive ( false);

            CurrentCrackle++;
        }

        if(health <= 0)
            Destroy ();
    }


    //Check all the neighbours. and add them to neighbors list.
    public void CheckNeighbors()
    {
        Debug.Log ("Tarkistetaan naapurit sisalla");
        RaycastHit hit;
        Ray ray;
        int CurrentNeighbor = 0;
        //First rayposition to lefttop corner block
        Vector3 RayPosition = this.transform.position + new Vector3 (-1.28f,1.28f,-2f);
        for(int y = 0;y < 3;y++)
        {
```

```
            for(int x = 0;x < 3;x++)
            {
                if(Physics.Raycast(RayPosition, Vector3.forward, out hit, 2f))
                {
//                  Debug.Log ("Raycast osui");
//                  Debug.DrawLine ( RayPosition + new Vector3(0,0.01f), hit.point , Color.blue);
//                  Debug.Break();

                    //If it hits and it has a block component
                    if(hit.transform.GetComponent<Block>())
                        naapurit[CurrentNeighbor] = hit.transform.gameObject;
                    //if hits something but it isnt a block, add null
                    else naapurit[CurrentNeighbor] = null;
                    CurrentNeighbor++;
                    //Move one block to right.
                    RayPosition += new Vector3(1.28f,0,0);
                }
                //If doesnt hit anything add null to the list.
                else
                {
//                  Debug.Log ("Raycast ei osunut");
//                  Debug.DrawLine ( RayPosition + new Vector3(0,0,0.01f), hit.point , Color.red);
//                  Debug.Break();

//                  naapurit[CurrentNeighbor] = null;
                    CurrentNeighbor++;
                    RayPosition += new Vector3(1.28f,0,0);
                }
            }
            //Ray.position changed one line down and back to the start of row.
            RayPosition += new Vector3 (-3.84f,-1.28f,0);
        }

    }

    void SetNeighborsVisualCornerBlocks()
    {
        //Naapurijärjestys
        // 0 | 1 | 2
        // 3 | 4 | 5
        // 6 | 7 | 8

        //Activates CornerPieces
        //Jos naapuri 3 on block (Vasen keski)
        if(naapurit[3] != null && naapurit[3].GetComponent<Block>())
        {
//          jos naapuri 1 (keskimmäinen ylhäältä) on block niin aktivoidaan naapuri 3 yläkulmapala
            if(naapurit[1] != null && naapurit[1].GetComponent<Block>())
            {
                Block testi = naapurit[3].GetComponent<Block>();
                int random = GetRandomFrom(RightTopCornerPiece.Count);
                GameObject Kulma = testi.RightTopCornerPiece[random];
                Kulma.SetActive(true);
            }
            //Jos naapuri 7 on block (alakeski block)
            if(naapurit[7] != null && naapurit[7].GetComponent<Block>())
            {
                naapurit[3].GetComponent<Block>().RightBottomCornerPiece[GetRandomFrom(RightBottomCornerPiece.Count)].SetActive (true);
            }
        }
        //Jos naapuri 5 on block (oikea keski)
        if(naapurit [5] != null && naapurit[5].GetComponent<Block>())
        {
//          jos naapuri 1 (keskimmäinen ylhäältä) on block
            if(naapurit[1] != null && naapurit[1].GetComponent<Block>())
            {
                naapurit[5].GetComponent<Block>().LeftTopCornerPiece[GetRandomFrom(LeftTopCornerPiece.Count)].SetActive(true);
            }
            //Jos naapuri 7 on block (alakeski block)
            if(naapurit[7] != null && naapurit[7].GetComponent<Block>())
            {
                naapurit[5].GetComponent<Block>().LeftBottomCornerPiece[GetRandomFrom(LeftBottomCornerPiece.Count)].SetActive(true);
            }
        }
    }

```

```
    void SetNeighborsVisualCornerBlocksOnStart()
    {
        //Naapurijärjestys
        // 0 | 1 | 2
        // 3 | 4 | 5
        // 6 | 7 | 8

        //Activates CornerPieces

//      Blockin vasemmalle puolelle olevat palikat
        if(naapurit[3] == null)
        {
            if(naapurit[0] != null && naapurit[0].GetComponent<Block>())
                LeftTopCornerPiece[GetRandomFrom(LeftTopCornerPiece.Count)].SetActive (true);
            if(naapurit[6] != null &&naapurit[6].GetComponent<Block>())
                LeftBottomCornerPiece[GetRandomFrom(LeftBottomCornerPiece.Count)].SetActive(true);
        }
        if(naapurit[5] == null)
        {
            if(naapurit[2] != null && naapurit[2].GetComponent<Block>())
                RightTopCornerPiece[GetRandomFrom(RightTopCornerPiece.Count)].SetActive(true);
            if(naapurit[8] != null &&naapurit[8].GetComponent<Block>())
                RightBottomCornerPiece[GetRandomFrom(RightBottomCornerPiece.Count)].SetActive(true);
        }
    }

    void SetVisualCornerBlocksOff()
    {
        //Naapurijärjestys
        // 0 | 1 | 2
        // 3 | 4 | 5
        // 6 | 7 | 8

