Joni Uusimäki

Mobiilipelin suunnittelu, toteutus ja testaus Unity3D-pelimoottorilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

8.11.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Joni Uusimäki Mobiilipelin suunnittelu, toteutus ja testaus Unity3D-pelimoottorilla 35 sivua 8.11.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Lehtori Juha Kämäri

Opinnäytetyön tavoitteena on näyttää eri työvaiheet, jotka kuuluvat mobiilipelin suunnittelemiseen, toteuttamiseen ja testaamiseen Unity3D-pelimoottorilla. Työn toimeksiantajana toimi Marimo Games. Työssä keskitytään pääasiassa Kosunin pelin pelattavuuden kehittämiseen ja optimoimiseen.

Peli kehitettiin C#-ohjelmointikielellä Unity-ympäristössä. Pilveen tallennettavien tietojen kanssa käytettiin GameSparks-sivuston palveluita.

Työn aikana Unity3D-pelimoottori todettiin monipuoliseksi ja toimivaksi alustaksi mobiilipelien kehittämiseen. Mobiilipelin toteuttaminen ja optimointi onnistuivat ongelmitta seuraamalla Unityn tarjoamaa dokumentaatiota. Pelin testaus suoritettiin iOS-käyttöjärjestelmässä.

Avainsanat	pelinkehitys, Unity, mobiilipeli

Author(s) Title Number of Pages Date	Joni Uusimäki Design, Implementation and Testing of Mobile Game on Unity3D Game Gngine 35 pages 8 November 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Development
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Juha Kämäri, Senior Lecturer

The purpose of this thesis was to show all the different steps in planning, executing and testing a mobile game on the Unity3D game engine. The thesis was commissioned by Marimo Games. In this thesis the focus is mainly on the creating and optimizing of the gameplay in the game Kosunin.

The game was developed using the C# programming language in the Unity environment. For uploading game information in to the cloud services provided by GameSparks were used.

During this project the Unity3D game engine was proven a versatile and practical platform for mobile game development. The executing and testing of the mobile game worked out well as long as Unitys provided documentation was followed. The testing of the mobile game was done on the iOS operating system.

Keywords	game development, Unity, mobile game

Sisällys

Lyhenteet

Asset	Mikä tahansa pelissä käytettävä resurssi. Esimerkiksi tekstuuri, 3D-malli tai musiikkititiedosto.
AssetBundle	Ryhmä assetteja, jotka voidaan hakea ja instantioida pelitiedostojen ulkopuolelta.
Cooldown	Aika jonka pelihahmon on odotettava, jotta hän voi käyttää erikoistaitoaan uudestaan.
GameObject	Unity3D-pelimoottorin toiminto. Skeneen lisätty assetti.
Prefab	Tiedosto joka sisältää joukon GameObjecteja.
Shuriken	Pelissä esiintyvien vihollisten käyttämä ase. Japanilainen heittotähti.
Skene	Ryhmä GameObjecteja jotka esittävät yhtä pelikohtausta.
Skripti	Ohjelmakoodia sisältävä komponentti joka liitetään GameObjectiin.

1 Johdanto

Kosunin on tasohyppelypeli, jossa on tarkoituksena väistää vihollisia samalla keräten kolikoita matkan varrella. Peli on alagenreä “runner”, joka tarkoittaa, että hahmo jatkaa juoksemistaan jatkuvasti. Hahmoa ei voi pysäyttää missään vaiheessa, sitä voit ainoastaan ohjata hahmoasi hyppäämällä. Peli sisältää myös roolipelielementtejä kuten hahmon kehitystä ja lisätarvikkeiden rakentamista.

Kenttien läpäiseminen antaa pelaajalle kokemuspisteitä ja kolikoita. Kokemuspisteillä pelaaja voi kehittää omia asujaan ja tehdä niistä voimakkaampia. Kolikoilla pelaaja voi ostaa pelin sisäisestä kaupasta erilaisia lisätarvikkeita sekä uusia hahmoja. Kuvassa 1 näemme yhden pelin ensimmäisistä kentistä.

Peli on suunniteltu mobiililaitteille, ja kyseisessä projektissa sen käyttöliittymänä toimii iOS. Työssä keskitytään lähinnä pelin pelattavuutta käsitteleviin aiheisiin. Ensin selitetään, miten pelin maailma on suunniteltu, sitten keskitytään siihen, miten se on toteutettu. Lopuksi pohdimme vielä, miten peliä voidaan optimoida Unity-pelimoottorissa. Tämä työ ei käsittele käyttöliittymän suunnittelua tai pelin graafista suunnittelua.

Kuva 1. Vasemmalla oleva hahmo on pelaaja, joka on juuri käyttänyt jänis-asun erikoistaidon. Oikealla joukko vihollisen shurikeneita ja muutama kolikko.

2 Projektin työkalut

Keskeisimmät työkalut projektissamme ovat Unity3D ja GameSparks.

2.1 Unity 5

Unity3D on Unity Techonologiesin kehittämä pelimoottori, joka on erityisen tunnettu sen kyvystä luoda alustariippumattomia 2D- ja 3D-pelejä. Unityllä voi kehittää pelejä niin PC:lle, videopelikonsoleille, mobiililaitteille kuin nettisivuille. Unityn ohjelmointiympäristö toimii Mono:lla, joka on avoimen lähdekoodin versio .Net Frameworkista. Unity itse on kirjoitettu C++:lla. (Brodkin 2013.)

Kosunin-pelin luomisessa käytettiin Unity 5.1 -versiota. Unity 4 toi monia parannuksia ja uusia ominaisuuksia mukanaan. Unity 5 parantaa kokemusta entisestään tarjoamalla edullisen ja yksinkertaisen pelimoottorin. Unityn suosio on kasvanut moninkertaisesti parin viime vuoden aikana. (Studica 2015.)

Unity 5:stä lähtien ei pelimoottorin käyttämiseen tarvitse enää maksullista lisenssiä. Lisenssimaksun poistaminen pelimoottorin perusominaisuuksista tarkoittaa sitä, että toiminnot, jotka ennen maksoivat yhteensä noin 700 euroa, eivät enää maksa mitään. (Etherington 2013.)

Unityn työkalujen avulla voidaan luoda halvemmalla ja pienemmällä työllä hyvin monimutkaisiakin interaktiivisia tuotoksia (Studica 2015.). Unityn kanssa tuleva integroitu ohjelmistoympäristö (IDE) on MonoDevelop. Skriptien ohjelmoinnissa voidaan käyttää JavaScriptiä, C#:aa tai Boo:ta. (Aleksandr 2014.)

Unityn avulla voi luoda pelejä yli viidelletoista eri alustalle. Tuettuja alustoja ovat muun muassa BlackBerry 10, Windows Phone 8, Windows, OS X, Android, iOS, Unity Web Player (myös Facebook), PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, Nintendo 3DS, Wii ja Wii U. (Unity Technologies 2015a.)

2.2 GameSparks

GameSparksin on hyvin monimuotoinen työkalu, jonka avulla voidaan käsitellä pelaajakohtaisia tietoja pilvessä. Rajapinnan avulla voidaan autentikoida pelaajat ja laitteet sekä tallentaa heidän tilikohtaiset tiedot ja tulokset pilveen. Palvelun avulla voimme myös luoda kaverilistoja ja pistetaulukoita. Myös erilaiset push notifikaatiot kuten toisten pelaajien haastaminen ovat mahdollisia. (GameSparks 2015.)

Pilvessä ajettava koodi toimii JavaScriptillä. Tiedot tallentuvat JSON-muodossa NoSQL-taulukkoihin. Unityn kanssa GameSparksia voidaan käyttää lisäämällä GameSparks SDK -rajapinta peliprojektiin. GameSparksin pilvipuolen skriptejä voidaan kutsua Unityssä suoraan skripteistä käsin. (GameSparks 2015.)

GameSparksia käytetään pelissä tietojen ja muuttujien tallentamisessa pilveen. Hyödyt siinä, että tiedot löytyvät pilvestä, ovat muun muassa mahdollisuus päivittää pelikohtaisia tietoja ilman softapäivityksiä ja mahdollisuus käyttäjille vaihtaa mobiililaitetta menettämättä käyttäjäkohtaisia tietojaan. Paketoidut tiedot latautuvat suoraan GameSparksin palvelimilta. Ainoa huono puoli tässä on se, että pelin pelaaminen vaatii internetyhteyden.

3 Pelin ominaisuudet

Kosunin-pelin kanssa tähdättiin nopeatempoiseen ja sulavaan pelattavuuteen. Pelattavuus on arcade-tyylistä ja helppoa ymmärtää. Pelimekaniikat ovat yksinkertaisia ja nopeasti opittavissa.

Pelin roolipelielementit luovat monipuolisuutta ja uudelleenpeluuarvoa. Kehittämällä pelin eri hahmoja voi tehokkaammin ja nopeammin päihittää kenttiä ja saavuttaa uusia huippupisteitä.

Kentistä kerätyillä kolikoilla voi ostaa pelin sisäisestä kaupasta uusia tarvikkeita ja asuja. Pelin erilaiset asut luovat monipuolisuutta pelattavuuteen tarjoamalla uusia pelihahmoja täysin erilaisilla ominaisuuksilla ja erikoistaidoilla.

3.1 Kentät ja maat

Pelin eri kentät jaettiin kahdeksaan eri maahan. Jokaisella maalla on oma teemansa, ulkonäkönsä ja kenttänsä. Eri teemat luovat vaihtelevuutta pelattavuuteen ja pitävät pelaajat kiinnostuneina.

Kenttiä on kahta eri tyyppiä: single plane ja double plane. Single plane -kentissä on vain lattia, jota pitkin pelaaja voi juosta. Double plane -kentissä on lattia ja katto, jota pitkin pelaaja voi juosta. Kattoa pitkin juostessa pelaaja on käännetty ylösalaisin. Tämän lisäksi vihollisten hyökkäykset voivat kentästä riippuen tulla joko hahmon edestä tai hahmon takaa. Jokaisella kentällä on omat säännöt, esim. kentän pituus tai kentän tyyppi, ja omat kombinaatiot vihollisten hyökkäyskuvioita.

3.2 Kenttien rakenne

Pelimoodista riippumatta kenttien rakenne pysyy aikalailla samana. Suurimmat erot löytyvät kentistä, jotka sisältävät loppuvastuksen.

Kampanjan kentissä ja endless-pelimoodin kentissä rakenne on hyvin samanlainen. Kentän alusta loppuun tuodaan hyökkäyskuvioita ja kolikkojoukkoja peliin yksi toisen perään. Kampanjan kentissä vihollisjoukot ovat ennalta määrättyjä ja endless-pelimoodissa ne ovat satunnaisesti arvottuja. Kolikkojoukot ovat aina satunnaisesti arvottuja.

