KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Joni Kurki

CUBEPLOSION-PELI UNITY-TYÖKALULLA

Opinnäytetyö
Kesäkuu 2016

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Toukokuu 2016** **Tietotekniikan koulutusohjelma** Karjalankatu 3 80200 JOENSUU 013 260 600

Tekijä(t)
Joni Kurki

Nimeke
CubePlosion-Peli Unity-työkalulla

Toimeksiantaja
-

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa peli Unity työkalua käyttäen. Idea peliin oli jo valmiina Jouni Laitisella / GPB Studios, mutta hänellä ei ollut aikaa lähteä toimeksiantajaksi, joten työllä ei ollut toimeksiantajaa, vaan se tehtiin itsenäisesti.

Opinnäytetyö alkoi aiheen esitutkimuksella ja Unity-työkaluun tutustumisella erilaisten oppaiden avulla. Esitutkimuksen jälkeen alkoi pelin määrittely ja suunnittelu. Kun suunnittelu ja määrittely olivat valmiina, alkoi pelin toteutus. Pelin testausta suoritettiin samanaikaisesti toteutuksen kanssa.

Pelin toteutus aloitettiin pelialueen rakentamisella. Seuraavaksi vuorossa oli pelaajan luonti sekä hänen liikkumisensa ohjelmointi. Kentän tekstuurit ja materiaalit luotiin seuraavaksi. Tämän jälkeen luotiin valikot ja käyttöliittymä. Viimeisenä luotiin tekoäly, joka osaa itse liikkua pelikentällä sekä tuhota merkittyjä kuutiota. Peli rakennettiin Unity Web Player-alustalle.

Opinnäytetyö onnistui hyvin, vaikka kaikkia ominaisuuksia ei peliin saatu toteutettua. Tulevaisuudessa kehitän pelistä luultavasti mobiiliversion.

Kieli suomi	Sivuja 34 Liitteet 1

Asiasanat

Pelikehitys, Unity 3D, C#, Unity Web Player

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2016
Degree Programme in Information Technology

Karjalankatu 3
80200 JOENSUU
FINLAND
013 260 600

Author (s)
Joni Kurki

Title
CubePlosion – Game Made with Unity Game Development platform

Commissioned by
-

Abstract

The main goal for this thesis was to design and create a game using Unity game development platform. The original idea came from Jouni Laitinen at GPB Studios, but unfortunately Laitinen did not have enough time for this project so this thesis was done independently.

The work began by collecting material for the feasibility study and reading articles about Unity and game making. Definition and design phases were done after the feasibility study was completed. Next, the coding of the game began. Testing was done simultaneously with the coding phase.

Coding was started by making the game area first. The next step was to create the player character and moving capability. Textures and materials for the game area and character were made next. After that menus and user interface were made. The last step was to create artificial intelligence, which could move and destroy marked cubes. The game was built to Unity Web Player platform.

The thesis succeeded nicely, although some functionality could not be realized at this time. In the future, I will likely create a mobile version of the game.

Language

Finnish

Pages 34

Appendices 1

Keywords

game development, Unity 3D, C#, Unity Web Player

Sisältö

Liite Alkuperäinen luonnos, Jouni Laitinen

Keskeiset Käsitteet ja lyhenteet

Asset	Voi olla mikä tahansa peliin kuuluva tiedosto, esimerkiksi kuva- tai äänitiedosto, joka on tuotu Unityyn.
Bugi	Tarkoittaa ohjelmointivirhettä, ohjelma ei toimi niin kuin pitäisi.
Debug	Debuggingilla tarkoitetaan menetelmää, jolla voidaan havaita buge- ja tai ei-toivottuja toimintoja.
DirectX	Microsoftin luoma ohjelmointirajapinta. Se toimii laitteiston ja ohjelmiston välissä.
Frame	Tarkoittaa tietyllä hetkellä renderöityä kuvaa.
GIMP	GNU Image Manipulation Program. GIMP on ilmainen avoimen lähdekoodin kuvanmanipulointiohjelmisto.
Refactoring	Muutetaan olemassa olevaa koodia paremmaksi.
Render	Kuvan piirtäminen tietokoneohjelmaa käyttäen.
UnityEngine	Unity pelinkehitysalustan pohjalla toimiva rajapinta, joka sisältää valmiita toiminnallisuuksia.
VR	Virtual Reality, virtuaalinen todellisuus.

1 Johdanto

Tässä toiminnallisessa opinnäytetyössä oli tarkoitus tehdä olemassa olevan idean pohjalta julkaisuvalmis peli. Alkuperäinen idea peliin tuli Jouni Laitiselta / GPB Studios, mutta koska hänellä ei ollut aikaa lähteä toimeksiantajaksi, tehtiin peli ilman toimeksiantajaa itsenäisesti. Peliä ei kehitetty kuitenkaan täysin alkuperäisen Laitisen toimittaman luonnoksen (liite) mukaan, vaan mukana on omia ideoita ja toteutustapoja.

Opinnäytetyön teoreettisessa osiossa paneudutaan erittäin suosittuun kehitysympäristöön, Unityyn ja sen keskeisiin ominaisuuksiin pelin kehityksessä. Unity on Unity Technologiesin kehittämä kehitysalusta, jolla kuka tahansa voi tehdä omia pelejä. Se on erittäin suosittu pelinkehitysalusta pelinkehittäjien parissa. Unityllä on mahdollista kehittää pelejä PC:lle, konsoleille, mobiililaitteille sekä verkkosivuille. Unityllä on myös mahdollista kehittää erilaisia ohjelmia ja simulaatiota, mutta tässä opinnäytetyössä keskitytään vain pelien kehitykseen.

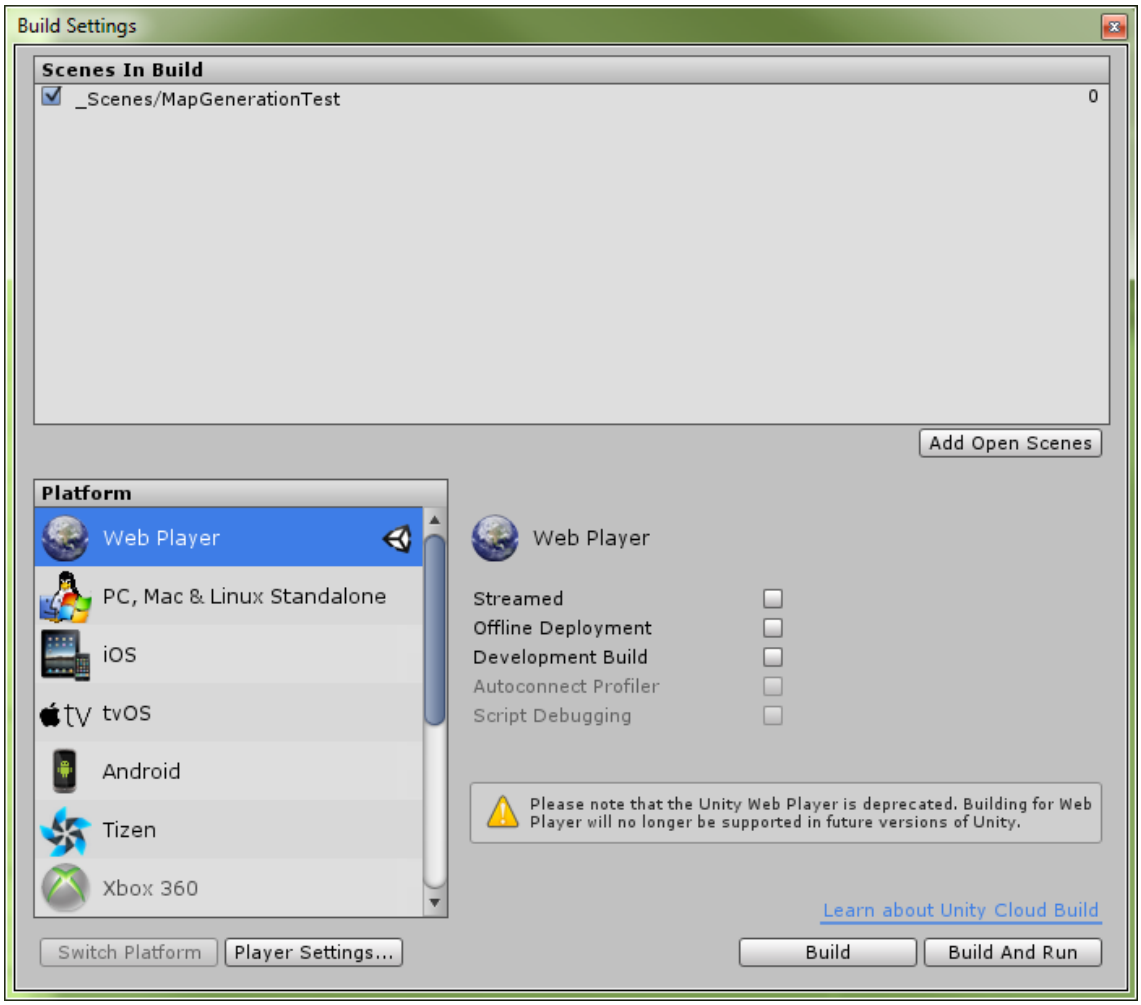
Opinnäytetyössä käydään läpi pelinkehitys prosessia ideasta valmiiseen tuotteeseen asti vaihe vaiheelta. Lisäksi käydään läpi Unityn tärkeimpiä ominaisuuksia ja mitä muita työkaluja on hyödynnetty prosessin aikana.

2 Käytetyt työkalut sekä Unityn tärkeimmät ominaisuudet

2.1 Unity

Unity on pelin kehitys työkalu, jonka Unity Technologies on kehittänyt. Unityllä on mahdollista kehittää pelejä tai erilaisia graafisia ohjelmistoja 2D- ja/tai 3D-ympäristöön. Unityllä on mahdollista kehittää pelejä PC:lle, konsoleille, eri mobiililaitteille sekä verkkosivuille.

Unityssä on todella monia erilaisia ominaisuuksia, jotka helpottavat ja nopeuttavat pelien kehitystä. Unityllä on mahdollista kehittää pelejä monella alustalle samalla kertaa. Kehitysalustasta löytyy kohta, josta on mahdollista valita, mille alustalle haluaa pelin rakentaa (kuva 1).



Kuva 1. Unityn Build Settings -valikko.

2.1.1 Historia

Unity Technologies perustettiin vuonna 2004 David Helgasonin, Nicholas Francisin ja Joachim Anten toimesta Kööpenhaminassa, Tanskassa [1]. Unity sai alkunsa, kun perustajat halusivat luoda pelinkehitysalustan, joka olisi edullinen ja sisältäisi hyviä työkaluja amatööripelinkehittäjille. Näin pelinkehittäjät pystyisivät keskittymään pelinkehitykseen sen teknologian kehityksen sijasta. [2.]

