OULUN AMMATTIKORKEAKOULU

Laura Vilhunen

2D-PELIDEMON TEKEMINEN DEFOLD-PELIMOOTTORILLA

2D-PELIDEMON TEKEMINEN DEFOLD-PELIMOOTTORILLA

Laura Vilhunen
Opinnäytetyö
Kevät 2017
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Internetpalvelut ja digitaalinen media

Tekijä(t): Laura Vilhunen
Opinnäytetyön nimi: 2D-pelidemon tekeminen Defold-pelimoottorilla
Työn ohjaaja: Matti Viitala
Työn valmistumislukukausi- ja vuosi: Kevät 2017 Sivumäärä: 48 + 2

Opinnäytetyön aiheena on luoda kaksiulotteinen teknillinen pelidemo vielä kehitysvaiheessa olevalla Defold-pelimoottorilla, samalla tutustuen pelimoottorin taustoihin, pelintekomekaniikkoihin ja muokkausohjelmistoon. Opinnäytetyöllä ei ole toimeksiantajaa, ja aihe valittiin oman mielenkiinnon mukaan.

Ihan ensimmäiseksi opinnäytetyön raportissa käydään läpi kolmea eri pelimoottoria, joita yleisesti käytetään kaksiulotteisten pelien kehittämiseen. Näitä ovat Game Maker: Studio, Clickteam Fusion ja Unity. Nämä kolme pelimoottoria esitellään pintaraapaisulla kertoen nopeasti taustoista, käytettävistä koodikielistä ja pelintekomekaniikoista. Lisäksi vertaillaan pelimoottorien eri versioita ominaisuuksien ja hintojen kannalta. Lopulta kuitenkin päädyttiin Defoldiin sen täyden ilmaisuuden ja uuteen pelimoottoriin tutustumisen halun vuoksi.

Seuraavaksi käsitellään itse Defold-pelimoottoria teoriapohjalta. Aluksi käydään läpi pelimoottorin taustaa tarkemmin kuin edellisiä pelimoottoreita. Lisäksi kerrotaan pelimoottorin ominaisuuksista ja mille alustoille sitä on saatavilla. Osion ensimmäisessä kappaleessa mainitaan myös tuetut tiedostomuodot. Tämän jälkeen selitetään lyhyesti pelimoottorin omistajasta, Kingistä, joka osti Defoldiin oikeudet. Tämän jälkeen käydään läpi pelimoottorin muokkainohjelmiston käyttöliittymää kohtuutarkasti tärkeimpien työkalujen osalta. Lopuksi tämä osio käsittelee lyhyesti Lua-koodikieltä taustoineen.

Kolmas osio on omistettu tämän opinnäytetyön pelidemon teon selittämiselle hyvin tarkasti vaihe vaiheelta. Ensimmäiseksi kerrotaan pelidemon konseptista, joka on hyvin yksinkertainen tasohyppely, jossa pelattavana hahmona käytetään pöllönpoikasta. Sitten selitetään peliprojektin luomisesta, kentän rakentamisesta ja lopuksi hahmon luomisesta. Hahmolle luodaan myös skripti, joka käydään läpi pala palalta. Lopuksi kootaan pelidemo yhteen pakettiin.

Lopputuloksena on pelidemo, jossa pöllönpoikanen pystyy liikkumaan ja hyppimään mobiilinäytölle sopivalla alueella. Pelidemolle ehdotetaan sitten jatkokehitysmahdollisuuksiksi satunnaisesti ilmestyvät tasanteet ja kentän lopun teko. Lopulta pohditaan opinnäytetyön kulkua ja mikä olisi voinut mennä paremmin. Opinnäytetyön kanssa oli aikatauluongelmia, joista kuitenkin päästiin loppua kohden ylitse. Kehitystyö tulee todennäköisimmin jatkumaan harrastusmielessä.

Asiasanat: defold, pelit, peliala, digitaaliset pelit, mobiilipelit, tietokonepelit

ABSTRACT

Oulu University of Applied Sciences
Business Information Systems, Internet Services and Digital Media

---

Author(s): Laura Vilhunen
Title of Bachelor´s thesis: 2D-gamedemo's making with Defold game engine
Supervisor(s): Matti Viitala
Term and year of completion: Spring 2017 Number of pages: 48 + 2

The subject of the thesis is to create a two-dimensional tech-demo on a still-in-development game engine Defold. It'll also take a look into the background, game making mechanics and the editor. The thesis doesn't have a client, and the subject was chosen based on writer's own interest.

First, the thesis takes a look at three different game engines that are typically used for two dimensional games: Game Maker: Studio, Clickteam Fusion and Unity. This part quickly explains the backgrounds, used programming languages and game making mechanics. In addition, different versions of the engine are being compared between features and prices. In the end, Defold was chosen as the main game engine due to it being completely free, and due to own interests of getting to know a whole new engine.

Second part explains the Defold engine on theoretical level. Firstly, it explains the background of Defold on a deeper level than was done with former engines. It also explains the main features and available platforms as well as supported file formats. After this, King, the owner of the engine, is briefly introduced. After this, the editor is explained in detail focusing on the most important parts. Lastly, this part of the thesis tells about Lua, the programming language used in Defold.

Third part of the thesis focuses on the process of the game demo making itself in detail. It first explains the basic concept used in the demo, which is a very simple platformer, where the playable character is a baby owl. Then it'll go into detail on how to create the project itself, how to create the level and lastly the character itself. The script used for character movement and physics are examined in detail.

The result is a game demo, where you can play as a baby owl, and move to the left and right while also jumping. Lastly, the possible future of the demo is by developing randomly generated platforms and creating the stage end. In the end, the process of the thesis was a hard journey to keep in check, but problems were overcome in time. The development will most likely continue on a hobby basis.

Keywords: defold, games, game industry, digital games, mobile games, computer games

SISÄLLYS

1 JOHDANTO 6
2 2D-PELINTEKO-OHJELMAT 7
2.1 Game Maker: Studio 7
2.2 Clickteam Fusion 9
2.3 Unity 10
2.4 Miksi Defold? 12
3 DEFOLD 13
3.1 King 15
3.2 Editorin käyttöliittymä 16
3.3 Lua-ohjelmointikieli 17
4 PELIDEMO 19
4.1 Konsepti 19
4.2 Projektin luonti 20
4.3 Projektin asetusten säätö 22
4.3.1 Yleisasetukset 22
4.3.2 Näyttö 22
4.4 Kansioiden rakenne 23
4.5 Kentän luonti 24
4.6 Pelihahmon luonti 33
4.6.1 Kuvasarja 34
4.6.2 Koodaus 35
4.6.3 Hahmon osien koonti 43
5 TULOKSET 46
6 POHDINTA 48
LÄHTEET 49

1 JOHDANTO

Tämän opinnäytetyön aiheena on tutustua peliyhtiö Kingin omistamaan 2D-pelimoottoriin nimeltä Defold ja toteuttaa pienimuotoisen pelisuunnitelman avulla toimiva teknillinen demo. Valitsin aiheen oman mielenkiinnon kautta: pidän hyvin paljon 2D-peleistä, joissa on edes jonkin verran tasohyppelyelementtejä. Itseäni kiinnostaa myös paljon pelialalle suuntautuminen, joten uuteen pelimoottoriin tutustuminen tuntui mukavalta idealta syventyä asiaan enemmän.