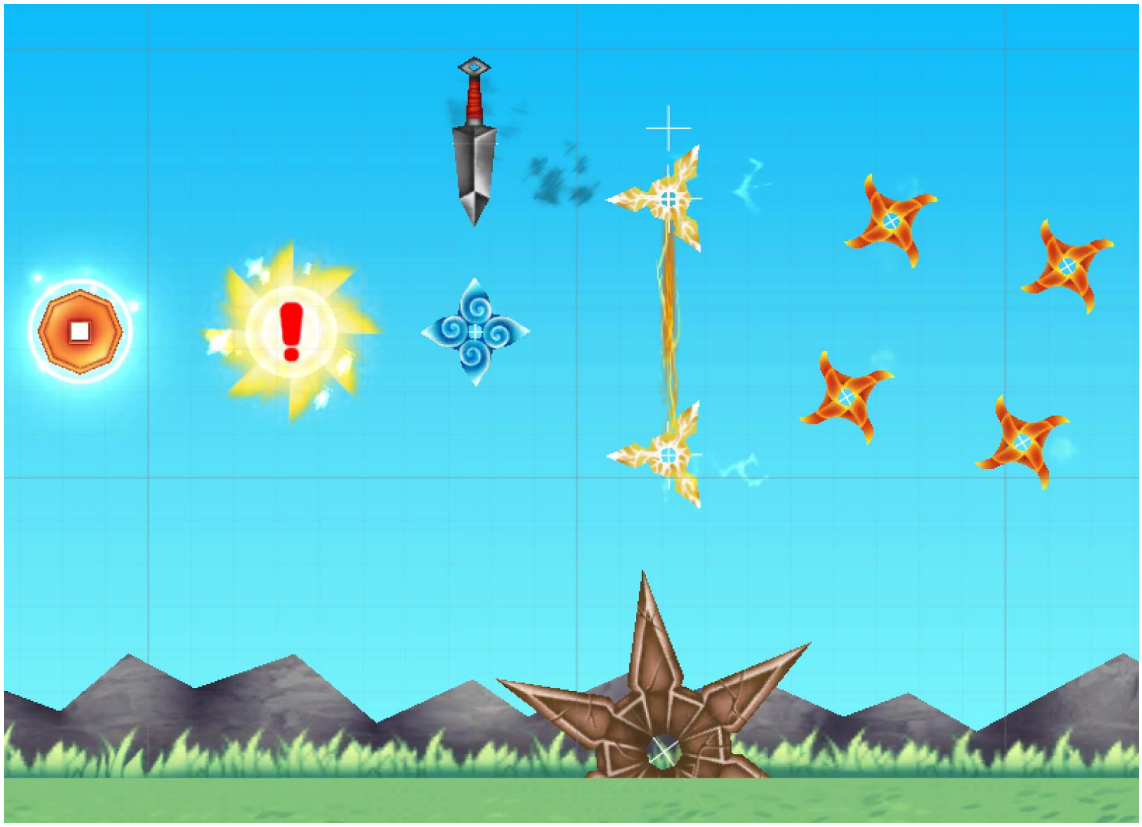
Kentän alkaessa kameran näkemä alue on vielä vihollisista tyhjä, mutta heti pelaajan aloitettua juoksemisen, tuodaan jo ensimmäinen hyökkäyskuvio ja kolikkokuvio kenttään. Ensimmäisen hyökkäyskuvion poistuttua näkyvistä tuodaan seuraava hyökkäyskuvio kenttään. Vihollisjoukot ovat eripituisia ja kokoisia. Hyökkäyskuvioiden poistaminen pelistä perustuu niiden ankkurin positioon. Hyökkäyskuvio poistetaan, kun sen ankkuri on kameran näkemän alueen ulkopuolella. Kolikkojoukot ovat aina samanpituisia, ja myös niiden poistaminen tapahtuu samalla tavalla. Kolikkokuvioiden samanpituisuus perustuu siihen, että vihollisten hyökkäyskuvioiden välinen tauko on aina ajaltaan samanpituinen.

Kentät, joissa on loppuvastus, alkavat loppuvastuksen astuttua pelialueelle ja kameran näkyviin. Loppuvastuksilla on jokaisella kolme eri hyökkäystä, ja niiden järjestys on satunnaisesti arvottu. Mitä lähempänä kuolemaa loppuvastus on, sitä suuremmalla todennäköisyydellä hän käyttää vaikeampia hyökkäyksiä. Kenttä loppuu, kun loppuvastus kuolee. Boss rush -pelimoodissa kenttä loppuu, kun kaikki loppuvastukset ovat kuolleet.

3.3 Shurikenit

Pelin vihollisten hyökkäyskuviot ovat ryhmä shurikeneita, jotka liikkuvat pelissä pelaajaa kohti. Shurikeneita on monia erilaisia. Eri vihollisten hyökkäyskuviot luotiin käsin ja niitä löytyy yli sataviisikymmentä erilaista. Vihollisten hyökkäyskuvioista tehtiin mahdollisimman monipuolisia ja erilaisia, jotta pelattavuus ei toistaisi liikaa itseään.

Hyökkäyskuviot muodostuvat vihollisten eri shurikeneiden lisäksi kolikoista, vihollisytimistä ja energiapalloista. Nämä kaikki voidaan nähdä kuvassa 2. Shurikeneita on viittä eri tyyppiä: normaali, tikari, salama, tuli ja maa. Normaalit shurikenit liikkuvat suoraa viivaa eteenpäin. Tikarit leijuvat vasemmalta oikealle tai ylhäältä alas. Salama-shurikenit luovat sähköä toistensa välille. Tuli-shurikenit pyörivät ympyrää joko myötäpäivään tai vastapäivään. Maa-shurikenit ovat suurempi versio normaalista shurikenista, jotka kulkevat joko lattiaa tai kattoa pitkin. Hyökkäyskuvioissa esiintyvät vihollisytimet ovat palloja, jotka pelaaja voi ottaa kiinni tuhotakseen koko vihollisen hyökkäyskuvion yhdellä iskulla. Energiapallot ovat palloja, jotka esiintyvät kentissä, joissa on loppuvastus. Keräämällä energiapalloja pelaaja voi vahingoittaa loppuvastusta.



Kuva 2. Ylhäältä ylös ja vasemmalta oikealle: Tikari, kolikko, vihollisydin, normaali shuriken, salama-shuriken, neljä tuli-shurikenia ja maa-shuriken.

3.4 Liikkuvuus

Pelin kentissä hahmo jatkaa automaattisesti juoksemistaan kentän alusta loppuun asti. Matkalla kentän loppuun pelaajan on väistettävä vastaantulevia vihollisten shurikeneita samalla keräten kolikoita. Shurikeneita väistetään hyppäämällä tai käyttämällä hahmojen erikoistaitoja.

Hahmon hyppykorkeus vaihtelee kentän tyypin mukaan. Single plane -kentissä hahmon hyppykorkeus on kentän puoleen väliin. Double plane -kentissä hyppykorkeus on aavistuksen korkeampi, koska pelaaja voivat halutessaan hypätä ylös kentän kattoon.

Normaalin hypyn lisäksi pelaaja voi myös tehdä jetstep-liikkeen. Jetstep on nopea alaspäinhyppy ilmasta vinosti alas lattiaan tai kattoon. Hypyn ja jetstepin lisäksi pelaaja voi myös syöksyä ilmassa neljään eri suuntaan: ylös, alas, oikealle tai vasemmalle. Pelaaja

voi hypyn jälkeen syöksyä neljä kertaa peräkkäin ilmassa. Koskettuaan taas maata pelaaja saa taas uudet neljä syöksyä käyttöönsä. Tyypillinen väistöliike voi esimerkiksi olla hyppy ylös, syöksy oikealle ja jetstep alas.

3.5 Hahmot

Pelaaja saa pelissä erikoistaitonsa pukemalla erilaisia eläinasuja päälleen. Jokaisella eläinasulla on omat taitonsa ja vahvuutensa. Antamalla pelaajalle erilaisia pelihahmoja käyttöön luo uusia mahdollisuuksia ja tapoja läpäistä kenttiä. Jotkin kentät saattavat olla helpompia läpäistä käyttämällä tiettyä eläinasua tietyllä erikoistaidolla. Erilaiset pelihahmot tekevät pelattavuudesta monipuolisempaa ja vaihtelevampaa. Uusia asuja pelaaja voi hankkia keräämällä loppuvastusten pudottamia osia ja rakentamalla niistä uusia eläinasuja. Jokaisella asulla ovat myös omat kehityspisteensä jotka jakautuvat neljään eri ryhmään. Ryhmiä ovat: HP eli kestävyyspiteet, STR eli voimakkuus, SPD eli nopeus ja CD eli cooldown. Cooldown määrittää, kuinka kauan pelaajan on odotettava, kunnes hän voi käyttää pelihahmonsa erikoistaitoa uudestaan.

Esimerkiksi jänisasu, jolla pelaaja aloittaa, on nopeampi juoksemaan kuin mikään muu pelin asu. Jäniksen erikoistaidon avulla pelaaja voi luoda suojan ympärilleen samalla juosten nopeampaa. Nopeampi juoksutahti pysyy siihen asti, kunnes suoja rikkoutuu.

Muita asuja pelissä ovat esimerkiksi panda, joka voi heittää bambun oksia, härkä, joka voi puskea vihollisensa pois tieltään, kissa, joka voi muuttaa viholliset kolikoiksi, lammas, joka voi luoda suojan ympärilleen, ja lohikäärme, joka voi ampua tulta ympärilleen. Erikoistaitojen lisäksi jokaisella eläinasulla on omat kehityspisteensä ja polkunsa hahmonkehityksessä. Yksi hahmo voi esimerkiksi olla erityisen nopea juoksemaan ja toinen erityisen kestävä. Lisäksi hahmoilla kuten lohikäärmeellä on erityisen voimakas erikoistaito, jolloin hahmon muut tavallisemmat voimat ovat aavistuksen heikompia kuin muilla pelihahmoilla.

3.6 Loppuvastukset

Noin joka toisen maan lopussa pelaaja kohtaa loppuvastuksen. Ninusok-nimellä kulkevat loppuvastukset ovat naamioituneet eläinasuihin ja niillä on asujen antamat erikoistaidot.

Jokaisella loppuvastuksella on omat hyökkäyskuviot ja erikoishyökkäykset, kuten laseria tai kivien heittelyä. Loppuvastus tuhotaan keräämällä energiapalloja viholliskuvioiden keskeltä. Kun olet kerännyt kolme energiapalloa, niin voit hyökätä ja tehdä kokemustasostasi riippuen tietty määrä tuhoa loppuvastukseen. Loppuvastus kuolee, kun pelaaja on tehnyt häneen tarpeeksi tuhoa.

Loppuvastuksen tappamisesta saa pelaaja palkinnoksi kokemuspisteitä ja yhden kolmesta tarvittavasta osasta vastuksen käyttämän eläinasun rakentamiseen. Tämän lisäksi pelaajalle annetaan lupa siirtyä kampanjan seuraavaan maahan. Kerättyään kaikki kolme asun osaa voi pelaaja luoda loppuvastuksen käyttämän asun itselleen.

3.7 Pelimoodit

Luomalla monia erilaisia pelimoodeja voidaan lisätä pelin uudelleenpelattavuusarvoa.

3.7.1 Kampanja

Kampanja on pelin pääpelimoodi. Pelimoodin tarkoituksena on läpäistä pelin kaikki kentät yksi kerrallaan. Kampanjan eri kentät on jaettu kahdeksaan eri maahan. Jokaisessa maassa on noin kymmenen erilaista kenttää. Mitä pidemmälle kampanjassa pelaaja etenee, sitä vaikeammaksi kentät muuttuvat. Pelin muut pelimoodit avataan etenemällä kampanjassa tarvittavan pitkälle. Jokainen kampanjan kenttä on käsintehty ja ainutlaatuinen.

3.7.2 Endless mode

Endless mode on pelimoodi, jossa pelaaja ottaa vastaan loputtoman määrän vihollisia. Pelimoodin tarkoituksena on koittaa selviytyä hengissä niin pitkään kuin mahdollista. Vastaan tulevien hyökkäyskuvioiden määrä, vaikeustaso ja järjestys ovat satunnaisesti arvottuja.

Endless mode -kentät ovat pelin alussa lukittuina. Kampanjan jokaisen maan viidennes kenttää avaa uuden sen maan teemaa mukailevan endless mode -kentän.

Erilaisia endless mode -kenttiä on kahdeksan, yksi jokaista pelin maata kohti. Jokaisella eri endless moden kentällä on oma teemansa. Yksi voi olla single plane –tyylinen, kun taas toinen on double plane -tyylinen. Endless mode -kentät ovat helppo ja nopea tapa ansaita kokemuspisteitä ja kolikoita tavallisten kenttien ulkopuolelta.