Unityn 1.0 versio julkaistiin kesäkuussa 2005 Applen Worldwide Developers konferenssissa, tuolloin Unityä oli mahdollista käyttää vain OS X käyttöjärjestelmällä [1]. Tällä hetkellä Unityä on mahdollista käyttää 24:llä eri alustalla, mukaanlukien neljä eri VR-alustaa [3].

Lokakuussa vuonna 2007 julkaistiin 2.0 versio Unitystä, joka sisälsi muun muuassa DirectX 9.0 tuen ja reaaliaikaisen varjojen luonnin [4]. 3.0 versiossa, joka julkaistiin syyskuussa 2010, paneuduttiin valaistukseen ja alustan suorituskykyyn. Myös Occlusion Culling-teknologia sai täyden tuen. [5.] Toiseksi uusin iso julkaisu, eli versio 4.0, johon on vielä kirjoitushetkellä tuki, julkaistiin marraskuussa 2012. 4.0 sisälsi muun muassa tuen reaaliajan varjostukselle kaikille alustoille, sekä DirectX 11 tuen. [6.] 5.0 versio julkaistiin maaliskuussa 2015 ja se sisälsi paljon uusia ääni- ja valo-ominaisuuksia sekä Nvidia:n PhysX 3.3 tuen [7]. Tämän hetkinen uusin Unity versio on 5.3.5, joka julkaistiin 20.5.2016 [8].

2.1.2 Lisenssit

Unitystä on saatavilla Unity Personal -lisenssi, joka on ilmainen versio, sekä erillisiä maksullisia Professional -lisenssejä. Maksulliset lisenssit sisältävät paljon enemmän ominaisuuksia ja ovat suunnattu enemmän isoille kaupallisille yrityksille. Unity Professional -lisenssin voi vuokrata hintaan 57€ / kuukausi tai 1140€ kertamaksulla. Professional versioon löytyy myös kolme ilmaista lisäosaa Team License, Windows Phone 8 Pro ja Windows Store Apps Pro. Lisäksi Unity Professionaliin voi ostaa iOS Pro sekä Android Pro -lisäosat, jotka maksavat vastaavasti 57€ kuukaudessa tai 1140€ kertamaksulla. Kertamaksun ja vuokrauksen ero on siinä, että vuokratussa versiossa on tuki kokoajan. Kirjoitushetkellä jos ostaisi Unity Professional version, niin tuki uusille ominaisuuksille, parannuksille, korjauksille tai palveluille loppuisi maaliskuussa 2017. [9.] Tätä opinnäytetyötä varten Unityn Personal versio oli riittävä, koska se sisälsi kaikki tarvittavat ominaisuudet pelin kehittämistä varten.

Unityn Personal sisältää Unityn pelimoottorin kaikilla ominaisuuksilla. Unity Personal on rojaltivapaa, eli omasta tuotteesta ei tarvitse maksaa rojalteja. Unity Personalia on mahdollista käyttää kaikilla alustoilla kehittämiseen. Myös Personal-version käyttäjät voivat käyttää Unityn uusimpia Beta versioita. [10.]

GameObject

GameObjectit ovat Unityn yksiä tärkeimmistä objekteista, joita voidaan luoda peliin. GameObject voi olla esimerkiksi rakennuksen seinän palanen tai pelaajahahmo. GameObject on eräänlainen säiliö sen kaikille eri ominaisuuksille ja toiminnallisuuksille, näitä kutsutaan komponenteiksi. GameObjectiin voi olla kiinitettynä monia erilaisia komponentteja. [11.] (Unity - Manual: GameObject, 2016)

Scene

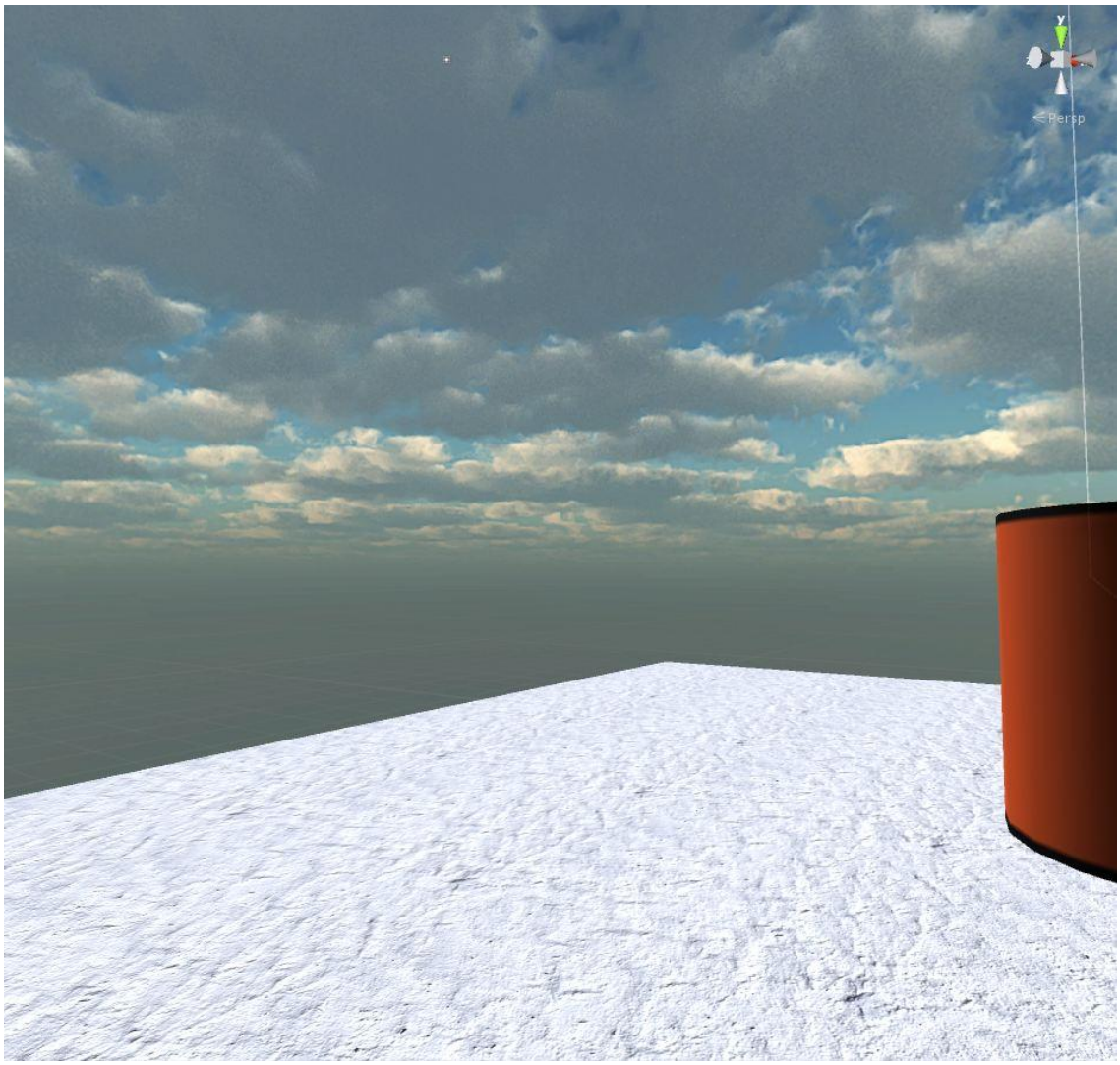
Scene pitää sisällään pelissä sen hetkiset käytetyt GameObjectit. Jokaisella Unityn pelillä pitää olla ainakin yksi scene. Yleensä scenejä on monia, esimerkiksi valikoille oma sekä kaikille kentille omat scenet. Scenet ovat yleensä sisällöiltään erilaisia. Yhdessä pelissä voi käyttää montaa sceneä yhtä aikaa. [12.] (Unity - Manual: Scenes, 2016)

Skybox

Unityssä on mahdollista luoda peliin skybox. Se on eräänlainen näkymä joka luodaan koko scenen ympärille. [13.] Skyboxilla voidaan luoda esimerkiksi seuraava vaikutelma: pelimaailma jatkuu pelikenttää pidemmälle horisonttiin. (Kuva 2.)

Shader

Shader on Unityn oma skripti, joka sisältää matemaattisia laskelmia ja algoritmeja, joiden mukaan se laskee jokaisen piirrettävän pikselin (Unity - Manual: Materials, Shaders & Textures, 2016) [14].

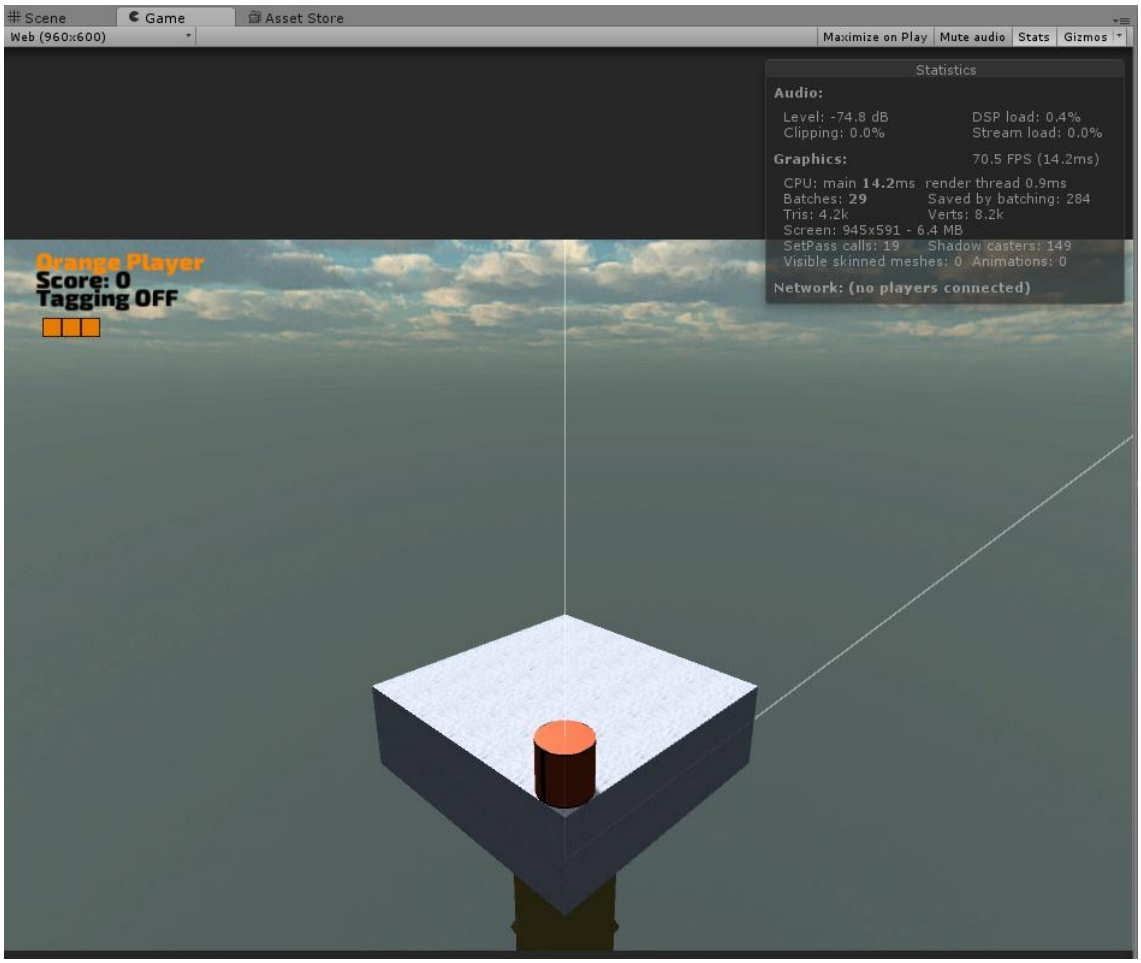


Kuva 2. Skybox pelinäkymässä.