Tässä opinnäytetyössä esitellään ensimmäiseksi erilaisia 2D-peleille tarkoitettuja pelimoottoreita lyhyesti, jotta lukija ymmärtää, mitä pelimoottoreita 2D-pelien tekemiseen voi käyttää, ja millä hinnalla. Lisäksi mainitaan, mitä ohjelmointikieliä mikäkin pelimoottori käyttää. Osion lopussa selitän, miksi päädyin valitsemaan Defoldin oman pelidemon alustaksi.

Toisessa osiossa selitetään Defoldin taustaa ja esitellään ominaisuuksia julkaisualustoista käytettäviin tiedostomuotoihin ja ohjelmiston käyttöliittymää. Lisäksi tässä osiossa kerrotaan pikaisesti ohjelmiston omistajasta Kingistä ja Defoldissa käytettävästä ohjelmointikielestä Luasta.

Teoriaosuuden jälkeen päästään opinnäytetyön pääasiaan, eli itse pelidemon tekemiseen. Ensimmäiseksi selitetään konseptia, jonka kehitin pelimoottorin testausta varten ja sitten kerrotaan vaihe vaiheelta pelidemon etenemisestä. Koska aika on vähäinen, pelidemo keskittyy keskeisimpiin elementteihin, eli ohjattavuuteen, kentän luontiin, hahmon luontiin ja äänien liittämiseen peliin.

Lopussa pohdiskellaan opinnäytetyön etenemistahtia ja lopputulosta. Mietitään mikä sujui ja mikä olisi voinut mennä paremmin. Lisäksi pohditaan projektin mahdollista tulevaisuutta.

2 2D-PELINTEKO-OHJELMAT

2D-peleille tarkoitettuja pelinteko-ohjelmia löytyy markkinoilta useita. Nykypäivänä usea näistä tarjoaa sekä ilmaisia että maksullisia lisenssejä, joissa jälkimmäisessä on lisäominaisuuksia, jotka helpottavat pelintekoa huomattavasti. Seuraavaksi esitellään muutamia pelinteko-ohjelmia, joita käytetään 2D-pelien tekemiseen.

2.1 Game Maker: Studio

Game Maker: Studio on Yoyo Gamesin omistama, alkuperältään alankomaalaisen Mark Overmansin kehittämä pelinteko-ohjelma. Ohjelma perustuu drag & drop, suomennettuna "raahaa ja pudota" tekniikan käyttöön, eli ohjelman käyttäjältä ei vaadita laajoja ohjelmointitaitoja esimmerkiksi C#-kielen tai Javascriptin kanssa. Ohjelma käyttää ohjelmointikielenään omaa *Game Maker Language* -ohjelmointikieltä. Game Makerilla voi pääasiassa tehdä kaksiulotteisia pelejä, mutta kolmiulotteisia elementtejäkin voi tehdä rajatusti. (Wikipedia, viitattu 3.12.2016)

Alkujaan Animo-nimisenä 2D-animaatio-ohjelmana tunnettu Game Maker sai alkunsa vuonna 1999. Sitä kehitettiin niin, että ohjelmointia ei-osaavat voisivat rakentaa hyvälaatuisia pelejä helposti. Game Makerillä ei kuitenkaan pysty tekemään kovinkaan vaativaa peliä (PC Gamer, viitattu 7.12.2016). Pelinteko-ohjelma menestyy hyvin lähinnä indie-pelien eli itsenäisesti tehtävien pelien tekemisessä.

Game Maker Studioon voi tarvittaessa ladata lisäpalikoita eli assetteja Yoyo Gamesin omilta sivuilta löytyvästä Market Place -kaupasta. Sinne voi myös laittaa myyntiin Game Makerillä tehtyjä pelejä ennen kuin yrittää suuremmille markkinoille kuten Steamiin. Game Maker: Studiosta voi valita kolmen eri lisenssin välillä, jotka ovat nimeltään Studio Free, Studio Professional ja Studio Master Collection. (Yoyo Games 2016, viitattu 3.12.2016.)

Studio Free on täysin ilmainen, vapaasti ladattava versio, joka sisältää pelimoottorin kokonaisuudessaan. Se ei vaadi tekijänpalkkiota, eli käyttäjä saa julkaista pelimoottorilla tehdyt pelit ilman lisämak-

suja. Ilmaisversiolla voi käyttää myös Marketplacesta löytyviä lisäpalikoita. Julkaisualustoina voi käyttää ainoastaan Windowsia. (Yoyo Games 2016, viitattu 3.12.2016.)

Studio Professional on noin 150 euron lähtöhinnalla ostettava lisenssiversio, joka ilmaisen version ominaisuuksien lisäksi antaa mahdollisuuden julkaista useammalle alustalle, joita ovat mm. Xbox One, Playstation 3, 4 & Vita, sekä lisämaksuilla mm. IOS, Android, Ubuntu Linux ja myös HTML5. Lisäksi Professional-versiossa voi luoda oman Splash Screenin eli pelin käynnistyksissä näkyvät logot yn muut sellaiset, sekä saa luvan varhaisiin julkaisuihin ja palikoiden myymiseen YoyoGamesin Marketplacessa. Lisäksi tässä versiossa on mahdollisuus mobiilialustalla testaamiseen. (Yoyo Games 2016, viitattu 3.12.2016.)

Master Collection on muuten samanlainen versio Professionalin kanssa, mutta se sisältää kaikki lisäpalikat. Tämän version hinta on noin 800 euroa. (Yoyo Games 2016, viitattu 3.12.2016.)

KUVIO 1. Game Maker –pelimoottorilla tehty peli Undertale.

Game Maker: Studiolla on luotu esimerkiksi *Undertale (KUVIO 1),* Mossmouthin *Spelunky, Hyper Light Drifter* ja *Hotline Miami.* Game Maker Studiosta on tällä hetkellä beeta-vaiheessa uusi ohjelmistoversio, Studio 2 (Yoyo Games 2016, viitattu 3.12.2016). Game Makerin voi ostaa joko Yoyogamesin sivuilta tai Steam-kaupasta.

2.2 Clickteam Fusion

Clickteamin Fusion 2.5, entiseltä nimeltään Multimedia Fusion 2.5, on Game Maker: Studion tapaan drag & drop –mekaniikkaan pohjautuva pelinteko-ohjelma, jota voi käyttää myös sovellusten luomiseen. Clickteam Fusionissa ei pääse helpolla sorkkimaan koodipohjaa, mikä tekee siitä vaikeasti muokattavan. Siitä on saatavilla kolme eri versiota: ilmainen Free Edition, normaali versio ja Developer. (Clickteam Versions Comparision 2016, viitattu 27.12.2016.)

Ilmainen versio, **Free Edition** on hyvin riisuttu normaaliversiosta. Mahdollisista pelialustoista voi julkaista vain html5-pohjaisia pelejä, ja niitäkin vain rajatuilla ehdoilla. Tällä versiolla kuitenkin pystyy tekemään pelille Editorin kautta Story Boardin (Tarinankerronta), muokkaamaan kuvaruutuja, tapahtumia, dataa, animaatioita ja kuvia. Tuettuja tiedostomuotoja ovat BMP, GIF, JPG, PNG, PCX ja FLC. (Clickteam Versions Comparision 2016, viitattu 27.12.2016.)