3.7.3 Boss Rush

Boss rush on pelimoodi, joka avataan pelaajalle hänen läpäistyään koko kampanjan. Boss rush -pelimoodi on kaikista pelimoodeista haastavin. Se on yksi haaste lisää pelaajalle hänen tehtyään jo kaiken muun pelissä. Pelaajan läpäistyä boss rush -pelimoodin hänelle annetaan palkinnoksi saavutus, kolikoita ja kokemuspisteitä.

Boss rush -pelimoodin tarkoituksena on tappaa kaikki kampanjassa vastaan tulleet loppuvastukset peräkkäin, alusta loppuun, yhteen vetoon. Loppuvastuksilla on samat hyökkäykset ja samat ominaisuudet kuin kampanjassa. Ainoa ero on, että nyt pelaajalle ei anneta hengähdystaukoa taisteluiden välissä.

3.7.4 Tutoriaali

Tutoriaalin tarkoituksena on esitellä peli ja pelin ominaisuudet pelaajalle. Kun uusi käyttäjä rekisteröityy ensimmäistä kertaa, niin hänet viedään automaattisesti tutoriaalikenttään. Tutoriaalikenttään voi tulla uudestaan myös myöhemmin pelissä.

Tutoriaalissa näytetään pelaajalle ohjeita, missä kerrotaan ohjeita pelin pelaamiseen. Ohjeita kuten miten pelaaja voi ohjata hahmoaan ja miten hän voi oikeaoppisesti väistää vihollisten hyökkäyskuvioita. Tutoriaalikenttä suunniteltiin helpoksi mutta opettavaiseksi. Vastaan tulevat hyökkäyskuviot ovat yksinkertaisia, mutta niiden ohi pääsee vain tekemällä tietynlaiset väistöliikkeet. Jos ei pelaaja onnistu väistöliikkeiden tekemisessä, niin sama hyökkäyskuvio tulee uudestaan vastaan, jotta pelaaja voi harjoitella väistämistä uudestaan. Kentän läpäistyä pelaajalle näytetään vielä nopeasti ohjeita pelin valikoista ja niiden ominaisuuksista.

3.8 Pisteytys ja pistetaulukko

Jokaisen kentän läpäisystä pelaaja ansaitsee pisteitä ja päätyy kentän pistetaulukkoon. Pisteitä antavat muun muassa kentän kesto, kentän pituus, hyppyjen määrä, tuhottujen vihollisten määrä ja kolikoiden kerätty määrä. Pisteet kertovat pelaajalle, kuinka hyvin hän pärjäsi kentässä. Jokaisella kentällä on pistetaulukkoja kaksi, globaali ja kavereiden keskeinen.

Joka viikko peliin ilmestyy uusia tapahtumia. Tapahtumat ovat tietynlaisia haasteita missä pelaajat voivat kilpailla keskenään saavuttaakseen parhaan mahdollisen pistemäärän. Jokaisella tapahtumalla on tietyt vaatimukset, esimerkiksi kenttä, jonka pitää läpäistä tiettyä asua käyttäen. Parhaan pistemäärän saavuttanut pelaaja palkitaan, kun tapahtuma on ohi.

Tapahtumien lisäksi pelaajat voivat haastaa kavereitaan. Haastaja voi valita, millä kentällä pelataan ja millä panoksella. Panoksena toimivat kolikot. Kummankin pelaajan hyväksyttyä haasteen he kummatkin pelaavat kentän läpi ja yrittävät saavuttaa mahdollisimman suuren pistetuloksen. Paremman pistetuloksen saavuttanut pelaaja saa palkinnoksi itselleen kummankin asettaman panoksen.

3.9 Saavutukset

Saavutukset ovat pelistä löytyviä haasteita, joita suorittamalla pelaajat voivat ansaita kolikoita ja kokemuspisteitä. Jokaisen saavutuksen pelaaja voi ansaita vain kerran. Saavutuksien tarkoituksena on lisätä monipuolisuutta peliin antamalla pelaajille uusia haasteita ja kokemuksia. Tehtävät, joita pelaajan on suoritettava saavutuksien ansaitsemiseksi liittyvät kaikki jollain tapaa pelattavuuteen. Esimerkiksi alkupelin saavutuksia ovat “tapa pelin ensimmäinen bossi” ja “tuhoa 50 vihollista”.

4 Pelattavuuden tasapainottelu

Jos peliltä toivotaan uudelleenpelattavuusarvoa, niin sen pelattavuus on oltava hyvin tasapainotettu. Miten peli on tasapainotettu vaikuttaa suuresti siihen, mitä päätöksiä pe-

laajat tekevät pelatessaan peliä. Huonosti tasapainotetussa pelissä jotkin pelityylit saattavat olla muita pelityylejä huomattavasti kannattavampia. Ne tekevät muista pelityyleistä pelaajalle turhia. Pelaajien mielenkiinto peliin on pysyttävä vakaana pelin loppuun asti. Pelaajille on jatkuvasti esitettävä uusia haasteita. (Burgun 2011.)

4.1 Kentät

Kampanjapelimoodi on suunniteltu niin, että sen ensimmäiset kentät ovat helppoja ja opettavaisia. Pelaajille annetaan mahdollisuus oppia pelin mekaniikat kunnolla ennen vaikeampien hyökkäyskuvioiden esittelemistä. Mitä pidemmälle pelissä kampanjassa pelaaja pääsee, sitä vaikeammaksi kenttien eri hyökkäyskuviot muuttuvat. Hyökkäyskuvioiden vaikeutumisen lisäksi myös vihollisten hyökkäysnopeus nousee myöhemmissä kampanjan maissa. Mitä nopeammalla nopeudella hyökkäyskuvio tulee pelaajaa vastaan, sitä vaikeampaa sen väistäminen on.

Kampanjan ensimmäinen maa sisältää pelkästään single plane -kenttiä, kampanjan toinen maa taas sisältää pelkästään double plane -kenttiä ja kampanjan kolmas maa pelkästään toisin päin käännettyjä kenttiä. Tällä tavoin pelaajille annetaan mahdollisuus tutustua jokaiseen erilaiseen kenttätyyppiin yksi kerrallaan rauhassa. Neljännestä maasta eteenpäin eri maat käyttävät jokaista kolmea eri kenttätyyppiä sekaisin.

Endless moden tasapainottaminen oli kaikista yksinkertaisinta. Mitä pitemmälle endless modessa pelaaja juoksee, sitä vaikeammaksi se muuttuu. Vaikeutta lisäämme nostamalla vastaan tulevien hyökkäyskuvioiden vaikeustasoa ja niiden hyökkäysnopeutta.

4.2 Hyökkäyskuviot

Eri pelin hyökkäyskuviot luotiin ja tasapainotettiin ensin silmämääräisesti. Hyökkäyskuvioiden testaamisen jälkeen tehtiin kuvioihin tarvittaessa lisämuokkauksia. Tasapainottamisessa haluttiin, että olisi mahdollista ohittaa jokaisen vihollisen hyökkäyskuvion ottamatta osumaa. Lisäksi kolikoiden avulla tehtiin tietynlaisia polkuja hyökkäyskuvioiden läpi, jotka näyttävät pelaajalle yhden mahdollisista reiteistä kuvion ohittamiseen. Kolikoita seuraamalla ja hyppäämällä oikein pelaaja voi löytää polun, joka antaa eniten kolikoita laittamatta pelaajaa vaaraan. Eri hyökkäyskuviot jaettiin viiteen eri vaikeustasoon.

Vaikeampia hyökkäyskuvioita käytetään silloin, kun halutaan kenttien olevan haastavampia.

4.3 Kolikot

Yksi pelin tärkein pelattavuuteen vaikuttava elementti on kolikoiden kerääminen. Kolikoiden avulla pelaaja voi ostaa pelin sisäisestä kaupasta uusia asuja ja tarvikkeita. Kolikoiden kerääminen luo myös pelaajille lisätekemistä kentissä vihollisten väistämisen lisäksi.

On tärkeää pitää huoli siitä, että kolikoita ei esiinny kentissä liian vähän eikä liian paljon. Oikea määrä kolikoita tekee pelistä hauskemman. Liian pieni määrä kolikoita olisi pelattavuuden näkökulmasta tylsää, sillä pelaaja saattaisi kokea, että niiden kaikkien kerääminen olisi liian helppoa. Liian suuri määrä kolikoita taas tekisi pelistä liian levottoman.

Kolikoilla on myös tärkeä osa pelin pistelaskennassa. Ei haluttu, että olisi liian helppoa kerätä pelissä kaikki kolikot jokaisesta kentästä, sillä silloin ei syntyisi pelaajien keskeistä kilpailua kenttien huippupisteiden saavuttamisesta. Kolikoiden määrä on oikea silloin, kun on vaikea kerätä ne kaikki, mutta niiden määrä ei silti aiheuta pelaajissa toivottomuutta vaikuttamalla liian mahdottomilta kerätä.

4.4 Loppuvastukset

Pelin viisi loppuvastusta ovat toinen toistaan vaikeampia. Loppuvastusten vaikeus määräytyy heidän kestävyyden ja hyökkäysten monimuotoisuuden mukaan. Mitä korkeammalla tasolla pelaajan asu on, sitä enemmän tuhoa hänen hyökkäyksensä loppuvastukseen tekee. Siksi on tärkeää, että loppuvastuksen kestävyys on pelaajan asun nykyisen tason kanssa tasapainossa. Esimerkiksi korkeatasoista asua käyttäen pelin ensimmäinen loppuvastus saattaa olla hyvin helppo, mutta matalatasoisella asulla sen vaikeustaso on juuri oikein.

4.5 Asut

Jokaisella asulla on omat kehityspisteensä. Yksi asu saattaa olla toista kestävämpi, mutta on taas jossain toisessa osa-alueessa huonompi. Korkea määrä HP-kehityspisteitä tekee hahmosta kestävämmän ja saattaa olla hyvä pelaajalle, joka on huono väistämään vihollisten hyökkäyskuvioita. Asu, jolla on korkea määrä STR-kehityspistettä, on taas parempi loppuvastuksia vastaan, sillä STR vaikuttaa siihen, kuinka paljon tuhoa pelaajan hyökkäykset loppuvastuksiin tekee. Asut on tasapainotettu eri pelityylien mukaan. Jokaiselle eri pelaajalle ja pelityylille löytyy oma asunsa.

4.6 Pelihahmojen erikoistaidot

Asujen eri erikoistaidot tasapainotettiin antamalla niille omat erikoisuusalueensa. Esimerkiksi kissa- ja lohikäärmeasut ovat hyvin samanlaisia, mutta heillä ovat omat paikkansa pelissä. Kissa-asun erikoistaito tuhoaa kaikki viholliset ympärillään ja tekee heistä kolikoita. Lohikäärmeasun erikoistaito taas tuhoaa kaikki viholliset, mutta ei anna pelaajalle yhtään lisäkolikoita. Eikö kissa-asun silloin pitäisi olla huomattavasti parempi? Ei, sillä kissa-asu ei pysty vahingoittamaan pelin loppuvastuksia, kun taas lohikäärmeasu pystyy. Kissa-asu on parempi kampanjan tavallisissa kentissä, kun taas lohikäärmeasu on parempi loppuvastuksia vastaan.