2.1.3 Käyttöliittymä

Unityn käyttöliittymä on hyvin selkeä ja sitä voi muokata haluamakseen helposti. Kuvassa 3 näkyy Game-välilehti, jossa on näkymä omasta kehitettävästä pelistä.

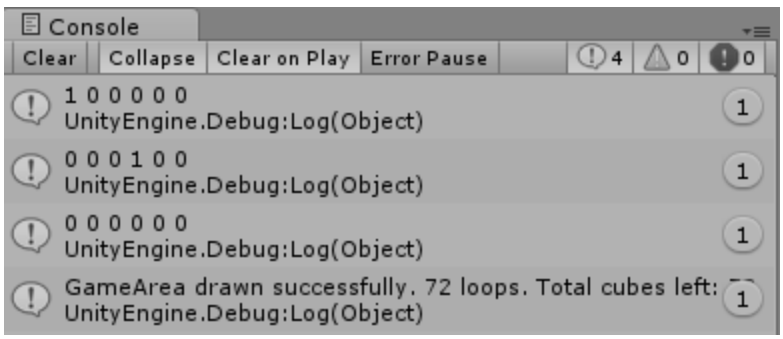
Kuvassa 4 näkyy pelin hallintaan liittyvä näppäinrivi. Vasemman puoleinen nappi käynnistää tai lopettaa pelin. Keskimmäinen on pause-nappi, eli pelin voi pysäyttää. Oikeanpuoleisin nappi toimii vain jos pause-nappi on pohjassa, tällöin edetään yksi frame pelissä.



Kuva 3. Unity editorin Game-näkymä.

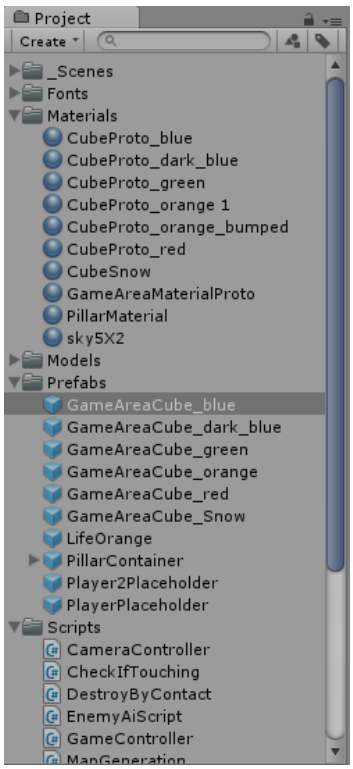
Kuva 4. Näppäinrivistö pelin hallintaa varten.

Kuvassa 5 esitetään konsoli-ikkunan syötettä pelistä. Konsoli on erittäin hyödyllinen esimerkiksi bugien kiinni saamisen kannalta tai jos halutaan seurata muuttujan arvoa jossakin tietyssä kohtaa peliä.



Kuva 5. Esimerkki pelin Console-välilehti.

Kuvassa 6 on esitetty projektin kaikkien assettien hakemistorakenne. Unityn käyttäjä voi itse määritellä kansioiden rakenteet ja nimetä kaikki objektit haluamallaan tavalla. Uuden assetin pystyy tuomaan suoraan vetämällä tiedoston Unityyn tai kopiomalla tiedoston Unityn projektin hakemistoon, jolloin Unity huomaa, että tiedosto on lisätty hakemistoon, jolloin se lisää sen projektiin. Unity tarjoaa tuen suoraan todella monelle eri tiedostomuodolle [15](Taulukko 1).

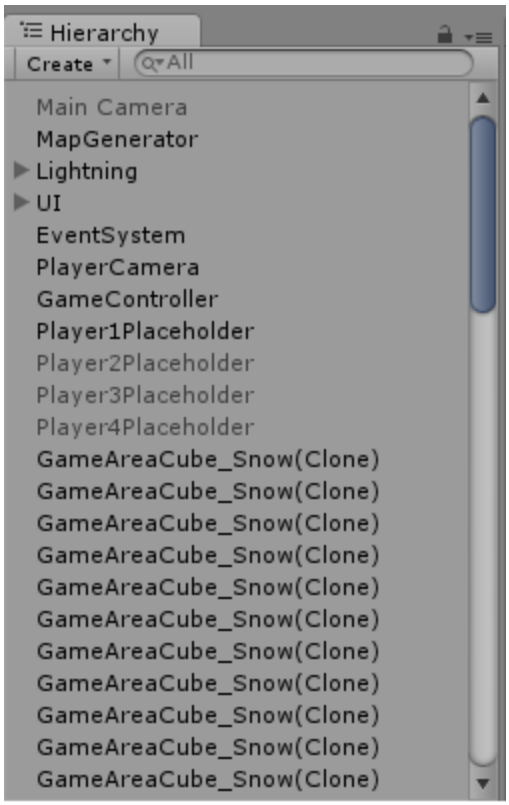


Kuva 6. Project-välilehti Unityssä näyttää kaikki assetit.

Taulukko 1. Unityn tukemat tiedostomuodot [14].

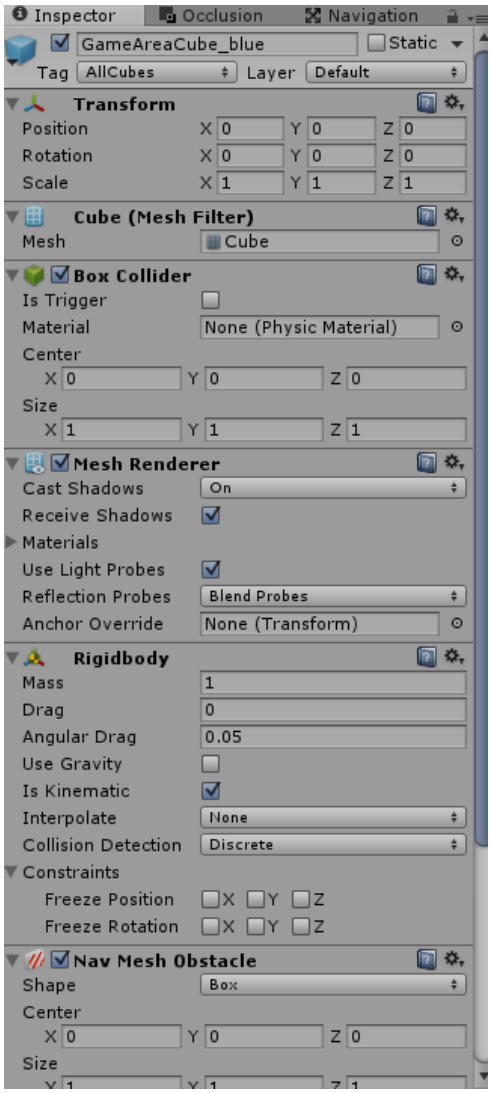
Kuvaformaatit	.psd	.jpg	.png	.gif	.bmp	.tga	.tiff	.iff	.pict	.dds
Ääniformaatit	.mp3	.ogg	.aiff	.wav	.mod	.it	.sm3			
Videoformaatit	.mov	.avi	.asf	.mpg	.mpeg	.mp4				
Tekstiformaatit	.txt	.htm	.html	.xml	.bytes					

Kuvassa 7 näkyy Hierarchy-välilehti, eli sen hetkisen scenen sisältö. Hierarchy-välilehdeltä käy ilmi kaikki GameObjectit, jotka ovat sillä hetkellä mukana pelissä. Tämä helpottaa myös paljon pelin debuggaamista. Ei-aktiiviset GameObjectit näkyvät harmaammalla Hierarchy-välilehdellä. Ei-aktiiviset GameObjectit ovat sillä hetkellä scenessä, mutta ne ovat disabloituja.

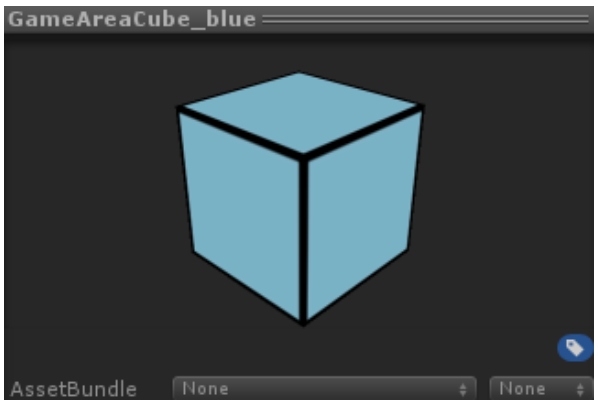


Kuva 7. Hierarchy-välilehti Unityssä.

Unityssä on mahdollista myös tarkastella GameObjecteja, jotka ovat jo pelissä tai joita voidaan käyttää pelissä myöhemmin. Tällöin Inspector-välilehdestä on hyötyä (Kuva 8). GameObjectin tarkastelu 3D-ympäristössä onnistuu Preview-välilehden kautta (Kuva 9).



Kuva 8. Inspector-välilehti GameAreaCube_blue assetista.



Kuva 9. Preview-välilehti GameAreaCube_blue assetista.

2.1.4 Skriptit ja Unityn ohjelmointikielet

Skriptit ovat GameObjecteihin kiinnitettäviä komponentteja, joilla voidaan ohjata ja muokata, miten GameObject käyttäytyy missäkin tilanteessa. Esimerkiksi jos halutaan saada pallo liikkumaan nuolinäppäimestä painamalla, niin siihen täytyy lisätä skripti, joka hoitaa pallon liikutuksen nuolinäppäimen painamisen johdosta.

Unityssä on mahdollista luoda skriptejä kolmella eri ohjelmointikielellä: Boo, C# ja UnityScript. Tässä opinnäytetyössä on käytetty pelkästään C#-ohjelmointikieltä. Unityn oman blogin mukaan C# on suosituin ohjelmointikieli Unityssä (80,4 %). UnityScript (18,9 %) ja Boo (0,44 %) eivät yllä lähellekään samaa suosiota kuin C#-ohjelmointikieli. [16.]