Pelimoottorin **normaalissa versiossa** pystyy ilmaisversion ominaisuuksien lisäksi julkaisemaan pelejä Windowsille, Xboxille, Androidille, iOSille ja Flash-muodolle. Mac-version luomiselle sopivaa moduulia luvataan myöhemmälle ajankohdalle. Editorissa on lisänä tapahtumalistan muokkaus, projektitiedostot, alfakanavat ja binääritiedostot. Tuetuissa tiedostomuodoissa on lisänä AVI ja TGA. Tässä versiossa voi myös muun muassa luoda omia lisäpalikoita SDK-kitin avulla. Normaaliversion hinta on noin 80 euroa Steamissa. (Clickteam Versions Comparision 2016, viitattu 27.12.2016.)

Developer versiossa lisäominaisuudet liittyvät lähinnä pelin sisäisiin ominaisuuksiin, esimerkkinä mahdollisuus muokata EXE tiedostojen versiotietoja. (Clickteam Versions Comparision 2016, viitattu 27.12.2016.)

KUVIO 2. Clickteam Fusionilla toteutettu Freedom Planet.

Clickteam Fusion 2.5 ohjelmistolla on luotu esimerkiksi *Freedom Planet (KUVIO 2), Five nights at Freddy's* ja *The Escapists.* (Steam 2016, viitattu 13.12.2016.)

2.3 Unity

Unity Technologiesin Unity on Clickteam Fusionista ja Game Maker: Studiosta poiketen 3D-pelimoottori, johon on uusimpien versioiden ja suuren yleisön halun myötä tullut tuki 2D-pelien tekoon. Ohjelmointikielinä Unityssa voi käyttää JavaScriptiä (UnityScript), C#-kieltä ja Boo-kieltä. Ohjelmoinnin lisäksi tässä pelimoottorissa voi rakentaa pelin myös pelipalikoista ilman kooditaitoa. Unity on täysin roalty-free, eli yhtiö ei peri pelimoottorilla tehdyistä peleistä rahaa itselleen.

Unity julkistettiin vuonna 2005 Applen kehittäjämessuilla alkujaan Mac-pelien kehitykseen, mistä se on myöhemmin laajentanut alustojaan 21 eri vaihtoehtoon. Unitysta on tähän mennessä julkaistu viisi isompaa ohjelmistoversiota, uusimpana versionumerona 5.5.0. Unity tarjoaa omien ladattavien lisäpakettien lisäksi kaupan, josta saa ostettua tai ladattua ilmaiseksi sekä Unity-yrityksen että yhteisön tekemiä lisäpaketteja. (Unity Wikipedia, viitattu 12.1.2017.)

Unitysta on saatavilla neljä eri versiota ja niiden käytettävyys riippuu yrityksen tuotoista. Nämä versiot ovat nimiltään Personal, Plus, Pro ja Enterprise. Jokaisella versiolla voi tehdä pelejä kaikille mahdollisille alustoilla, joita on muun muassa PS4, PC, Linux, Mac, PS3, Wii U, android, iOS ja uusimpana Switch. Jokainen versio myös sisältää kaikki pelimoottorin ominaisuudet ja kaikki päivitykset ovat saatavilla kaikille versioille. (Unity Store 2017, viitattu 30.1.2017.)

Personal on ilmaiseksi ladattava versio, jonka sanotaan olevan hyväksi aloittelijoille ja harrastajille. Koska tämä on ilmainen versio, pelien käynnistyessä välähtää aina *Made with Unity* -käynnistyskuva (MWU Splash Screen). Unityn pilvipalveluun joutuu odottamaan jonossa pisimmän ajan, ja monipeleihin annetaan olevan paikalla vain maksimissaan 20 pelaajaa. Ilmaisversiota voi käyttää sillä ehdoin, ettei peli tuota 100 000 euroa enempää vuodessa. (Unity Store 2017, viitattu 30.1.2017.)

Plus-versio on saatavilla kuukausiveloituksella hintaan 32 euroa yhdelle koneelle ja sitä esitellään sopivaksi vakaville pelinkehittäjille. Tämä versio antaa luvan poistaa MWU-käynnistyskuvan ja joko käyttää omatekemää käynnistyskuvaa, tai poistaa kyseisen ominaisuuden omasta peliprojektistaan kokonaan. Pilvipalveluun pääsee jonotukseen nopeammin kuin ilmaisversiossa, ja moninpeleissä voi olla nyt maksimissaan jopa 50 pelaajaa. Lisäksi tästä versiosta lähtien pääsee käsiksi Pro UI Editoriin, suoritusraportteihin ja Unity-tiimin hallinnointityökaluihin. Lisäksi tämän version tilaamalla saa Asset Storessa 20% alennusta. Tätä versiota saa käyttää 200 000 euron vuositulorajalla. (Unity Store 2017, viitattu 30.1.2017.)

Pro-versio nostaa kuukausihinnan 115 euroon ja sen sanotaan olevan oiva valinta ammattitasolle. Sillä ei ole tulorajakattoa, ja projektista saa laitettua Unityn omaan pilvipalveluun useamman eri koontiversion lyhyimmällä jonotusajalla. Moninpeliin pystyy tässä versiossa osallistumaan jopa 200 pelaajaa yhtä aikaa. Lisäksi tässä versiossa saa mahdollisuuden päästä käsiksi lähdekoodiin kysymällä Unityn myyntipäälliköltä. Asset Storessa saa nyt 40% alennuksen. (Unity Store 2017, viitattu 30.1.2017.)

Enterprise on yrityksille tarkoitettu versio, jonka hinnasta pitää puhua Unityn kanssa. Monipeleissä ei ole enää käyttäjäkattoa, ja Unityn pilvipalveluun saa oman osion käyttöönsä nopeimpia latausaikoja varten. (Unity Store 2017, viitattu 30.1.2017.)

KUVIO 3. Unitylla tehty Ori and the Blind Forest.

Unityn tunnetuimpia julkaistuja 2D-pelejä ovat esimerkiksi *Ori and the Blind Forest* (KUVIO 3), *BROFORCE* ja *Enter the Gungeon.* (Made with Unity 2017, viitattu 30.1.2017.)

2.4 Miksi Defold?

Tämä osio on nyt käsitellyt pelimaailman kolme suosituinta 2D-peleihin käytettävää pelimoottoria. Jokaisessa on hyvin yksinkertaiset tavat saada pelinteko alkuun, ja muokattavuus on Unityssa parhainta. Näistä kolmesta poiketen Defold on kuitenkin täysin uusi pelimoottori markkinoilla, eikä sillä ole ehditty tekemään mobiilialustan ulkopuolella hittipelejä.

Seuraavassa kappaleessa tutkitaan Defoldin taustoja, ominaisuuksia ja sitä, mitä kaikkea sillä pystyy tekemään. Lisäksi käydään pikaisesti läpi peliyhtiö King ja Lua-skriptikieli.

3 DEFOLD

Defold on ruotsalaisen peliyhtiö Kingin omistama, vielä raportin kirjoitushetken aikana beeta-vaiheessa oleva kaksiulotteisten (2D) pelien pelimoottori. Sen tekijöitä ovat Ragnar Svensson ja Christian Murray, jotka aloittivat pelimoottorin luomisen vuonna 2009. Itse King tuli pelimoottorin projektiin mukaan yhteistyöläiseksi vuonna 2013 ja on toteuttanut sillä esimerkiksi Blossom Blast Saga -pelin (KUVIO 4). Defold tuli Kingin omistukseen vuonna 2014 ja julkaistiin julkiseen käyttöön vuonna 2016. (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

KUVIO 4. Blossom Blast Saga, Defoldilla tehty peli.