4.7 Cooldown

Pelin eri asujen erikoistaidot on tasapainotettu muun muassa siten, että niille on annettu cooldownit. Jokaisella erikoistaidolla on oma peruscooldown. Mitä voimakkaampi erikoistaito, sitä pidempi cooldown sillä on. Esimerkiksi jäniksen erikoistaidon cooldown on kaksitoista sekuntia, kissan erikoistaidon cooldown on kaksikymmentä sekuntia ja pandan erikoistaidon cooldown on kuusi sekuntia. Perusarvojen lisäksi cooldownien pituuteen vaikuttaa jokaisen asun CD-kehityspisteen arvo. Jokainen piste-CD kehityspiteessä vastaa yhtä prosenttia pois cooldownin pituudesta, esimerkiksi CD-arvolla kuusitoista saat kuusitoista prosenttia lyhyemmän cooldownin.

5 Tekninen toteutus

5.1 Hahmon ohjaus kosketusnäytöllä

Pelaaja voi ohjata kosketusnäytöllä hahmoaan kahdella tapaa, painamalla näyttöä tai vetämällä sormea näyttöä pitkin. Pelin kentille on luotu oma skene, joka sisältää kaikki kenttien pelaamiseen liittyvät GameObjectit ja toiminnot. Kosketusnäytön painallukset luetaan näkymättömästä unity-rajapinnan UI.Button-napista, joka on liitetty skeneen. Sormen vedot kosketusnäyttöä pitkin luetaan unity-rajapinnan BaseInputModule-toimintoa käyttäen.

Hypyt ylös ilmaan ja jetstep-liikkeet takaisin maahan pelaaja tekee painamalla kosketusnäyttöä. Painalluksen lukee näkymätön UI.Button-nappi, joka peittää koko ruudun. Napin painallus kutsuu skeneen liitetyn skriptin OnPointerClick-funktiota, joka kertoo, että hahmon pitää hypätä. Periaatteessa UI.Button on tarkoitettu lähinnä käyttöliittymien tekemistä varten, mutta se on kätevä toiminto myös silloin, kun kosketusaluetta halutaan helposti rajata. Näytön oikealla puolella kenttäskenessä sijaitsee myös kaksi muuta nappia, asetukset ja hahmon erikoistaito. Voimme rajata UI.Button-toiminnon kanssa kosketusalueet toisistaan niin, että pelaaja ei vahingossa paina väärä nappia pelatessaan.

Pelaaja voi syöksyä pelihahmollaan ilmassa vetämällä sormea näyttöä pitkin. Syöksyn voi tehdä neljään eri suuntaan: ylös, alas, vasemmalle tai oikealle. Syöksyt antavat pelaajalle paremmat mahdollisuudet väistää vihollisia ilmassa. Vedot luetaan BaseInputModulen OnBeginDrag ja OnEndDrag funktioilla.

OnBeginDrag lukee vedon alkupisteen ja tallentaa sen x- ja y-koordinaatit Vector2-muuttujaan startPosition. OnEndDrag lukee, missä veto loppuu, ja tallentaa vedon loppupisteen x- ja y-koordinaatit Vector2-muuttujaan endPosition. Alku- ja loppupisteiden avulla saamme vektorin, jonka avulla voimme laskea vedon suunnan.

Vedon suunta saadaan laskemalla saadun vektorin ja x-akselin kulma. Tämä voidaan saavuttaa käyttämällä unity-rajapinnan funktiota Mathf.Atan2. Samalla muutamme saamme radiaanit asteiksi käyttämällä Mathf.Rad2Deg -funktiota. Lopuksi tarkistetaan vedon suunta yksinkertaisella if-lauseella.

5.2 Hahmojen erikoistaidot

Pelin ensimmäisen version julkaisussa pelihahmoja tulee olemaan kuusi erilaista. Joidenkin erikoistaitojen toteutuksien samanlaisuuden takia käsittelen tässä vain kolmen eri pelihahmon erikoistaidon toteutusta. Jäniksen, kissan ja pandan erikoistaidot ovat toteutuksiltaan hyvin erilaisia ja tulivat siksi valituiksi.

5.2.1 Jänis

Jäniksen erikoistaito antaa hänelle nopeamman juoksunopeuden ja suojan, joka suojaa häntä yhdeltä vihollisen iskulta. Taidon antama juoksunopeus riippuu jäniksen nykyisestä tasosta ja SPD-kehityspisteen määrästä. Taito suojaa aina yhdeltä iskulta ja taito loppuu pelaajan otettuaan osuman.

Jänis-asu on ainoa asu, joka pelaajalle annetaan pelin alussa. Siksi asusta suunniteltiin mahdollisimman yksinkertainen mutta samalla monipuolinen. Halusimme sen olevan hyvä aloitushahmo pelin oppimiseen. Nopeampi juoksutahti tarkoittaa sitä, että pelaaja voi läpäistä kentät hieman normaalia nopeammin ja suoja oikein käytettynä helpottaa monien vihollisten hyökkäyskuvioiden väistämisessä.

Jäniksen erikoistaidon juoksunopeus lasketaan pelihahmoon liitetyssä skriptissä. Laskussa otetaan huomioon nykyisen kentän normaali juoksunopeus ja sitten lisätään siihen lisää nopeutta asun nykyisen tason ja sen SPD-kehityspisteen määrän mukaan. Mitä korkeampi asun SPD-kehityspiste on, sitä enemmän hänen nopeutensa erikoistaidon kanssa nousee. Visuaalinen palaute nopeammasta juoksunopeudesta annetaan pelaajalle liikuttamalla kentän taustaa nopeammin.

Jäniksen suoja on kupla hänen ympärillään, joka piirretään partikkeliefektinä. Suojan ollessaan päällä jäniksen sphere colliderin kokoa kasvatetaan partikkeliefektin kokoiseksi. Kun suoja törmää viholliseen, niin sen partikkeliefekti tapetaan ja sen sphere colliderin koko palautetaan takaisin entiselleen.

5.2.2 Panda

Pandan erikoistaito on bambuoksien heittäminen. Bambuoksat tuhoavat ne viholliset, joihin ne osuvat. Heitettyjen bambu oksien määrä riippuu asun tasosta ja sen nykyisestä

STR-kehityspisteen määrästä. Aluksi panda heittää vain yhden bambuoksan, mutta korkeammalla tasolla panda voi heittää niitä jopa kolme kerralla.

Bambuoksa ovat tallennettuna AssetBundlessa prefabina. Prefabi sisältää bambuoksan meshin, tekstuurin, RigidBodyn ja SphereColliderin. Kun pandan erikoistaitoa käytetään, instantioidaan bambuoksien prefabit pelihahmoon liitetyssä skriptissä. Bambu oksien määrä ja suunta annetaan skriptissä. Bambu oksien RigidBodyt on laitettu kinemaattisiksi, mikä tarkoittaa sitä, että niihin ei vaikuta fysiikkamoottorin omat laskut, ainoastaan skriptissä annetut positioiden vaihdot. Bambuoksien heittosuunnat ovat ennalta määritetyt.

Bambuoksien prefabiin on liitetty skripti, joka pitää huolen törmäyksistä. Skriptissä käytetään OnTriggerEnter funktiota, joka tunnistaa törmäyksen toisen osapuolen tagien avulla. Jos toinen osapuoli törmäyksessä on vihollinen, niin skripti tuhoaa sen ja itsensä. Jos toinen osapuoli törmäyksessä on loppuvastus, niin skripti vaurioittaa loppuvastusta pandan tason ja sen STR-kehityspisteen määrän mukaan. Mitä korkeampi STR-määrä on, niin sitä enemmän tuhoa bambu oksat tekevät loppuvastuksiin.

5.2.3 Kissa

Kissan erikoistaito muuttaa ympärillä olevat viholliset kolikoiksi. Kolikoiksi muutetut viholliset lentävät automaattisesti pelaajan luo. Taito on erittäin vahva, mutta sitä voi käyttää harvemmin kuin muita korkean cooldownin takia. Kissan erikoistaito on oiva tapa kerätä pelissä kolikoita nopeasti.

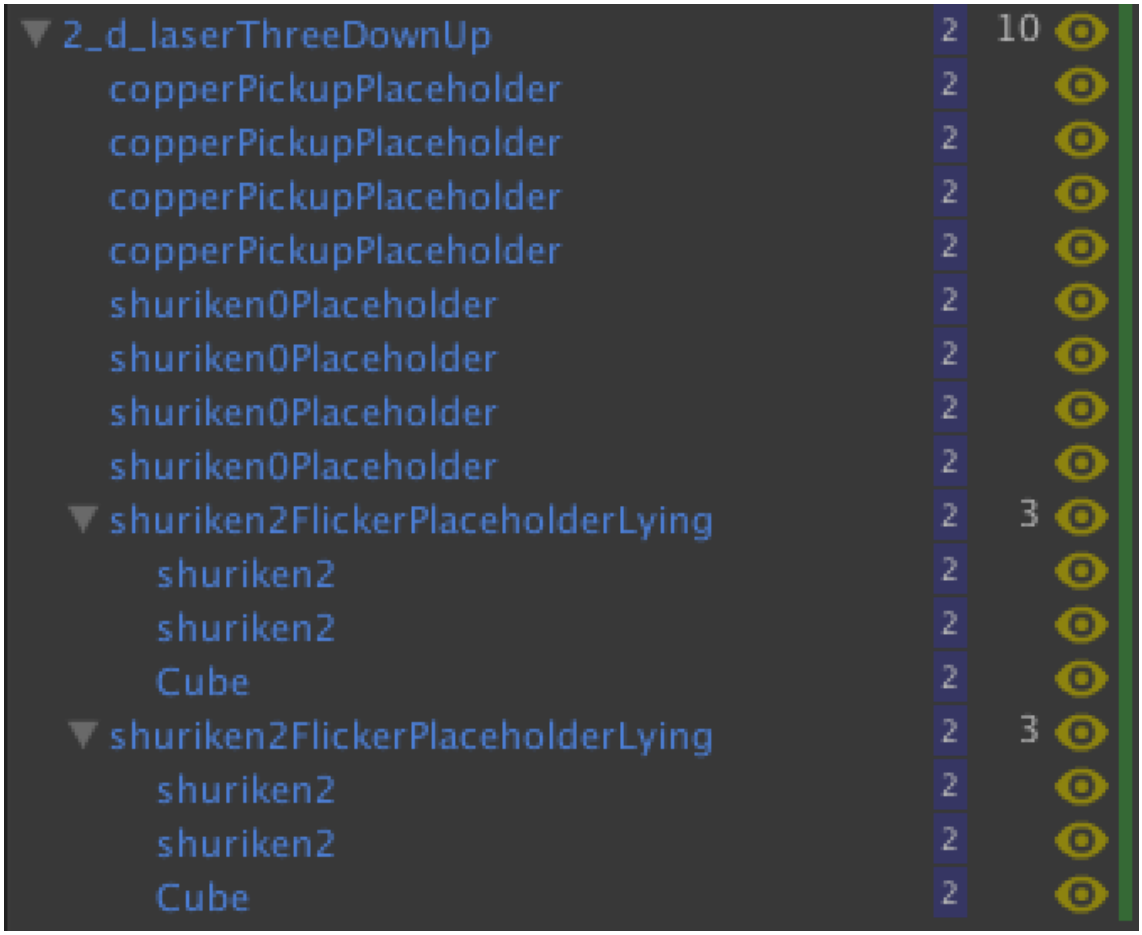
Kissan erikoistaidon funktio löytyy skriptistä, joka liitetään pelihahmoon. Funktion alussa kissa ensin siirretään ruudun keskelle, missä erikoistaidon animaatio aloitetaan. Samalla kun animaatio alkaa, niin luodaan iso sphere collideri kissan ympärille, joka toimii erikoistaidon törmäysalueena. Kaikki viholliset, jotka ovat törmäysalueen sisällä, muutetaan kolikoiksi. Kolikot sitten siirretään pelaajan positioon, missä ne poistetaan pelistä. Tuhottujen vihollisten määrä lasketaan ja pelaajalle annetaan kolikoita sen mukaan. Erikoistaidon loputtua pelaaja siirretään takaisin maahan.