C#-ohjelmointikieli kehitettiin Microsoftin toimesta vuonna 2000 Microsoftin omaa .NET-konseptia varten. C# on oliopohjainen ohjelmointkieli. C#:n syntaksi on hyvin samantapaista kuin C, C++ tai Java-ohjelmointikielissä. [17.] Alla esimerkki C#-ohjelmointikielestä Unityssä.

```
using UnityEngine;
using System.Collections;

public class ExampleProgram : MonoBehaviour {
	int myInt  = 24;

		int myFunction(){
			return myInt;
		}

}
```

Unityssä käytettävä ohjelmointikieli UnityScript on samantyylinen kuin JavaScript-ohjelmointikieli, jota käytetään esimerkiksi web-ohjelmoinnissa. UnityScripstä käytetään kuitenkin nimeä JavaScript joissain teksteissä, vaikka kyseessä on kaksi eri ohjelmointikieltä. [18.] Alla näkyy esimerkki Unityssä käytetystä UnityScriptstä.

```
#pragma strict
```

```
class Foo{
        var myInt : int = 24;
        function myFunction(){
                return myInt;
        }
}
```

Boo on oliopohjainen ohjelmointikieli, joka on saanut vaikutteita Pythonista ja C#-ohjelmointikielistä. Boo-ohjelmointikielen syntaksi on tehty selkeäksi ja se on osittain samantapainen kuin Pythonin syntaksi. [19.] Alla näkyy esimerkki Boo-ohjelmointikielen syntaksista Unityssä.

```
import UnityEngine
import System.Collections

public class ExampleProgram(MonoBehaviour):

        myInt = 24

                def myFunction():
                        return myInt
```

2.1.5 NavMesh ja NavMeshAgent

NavMesh on yksi Unityn tärkeimmistä ominaisuuksista. NavMesh laskee scenen kaikista Navigation Static merkatuista GameObjecteista eräänlaisen kartan, Navigation Meshin, tätä prosessia kutsutaan NavMesh Bakingiksi. [20.]

NavMeshAgent on eräänlainen valmis tekoäly, joka osaa itsenäisesti liikkua paikasta A paikkaan B ilman, että se törmäilee esineisiin. Jotta NavMeshAgent toimii, pitää NavMesh Baking olla sitä ennen tehtynä ja GameObjectiin pitää olla NavMeshAgent-komponentti kiinnitettynä. [21.]

2.1.6 Occlusion Culling

Occlusion Culling on Unityn tarjoama ominaisuus. Occlusion Culling toimii siten, että Unity ei renderöi kaikkia Scenessä olevia GameObjecteja, vaan pelkästään ne, jotka näkyvät kamerassa. Joten jos vaikka pelaaja katsoo seinään ja seinän takana on paljon muita objekteja, niin näitä ei renderöidä seinän takaa. Tämä ominaisuus nostaa pelin suorituskykyä. [22.]

Jotta Occlusion Culling toimisi hyvin, täytyy GameObjectit baketa. ”Bake” tarkoittaa että se lasketaan etukäteen. Occlusion Culling siis laskee kaikki staattiset GameObjectit ennen pelin aloitusta ja voi näin laskea mikä objekti renderöidään milloinkin.

2.2 Microsoft Visual Studio 2015 community ja MonoDevelop

Opinnäytetyössä skriptien luontiin käytettiin ensin Microsoftin Visual Studio 2015 Communityä, mutta tästä loppui ilmaislisenssi, minkä takia siirryttiin käyttämään Unityn mukana tullutta MonoDevelopia.

Molemmista editoreista löytyy tärkeimmät ominaisuudet, kuten refactoring ja koodin värjäys.

Blender

Blender on 3D-mallinnus ohjelmisto, jolla voi tehdä malleja esimerkiksi peleihin tai piirtää grafiikkaa. Blenderillä tehdyt mallit ovat suoraan yhteensopivia Unityn kanssa.

GIMP

GIMP on ilmainen avoimen lähdekoodin kuvanmuokkausohjelmisto, jolla voidaan piirtää tai muokata kuvia. Tässä opinnäytetyössä GIMPiä käytettiin Menuvalikon taustakuvan luontiin, kuutioiden materiaalin tekstuurin värjäykseen sekä hahmojen elämiä esittäviin kuviin.

2.3 Asset Store

Asset Store on eräänlainen kauppa, josta voi ladata erilaisia valmiita tai keskeneräisiä assetteja omaan peliin. Asset storessa on mainittu, onko käytettävä assetti ilmainen vai maksullinen ja minkälaisen lisenssinalaisena kyseinen assetti on.

2.4 Tekoäly

Tekoälyllä tarkoitetaan tietokonetta tai tietokoneohjelmaa, joka pystyy suorittamaan itsenäisesti toimintoja, jotka luokitellaan älykkääksi [23]. Opinnäytetyötä varten luotua tekoälyä ei sinänsä voida luokitella oikeaksi tekoälyksi, koska se ei laske liikettä monen liikkeen päähän, vaan se laskee ainoastaan seuraavan liikkeen. Se ei pysty perustamaan tehtävää liikkumispäätöstä, kuin seuraavaan loogiseen vertailulausekkeeseen, joka annetaan sille skriptissä.

Tässä opinnäytetyössä tekoälyllä viitataan tietokonepelaajiin, jotka kykenevät itse laskemaan liikkumisen suunnan ja satunnaisesti väistelemään tyhjiä ruutuja kentässä.

3 Opinnäytetyön lähtökohdat ja ohjelmiston elinkaari

Opinnäytetyön aiheen valintaan vaikuttivat omat mieltymykset pelinkehitykseen, pelimekaaniikkoihin, sekä se että en saanut paikallisilta yrityksiltä valmista toimeksiantoa opinnäytetyöhön. Myös intohimo erilaisia videopelejä kohtaan vaikutti suuresti päätöksentekoon. Lisäksi aiheen valintaan vaikutti se, että olin halunnut opetella käyttämään Unityä, mutta minulla ei ole ollut aikaa tutustua Unityyn aikaisemmin, joten opinnäytetyö osui hyvään saumaan.

3.1 Ohjelmiston elinkaari

Ohjelmistojen kehityksen elinkaari vesiputousmallissa menee yleensä seuraavalla kaavalla: Ensiksi määritellään vaatimukset ja tehdään esitutkimusta aihealueesta, sekä tehdään tarvittavat taustatutkimukset. Kun tarvittavat taustatutkimukset sekä määrittelyt on tehty ja hyväksytty, aletaan suunnittelemaan ohjelmistoa.

Suunnitteluvaiheessa päätetään miten järjestelmä toteutetaan teknisesti. Kun suunnittelu on valmis, aletaan toteuttaa ohjelmistoa aiempien kohtien mukaan, eli tässä vaiheessa ohjelmoidaan ohjelmisto. Seuraava vaihe on testaus, yleensä testausta voidaan hoitaa myös samaan aikaan kuin itse ohjelmointia, jolloin voidaan löytää ei-toivottuja bugeja jo varhaisessa vaiheessa. Lopuksi tuote julkaistaan tai otetaan käyttöön. [24.]

Vesiputousmallissa toteutetaan yksi vaihe kerrallaan ja seuraava vaihe on aiemmasta vaiheesta kiinni. Tässä opinnäytetyössä käytettiin vesiputousmallia hieman sovelletusti.

On myös tärkeää tiedostaa, että kaikkea ei välttämättä voida toteuttaa niin kuin alun perin on suunniteltu, jolloin toteutus saattaa poiketa alkuperäisestä suunnitelmasta.

3.2 Vaatimukset ja määrittely

Tämän opinnäytetyön kohdalla taustamateriaalin hankkiminen, ohjeet sekä oppaat olivat todella helposti saatavilla, sillä Unityn sivuston Learn- ja Community-osiot [25; 26] tarjoavat todella hyviä ohjeita ja artikkeleita liki kaikista Unityn pääominaisuuksista. Unityn sivustolta löytyy myös dokumentaatio kaikkiin UnityEnginen osa-alueisiin [27].
Peliä varten määriteltiin vaatimukset ja ne jaettiin toiminnallisiin, laadullisiin sekä resurssi vaatimuksiin.

3.2.1 Toiminnalliset vaatimukset

Käyttöliittymän tulee olla helppokäyttöinen ja helposti ymmärrettävä. Käyttöliittymästä on käytävä ilmi pelaajien pisteet ja elämäpisteet. Peli ilmoittaa myös jos pelaajahahmo tai tekoälyhahmo häviää tai voittaa pelin.

Pelin päävalikosta on pystyttävä näkemään Help- ja Credits-osiot. Help-osiosta on myös pystyttävä näkemään pelin kontrollit. Peliä aloittaessa on voitava määrittää kuinka monta pelaajaa pelissä on.

Sekä pelaaja- että tekoälyhahmojen pitää pystyä liikkumaan pelikentällä kahdeksaan eri suuntaan ja pudottamaan pommeja. Pommin pudotus värjää alla olevan kuution pelaajan värillä, muuten pommi ei näy visuaalisesti. Hahmon pitää pystyä räjäyttämään merkatut ruudut ja saada siitä pisteet. Yhden kuution räjäyttäminen antaa yhden pisteen. Merkattuja kuutiota voi olla enemmän kun yksi kerrallaan pelikentässä. Mikäli pelikentän kaikki kuutiot ovat räjäytetty, kenttä on suoritettu loppuun ja tällöin tulee uusi valikko, mistä voi jatkaa peliä seuraavaan kenttään.

Hahmojen tulee voida kävellä toisten hahmojen läpi. Hahmon ei ole mahdollista kävellä kuutioiden läpi. Hahmot voivat merkata toisten hahmojen merkkaamia kuutioita. Hahmot liikkuvat yhden kuution kerrallaan per liike. Liikkeeseen menee ennalta määritetty aika, kuitenkin siten, että peli ei ole vuoropohjainen, vaan reaaliaikainen.

Pelikentän tulee koostua monesta kuutiosta ja sen tulee olla symmetrinen luontivaiheessa, jotta peli olisi kaikille hahmoille tasapuolinen. Painovoima ei vaikuta kuutioihin, mutta vaikuttaa hahmoihin. Jos hahmo tippuu tai kävelee kohtaan, jossa ei ole kuutioita, hahmo tippuu alaspäin. Mikäli hahmo tippuu kohtaan, joka ei ole pelialueella, menettää hahmo yhden elämän ja hahmo luodaan uuteen paikkaan. Mikäli elämät loppuvat, hahmo häviää pelin.

Tekoälyhahmo yrittää voittaa pelin keräämällä mahdollisimman monta pistettä. Tekoälypelaajan vaikeustasoa voidaan muuttaa. Vaikeustason muutos vaikuttaa tekoälypelaajan liikkumisnopeuteen.

3.2.2 Laadulliset ja resurssivaatimukset

Peli toteutetaan Unityllä siten, että se on modulaarinen, jolloin siihen on tulevaisuudessa helppo lisätä sisältöä. Pelin pitää olla suorituskykyinen ja helppokäyttöinen.