Blossom Blast Saga on mobiilipeli, jossa tavoitteena on kasvatella kukkia yhdistelemällä kolme tai useampi samanvärinen kukka. Pelissä voi myös käyttää niin sanottuja boostereita, jotka auttavat pelinkulussa ja lisäksi pelinkulkua vaikeuttamassa on niin sanottuja blockereita. (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

Defoldia mainostetaan helpoksi kehittämisalustaksi peleille, koska sen kanssa ei tarvitse miettiä eri alustojen omia koodeja, vaan pelintekijä voi julkaista oman tuotoksensa kuudelle eri alustalle yhdellä napinpainalluksella. Tällä hetkellä tuotettuja alustoja ovat Windows, Mac OS X, Linux, iOS, Android ja HTML5. Pelimoottoria myös mainitaan hyvin optimoiduksi nopean pelinluonnin ja editorin pienen koon takia. Editori vie koneelta tilaa kirjoitushetkellä vain 5 megatavua ja suunnitteilla on jopa tiiviimpi paketti. (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

Pelimoottorin editori toimii Windowsilla, Linuxilla ja Mac OSllä ja sen voi ladata ilmaiseksi omaan käyttöön Defoldin omilta sivuilta. Editorista ei tarvitse maksaa tekijänoikeusmaksuja (royalty), lisenssejä tai käyttöoikeuksia parempaan versioon. Syyksi King sanoo, että "mitä useampi ihminen käyttää heidän pelimoottoriaan, sen paremmaksi se saadaan kehitettyä" (King 2016, viitattu 27.5.2016). He uskovat, että pelimoottorin jakaminen julkiseen käyttöön auttaa Defoldin kehittämisessä, kun ihmiset tekevät tutoriaaleja, ilmoittavat löydetyistä bugeista ja auttavat täydentämään Kingin omaa dokumentaatiota. He myös sanovat itse käyttävänsä tätä pelimoottoria jokapäiväisesti, joka motivoi kehittämään sitä jatkuvasti. Pelilogiikan toteutus hoituu Lua-skriptillä, joka on hyvin kevyt, tehokas ja nopea ohjelmointikieli, ja hyvin helppo oppia laajan netistä löytyvän dokumentaation ansiosta. (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

Defoldilla pystyy myös rakentamaan pelille tarvittavat animaatiot. Nämä voi olla tehtynä joko niin sanotusti animaatiokirjan (flipbook) avulla, tai jos omistaa erillisen ohjelman, spine-luuanimaatioita käyttäen. Editori tukee avainkehyksiä (keyframe) ja kaikkia graafisia käyttöliittymäelementtejä, myös partikkeleita, pystyy animoimaan. Kyseisiä partikkeleita pystyy luomaan ja muokkaamaan editorista käsin reaaliaikaisen esikatselun kanssa. Partikkeleiden arvoja pystyy muokkaamaan kurvieditorissa. Grafiikkapuoli sisältää partikkeleiden lisäksi materiaalit ja varjostimet (shader). (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

Graafisen käyttöliittymän (Graphical User Interface, GUI) muokkaaminen tapahtuu Defoldin editorin avulla ja siihen kuuluvat tasot, tekstit, kuvat ja tasomaskit. GUI:n ulkoasu on automaattisesti muokkautuva ja tukee suuntamuutoksia. Pelimoottori tukee myös 2D- ja 3D-fysiikoita kokonaan integroituna. Äänien kannalta Defoldissa on tuki OGG- ja WAV-äänitiedostoille, joita voi miksata ja ohjelmoida toimimaan halutulla tavalla. (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

Defoldin käyttämiseen vaaditaan Google-tiliä, ja peliprojektit tallentuvat tällä hetkellä Defoldin omalle palvelimelle. Syyksi King on tälle antanut sen, että kyseisen pelimoottorin oli alun perin suunniteltu olevan ns. palvelukokonaisuus, johon sisältyy tallennuspaikka ja työskentelyvälineet. He kuitenkin ovat tekemässä Defoldista versiota, jossa projektin voi tallentaa käyttäjän haluamaan tilaan. Versionkontrollointi tapahtuu gitin, eli versionhallintaohjelmiston, avulla, mikä tekee ryhmätyöskentelystä helppoa. (Defold: Free 2D Game Engine for 2D games 2016, viitattu 25.5.2016.)

3.1 King

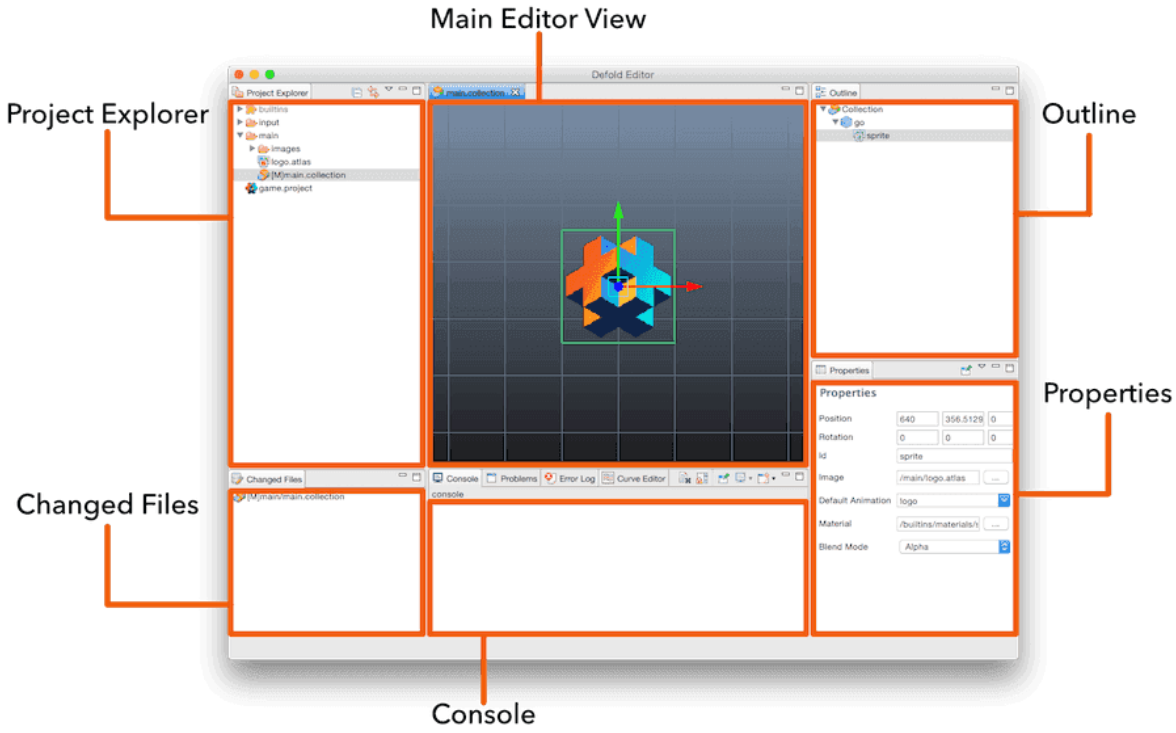
King Digital Entertainment PLC, lyhyemmin King, on vuoden 2003 elokuussa perustettu ruotsalainen peliyhtiö, joka keskittyy sosiaalisten pelien tekemiseen. Yhtiö on tätä tekstiä kirjoittaessa mobiilimarkkinoilla johtavin interaktiivinen viihdeyhtiö, jolla on yli 200 pelinimikettä allaan. (King.com 2016. Viitattu 15.6.2016.)