5.3 Vihollisten ja hyökkäyskuvioiden luominen ja piirtäminen

Vihollisten hyökkäyskuviot luotiin käsin ja tallennettiin AssetBundleen prefabina. Prefabit ovat kokoelma placeholdereita, jotka sisältävät pelkästään shurikeneitten transformit, colliderit ja mahdolliset skriptit. Placeholderilla tarkoitetaan mahdollisimman kevyttä GameObjectia, jonka avulla varataan paikka suurempaa ja kattavampaa tulevaisuuden GameObjectia varten. Prefabeista jätettiin meshit, partikkeliefektit ja tekstuurit pois, jotta prefabien tiedostokoot eivät olisi liian suuria. Tällä tavoin yhden prefabin koko on pelkästään noin kaksisataa tavua, kun se tavallisesti voisi olla jopa yli yhden megatavun kokoinen. Jättämällä kaikki visuaaliset osat prefabeista pois on oiva tapa säästää tilaa pelin tiedostokoossa.

Pelissä vihollisten hyökkäyskuviot piirretään peliin lukemalla placeholdereitten paikat ja sitten luomalla shurikeneitten meshit, tekstuurit ja partikkeliefektit saatuihin paikkoihin. Jokaisen shurikenin meshi, tekstuuri ja partikkeliefekti löytyy erillisestä prefabista, joista ne sitten haetaan. Tällaisia tekstuurin, partikkeliefektin ja meshin sisältäviä prefabeja tarvitaan vain yksi jokaista eri shurikenia varten. Shurikenit erotetaan koodissa toisistaan merkitsemällä shurikenin nimi ja numero skriptiin, joka on shurikeniin kiinnitetty.

Vihollisten hyökkäyskuvioiden prefabit on rakennettu joukosta parentteja ja childeja. Parentilla tarkoitamme hierarkian ylintä objektia ja childilla jokaista objektia, joka sijaitsee parentin alla. Prefabin ylimpänä on parentti, joka sisältää hyökkäyskuvion ankkurin ja viittauksen skriptiin EnemyPatternManager. Patternin alla on joukko lapsia, jotka ovat shurikeneitten placeholdereita. Monimutkaisemmat shurikenkuviot, jotka vaativat useampaa osaa, pistetään yhteisen parentin alle. Parenttiin useimmiten lisätään skripti, joka sisältää kuvion tiedot ja toiminnot. Tyypillinen vihollisten hyökkäyskuvion rakenne nähdään kuvassa 3.



Kuva 3. Tyypillinen vihollisten hyökkäyskuvio unity editorin hierarkiassa.

Vihollisten hyökkäyskuvioiden luomiskohta kenttään on merkitty tarkasti. Hyökkäyskuviot luodaan kameran ulkopuolelle joko kentän vasemmalle tai oikealle puolelle riippuen kentän suunnasta. Luomisen jälkeen vihollisjoukot rupeavat liikkumaan pelaaja päin niille annetun hyökkäysnopeuden mukaisesti. Hyökkäyskuvioiden poistaminen pelistä tapahtuu silloin, kun ne ovat siirtyneet pelaajan ohi ja kameran näkyvyysalueen ulkopuolelle. Kohta jossa hyökkäyskuvio poistetaan, tarkistetaan ankkurilla, joka on liitetty hyökkäyskuvioon. Ankkurin osuttua kameran ulkopuolelta löytyvään seinään, niin hyökkäyskuvio poistetaan pelistä.

5.4 Vihollisten hyökkäyskuvioiden kanssa käytetyt skriptit

EnemyPatternManager on skripti, joka annetaan vihollisjoukon ylimmälle parentille. Skripti kertoo pelille, mihin kohtaa kenttää hyökkäyskuvio luodaan ja mitä tekstuureja, partikkeliefektejä ja meshejä kullekin placeholderille annetaan.

EnemyBehavior on skripti joka annetaan jokaiselle placeholderille. Skripti kertoo, mitä shurikenia placeholder esittää ja mitä toimintoja sen kuuluu osata. Shurikeneitten törmäykset tarkistetaan myös tässä skriptissä.

GenericRotate on skripti, joka liitetään shurikeneitten placeholdereihin. Se pyörittää shurikenia ympäri ja antaa vaikutuksen, kuin shuriken olisi heitetty.

GenericHover on skripti, joka siirtää shurikenia joko horisontaalisesti tai vertikaalisesti annetun nopeuden ja pituuden mukaan.

5.5 Miten shurikenit toimivat

Normaalin shurikenin ja maa-shurikenin toteutukset ovat yksinkertaisimmat. Ainoa ero niiden välillä on koko, maa-shurikeni on huomattavasti isompi. Normaali shuriken ja maa-shuriken tarvitsevat ainoastaan EnemyBehavior-skriptit toimiakseen. Skripti kertoo moottorille mihin shurikenit luodaan sekä pitää huolen niiden liikkumisesta ja törmäyksistä.

Tikarille annetaan EnemyBehavior-skriptin lisäksi GenericHover-skripti. Kun tikari liitetään hyökkäyskuvioon, sen GenericHover-skriptille kerrotaan, jos se liikkuu horisontaalisesti tai vertikaalisesti sekä millä nopeudella ja suuruudella kyseinen liike suoritetaan.

Salama-shuriken käyttää pelkästään EnemyBehavior-skriptiä. Salama-shurikenin placeholder sisältää kaksi sphere-collideria, mihin shurikenit tulevat, ja yksi box collider shurikeneitten välissä, mihin salama tulee. Sphere-collidereiden kohdalle luodaan shurikenit ja box-colliderin kohdalle partikkeliefekti, joka edustaa salamaa. Ajoitus, milloin salama on päällä ja milloin ei, hoidetaan coroutinella, jonka lopussa on WaitForSeconds-kutsu, odotuksen pituus annetaan coroutinelle parametrinä.

Tuli-shurikenille annetaan EnemyBehavior-skriptin lisäksi GenericRotate-skripti. GenericRotate-skripti pyörittää tuli-shurikeneita keskipisteen ympäri.

5.6 Unityn coroutinet

Ajastetut funktiot ja tarkistukset voi unityssä suorittaa joko käyttämällä updatea, invokea tai coroutinea. Update on funktio, joka aktivoituna kutsuu itseään kerran jokaisen ruudunpäivityksen aikana. Update on unityssä yleisin ja eniten käytetty toiminto pelilogiikan toteuttamisessa. (Unity Technologies 2015b.)

Invoke on funktio, joka kutsuu jotain toista funktiota tietyn annetun ajan kuluttua. Invokesta on myös toinen versio RepeatInvoke, joka kutsuu ennalta määrättyä funktiota aina uudestaan tiettyjen ajankohtien välillä. (Unity Technologies 2015c.)

Coroutine on unity-ominaisuus, joka tarjoaa mahdollisuuden ajaa funktioita ruudunpäivityksen ulkopuolella. Tavallisesti funktio ajettaisiin yhden ruudunpäivityksen aikana alusta loppuun, mutta coroutinet ovat täysin aikariippuvaisia. Coroutine on funktio, jolla on kyky pysäyttää suoritus ja palauttaa hallinta takaisin Unitylle, mutta sitten myöhemmin jatkaa täysin samasta paikasta, mihin se jäi aikaisemman ruudunpäivityksen aikana. (Unity Technologies 2015d.) Coroutinet eivät pyöri omassa säikeessä.

Coroutinejen käyttäminen vaatii tietyissä tapauksissa vähemmän suorituskykyä kuin update tai invoke. Update ajetaan jokaisen ruudunpäivityksen aikana, myös silloin, kun se ei päivitä mitään. Käyttäessään updatea tulee jatkuvasti vastaan tilanteita, jolloin funktiota ajetaan turhaan. Coroutine taas ajetaan vain kerran alusta loppuun. Coroutine ajetaan vain silloin, kun se käynnistetään (Unity Technologies 2015d.). Invoke on funktio, jonka avulla voi käynnistää minkä vain toisen funktion tietyn annetun ajankohdan jälkeen. Invoken kanssa voi valita, milloin funktio alkaa, mutta sitä ei pysty pysäyttämään jälkeenpäin. InvokeRepeating toimii samalla tavalla. InvokeRepeatingia on kritisoitu hitaudestaan. Sen ajaminen vaatii enemmän suoritustehoa kuin update tai coroutine.

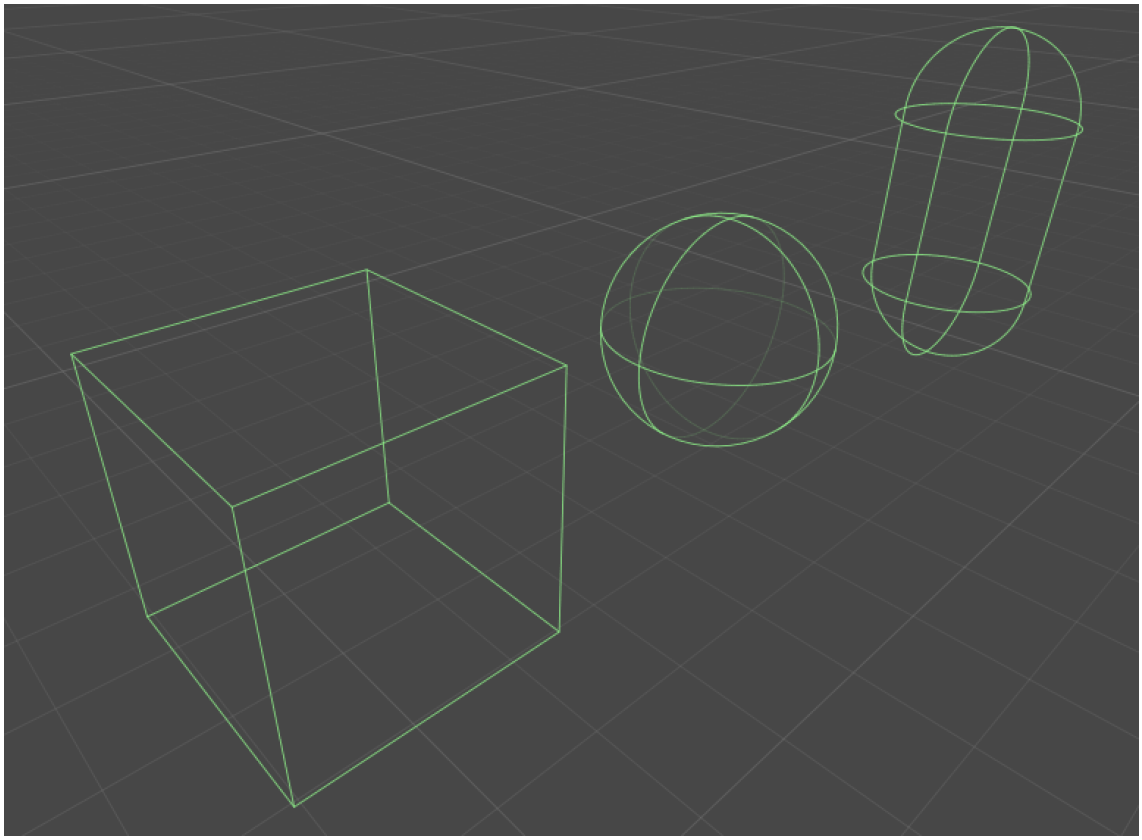
Coroutinen ajon voi pysäyttää ja jättää odottamaan esimerkiksi käyttämällä funktioita yield return null, yield return new WaitForEndOfFrame() tai yield return new WaitForSeconds. Yield return null viestittää coroutinelle, että pysäyttää funktion ja odottaa seuraavan ruudunpäivityksen alkuun asti. Yield return new WaitForEndOfFrame taas kertoo coroutinelle, että pysäyttää funktion ja odottaa, kunnes seuraava ruudunpäivitys on renderöity. Yield return new WaitForSeconds on ruudunpäivityksistä riippumaton ja pysäyttää funktion siihen asti, kunnes annettu aika sekunteina on kulunut. (Unity Technologies 2015d.)