Pelistä tehdään joko mobiiliversio ja/tai Unity Web Playerillä pyörivä toteutettu versio. Näistä valitaan joko toinen tai molemmat toteutusvaiheessa. Pelissä käytetään avoimen lähdekoodin resursseja, mikäli valmiita ratkaisuja tarvitaan.

3.3 Suunnittelu ja toteutus

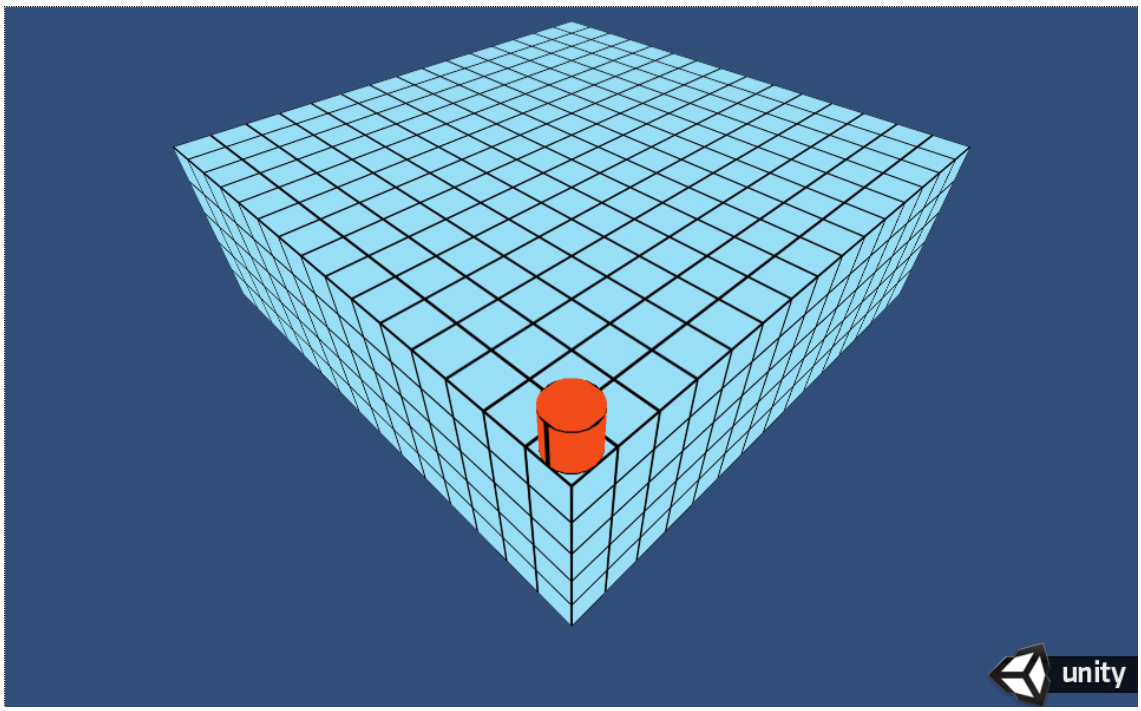
Suunnitteluvaiheessa pelille määriteltiin yksi osa eli moduuli kerrallaan, koska moduulit ovat toisistaan kriittisesti riippuvaisia. Mikäli hahmo tehtäisiin ennen pelikenttää, niin hahmon liikuttelu ei välttämättä toimisi oikein kentässä. Toteutusjärjestys määriteltiin seuraavasti:

1. Pelikenttä
2. Pelihahmo ja hahmon ominaisuudet
3. Kentän tekstuuri ja materiaali
4. Pelivalikot
5. Tekoäly pelaajat

3.3.1 Pelikenttä ja pelaajahahmo

Ensimmäinen vaihe oli kentän muodostaminen. Pelikenttä muodostuu kuutiosta ja se rakennetaan dynaamisesti pelikentän alussa. Pelikentän arvoja pidetään

kolmiulotteisessa taulukossa, jonka mukaan sitten kenttä aina piirretään, mikäli siihen tehdään muutoksia. Kuvassa 10 nähdään Unityn kautta luotu pelikenttä, joka on kooltaan 15x15x5 kuutiota. Kuvassa 11 nähdään Instantiate–funktio, jolla luodaan GameAreaCubeSnow niminen GameObject. Position-muuttuja ilmoittaa mihin kohtaan pelialuetta GameObject piirretään. Rotation-muuttuja ilmoittaa missä kulmassa GameObject piirretään.



Kuva 10. Pelikenttä piirretty Unityssä.

```
// palikan sijainti x,y,z koordinaatistossa
Vector3 position = new Vector3 (x, z, y);
// ei anneta palikoille yhtään kiertoa
Quaternion rotation = new Quaternion ();
// luodaan palikka kohtaan x,y,z - 0 kierrolla.
Instantiate (GameAreaCubeSnow, position, rotation);
```

Kuva 11. GameAreaCubeSnow Intantiate-funktio.

Kentän keskipalkin malli luotiin ensiksi Blenderissä. Blenderissä rakennettu 3D-malli tuotiin Unityyn. Blenderissä tehdyn 3D-mallin tiedostomuoto oli ”.fbx”, joten Unity tuki tiedostomuotoa suoraan. Unity tuo 3D-mallin mahdollisesti väärässä skaalassa, joten pitää tarkistaa että Scale Factor on arvossa 1. Koska Blenderissä ja Unityssa akselit ovat erisuuntaiset, täytyy tuodulle 3D-mallille antaa Uni-

tyn puolella x-akselille 90° kierto, jotta 3D-mallin kierto on samalla tavalla kuin se on Blenderissä luotu.

Pelissä käytetään valmista skybox assettia nimeltä Sky5X One [28]. Kyseinen asset on haettu suoraan Unityn Asset Storesta ja se on ilmaisen lisenssin alla. Kun pelikentän muodostaminen oli saatu valmiiksi, oli pelihahmon luonnin vuoro. Pelihahmona toimii oranssi lieriö (Kuva 10). Pelihahmojen GameObjecteille on kiinnitetty Rigidbody-komponentti, jotta fysikaaliset ilmiöt ja voimat kohdistuisivat GameObjectiin.

Pelihahmolle luotiin yksinkertaiset kontrollit, joilla pelihahmoa voidaan liikuttaa kahdeksaan eri suuntaan. Kuvassa 12 esitetään miten pelaajan syöte otettiin vastaan ja kuvassa 13, miten pelaajahahmoa liikutettiin pelin alkuvaiheessa.

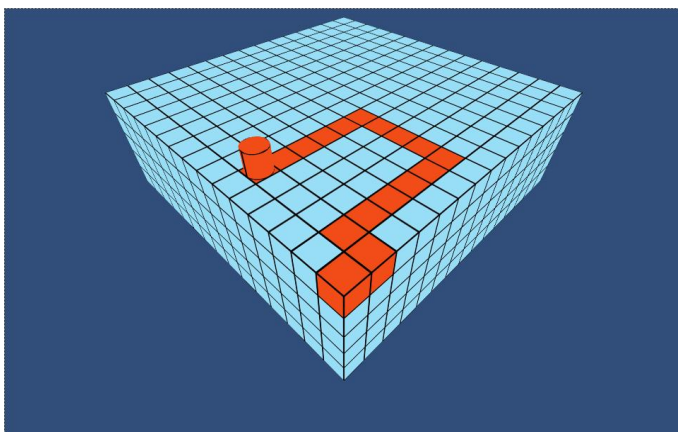
```
if (Input.GetKeyDown (KeyCode.Keypad9)) {
```

Kuva 12. Näppäimen numpad9 syöte voidaan havaita kuvassa näkyvällä koodilla.

```
    player.transform.Translate(x,y,z);
```

Kuva 13. GameObjectia voi liikuttaa transform.Translate -funktiolla haluamaansa paikkaan.

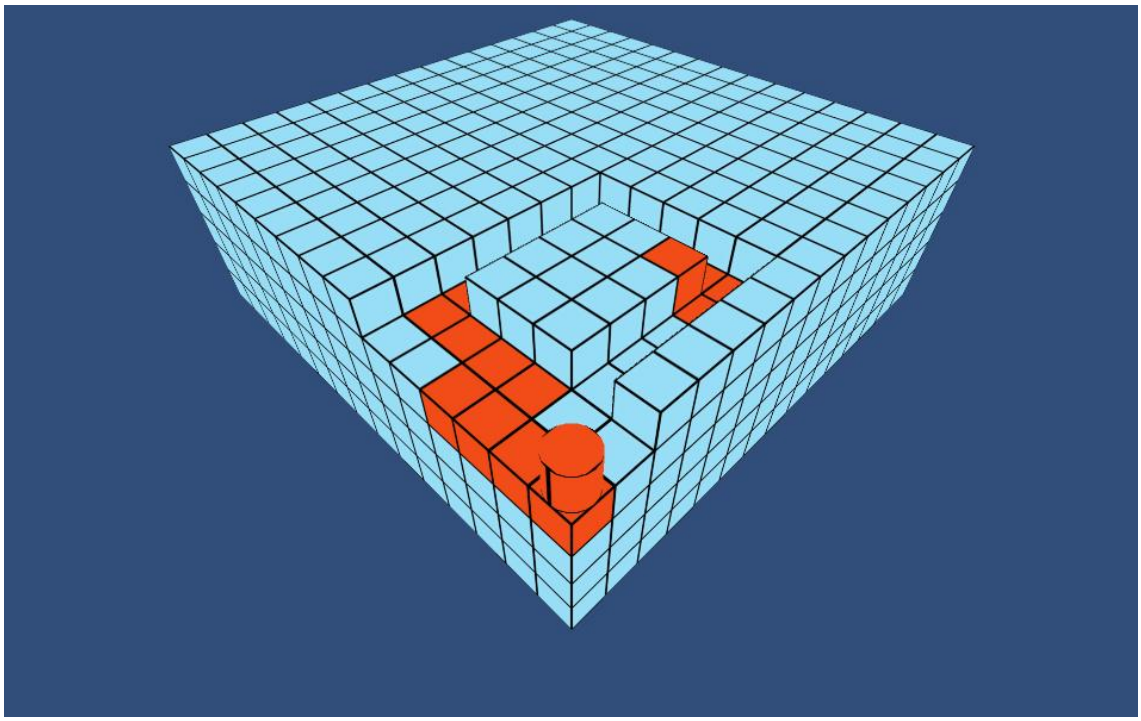
Seuraavaksi vuorossa on kuution merkkaamisen tekeminen. Pelihahmon pitää pystyä merkkaamaan alla oleva kuutio omalla värillä (Kuva 14). Pelihahmo merkkaa alla olevan kuution omalla värillään, mikäli pelihahmo on kuution päällä ja pelaajahahmon TaggingSW -funktio palauttaa arvon ”true”.



Kuva 14. Pelihahmo voi merkata allaan olevan kuution.