Peliyhtiö keskittyy suunnittelemaan pelejä sillä ajatuksella, että niitä voi pelata hetken aikaa, sitten jatkaa päivää normaaliin malliin ja jatkaa pelaamista myöhemmällä hetkellä. Kingin pelit yleensä sisältävät myös sisäisen synkronisoinnin, joka mahdollistaa useamman laitteen välillä pelaamisen samalla pelidatalla. (King.com 2016. Viitattu 15.6.2016.)

Kingillä on eri pelistudioita muun muassa Tukholmassa, Lontoossa, Berliinissä ja Seattlessa, kuin myös toimistoja muun muassa Tokiossa, San Fransiscossa ja Soulissa. Helmikuusta 2016 lähtien, King on ollut Activision Blizzardin omistuksessa itsenäisenä yksikkönä. (King.com 2016. Viitattu 15.6.2016.)

3.2 Editorin käyttöliittymä

Defoldin editori (KUVIO 5) on käyttöliittymältään hyvin yksinkertainen. Päänäkymä on jaettu seuraaviin eri osioihin: Project Explorer (Projektinhallinta), Main Editor View (Päämuokkausnäkymä), Outline, Changed Files (Muutetut tiedostot), Console (Konsoli), Problems (Ongelmat) ja Properties. Jokaista osiota voi liikutella vapaasti editorista irtonaisena, mikä mahdollistaa editorin näkymän muokkaamisen käyttäjän halun mukaan. (Defold game development manual 2016, viitattu 10.8.2016.)



KUVIO 5. Defoldin editorinäkymä jaoteltuna

Project Explorerissa näkyy kaikki projektiin liittyvät tiedostot. Eri tiedostotyypit näkyvät eri pikkukuvina, ja tiedostot saa auki päämuokkausnäkymään kaksoisklikkaamalla. Tiedostojen seasta löytyy jokaisessa projektissa ainoastaan luettavissa oleva kansio nimeltä ”builtins”, joka sisältää jokaisen projektin kannalta tärkeät tiedostot, kuten esimerkiksi renderöintiskripti, fontti ja renderöintejä varten tarkoitettuja materiaaleja. (Defold game development manual 2016, viitattu 10.8.2016.)

Päämuokkausnäkymässä (Main Editor View) näkyy sillä hetkellä muokkauksessa oleva tiedosto, kuten pelinäkymä, skriptit ja kokoelmat. Jokainen muokkauksessa oleva tiedosto avautuu omaan välilehteen. (Defold game development manual 2016, viitattu 10.8.2016.)

Outlinessa näkyy sen hetkinen muokkauksessa oleva tiedosto jaotellussa näkymässä. Tässä paneelissa voi muokata, poistaa, lisätä ja valita objekteja sekä komponentteja. (Defold game development manual 2016, viitattu 10.8.2016.)

Muokatuissa tiedostoissa (Changed Files) näkyy viimeisimmät muokkaukset projektin sisällä viimeisimmän projektisynkronoinnin aikana. Eli jos näet tässä paneelissa mitään merkintöjä, projektissa on tehnyt muokkauksia, joita ei ole vielä synkronoitu projektin tallennuspalvelimelle. (Defold game development manual 2016, viitattu 10.8.2016.)

Konsoli (Console) näyttää pelimoottorista tulevat logit, virheet ja debugaukset silloin kun peli on käynnissä. Tätä paneelia vilkuillaan varsinkin silloin, kun peli ei tahdo käynnistyä. Konsolin takaa löytyy myös välilehtiä virheilmoituksille ja kurvieditorille (curve editor), jota käytetään partikkeliefektien tekoon. (Defold game development manual 2016, viitattu 10.8.2016.)

Arvoissa (Properties) näkyy valitun objektin tai komponentin eri arvot. (Defold game development manual 2016, viitattu 10.8.2016.)

3.3 Lua-ohjelmointikieli

Lua on ilmainen skriptauskieli, joka on tehokas, käytännöllinen, kevyt ja hyvin uppoutuva. Luan on suunnitellut, luonut ja ylläpidossa Rio De Janeiron Paavillisen katollisen yliopiston tiimillä. Skriptauskielen nimi tulee portugalin kielestä, ja tarkoittaa kuuta. Luan sanotaan olevan skriptikielistä nopeimmin toimiva sekä testausympäristöissä että oikeassa elämässä. (Lua: about 2016, viitattu 27.05.2016.)

Lua eroaa muista koodikielistä yksinkertaisuudellaan. Missä esim. C#-kielessä pitää käyttää erinäisiä päätteitä sulkemaan eri skriptilauseita, Luassa ei ole tarvetta kirjoittaa päätteitä rivin loppuun, ja funk-

tiolauseet suljetaan yksinkertaisesti sanalla end. Kuvio 6 skripti toimii esimerkkinä Lua-skriptirakenteesta, joka aukaistaan tarkemmin luvussa 4.2.6.

```
local function update_animations(self)
    -- make sure the player character faces the right way
    sprite.set_hflip("#sprite", self.direction < 0)
    -- make sure the right animation is playing
    if self.ground_contact then
        if self.velocity.x == 0 then
            play_animation(self, anim_idle)
        else
            play_animation(self, anim_run)
        end
    else
        if self.velocity.y > 0 then
            play_animation(self, anim_jump)
        else
            play_animation(self, anim_fall)
        end
    end
end
```

KUVIO 6. Esimerkki Lua-skriptikielestä

4 PELIDEMO

Tässä osiossa käsitellään sitä, miten Defold-pelimoottorilla voidaan tehdä pelien perusasiat eli tärkeimmät asetukset, hahmon ja kentän luonti. Alussa esitellään konsepti, joka pyrkii olemaan pohjana pelidemolle. Tämän jälkeen selitetään kohtuutarkasti projektin luomisesta lähtien, lopettaen pelidemon osien yhdistämiseen kokonaisuudeksi.

4.1 Konsepti

Jotta pelimoottoria saa kunnolla tutkittua, on tutkimuksen rinnalla hyvä olla pelisuunnitelma siitä, mitä aikoo alkaa rakentaa. Tästä syystä projektille alettiin kehittää pienimuotoista pelisuunnitelmadokumenttia, johon kirjoitettiin pelissä tarvittavat elementit.

Pelikonsepti, jota alettiin työstään toimivaksi demoksi Defoldissa, on yksinkertainen ylöspäin etenevä 2D-tasohyppely. Pelaaja on pöllönpoikanen (KUVIO 7), joka on tippunut pesästä, ja koska ei osaa vielä lentää, alkaa hyppiä takaisin pesäänsä kohti. Koska aika on kuitenkin vähäinen, keskitytään tässä opinnäytetyössä pelihahmon ja kentän osien toimivuuteen, eikä demolla tule siis olemaan loppukohtausta.