Pelin skripteissä haluttiin käyttää coroutineja kaikkialla päin koodia, missä vain mahdollista. Updaten käyttämistä vältettiin, jotta ei tyhjiä tai turhia funktiokutsuja syntisi ruudunpäivityksien aikana. Invokeja kutsuttiin skripteissä ainoastaan silloin, kun haluttiin muuttaa tietyn muuttujan arvoa tarkkana ajankohtana. Coroutinen suurin etu on sen kyky toimia kellon mukaan eikä ruudunpäivityksen mukaan. Funktion toiminnot voidaan ajoittaa paljon helpommin ja tarkemmin.

5.7 Törmäysten tarkistaminen

Unityssä RigidBody on pääkomponentti, joka mahdollistaa fysiikan laskemisen GameObjectille. Kun GameObjectilla on RigidBody, niin se reagoi painovoimaan ja fysiikkamoottorin laskuihin automaattisesti. Jos RigidBodyn lisäksi GameObjectiin lisätää Collider komponentti, niin GameObjecti reagoi myös törmäyksiin. Koska RigidBody komponentit päättävät GameObjectin liikkuvuudesta, ei GameObjectia kannata enää koodista käsin liikuttaa. On järkevämpää jättää kaikki GameObjectin liikkuvuus fysikkamoottorin laskujen varaan sillä se on suoritustehoa ajatellen fysiikkamoottorille kevyempää. (Unity Technologies 2015e.)

Collider-komponentit määräävät GameObjectin muodon mahdollisten törmäysten laskemista varten. Colliderin, joka on näkymätön, kuuluu olla samankokoinen ja muotoinen kuin GameObjectin meshi. Vähiten suoritustehoa vaativat colliderit ovat niin sanottuja primitiivisiä collidereita. 3D-peleissä nämä primitiiviset colliderit ovat: box-collider, sphere-collider ja capsule-collider. Collidereiden muodot nähdään kuvassa 4. (Unity Technologies 2015f.)



Kuva 4. Pelissä käytettävät eri colliderit. Vasemmalta oikealle: box-collider, sphere-collider ja capsule-collider.

Collideri voidaan lisätä myös GameObjectiin, jolla ei ole RigidBodya, mutta GameObjectin on oltava liikkumaton, kuten esimerkiksi. lattia tai seinä. Näitä kutsutaan staattisiksi collidereiksi. GameObjectit, jotka sisältävät colliderin mutta eivät RigidBodya, ovat dynaamisia colldereita. Stattiset colliderit voivat olla vuorovaikutuksessa dynaamisten collidereiden kanssa, mutta koska niillä ei ole RigidBodya niin ei kumpikaan GameObjecti liiku törmäyksen seurauksena. (Unity Technologies 2015f.)

Skriptissä voidaan huomata ja ohjata törmäyksiä käyttämällä Unityn OnCollisionEnter-funktiota. On myös mahdollista kutsua skriptiä silloin, kun colliderit liikkuvat toistensa alueelle mutta eivät aiheuta törmäystä. Collideri, joka on konfiguroitu Triggeriksi, ei käyttäydy kuten tyypillinen kiinteä GameObjecti, ja yksinkertaisesti antaa muiden GameObjectien kulkea sen läpi. Kun toinen collider astuu sen alueelle, niin se kutsuu skriptissään OnTriggerEnter funktiota. (Unity Technologies 2015f.)

Törmäyksien tapahtuessa fysiikkamoottori kutsuu asiaan liittyviä funktioita mistä vain skrpitistä, joka on liitetty törmäävään GameObjectiin. GameObjectien törmäyksen alussa

kutsutaan ensin OnCollisionEnter-funktiota. Jos törmäys jatkuu, kutsutaan funktiota OnCollisionStay jokaisen törmäyksen aikaisen silmukan aikana. Lopulta kun törmäys loppuu, niin kutsutaan funktiota OnCollisionExit. Trigger colliderit kutsuvat taas vastaavia funktioitaan OnTriggerEnter, OnTriggerStay ja OnTriggerExit. (Unity Technologies 2015f.)

Unityn PhysicsManagerissa voidaan päättää, mitkä colliderit törmäävät toistensa kanssa. GameObjecteille annetaan omat layerit, kuten esimerkiksi viholliset voidaan laittaa omaan layeriin, pelaaja omaan layeriin ja myös kolikot omaan layeriin. PhysicsManagerissa voidaan sitten päättää, että pelaaja törmää kolikoiden ja vihollisten kanssa, mutta kolikot ja viholliset eivät törmää keskenään. (Unity Technologies 2015g.)

Skriptissä eri GameObjectit voidaan tunnistaa niiden tagien perusteella. Jos vihollinen törmää ja sen skriptissä kutsutaan OnCollisionEnteriä, niin GameObjecti, johon törmäys tapahtui, saadaan selville lukemalla sen tagi. Eri tagit voidaan mennä läpi yksinkertaisella switch case silmukalla. (Unity Technologies 2015h.)

5.8 Pelimoodit

Kampanjan kenttien tiedot ovat tallennettuna GameSparksin puolella pilvikoodina. Pelaajan aloittaessaan kentän peli hakee kentän shortcoden perusteella tiedot kentän pituudesta, vihollisista, musiikista ynnä muista asioista. Tiedot säilytetään GameSparksin pilvessä siksi, että olisi helppo muuttaa kenttien tietoja ilman ylimääräistä päivitystä peliin. Kampanjan kentät, jotka sisältävät loppuvastuksen ja boss rush -kenttä toimivat samalla periaatteella.

Kaikki endless modessa käytettävät vihollisjoukot ovat GameSparksin puolella listattuna pilvikoodissa. Kentän alussa lista vihollisjoukosta haetaan pilvestä ja tallennetaan taulukkoon pelin ajaksi. Vihollisjoukot on jaettu viiteen eri vaikeustasoon. Vaikeustaso on merkattu numerolla vihollisjoukon shortcoden alussa. hyökkäyskuvioiden tiedot ovat tallennettu GameSparksin pilveen siksi, että niitä olisi helppo muokata ilman koko pelin päivittämistä.

Endless moden kenttä on jaettu eri vaiheisiin. Mitä pidemmälle pelaaja kentässä pääsee, sitä vaikeammaksi se muuttuu. Endless modessa vaikeustasoja on kaksi. Yksi alkaa endless moden ensimmäisestä vaiheesta sekä sisältää ainoastaan vihollisten tavallisia hyökkäyskuvioita ja toinen alkaa hieman myöhemmästä vaiheesta sekä käyttää pelin kaikkia mahdollisia vihollisten hyökkäyskuvioita.

Jokainen endless moden vaihe kestää noin kolmekymmentä sekuntia. Vaiheen loputtua skriptissä arvotaan uuden vaiheen kesto sekä kerrotaan koodille, minkä vaikeustason hyökkäyskuvioita sen nyt kuuluu luoda peliin. Eri vaiheita missä hyökkäyskuvioiden vaikeustasoa muutetaan, on endless modessa kokonaisuudessaan kuusi. Sen jälkeen, kun kaikki kuusi ensimmäistä vaihetta on läpäisty, pysyy annettu vaikeustaso samana ja skriptissä nostetaan enää vihollisten hyökkäyskuvioiden hyökkäysnopeutta niin, että mitä pidemmälle pelaajaa pääsee, sitä nopeammin hyökkäyskuviot liikkuvat pelaajaa vastaan.

5.9 Pistetaulukko

Pistetaulukkotoiminto GameSparks-järjestelmässä ei ole rajattu pelkästään perinteisen kaavan pistetaulukkoihin. Pisteiden laskemisen lisäksi voidaan seurata, verrata ja tulostaa pelaajien suorituksia pelissä. Toiminnot tarjoavat mahdollisuuden luoda sosiaalisia ja kilpailuhenkisiä ominaisuuksia peliin. (Page 2014a.)

Pistetaulukkoja voidaan myös jakaa useampiin. Voidaan ensin luoda yksi geneerinen konfiguraatiomalli pistetaulukosta ja sitten tarvittaessa automaattisesti luoda siitä erilaisia kopioita ja versioita. Hyvä esimerkki tästä ovat pistetaulukot pelin eri kentille. Jokainen kenttä käyttää samanlaista pistetaulukkoa. Pistetaulukosta on kopioitu useita kappaleita, missä kentän numero ja nimi on muuttunut. Tämä helpottaa työtä, sillä ei joudu käsin tekemään oman pistetaulukon jokaiselle sadalle eri kentälle. (Page 2014a.)

5.10 Saavutukset

Pelin aikana UserManager skripti pitää laskua kaikista pelaajan tekemistä asioista, kuten esimerkiksi pelaajan hyppyjen määrästä, kerättyjen kolikoiden määrästä, juostusta kokonaismatkasta ja omistamista asuista. Tasaisin väliajoin muuttujien arvot tallennetaan GameSparksin, pilveen mistä ne voidaan hakea ja tarkistaa tarvittaessa myöhemmin.

Tiedot saavutuksista lasketaan ja tallennetaan GameSparksin pilvikoodissa. Saavutusten tiedot kuten saavutuksen saamiseen tarvittavat vaatimukset tai saavutuksen antamat palkinnot sijaitsevat kaikki myös GameSparksin pilvikoodissa. Kun pilvikoodissa huomataan, että pelaaja on täyttänyt saavutuksen vaatimukset, niin lähetetään peliin tieto sen saamisesta sekä palkinnon suuruudesta.

Kun pelaaja seuraavan kerran käy pelin päävalikossa saavutuksen saamisen jälkeen hänelle annetaan palkinto ja ilmoitetaan saavutuksen saamisesta pelin sisäisellä viestillä. Päävalikosta löytyy myös lista saaduista ja lukituista saavutuksista ja niiden mahdollisista palkinnoista.

6 Optimointi

Suorityskyky on tärkeää nopeatempoisissa mobiilipeleissä. Tasaisen ruudunpäivityksen saavuttamisen lisäksi hyvin optimoitu mobiilipeli kuluttaa myös mobiililaitteilla vähemmän akkua. Unity-pelimoottori sisältää monia toimintoja jotka auttavat tasaisen ruudunpäivityksen saavuttamisessa.

6.1 Poolaus

GameObjectien poolaus on tekniikka, joka muistuttaa hyvin paljon varastoimista. Sen sijaan, että instantioitaisiin jokaisen tarvittavan pelin GameObjectin juuri sinä hetken, kun sitä tarvitaan, niin ne instantioidaan jo ennakkoon esimerkiksi latausruudun aikana. Instantioimisen jälkeen ne laitetaan varastoon odottamaan vuoroaan, milloin niitä tarvitaan pelissä käyttää. Sen sijaan, että tuhottaisiin jokainen GameObjecti silloin, kun sitä ei enää tarvita, niin ne siirretään takaisin varastoon odottamaan mahdollista seuraava käyttöään tulevaisuudessa. (Muklashy 2014.)

Klassinen esimerkki poolaamisesta ovat käsiaseen ammukset. Sen sijaan, että uudet ammukset instantioitaisiin jokainen kerta, kun pelaaja ampuu käsiaseellaan, niin on kannattavampaa ja helpompaa vain instantioida ne heti kentän alussa. Ammukset merkitään kentän ajaksi epäaktiiviseksi, kun niitä ei käytetä ja ne siirretään odottamaan vuoroaan jonnekin ruudun ulkopuolelle. Sitten kun ammusta tarvitaan, niin se laitetaan aktiiviseksi ja siirretään pelaajan käsiaseeseen. (Muklashy 2014.)