Pelihahmon pitää pystyä räjäyttämään merkkaamansa kuutiot, jolloin räjäytetyt kuutiot tuhotaan kentästä (Kuva 15). Mikäli räjäytettyjä kuutiota ei tuhottaisi kentästä, vaan pelkästään gameArea muuttujasta, niin tästä seuraisi että kokoajan luodaan uusia kuutiota, mutta vanhoja ei poisteta. Tämä johtaisi lopulta pelin lamaantumiseen ja muistin ylivuotoon (Kuva 16).

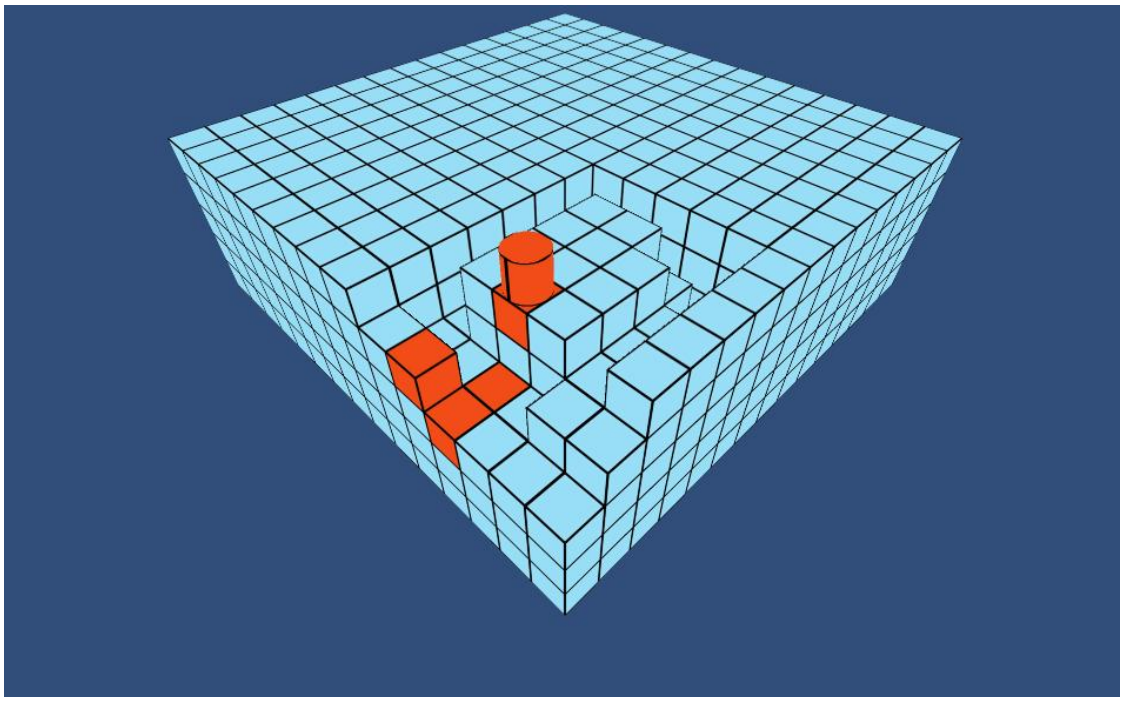
Pelihahmon pitää myös pystyä liikkumaan joko yhtä tai kahta kuutiota ylemmälle kerrokselle (Kuva 17). Tämä toteutettiin kuvan 18 mukaisesti. Mikäli viereiseen ruutuun ei voida liikkua, tarkistetaan y-koordinaatin +1 arvo, ja jos sekään ei ole vapaa, niin tarkistetaan y-koordinaatin +2 arvo. Jos ruutu ei ole vapaa, ei voida liikkua. Mikäli ruutu on vapaana, niin liikutaan ensimmäiseen vapaaseen ruutuun.



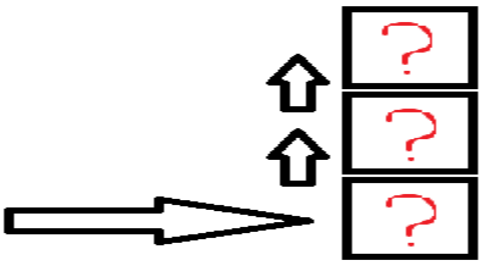
Kuva 15. Pelihahmo voi räjäyttää merkatut kuutiot.

```
GameObject[] gameObjects;
gameObjects = GameObject.FindGameObjectsWithTag("AllCubes");
for(int d = 0; d < gameObjects.Length; d++) {
    Destroy(gameObjects[d]);
}
```

Kuva 16. Esimerkki miten voidaan tuhota kaikki GameObjectit, joiden Tag on ”AllCubes”.



Kuva 17. Pelihahmon on mahdollista nousta ylemmälle tasolle.

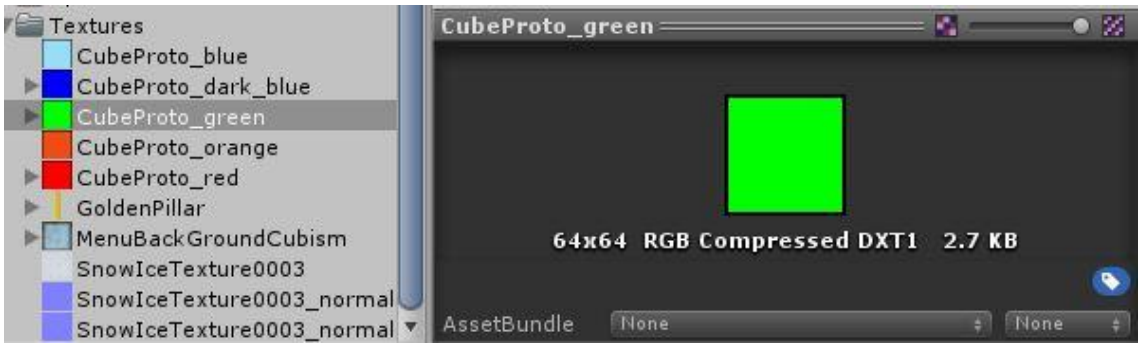


Kuva 18. Ylöspäin liikkuminen havainnollistettuna.

3.3.2 Materiaalit ja tekstuurit

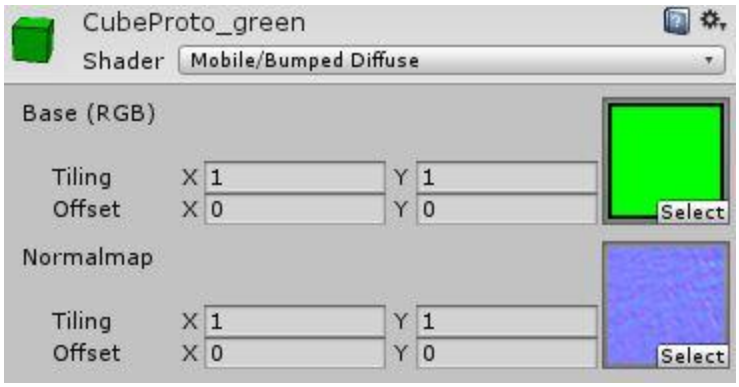
Unityssä on myös mahdollista tehdä erilaisia materiaaleja. Tässä opinnäytetyössä kaikki tekstuurit ovat itsetehtyjä, poislukien merkkaamattoman kuution tekstuuri. Merkkaamattoman kuution tekstuuri on haettu Brusheezy-sivustolta [29]. Valittu tekstuuri on ilmainen käyttää sekä yksityis- että yrityskäytössä.

Tässä opinnäytetyössä materiaalit on luotu seuraavalla tavalla: Tekstuuri on tehty itse GIMP-kuvankäsittelyohjelmistolla, jonka jälkeen kuva on tuotu Unityyn (kuva 19).

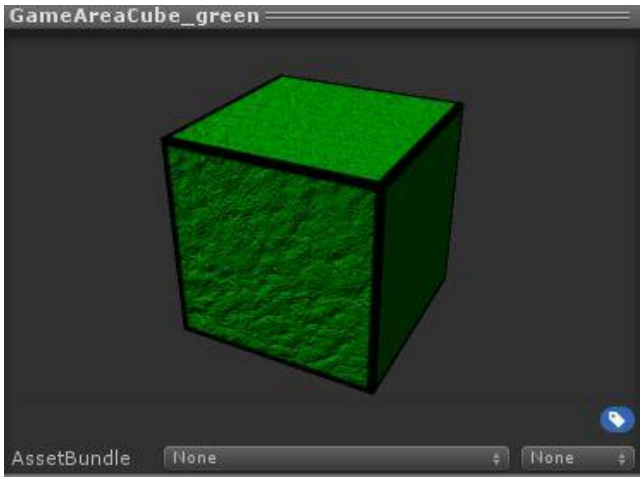


Kuva 19. CubeProto_green tuotu Unityyn.

Kuvan tuonnin jälkeen on luotu uusi materiaali Unityn valikoiden ”Create > Material”-valikon kautta. Tämän jälkeen materiaalille on mahdollista valita erilaisia shadereitä. CubeProto_green materiaalilla on käytetty ”Mobile/Bumbed Diffuse” shaderiä, jolla materiaalin saadaan syvyysvaikutelma käyttäen Normalmapia (Kuva 20). Tämän jälkeen materiaali-komponentti kiinnitetään haluttuun GameObjectiin, kuvassa 21 nähdään lopputulos.



Kuva 20. CubeProto_green materiaali.



Kuva 21. CubeAreaCube_green GameObjectiin on kiinnitetty materiaali.

3.3.3 Valikot ja pelilogiikka

Pelin valikoissa käytetty fontti on nimeltään Exo-2 ja se on avoimen lähdekoodin fontti [30]. Valikot on toteutettu seuraavalla tavalla GameController skriptissä. Vain yksi valikko voi olla päällä kerralla. Aina kun uusi valikko näytetään, niin aiempi näkyvissä ollut valikko deaktivoidaan. Kuvassa 22 esitetään ShowUIElements -funktio, joka hoitaa valikoiden aktivoimoisen ja deaktivoimisen. Oletuksena kaikki valikot ovat deaktivoituna. Deaktivoidut valikot voidaan kytkeä päälle kuvan 23 komennolla. Kuvassa 24 on kutsuttu funktiota, joka näyttää uuden pelin aloitusvalikon.

```
void ShowUIElements(int mainMenu = 0, int helpMenu = 0, int creditsMenu = 0,
                    int startNewGame = 0, int continueGame = 0, int lvlComplete = 0, int gameOver=0){
```

Kuva 22. ShowUIElements funktio.

```
if (mainMenu == 1)
    MainMenu.SetActive (true);
```

Kuva 23. Aktivoidaan MainMenu -valikko.

```
case 1: // start new menu
        ShowUIElements (0, 0, 0, 1);
```

Kuva 24. Esimerkki ShowUIElements funktion kutsusta.