        // DeActivates CornerPieces
        if(naapurit[0] != null && naapurit[0].GetComponent<Block>())
        {
            //DeAktivoidaan oikealapala naapuri nollasta
            foreach(GameObject cornerpiece in naapurit[0].GetComponent<Block>().RightBottomCornerPiece)
            {
                cornerpiece.SetActive(false);
            }

        }

        if(naapurit[2] != null && naapurit[2].GetComponent<Block>())
        {
            foreach(GameObject cornerpiece in naapurit[2].GetComponent<Block>().LeftBottomCornerPiece)
            {
                cornerpiece.SetActive(false);
            }
        }

        if(naapurit[6] != null && naapurit[6].GetComponent<Block>())
        {
            foreach(GameObject cornerpiece in naapurit[6].GetComponent<Block>().RightTopCornerPiece)
            {
                cornerpiece.SetActive(false);
            }
        }

        if(naapurit[8] != null && naapurit[8].GetComponent<Block>())
        {

            foreach(GameObject cornerpiece in naapurit[8].GetComponent<Block>().LeftTopCornerPiece)
            {
                cornerpiece.SetActive(false);
            }
        }


    }

```

```
    void SetVisualSoftenersOn()
    {
        //Naapurijärjestys
        // 0 | 1 | 2
        // 3 | 4 | 5
        // 6 | 7 | 8

        if(naapurit[1] == null || naapurit[1].GetComponent<Block>().aboutToDie)
        {
            TopVisualPiece[GetRandomFrom(TopVisualPiece.Count)].SetActive (true);
//          TopVisualPiece.SetActive(true);
        }
        if(naapurit[3] == null || naapurit[3].GetComponent<Block>().aboutToDie)
        {
            LeftVisualPiece[GetRandomFrom(LeftVisualPiece.Count)].SetActive(true);
//          LeftVisualPiece.SetActive (true);
        }
        if(naapurit[5] == null || naapurit[5].GetComponent<Block>().aboutToDie)
        {
            RightVisualPiece[GetRandomFrom(RightVisualPiece.Count)].SetActive(true);
//          RightVisualPiece.SetActive (true);
        }
        if(naapurit[7]  == null || naapurit[7].GetComponent<Block>().aboutToDie)
        {
            BottomVisualPiece[GetRandomFrom(BottomVisualPiece.Count)].SetActive(true);
//          BottomVisualPiece.SetActive (true);
        }
    }

    void SetVisualSoftenersOff()
    {
        if(naapurit[1] != null)
        {
            foreach(GameObject visualPiece in TopVisualPiece)
            {
                visualPiece.SetActive(false);
            }
        }
//          TopVisualPiece.SetActive(false);
        if(naapurit[3] != null)
        {
            foreach(GameObject visualPiece in LeftVisualPiece)
            {
                visualPiece.SetActive(false);
            }
//          LeftVisualPiece.SetActive (false);
        }

        if(naapurit[5] != null)
        {
            foreach(GameObject visualPiece in RightVisualPiece)
            {
                visualPiece.SetActive(false);
            }
//          RightVisualPiece.SetActive (false);
        }

        if(naapurit[7]  != null)
        {
            foreach(GameObject visualPiece in BottomVisualPiece)
            {
                visualPiece.SetActive(false);
            }
//          BottomVisualPiece.SetActive (false);
        }
    }

```

```
    void GoTroughAllNeighbours()
    {
        foreach(GameObject checking in naapurit)
        {
            if(checking != null)
            {
                checking.GetComponent<Block>().CheckNeighbors();
                checking.GetComponent<Block>().SetVisualSoftenersOff();
                checking.GetComponent<Block>().SetVisualSoftenersOn();
                checking.GetComponent<Block>().SetNeighborsVisualCornerBlocksOnStart();
                checking.GetComponent<Block>().SetVisualCornerBlocksOff();
            }
        }
    }


    void SetVisualSoftenersOnNeighbors()
    {
        foreach(GameObject checkin in naapurit)
            if(checkin != null)
                checkin.GetComponent<Block>().SetVisualSoftenersOn();
    }

    void GoTroughThisBlock()
    {
        CheckNeighbors();
        GoTroughAllNeighbours();
    }

    void SpawnResources()
    {
        ResourceDropDefiner ResourceDrops = this.GetComponent<ResourceDropDefiner>();

        List<ResourceBase> ResourcesToSpawn = ResourceDrops.CalculateDroppedResources();

        foreach(ResourceBase resource in ResourcesToSpawn)
        {
            GameObject resourceGameObject;
            resourceGameObject = Instantiate(resource.gameObject,transform.position, transform.rotation) as GameObject;
        }
    }

    int GetRandomFrom(int listSize)
    {
        return UnityEngine.Random.Range(0,listSize);
    }

}
```