GameObjectin poolaamisen hyödyt ovat muun muassa: vähemmän pelin hidastumisia kalliiden instantiointi- ja tuhokutsujen takia sekä parempi ennakkoarvio tarvittavasta muistitilasta, sillä poolatut GameObjectit ovat aina valmiiksi ladattuina muistiin. Ainoa huono puoli GameObjectien poolaamisessa on se, että se vaatii enemmän huomiota ja tarkkuutta ohjelmoijalta hänen kirjottaessaan pelilogiikkaa. (Muklashy 2014.)

GameObjectien poolaaminen on yksinkertainen tapa vähentää kertoja, jolloin muistipaikkoja varataan ja vapautetaan pelin aikana. Muistipaikkojen ympärillä tapahtuvan liikenteen vähentäminen parantaa pelin suorituskykyä.

6.2 Draw call batching

Draw Call tapahtuu jokainen kerta, kun GameObjecti renderöidään. Yksinkertaistettuna se on ohje suorittimella ja näytönohjaimelle, joka kertoo, että tämä GameObjecti renderöidään nyt. Draw call batching on Unityn ominaisuus, joka vähentää suorittimen rasitusta renderöinnin aikana. Draw call batching pitää huolen siitä, että ei yhtään turhaa draw callia ruudunpäivityksen aikana kutsuta. Draw call batching ottaa kaikki skenen samanlaiset pienet meshit ja yhdistää ne yhdeksi isoksi meshiksi. Normaalisti jokainen näistä pienistä mesheistä pitäisi renderöidä erikseen omalla draw callilla, mutta nyt ne kaikki voidaan renderöidä yhdellä yhteisellä draw callilla. Draw call batchausta on kahta erilaista: static ja dynamic. (Unity Technologies 2015j.)

Static batching antaa paremman suorituskyvyn, mutta sitä voidaan ainoastaan käyttää GameObjecteihin, jotka eivät liiku, pyöri tai skaalaudu skenen sisällä. (Unity Technologies 2015j.) Static batchingin tarkoitus on ryhmittää mahdollisimman monta eri skenen meshia mahdollisimman pieneen määrään piirtokertoja. Tällä tavoin unity voi renderöidä muutamia suuria meshejä lukuisten pienten meshien sijaan parantaen pelin suorituskykyä. Static batching käyttää dynamic batchingia vähemmän muistia ja on muutenkin huomattavasti tehokkaampi. Static batchingia kannattaa käyttää dynamic batchingin sijaan aina kun vain mahdollista, sillä se lisää pelin suorituskykyä huomattavasti. (Vliet 2012.)

Kun static batching ei ole mahdollisuus, niin käytetään dynamic batchingia. Unityssä dynamic batching on aina automaattisesti päällä kaikille pelin GameObjecteille, jotka eivät ole osaa static batchingia. (Unity Technologies 2015j.) Dynamic batchingin liittää automaattisesti liikkuvat GameObjectit samaan draw calliin, jos ne jakavat saman materiaalin

ja täyttävät muut tarvittavat vaatimukset. Dynaaminen batchaus toimii vain mesheillä, jotka sisältävät vähemmän kuin 900 pintaa ja käyttävät samaa skaalaa transformissa. (Vliet 2012.)

Pelin tekstuurien draw callien määrää voi vähentää laittamalla ne yhteiseen tekstuuriatlasiin. Tekstuuri0atlas on yksinkertaistettuna iso kuva, joka sisältää useita pienempiä kuvia. Sen sijaan, että moottori lataisi jokaisen pienen tekstuurin erikseen muistiin, niin käyttämällä tekstuuriatlasia voidaan ladata ne kaikki muistiin yhdellä kertaa. Esimerkiksi yhden kentän kaikki taustatekstuurit voivat olla yhdessä yhteisessä tekstuuriatlasissa. Täten koko kentän tausta voidaan ladata muistiin yhdellä kertaa. Tekstuuriatlasin käyttö säästää myös draw calleissa, koko tekstuuriatlas voidaan renderöidä yhdellä draw callilla. (Disley 2015.) Unity käyttää kaikkia saatavilla olevia tekstuuriatlaseja automaattisesti taustalla. (DrCrook 2014.)

Käyttämällä tekstuuriatlasia voi pelin suorituskykyä parantaa huomattavasti. Sen avulla voidaan säästää paljonkin muistitilaa. Kun tekstuuri tuodaan peliin, niin moottori muuttaa sen koon automaattisesti kahden potenssiin. Esimerkiksi 200 x 200 pikselin kokoinen tekstuuri muutetaan automaattisesti 256 x 256 pikselin kokoiseksi tekstuuriksi. Tyhjä tila täytetään tyhjillä pikseleillä. Tyhjä tila tekstuureissa on hukkaan heitettyä muistia. Tekstuuri atlasin avulla voit yhdistää tekstuurit sellaisella tavalla, että hukkaan heitetty tila jää minimiin. (Disley 2015.)

Unity sisältää lisätyökalun Sprite Packer, jonka avulla voidaan helposti luoda tekstuuri atlaseja. Sprite Packer generoi tekstuuri atlasin automaattisesti, kun vain ensin valitset siihen kuuluvat tekstuurit käsin. (Unity Technologies 2015i.)

6.3 Garbage Collector

Kun GameObjecti-muuttuja tai -taulukko luodaan, niin muisti sen säilyttämiseen varataan keosta. Kun GameObjectia ei enää käytetä, niin sen käyttämät muistipaikat voidaan vapauttaa ja antaa käyttöön muille asioille. Ennen oli ohjelmoijan työtä varata ja vapauttaa muistia keosta käyttämällä annettuja funktiokutsuja. Mutta nykyään, systeemit kuten Unityn Mono engine osaa hallita muistia automaattisesti. Automaattinen muistinhallinta vaatii vähemmän työtä ohjelmoijalta ja vähentää mahdollisten ongelmien ja riskien syntymistä huomattavasti. (Bond 2015.)

On kaksi tapaa optimoida unityn garbage collectoria. Yksi tapa on määrittää garbage collectorin koko niin pieneksi, että sen tyhjennys ei aiheuta pelin aikana liian suurta suoritustehon menetystä. Tällöin garbage collectoria tyhjennetään niin usein kuin mahdollista, ehkä jopa vain muutamien ruudunpäivityksien välein. Jatkuvan garbage collectorin tyhjennyksen hyöty on, että se vie huomattavasti vähemmän muistitilaa. Jatkuva tyhjennys vaatii kuitenkin enemmän suoritustehoa ja saattaa huonolla tuurilla aiheuttaa ruudunpäivityksen pätkimistä. Pätkiminen on erityisesti läsnä mobiililaitteissa suoritustehon puuttumisen seurauksena. (Fassihi 2013.)

Toinen tapa on tehdä garbage collectorista suuri, ja vain tyhjentää se aina ennaltaan määrättyinä ajankohtina. Garbage collectorin tyhjentäminen harvoin on kätevää silloin, kun suoritusteho on muistijäljen koota tärkeämpi. Kosunin pelissä käytettiin tätä tapaa. Garbage collector tyhjennetään aina skenen vaihdon yhteydessä. Tällä tavoin voidaan välttää garbage collectorin aiheuttamia ruudunpäivityksen pätkimisiä pelaamisen aikana. Garbage collectorille varatun muistitilan määrittämisessä on kuitenkin oltava tarkkoja. On huono ajatus varata garbage collectorille enemmän muistitilaa kuin mitä se tulee käyttämään.

6.4 Paketointi

AssetBundlet ovat assetteja sisältäviä tiedostoja unityssä, jotka voidaan viedä muualle. Tiedostot ovat kompressoituja ja niitä voidaan halutessaan ladata peliin skriptin kautta milloin tahansa. AssetBundlejen tarkoituksena on yksinkertaistaa sisällön kuten tekstuurien, äänitiedostojen tai jopa kokonaisten maisemien lataamisen ja tuomisen peliin. (Unity Technologies 2015k.)

Pelimme AssetBundlet sijaitsevat GameSparksin palvelimilla. Peli lähettää kutsun GameSparksin palvelimelle siitä, että mitä tiedostoja ja paketteja sieltä ladataan käyttäjän laitteeseen. GameSparksin palvelimille voidaan lisätä tiedostoja suoraan heidän nettisivulta. Tiedostolle määritetään shortcode, eli stringi muuttuja, jonka avulla tiedostoon voidaan viitata myöhemmin koodissa. (Page 2014b.)

Kosunin pelissä AssetBundleja käytettiin ladattavien tiedostojen pienentämiseen. Pelin eri tiedostot haluttiin pitää mahdollisemman pieninä. Tämä saavutettiin lisäämällä pelin

peruspakettiin ainoastaan kaikki perustoiminnoille välttämättömät asiat. Loput tiedot jaoimme eri AssetBundleihin, jotka käyttäjät lataavat GameSparksista käynnistäessään pelin. Esimerkiksi pelin tekstuuripaketit jaettiin eri AssetBundleihin. Käyttäjä lataa käynnistäessään pelin vain laitteelleen sopivan tekstuuripaketin. Tällä tavoin käyttäjän ei tarvitse ladata muita tekstuuripaketteja säästäen monta megatavua tilaa laitteessaan.

Toinen hyöty AssetBundlejen käyttämisessä on niiden joustavuus pelin päivittämisessä. AssetBundlet jaettiin noin kolmeenkymmeneen eri pakettiin. Yksi paketti sisältää esimerkiksi musiikit ja toinen hahmojen 3D-mallit. Jos päivitämme 3D-malleja, niin ei käyttäjän tarvitse ladata muita paketteja uudestaan, riittää, että hän lataa pelkästään sen AssetBundle paketin, joka sisältää uudet 3D-mallit. Tällä tavoin voimme myös pelin päivityksien yhteydessä vähentää ladattavien tiedostojen määrää.

AssetBundlejen määrä on kuitenkin käyttöjärjestelmäkohtaisesti hieman rajattu. Alun perin AssetBundleja oli huomattavasti enemmän, mutta iOS:n kanssa se osoittautui ongelmalliseksi. (Seno 2014.)

7 Testaus ja alustakohtaiset ominaisuudet

Julkaisualustana Kosunin-pelille on tarkoitus toimia iOS, Android ja Windows Phone. Testaamisen ja julkaisun yhteydessä keskitytään aluksi vain yhteen alustaan kerralla. Ensimmäiseksi testataan ja julkaistaan peli iOS-alustalle, sen jälkeen Androidille ja lopulta viimeisenä Windows Phonelle.

Kosuninin ruudunpäivityksen päätimme lukita kolmeenkymmeneen piirrettyyn kuvaan sekunnissa. Suunnittelun aikana olimme ensin päättäneet pitää Kosuninin ruudunpäivityksen kuudessakymmenessä, mutta vanhemmat mobiililaitteet eivät kyenneet niin korkeisiin kuvataajuuksiin. Esimerkiksi vanhemmat iPhonet, kuten iPhone 4 ja aikaisemmat, sisältävät lukitun kuvataajuuden. Ne kykenevät pelkästään kolmeenkymmeneen piirrettyyn kuvaan sekunnissa. Niinpä päätimme lukita pelin ruudunpäivitys kolmeenkymmeneen kaikilla laitteilla jotta peli tuntuisi samanlaiselta laitteesta riippumatta.