Pelin vaiheista vastaa GameController skriptissä oleva gameState muuttuja. GameState muuttujan arvon muuttaminen vaihtaa sillä hetkellä olevaa pelin vaihetta (Kuva 25). Esimerkiksi jos pelaaja haluaa mennä help-valikkoon, niin muuttujan arvoksi muuttuu 5, kun Help-valikon nappia painetaan.

```
private int gameState;
// Tämä skripti hoitaa pelin aloituksen, kentän vaihdot, jne...
// gameState =
//              0, main menu
//              1, start new game -> 11, start new game menu valikko
//              2, paused
//              3, continue game
//              4, help
//              5, credits
//              6, exit
//              7, level complete
//              8, restart level
//              9, game over
//              11, start new game menu
```

Kuva 25. Muuttujan gameState arvot.

Jos pelihahmon elämät loppuvat, niin pelihahmo deaktivoidaan. Deaktivoidut pelaajat voidaan aktivoida uudestaan tarvittaessa, jolloin pelaajien pisteet ja elämät resetoidaan. Pelihahmo kuolee, jos se tippuu pelialueen ulkopuolelle. Pelin aikainen käyttöliittymä näyttää pelaajan pisteet ja pelaajan värin (Kuva 26). Pelaajan elämät näkyvät pelaajan värisinä neliöinä. Ihmispelaaja näkee lisäksi onko ruudun merkkaaminen päällä vai pois päältä.

Kuva 26. Pelin aikainen käyttöliittymä.

3.3.4 Tekoäly

Tekoälypelaajien tavoite on liikkua satunnaiseen pisteeseen kentässä mahdollisimman vähillä siirroilla. Satunnainen piste on luotu skriptin kautta (Kuva 27). Tekoälypelaajan toiminnot ovat käytännössä loogisia lauseita, joiden mukaan tekoälypelaaja laskee aina seuraavan liikkeen suunnan. Tekoälypelaajan on mahdollista liikkua kahdeksaan eri suuntaan.

```
int randX = UnityEngine.Random.Range (0, w);
int randZ = UnityEngine.Random.Range (0, h);
int randY = UnityEngine.Random.Range (0, r);
wpLocation = new Vector3(randX, randY, randZ);
```

Kuva 27. Satunnaisen pisteen luonti EnemyAiScriptissä.

Tekoälyn logiikka toimii seuraavasti:

1. Tekoälylle luodaan satunnainen piste pelialueelta.
2. Tekoäly määrittää pitääkö hahmon liikkua suuntaan x, 0 vai -x.
3. Tekoäly määrittää pitääkö hahmon liikkua suuntaan z, 0 vai -z.

4. Tekoäly tarkastaa seuraavan siirron kohdan kentästä, mikäli ruutu on vapaa, se liikkuu siihen. Hyppää kohtaan 6.
5. Jos ruutu ei ole vapaa se etsii vierestä seuraavan vapaan ruudun. Mikäli x- ja z-akselilta ei löydy vapaata ruutua, niin tekoäly tekee uuden tarkastuskierroksen muuten samoilla arvoilla, mutta y:n arvoa muutetaan -1:llä.
6. Jos ruutu ei ole vapaa se etsii vierestä seuraavan vapaan ruudun. Mikäli x- ja z-akselilta ei löydy vapaata ruutua, niin tekoäly tekee uuden tarkastuskierroksen muuten samoilla arvoilla, mutta y:n arvoa muutetaan -2:lla.
7. Kun tekoälyllä on selvää seuraava vapaa ruutu, niin se tekee liikkeen.
8. Tekoäly laittaa kuution merkkauksen päälle.
9. Jos tekoäly saavuttaa satunnaisen ruudun, se laittaa ruudun merkkauksen pois päältä ja sille luodaan uusi satunnainen piste pelialueelta.
10. Tekoäly liikkuu yhden liikkeen ja räjäyttää merkityt kuutiot. Tekoäly saa tästä merkattujen ruutujen verran pisteitä.

Mikäli tekoäly ei ole saavuttanut satunnaista pistettä, niin se suorittaa vaiheita 1-7. Kun tekoäly on saavuttanut satunnaisen pisteen, se tekee vaiheet 8 ja 9, jonka jälkeen se suorittaa vaiheen 1.

3.4 Testaaminen

Testausta suoritettiin samanaikaisesti pelin toteutuksen aikana. Kun peli oli siinä vaiheessa, että siihen saatiin lisättyä tekoälypelaajia, niin kaverit pääsivät testaamaan peliä ja kertomaan mielipiteitään pelin toiminnallisuuksista ja mahdollisista ongelmista.

Näiden palautteiden perusteella on muutettu esimerkiksi pelihahmojen liikkuvuusnopeutta, sekä estetty pelaajien törmäily toisiinsa. Kuvassa 28 esitetään miten pelaajan ja tekoälyhahmojen törmäily estetään IgnoreLayerCollision-funktiolla. IgnoreLayerCollision-funktiolle määritetään mitkä layerit, eli kerrokset eivät voi törmätä toisiinsa. Pelaaja GameObjectille on määritetty layer 8 ja tekoälypelaajien GameObjecteille layer 9. Käytännössä tämä tarkoittaa sitä, että

törmäykset näiden layerien välillä on estetty ja hahmot voivat liikkua toistensa läpi.

```
// tällä ignorataan pelaaja ja enemyn collision
Physics.IgnoreLayerCollision (8, 9);
// tällä ignorataan enemyn ja enemyn collision
Physics.IgnoreLayerCollision (9, 9);
```

Kuva 28. IgnoreLayerCollision-funktion kutsu.

3.5 Julkaisualusta

Pelistä tehtiin vain Unity Web Player versio, joten sitä ei esimerkiksi julkaista mobiilialustalle tässä vaiheessa ennen mahdollista jatkokehitystä.
Mobiiliratkaisusta luovuttiin jo toteutusvaiheessa. Mobiiliversio olisi vaatinut paljon enemmän aikaa työn toteuttamiseen, sekä joitain pelin osia olisi jouduttu tekemään kokonaan uudelleen.

4 Toteutumattomat ideat ja ongelmat

Kaikkiin suunniteltuihin ideoihin ja ominaisuuksiin ei löytynyt toimivaa ratkaisua, joten ne jätettiin toteutettavasta versiosta pois. Yksi tärkeä ominaisuus, johon ei löytynyt ratkaisua: Kuutioiden tuhoaminen, joihin ei ole enää mahdollista päästä. Toinen toteuttamatta jäänyt ominaisuus oli pelin äänet, koska se oli suunniteltu tehtäväksi viimeisenä.

Suurin ongelma opinnäytetyön aikana oli tekoälyn liikkuminen pelialueella. Koska kenttä on muodostettu dynaamisesti, eikä staattisesti, niin ei voida käyttää Unityn tarjoamia NavMesh ja NavMeshAgent ominaisuuksia. Jos olisin löytänyt NavMesh ja NavMeshAgent ominaisuudet esitutkimusvaiheessa, niin olisin rakentanut kenttien muodostuksen staattisesti. Uskon, että tuolloin olisin säästänyt paljon aikaa ja vaivaa, jolloin olisin voinut keskittyä muihin ongelmiin enemmän. Nyt tekoäly saattaa jäädä jumittamaan tiettyjä kahta peräkkäistä kuutiota

tai se saattaa joissakin tilanteissa kävellä itse alas pelikentältä ja kuolla, joten täysin virheettömästi toimivaa tekoälyä en saanut toteutettua tähän peliin.

Toinen vastaan tullut ongelma oli Occlusion Culling ominaisuus, jota en voinut käyttää toteutuksessa samasta syystä kuin NavMeshiä tai NavMeshAgenttia. Jos olisin voinut käyttää Occlusion Cullingia, niin pelin suorituskyky olisi parantunut.

Unityn Asset Storesta löytyisi vaihtoehtoisia ratkaisuja tähän ongelmaan kuten SECTR VIS [31] tai InstantOC Dynamic Occlusion Culling + LOD [32]. Molemmat vaihtoehdot ovat maksullisia, joten näitä ei ollut mahdollista käyttää opinnäytetyössä.

5 Pohdinta

Omasta mielestäni opinnäytetyön aihe oli todella mielenkiintoinen ja itselle mieluinen. Pitkään erilaisia tietokonepelejä pelanneena oli erittäin mukava päästä tutustumaan prosessiin, joilla pelejä on luotu ja mitä kaikkea ne pitävät sisällään. Olen aikaisemmin tehnyt erilaisia pieniä pelejä ilman Unityä, mutta ne jäävät tämän projektin jalkoihin, niin ajankäytön kuin toiminnallisuuksien ja laajuudenkin suhteen.

Unityä en ollut käyttänyt aiemmin, joten sen käyttäminen oli täysin uusi aluevaltaus. Unity on omasta mielestäni mitä mainion työkalu pelinkehitykseen ja tulen myös käyttämään sitä jatkossa omissa projekteissa ja toivottavasti myös tulevaisuudessa työssäni. Koen että tämä opinnäytetyö on hyödyllinen omalle tulevaisuudelle ja ammatilliselle kasvulle.

Opinnäytetyö onnistui omasta mielestäni hyvin, vaikka joitakin alkuperäisessä suunnitelmassa määrittelemiäni ominaisuuksia jäikin puuttumaan. Opintosuunnitelmaani ei myöskään kuulunut yhtään kurssia peliohjelmointia, joten se toi li-

sää haastetta opinnäytetyön tekoon. Rajallisen ajan takia pelin suorituskyvyn optimointi ja peliäänien toteutus jäi pienemmälle prioriteetille.

Tulen luultavasti tekemään pelistä toisen version tulevaisuudessa. Tällä hetkellä ajatus olisi tehdä täysi mobiiliversio ainakin Androidille. Tämä vaatii kuitenkin vielä paljon tutkintaa ja luultavasti joudun tekemään koko pelin uudestaan alusta loppuun. Tällöin tulen kiinnittämään enemmän huomiota pelin teemaan, hahmoihin ja tekstuureihin.