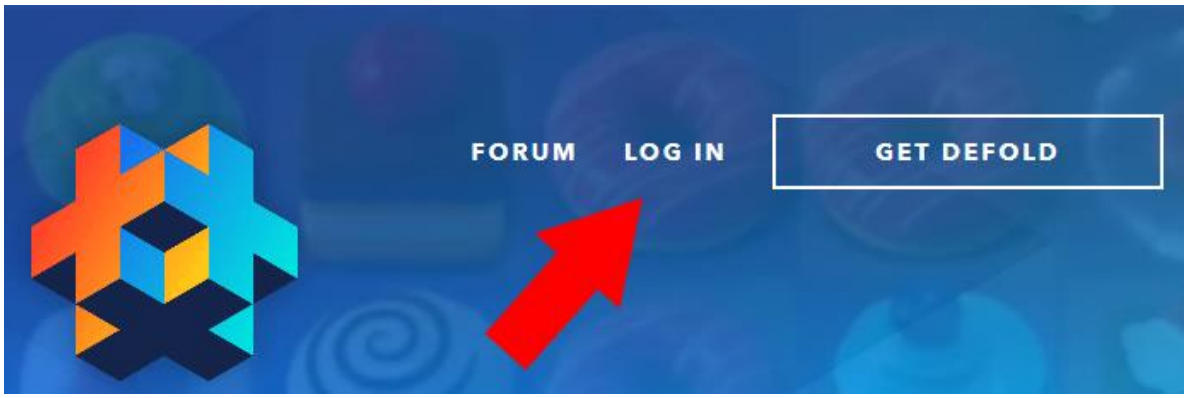
KUVIO 7. Demossa käytettävä pöllönpoikanen.

Pelin graafinen puoli pidetään yksinkertaisena, jotta niiden tekemiseen ei menisi liian kauan aikaa. Animaatiot pysyvät hyvin yksinkertaisella tasolla, tarkoittaen että animaatio ei ole täysin sujuvaa, mutta muuten hyvännäköistä.

Pelin näköalue on pystysuoran muotoinen, jolla pyritään rajaamaan alueen näkyvyyttä sopivaksi ylöspäin liikkuvalle tasohyppelylle. Ajatuksen takana on myös se, että peli sopisi myös mobiilinäytöille paremmin. Projektin kehittämisalustana toimii kuitenkin pääasiallisesti Windows, koska opinnäytetyön kirjoittajalta ei löydy Android-laitteistoa, ja HTML5 kaipaa vielä lisätukea toimiakseen kunnolla kirjoitushetkellä.

4.2 Projektin luonti

Projektin aloittamiseen tarvitaan google-tili ja internet. Projektin luonti tapahtuu Defoldin omilla sivuilla. Ensimmäiseksi kirjaudutaan sisään Defoldin palveluun pelimoottorin kotisivun etusivulla löytyvästä linkistä (KUVIO 8).

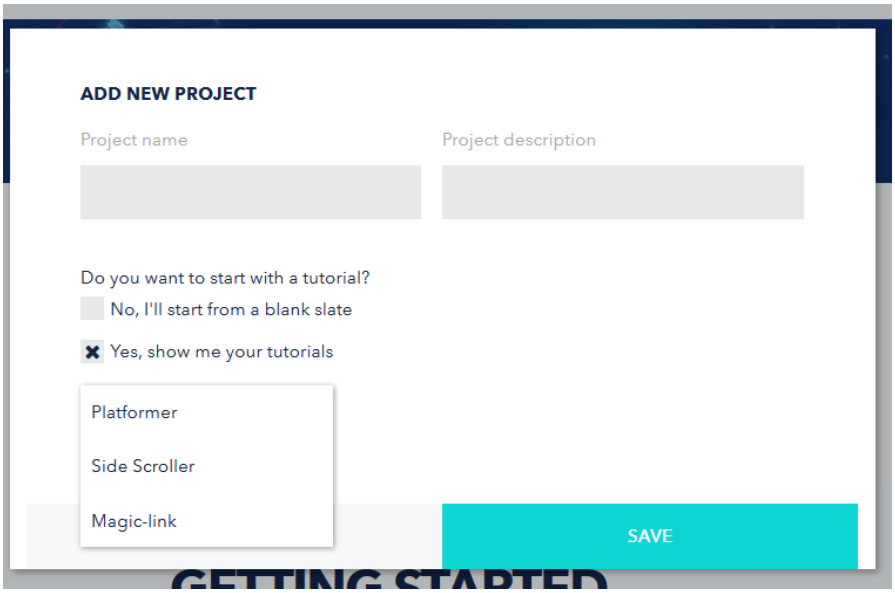


KUVIO 8. Defoldin etusivun Log In linkistä pääsee palveluun.

Palveluun eli Dashboardiin kirjautumisen jälkeen pystytään lataamaan Defoldin editori selaimen oikeasta yläreunasta. Samasta palvelusta pystyy myös helposti seuraamaan tutoriaaleja projektin aloittamisesta tutoriaalidemon loppuun.

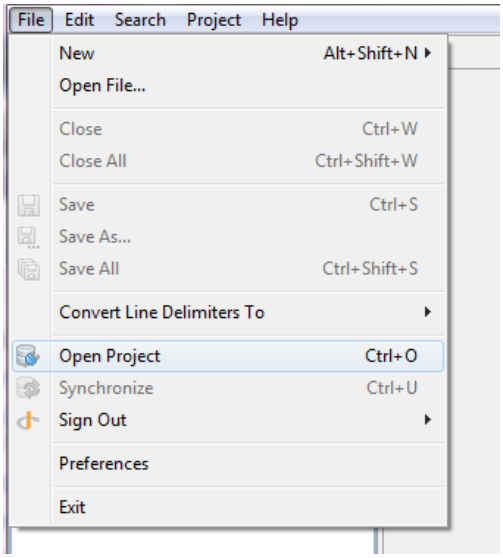
Projektin luonti tapahtuu vasemman puolen napeista. Klikataan +*Add Project*-nappia, ja selaimen keskelle ilmestyy **Add New Project** –ikkuna (KUVIO 9). Tähän ikkunaan pitää kirjoittaa projektin

nimi, kuvaus, ja voi myös valita aloittaako puhtaalta pöydältä, vai haluaako nähdä yhden kolmesta valittavasta tutoriaalista.



KUVIO 9. Defoldin Projektin luonti-ikkuna

Seuraavaksi aukaistaan projekti aiemmin ladatun editorin kautta. Editori vaatii Google-tilillä kirjautumisen, jotta voi aukaista kyseisellä tilillä olevia projekteja. Editorinäkymässä klikataan vasemmassa yläpalkissa *File*-nappia. Esille tulee lista, josta klikataan seuraavaksi nappia *Open Project* (KUVIO 10).



KUVIO 10. Defoldin projektin aukaisu.