Pelille tehtiin kolme eri tekstuuripakettia, SD, HD ja UD. SD-tekstuuripaketti sisältää pienimmät tekstuuri, HD SD:tä hieman isommat ja UD kaikista isoimmat ja tarkimmat tekstuurit. Laitteelle sopiva tekstuuripaketti valitaan sen keskusmuistin koon mukaan, sillä

mitä isompi tekstuuri, sitä enemmän keskusmuistia sen renderöinti vaatii. Käyttämällä Unity-rajapinnan SystemInfoa on mahdollista selvittää hyvinkin yksityiskohtaista tietoa eri laitteista. Esimerkiksi SystemInfo.systemMemorySize-funktio kertoo, kuinka paljon keskusmuistia laitteella on. Rajat määritettiin niin, että SD-tekstuuripaketti ladataan laitteille, joilla on alle 512 megatavua keskusmuistia, HD-tekstuuripaketti laitteille, joilla on yli 512 megatavua mutta alle 1024 megatavua keskusmuistia ja UD-tekstuuripaketti laitteille, joilla on yli 1024 megatavua keskusmuistia.

Mobiililaitteilla pelin orientaatio on lukittu landscape-orientaatioon. Pelin resoluutio ja kuvasuhde ovat laitekohtaisia, ja itse pelimoottori valitsee ne automaattisesti. Peli on suunniteltu niin, että sitä voi pelata yhtä hyvin kuvasuhteesta riippumatta, mitään pelattavuudelle tärkeää ei koskaan piiloteta kameran ulkopuolelle. Pelin resoluutio päätetään Unity API:n rajapinnan avulla. Valitsemamme resolution auto vaihtoehto valitsee pelille automaattisesti sen resoluution, joka antaa laitekohtaisesti parhaan suorituskyvyn. Kyseinen toimenpide kuitenkin esti meitä käyttämästä anti-aliaisingia, sillä se on käyttämällämme Unity 5.1 -versiossa rikki. Autoresoluution ja anti aliaisingin yhteiskäyttö kaataa pelin. (Unity Technologies 2015l.)

7.1 Testaajilta vastaanotettu palaute ja pelin muutokset

Ennen lopullisen version julkaisua haluttiin olla täysin varmoja siitä, että kaikki pelissä toimii suunnitellusti. Kun on kehittänyt peliä pitkän aikaa, voi helposti sokeutua sen kokonaiskuvalle, silloin on hyvä saada hieman ulkopuolista näkökulmaa asiaan.

Pelin alpha versio julkaistiin testattavaksi pienelle ryhmälle ihmisiä. Peli ei ollut vielä läheskään valmis, mutta se oli täysin pelattavissa ja oli mahdollista saada jo hyvä kuva siitä, miltä lopullinen versio voisi mahdollisesti näyttää.

Vastaanotettu palaute oli suurimmiltaan osin hyvin samanlaista. Pelaajat olivat pitäneet pelistä ja pitivät sitä hauskana. Yksi selvä kritiikki löytyi kuitenkin monelta, pelaajat pitivät pelihahmojen hyppykorkeutta hämmentävänä. Pelaajien oli vaikea ymmärtää miten jotkut vihollisten hyökkäyskuviot oli tarkoitus ohittaa ottamatta osumaa. Heidän mielestään olisi hieman alempi hyppykorkeus ollut parempi ja helpompi ymmärtää.

Toinen saamamme kritiikki oli muutaman sekunnin tauko, joka tapahtui jokaisen vihollisen hyökkäyskuvion l välissä. Olimme rakentaneet kenttämme niin, että aina ennen seuraavan vihollisten hyökkäyskuvion tuomista peliin olisi noin viiden sekunnin tauko. Tauot lisättiin peliin siksi, että pelaajilla olisi aikaa välillä hengittää. Emme halunneet kenttien olevan liian levottomia.

Kummatkin vastaanotetut kritiikit otettiin huomioon ja pelin kritisoituja pelialueita päätettiin hieman muokata. Single plane -kentissä hyppykorkeus oli ennen ruudun yläosaan asti. Hyppykorkeutta muokattiin niin, että nyt se on pelkästään ruudun keskiosaan asti. On kuitenkin edelleen mahdollista päästä ruudun yläosaan asti käyttämällä hyppyä ja sen jälkeen ylöspäin syöksyä. Double plane -kentissä hyppykorkeus oli jo valmiiksi melkein kentän keskiosassa, mutta sitä muokattiin silti hieman niin, että se olisi mahdollisimman lähellä single plane -kenttien hyppykorkeutta. Haluttiin, että pelaajien olisi helppo hahmottaa hyppykorkeudet molemmissa kentissä.

Hyppykorkeuden muuttaminen oli kuitenkin odotettua hankalampaa. Myös vihollisten hyökkäyskuvioita jouduttiin muokkaamaan niin, että ne toimisivat hyvin uusien hyppykorkeuksien kanssa. Muutoksien jälkeen tajuttiin kuitenkin miltei heti, että hyppykorkeuden muutos todellakin lisäsi enemmän monipuolisuutta pelihahmojen ohjaamiseen.

Hyökkäyskuvioiden väliset tauot täytettiin kolikkokuvioilla. Kolikkokuviot täyttävät tauot ilman levottomuuden lisäämistä. Kolikkojoukot eivät sisällä vihollisia, joten pelaajalla ei ole minkäänlaista tapaa satuttaa itseään ohittaessaan niitä. Kolikkojoukkojen tarkoitus onkin antaa pelaajille lisämahdollisuuksia kerätä enemmän kolikoita samalla antaen heille jotain tekemistä taukojen ajaksi. Pelaajat saavat edelleen hengähdystaukonsa, nyt heillä on vain jotain tekemistä myös niiden aikana.

8 Yhteenveto ja johtopäätökset

Unity3D tarjoaa mainiot työkalut pelinkehittämiseen. Unityn rajapinta on laaja ja monipuolinen. Internetistä löytyvä dokumentaatio on viimeisen päälle viimeistelty ja täynnä yksityiskohtia. Versiosta 4.6 lähtien on unityn mobiilipelikehitykseen upotettu paljon resursseja, ja kehitys on ollut valtavaa.

Kosunin pelin kaikki ominaisuudet, mitkä alkuvaiheessa suunniteltiin, niin pystyttiin myös pelimoottorin avulla toteuttamaan. Toteutus sujui suurimmiltaan osin täysin ongelmitta, vaikeuksia syntyi ainoastaan ulkopuolisten rajapintojen käyttämisen yhteydessä. Unity-pelimoottorin kanssa työskennellessä on oltava tarkkoja optimoinnin suhteen. Monet Unityn tarjoamat hieman hienommat ominaisuudet eivät käänny hyvin mobiilialustoille, on suunniteltava tarkkaan, mitä toimintoja voi käyttää ja mitä ei. Peli saatiin kuitenkin optimoitua hyvin. Ruudunpäivitys saatiin viilattua tasaiseksi myös mobiililaitteilla. Suuren roolin optimoinnissa ajoivat unity-pelimoottorin ominaisuudet kuten draw call batching ja coroutinet.

Testausprosessi oli pelin selviytymiselle väistämätöntä. Testaamisen aikana tietoon tulivat niin laitekohtaiset ongelmat kuin pelattavuudenkin ongelmat. Pelin kääntäminen eri käyttöjärjestelmille unity-pelimoottorilla ei ole täydellistä ja muutamia ongelmia esiintyi, ei kuitenkaan mitään ylitsepääsemätöntä. Loppujen lopuksi pelistä saatiin kehitettyä juuri sellainen kuin siitä kuviteltiin.

Lähteet

Aleksandr. 2014. Documentation, Unity scripting languages and you. Luettu 7.11.2015. http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/.

Bond Jason. 2015. iOS: anti-aliasing causes exc_bad_access exception. Luettu 7.11.2015. http://answers.unity3d.com/questions/988084/ios-anti-aliasing-causes-exc-bad-access-exception.html.

Brodkin Jon, 2013, How Unity3D Became a Game-Development Beast. Luettu 7.11.2015. http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/.

Burgun Keith. 2011. Understanding Balance in Video Games. Luettu 7.11.2015. http://www.gamasutra.com/view/feature/134768/understanding_balance_in_video_.php?print=1.

Disley Nicola. 2015. Texture Atlas and UV Remapping. 7.11.2015. http://nicoladisley.blogspot.fi/2015/03/texture-atlas-and-uv-remapping.html.

DrCrook. 2014. Object Pooling – A Great way to Increase Performance. Luettu 7.11.2015. http://indiedevspot.com/2014/07/08/object-pooling-great-way-to-increase-performance/.

Etherington Darrell. 2013. Unity Game Engine Goes Free For iOS, Android And BlackBerry 10 Developers. Luettu 7.11.2015. http://techcrunch.com/2013/05/21/unity-game-engine-goes-free-for-ios-and-android-developers/.

Fassihi Amir. 2013. “0 – 60 fps in 14 days!” What we learned trying to optimize our game using Unity3D. Luettu 7.11.2015. http://www.gamasutra.com/blogs/AmirHFassihi/20130828/199134/0__60_fps_in_14_days_What_we_learned_trying_to_optimize_our_game_using_Unity3D.php.

GameSparks. 2015. GameSparks FAQ. Luettu 7.11.2015. https://www.gamesparks.com/blog/gamesparks-faq/.

Muklashy Shadi. 2014. Object pooling for performance in Unity. Luettu 7.11.2015. http://www.sombr.com/2014/07/14/object-pooling-for-performance-in-unity/.

Page Gabriel. 2014a. Leaderboards. Luettu 7.11.2015. https://docs.games-parks.net/developer-portal/leaderboards.

Page Gabriel. 2014b. Downloadables. Luettu 7.11.2015. https://docs.games-parks.net/developer-portal/downloadables.

Seno. 2014. Problem when the ios build download AssetBundle via WWW.LoadFrom-CacheOrDownload. Luettu 7.11.2015. http://forum.unity3d.com/threads/problem-when-the-ios-build-download-assetbundle-via-www-loadfromcacheordownload.282487/.

Studica. 2015. Unity Pro 5: Create Games + Interactive Content. Luettu 7.11.2015. http://www.studica.com/about-unity-pro-5-3d-new-features-benefits.

Vliet Yorick Van. 2012. 4 Ways To Increase Performance of your Unity Game. 7.11.2015. http://www.paladinstudios.com/2012/07/30/4-ways-to-increase-perfor-mance-of-your-unity-game/.

Unity Technologies. 2015a. Build once deploy anywhere. Luettu 7.11.2015. http://unity3d.com/unity/multiplatform/.

Unity Technologies. 2015b. MonoBehaviour.Update(). Luettu 7.11.2015. http://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html.

Unity Technologies. 2015c. MonoBehaviour.Invoke. Luettu 7.11.2015. http://docs.unity3d.com/ScriptReference/MonoBehaviour.Invoke.html.

Unity Technologies. 2015d. Coroutines. Luettu 7.11.2015. http://docs.unity3d.com/Ma-nual/Coroutines.html.

Unity Technologies. 2015e. Rigidbodies. Luettu 7.11.2015. http://docs.unity3d.com/Ma-nual/RigidbodiesOverview.html.

Unity Technologies. 2015f. Colliders. Luettu 7.11.2015. http://docs.unity3d.com/Manual/CollidersOverview.html.

Unity Technologies. 2015g. Physics Manager. Luettu 7.11.2015. http://docs.unity3d.com/Manual/class-PhysicsManager.html.

Unity Technologies. 2015h. Tags. Luettu 7.11.2015. http://docs.unity3d.com/Manual/Tags.html.

Unity Technologies. 2015i. Sprite Packer. Luettu 7.11.2015. http://docs.unity3d.com/Manual/SpritePacker.html.

Unity Technologies. 2015j. Draw Call Batching. Luettu 7.11.2015. http://docs.unity3d.com/Manual/DrawCallBatching.html.

Unity Technologies. 2015k. AssetBundles. Luettu 7.11.2015. http://docs.unity3d.com/Manual/AssetBundlesIntro.html.

Unity Technologies. 2015l. Understanding Automatic Memory Management. Luettu 7.11.2015. http://docs.unity3d.com/Manual/UnderstandingAutomaticMemoryManagement.html.