Lähteet

1. Wikipedia. 2016. Unity Technologies. https://en.wikipedia.org/wiki/Unity_Technologies. 27.5.2016.
2. Brodkin, J. 2013. How Unity3D Became a Game-Development Beast. http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. 27.5.2016.
3. Unity. 2016. Multiplatform - Publish your game to over 10 platforms. http://unity3d.com/unity/multiplatform/. 25.5.2016.
4. Unite. 2007. https://unite.unity.com/archive/2007. 27.5.2016.
5. Unity. 2010. Unity Technologies Unveils Third Generation of Its Powerful Development Platform. https://unity3d.com/company/public-relations/news/unity-unveils-3rd-generation-platform-press. 27.5.2016.
6. Unity. 2016. Unity 4.0. https://unity3d.com/unity/whats-new/unity-4.0. 27.5.2016.
7. Unity. 2016. What's new in Unity 5.0. https://unity3d.com/unity/whats-new/unity-5.0. 27.5.2016.
8. Unity. 2016. Download Archive. https://unity3d.com/get-unity/download/archive. 27.5.2016.
9. Unity. 2016. Unity Store. https://store.unity3d.com/subscribe?currency=EUR. 25.5.2016.
10. Unity. 2016. Get Unity. https://unity3d.com/get-unity. 2.6.2016.
11. Unity. 2016. Manual: GameObject. http://docs.unity3d.com/Manual/class-GameObject.html. 27.5.2016.
12. Unity. 2016. Manual: Scenes. http://docs.unity3d.com/Manual/CreatingScenes.html. 27.5.2016.
13. Unity. 2016. Manual: Skybox. http://docs.unity3d.com/Manual/class-Skybox.html. 27.5.2016.
14. Unity. 2016. Manual: Materials, Shaders & Textures. http://docs.unity3d.com/Manual/Shaders.html. 27.5.2016
15. Unity. 2016. Editor. https://unity3d.com/unity/editor. 27.5.2016.
16. Aleksandr. 2014. Documentation, Unity Scripting Languages and You. http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/. 25.5.2016.
17. Wikipedia. 2016. C Sharp (Programming language). https://en.wikipedia.org/wiki/C_Sharp_(programming_language). 27.5.2016
18. Unity Community Wiki. 2014. UnityScript versus JavaScript http://wiki.unity3d.com/index.php?title=UnityScript_versus_JavaScript. 27.5.2016.
19. BOO-LANG. 2016. About Boo. http://boo-lang.org/Main.aspx?TAB=About%20Boo. 27.5.2016.
20. Unity. 2016. Manual: Building a NavMesh. http://docs.unity3d.com/Manual/nav-BuildingNavMesh.html. 22.5.2016.

21. Unity. 2016. Manual: Creating a NavMesh Agent. http://docs.unity3d.com/Manual/nav-CreateNavMeshAgent.html. 22.5.2016
22. Unity. 2016. Manual: Occlusion Culling. http://docs.unity3d.com/Manual/OcclusionCulling.html. 27.5.2016.
23. Wikipedia. 2016. Tekoäly. https://fi.wikipedia.org/wiki/Teko%C3%A4ly. 27.5.2016.
24. Ala-Mutka, K.;Rintala, M.;Savikko, V.;& Palviainen, J. (1996-2002). Ohjelmiston elinkaari. http://www.cs.tut.fi/etaopetus/titepk/luku21/elinkaari.html. 25.5.2016.
25. Unity. 2016. Learn. http://unity3d.com/learn. 2.6.2016.
26. Unity. 2016. Community. http://unity3d.com/community. 2.6.2016.
27. Unity. 2016. Manual: Unity Manual. http://docs.unity3d.com/Manual/index.html. 2.6.2016.
28. RKD. 2013. Sky5X One. https://www.assetstore.unity3d.com/en/#!/content/6332. 20.4.2016.
29. Goodtextures. 2016. Seamless Snow & Ice Textures. http://www.brusheezy.com/textures/20856-seamless-snow-ice-textures. 15.4.2016.
30. Gama, N. 2016. Google Fonts Exo 2. https://www.google.com/fonts/specimen/Exo+2. 25.4.2016.
31. Make Code Now!. 2016. SECTR VIS. https://www.assetstore.unity3d.com/en/#!/content/15353. 25.5.2016.
32. Frenchfaso Indie Apps. 2013. InstantOC Dynamic Occlusion Culling + LOD. https://www.assetstore.unity3d.com/en/#!/content/6391. 25.5.2016.

Alkuperäinen luonnos, Jouni Laitinen

Gameplay

Setup

Game takes place on symmetrical levels built for up to 4 players. Each player represents one color
(red, blue, green, yellow).
Image 1 shows a starting setup for a relatively difficult level.

Image 1. Symmetrical level with player starting points.

Basics

Each player controls his or her own character. Characters can only move on top of the boxes. As
characters move around, every box they step on is painted to their color (see Image 2). If a player
moves on a box that has already been painted by another player, the box will re-painted by the new
player.
At any time of their choosing, the player can choose to detonate all the boxes they currently have
painted (See Images 2 and 3). The player gains one point for every box they detonate.

Image 2. Boxes are painted to red player's color as they move around.

**Image 3. Player has detonated the 7 red boxes they had painted in Image 2. As a result, player
gains 7 points and falls down one level.**

After triggering the detonation, the player either falls down one level (See Image 3) as the box
below them also explodes OR in case there is nothing below the current layer, the player will
automatically jump to safety after explosion.
If player tries to move to an area that is higher or lower than their current level, the player will
automatically perform the required jump, up to 3 levels of height difference (Image 4).

Image 4. Player moves up one level. The jump is performed automatically.

Boxes and Support

Boxes cannot exist without support. Image 5 shows a situation where the player has blown up boxes
that support the segment on the right side of the image. The boxes that have lost their support will
fall down until they find new support (Image 6).
A box is considered to be part of the same segment if they are connected by a side or the upper edge
of the lower box. (needs visual clarification)

Image 5. Player has blown up another 7 boxes. A segment on right has lost its support.

Image 6. The segment fell down until it found new support (connected by edge)

If a player blows up the boxes in such a way that a segment loses all of its support, the segment is
considered **disconnected** and the player who disconnected the segment will score points based on
the number of boxes in that segment (See Image 8.)

Alkuperäinen luonnos, Jouni Laitinen

Moving

Player characters shouldn't move too fast to give everyone a chance to have a strategy or even try to counter other player's strategy. Movement shouldn't be too sluggish either as the game should feel fast-paced and hectic.

If a player tries to move to a box occupied by another player character, the player character on the box is knocked back to the previous box they were on. If the player cannot be knocked back, they'll stay on the box.

Falling off the map

A player will fall off the map if:

- Any player **disconnects** a segment the player is standing on.
- The box player is standing on is blown up and the player cannot jump to safety.

If the player falls off, they will suffer a short time penalty (3-5 second respawn timer) and then respawn in the center of the level (considered to be the safest area in entire game).

Goal

Each player attempts to score as many points as possible by blowing up boxes and disconnecting segments. Each level should have a time limit (should be determined per level?). Matches in medium sized levels should last about 45-90 seconds (requires testing).

The level ends if:

- Time runs out
- All the boxes connected to central pillar (see Image 9) have been destroyed. (and since boxes can't exist without support, this pretty much means all boxes have been destroyed…)

Image 7. Player moves up. Automatic jumps again.

Image 8. The segment on left lost all support. Red player scores 5 extra points.

Image 9. The level is supported by a central pillar. Any segments disconnected from the central pillar will fall down.

Gameplay objects

Characters

- If boxes take form of stone cubes / wooden boxes.. a human(oid) miner character with explosives might work? Not the most original choice though… could use ideas here ·
- Each player moves their own character
- Characters can also be AI controlled bots

Boxes

- Various visual appearances but same behavior
- Needs support or will become **disconnected.**
- Can be blown up by the player.
- Each blown up or disconnected box is worth 1 point.

Power-ups

Power-ups could spawn randomly on boxes but they shouldn't be anything too unbalancing.

Alkuperäinen luonnos, Jouni Laitinen

Some potential examples:
- Minor speed boost for 3-5 seconds.
- An arrow (when picked up, paints all boxes with your color in a line in the direction of the
arrow)
- A bomb that blows up 3x3 area around it (could be dangerous too but destroyed boxes give
points so…)
- A collectable item that gives bonus points (3 or 5 or so).

Design & Programming challenges

Camera

Keeping the player in view and giving the player the best possible view of the stage will be
challenging. This will be especially difficult on mobile platforms, as tap movement will become
much harder if the camera moves around too fast.

Visualizing the 'painting'

If boxes are made of stone / wood / whatever… tinting them to specific color might look pretty bad.
What would be the best way to visualize colored boxes?

Pathfinding

Tap-movement on mobile requires some sort of simple pathfinding.

AI and Bots

Decent AI is required for interesting single player mode or to fill up empty slots when playing with
2-3 players.

Multiplayer

Game should support 'local' multiplayer with WiFi/Bluetooth.
Global multiplayer/matchmaking lobby would require a server application as well.

Level design

Levels should be symmetrical in regards of player starting points to ensure fairness. More difficult
levels should include floating sections that can be dropped down to a lower level or entirely off the
game. Width / length of the levels should be uneven number as that will allow the level designer to
place starting points symmetrically in center of the level.
Levels could be a bit larger than what is shown in the pictures of this document. Average match in a
level should last for 1-2 minutes at most, but the level should also stay intact long enough for
players to actually have a strategy of sorts.
Easier levels should consist of 2-3 layers and generally the bottom layers should be bigger than the
top layers (pyramid shape). These levels will still be challenging despite the large bottom layer as
only the center of the area is supported by the central pillar. Disconnecting any section from the

Alkuperäinen luonnos, Jouni Laitinen

central pillar will cause that section to fall down.

Image 10. Easy level with 2 layers and pyramid shape

More difficult levels could be up to 4 layers deep (might become too difficult to handle camera and clarity beyond 4 layers, as well as jump related issues…). Building the difficult levels as "upsidedown pyramids" will make them more hectic towards the end and there will be more floating sections to drop down. As a downside, upside-down pyramids can make it harder for players to actually see themselves / boxes blocked by boxes on upper layers.

Art & Sounds

Visuals

Level art consists mainly of boxes. There could be several themes of boxes: wood, stone or even see-through glass/ice. Each theme should have several boxes with minor differences (cracks in stone, different colored wooden boxes, boxes with grass growing on them, etc) to keep the visuals appealing and non-repetitive.

Additional visuals of levels could include floating boxes with items/plants fitting the theme. Use of decals on boxes could also be considered although there may be some issues with that.

The background of levels consists of a large skysphere/cube with a panoramic texture. Additional visuals might include moving clouds, giant trees or whatever seems to fit the theme.

Sound

Nothing too special here. Explosion/popping sounds for blowing up the boxes. Generic blings & beeps for scoring. Menu sound effects. Background music.

User interface and Controls

Main menu

- Level selection
- Selecting the number of players / bots
- Selecting the multiplayer type (WiFi, Bluetooth, Global, etc..)

Gameplay UI

- Score display for all 4 players
- Detonate button (unless detonation is triggered by holding down finger on your own character… testing required to see how intuitive that would be…)

Controls

- Tap to move on mobile
- Either a separate "Detonate" button or tap and hold on your own character to Detonate

Monetization (pfft)

In-app purchases (eww)

This type of competitive game is somewhat difficult to monetize properly without giving players unfair advantages over other players. This part would also require a server software/hardware to store and serve player data.

Alkuperäinen luonnos, Jouni Laitinen

Currency
Players' score in every match is added to their total currency. Currency can also be purchased.
Currency can be used to permanently upgrade your character and to buy cosmetic changes.
Upgrades
- 'Equippable' one-use-per-match special abilities that could turn the game to your favor (eventually everyone would have at least one special ability so not too unfair?)
  - Regenerate boxes, area effect explosion around yourself, paint some shape / area, ondemand
speed boost, etc..
- Permanent speed boost (unfair)
- What else?
Cosmetics
- Since the game is top-down… Hats / Hairstyles
- Other accessories