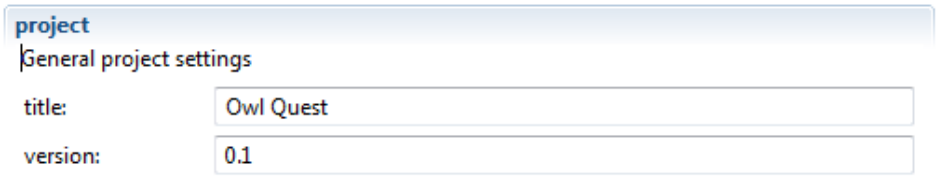
Seuraavaksi editori pyytää kirjautumaan Google-tilille, jos sitä ei ole vielä tehty. Sen jälkeen valitaan projekti, joka halutaan aukaista. Sitten luodaan projektille ns. oksa (branch), johon projekti tallentuu.

4.3 Projektin asetusten säätö

Kun projekti on avattu Defoldin editorissa, Project Explorerissa on jo esillä erilaisia kansioita ja tiedostoja. Asetusten kannalta tärkein tiedosto on *game.project*. Kaksoisklikkaamalla tätä tiedostoa aukeaa päänäkymäeditoriin kaikki peliprojektin keskeiset asetukset. Tässä alakappaleessa käsitellään ne asetukset, jotka olivat tämän projektin kannalta tärkeimpiä.

4.3.1 Yleisasetukset

Ensimmäinen alue **project** (KUVIO 11) määrittelee projektin yleiset asetuksen, kuten otsikon ja version.



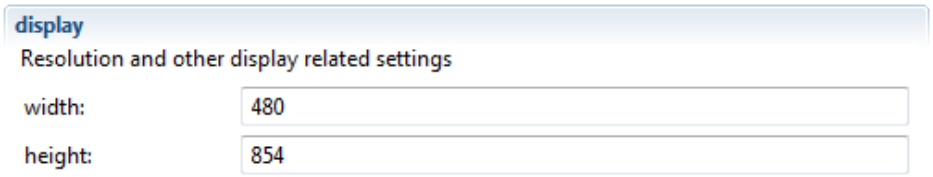
KUVIO 11. Projektin yleiset asetukset title ja version.

Otsikko (title) määrittää applikaation otsikon. Tässä projektissa laitettiin otsikoksi *Owl Quest.*

Versio (version) määrittää applikaation versionumeron. Tämä helpottaa pitämään peliversiot erillään toisitaan.

4.3.2 Näyttö

Toinen alue **display** sisältää näytön kannalta tärkeimmät asetukset, kuten leveyden, pituuden, kokonäytön ja ruudunpäivitysnopeuden.



KUVIO 12. Projektin näyttöasetukset width ja height.

Leveys (width) ja **korkeus (height)** (KUVIO 12) määrittelevät resoluutioalueen, millä pelinäkymä näytetään ruudulle. Tässä projektissa nämä arvot on laitettu vertikaalisesti näytölle sopivaan resoluutioon.

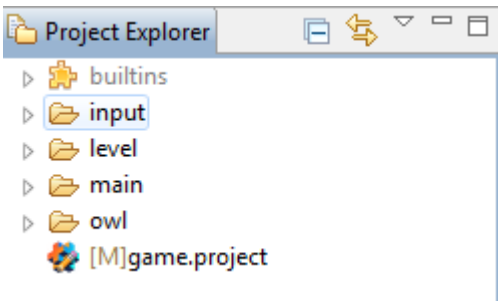
KUVIO 13. Näyttöasetukset fullscreen ja update_frequency.

Fullscreen (KUVIO 13) määrittää, halutaanko koko näytön kattava kuva vai ei. Tässä projektissa tämä ruutu on jätetty valitsematta.

Update_frequency (KUVIO 13) eli toisin sanoen **ruudunpäivitysnopeus (frames per second)** määrittää, kuinka nopeasti ruudun halutaan päivittyvän pelissä. Mitä isompi luku, sitä parempi nopeus.

4.4 Kansioiden rakenne

Jokaisen projektin kanssa on tärkeää, että tiedostot ovat helposti löydettävissä ja loogisissa paikoissa. Jokainen Defoldissa luotu projekti sisältää kolme eri kansiota: Builtins, Input ja Main.



KUVIO 14. Projektin kansioiden rakenne.

Builtins-kansio sisältää ne tiedostot, jotka ovat Defoldille hyvin tärkeitä, eikä niitä kannata mennä poistamaan. Tämän kansion alta löytyy muun muassa fontteja, renderöintiin liittyviä tiedostoja, skriptejä, materiaaleja jne.

Input-kansion alta löytyy tiedosto nimeltä *game.input_binding*, jossa määritellään pelissä tapahtuvien toimintojen näppäimet. Esimerkiksi voidaan määrittää liikkuminen haluttuihin näppäinkomentoihin, kosketusnäyttökomentoihin ja niin edelleen.

Main-kansio on se kansio, minkä alla itse pääpelikohtaus on. Tämän alle myös laitetaan usein kaikki omat peliprojektiin liitettävät tiedostot.

Näiden kolmen kansion lisäksi tähän projektiin on luotu kaksi uutta kansiota: level-kansio kentän osille ja owl-kansio pelihahmon osille.

4.5 Kentän luonti

Jotta tässä projektissa voidaan päästä kentän luomisen pariin, käydään ensin pari termiä läpi. Näitä ominaisuuksia tullaan käyttämään myös myöhemmin hahmon luonnissa.

Atlas on Defoldin käyttämä kuvakartasto, missä kuvat automaattisesti yhdistetään isommaksi kuvaksi (Defold Manual – 2D Graphics 2017, viitattu 22.2.2017). Atlasiin sisällytetään tiettyyn peliobjektiin tarvittavat kuvat. Esim. animaatioiden luonti tapahtuu atlasin sisäisellä *anim group* -toiminnolla.

Game Objectit eli **peliobjektit** ovat yksinkertaisia objekteja, joilla jokaisella on oma elinaika pelinkulun aikana. Nämä objektit ovat yleensä visuaalisesti tai äänillisesti koottuja. Niihin voi myös liittää käytöstapoja skriptikomponenttien avulla. Peliobjekteja luodaan ja laitetaan osaksi collectioneita editorissa tai laitetaan ilmestymään pelin käynnistyessä. (Defold Manual - Building Blocks 2017, viitattu 22.2.2017.)

Collisionobject eli **törmäytinobjekti** on komponentti, jota käytetään lisäämään peliobjekteihin fysikaalisia käytöstapoja. Tällä komponentilla voi määrittää peliobjektin painon, palautumisen ja kitkan. Sillä voi myös määrittää peliobjektin muodon. Defold tukee tällä hetkellä neljää eri törmäytintyyppiä:
Static eli staattiset objektit, jotka eivät liiku, mutta joihin voi osua.
Dynamic eli dynaamiset objektit, joita käytetään silloin, kun halutaan objektien käyttäytyvän realistisesti.
Kinematic eli kinemaattiset objektit, jotka törmäävät toisiin objekteihin, mutta reaktio on tekijän käsissä. Tätä käytetään myös silloin, kun halutaan, että objektin collider ei huomioi tiettyä collideria.
Triggers eli laukaisin, joka määrittää siihen osuessa jonkun tapahtuman käynnistämisen. (Defold Manual - Physics 2017, viitattu 22.2.2017.)

Collections eli **kokoelmat** on Defoldin oma tiedostomuoto uudelleenkäytettäville malleille, toisin sanoen prefabeille. Nämä kokoelmat voivat sisältää peliobjekteja ja muita kokoelmia. Kokoelmat, jotka on laitettu editorissa esille eivät ole muokattavissa. (Defold Manual - Building Blocks 2017, viitattu 22.2.2017.)

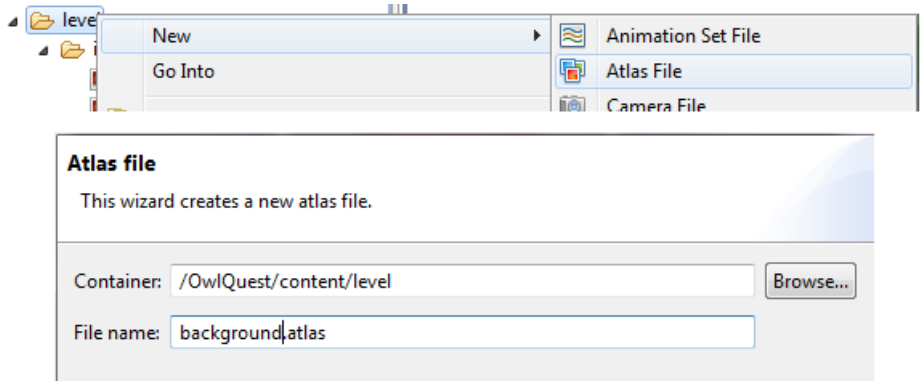
Sprite komponentteja käytetään grafiikkojen lisäämisen ja flip-book animaatioihin peliobjekteissa. Näitä tyypillisesti käytetään hahmojen ja kentän osien luomiseen. (Defold Manual – 2D Graphics 2017, viitattu 22.2.2017.)

Tässä projektissa kaikki, mitä tehdään tässä luvussa, laitetaan aiemmin luodun level-kansion alle. Kentän luominen aloitetaan tekemällä atlas kentän taustalle. Taustan kuvana käytetään tämän raportin tekijän tekemää puuta (KUVIO 15).

KUVIO 15. Pelin kentän taustana käytettävä puu.

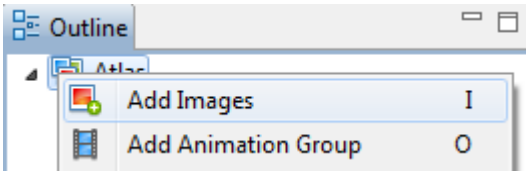
Ensimmäiseksi lisätään kyseinen puu projektiin. Project Explorerissa klikataan hiiren oikealla näppäimellä level-kansiota ja luodaan kuville oma kansio nimellä *image.* Tämän alle lisätään tämän puun tiedosto raahaamalla tiedosto sen lähtökansiosta projektin hierarkiaan. Puutiedoston nimenä käytetään bg.png.

Kun tiedosto on saatu kansiohierarkiaan, on aika luoda taustalle atlas-tiedosto. Atlas-tiedosto luodaan samaan tapaan kuin kansiokin. Eli oikealla hiirennäppäimellä klikataan level-kansiota ja valitaan *New* > *Atlas File* (KUVIO 16). Tämän tiedoston nimeksi laitetaan background.atlas.



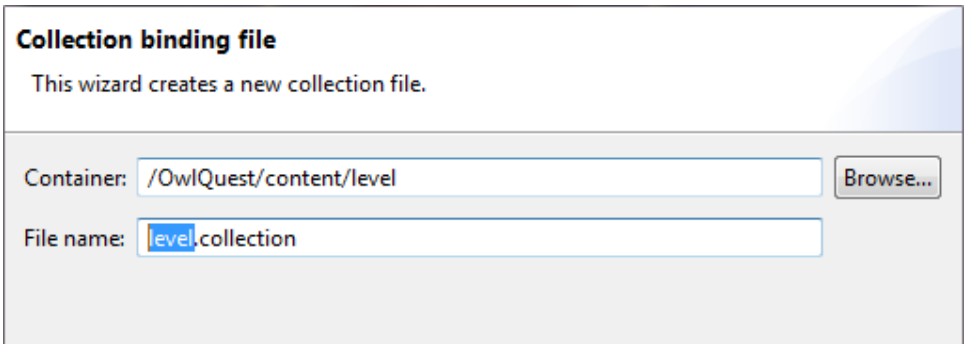
KUVIO 16. Atlas-tiedoston luonti.

Kun atlas on luotu, lisätään aiemmin projektiin tuotu taustakuvatiedosto atlasin hierarkiaan. Kun background.atlas on auki, Outline-ikkunassa klikataan hiiren oikealla näppäimellä *Atlas* > *Add Images* (KUVIO 17). Tämän jälkeen taustan kuvakirjasto on valmis käytettäväksi.



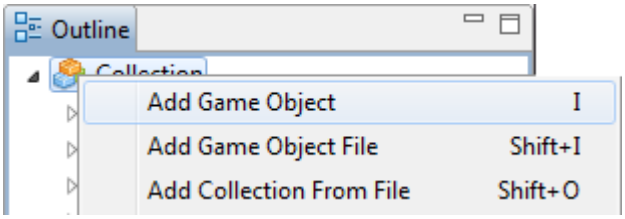
KUVIO 17. Atlasiin kuvien lisääminen.

Seuraavaksi luodaan kentälle collection-tiedosto, johon kootaan kenttä kokonaisuudessaan. Collection-tiedosto luodaan samalla tavalla kuin atlas eli Project Explorerissa hiiren oikean näppäimen klikkauksella level-kansion kohdalla *New* > *Collection File.* Nimetään tämä kokoelma nimellä level.collection (KUVIO 18).



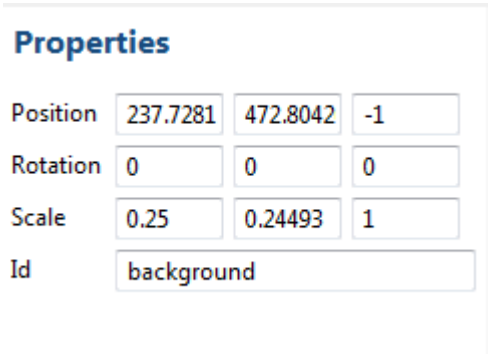
KUVIO 18. Collection-tiedoston luonti-ikkuna.

Seuraavaksi lisätään tähän collection-tiedostoon aiemmin tehty background.atlas. Tämä tapahtuu luomalla peliobjekti kokoelman hierarkiaan, ja lisäämällä siihen Sprite-komponentti. Eli kun level.collection on auki, Outlinessa klikkaa oikean hiirenpainikkeella *Collection* > *Add Game Object* (KUVIO 19).



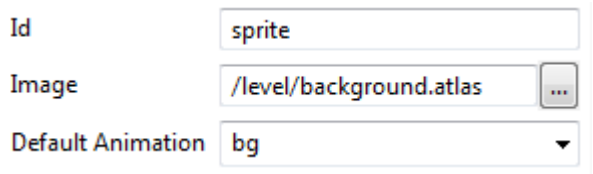
KUVIO 98. Peliobjektin lisäys Collection-tiedostoon.

Nimetään peliobjekti Properties-ikkunassa Id-kohdassa background. Position, Rotation ja Scale arvoja voi muuttaa oman halun mukaan. Tässä tapauksessa taustan sijaintia ei muuten muutettu, mutta kolmas arvo on laitettu arvoksi –1, jotta se olisi varmasti kaiken kentällä esiintyvien objektien takana (KUVIO 20).



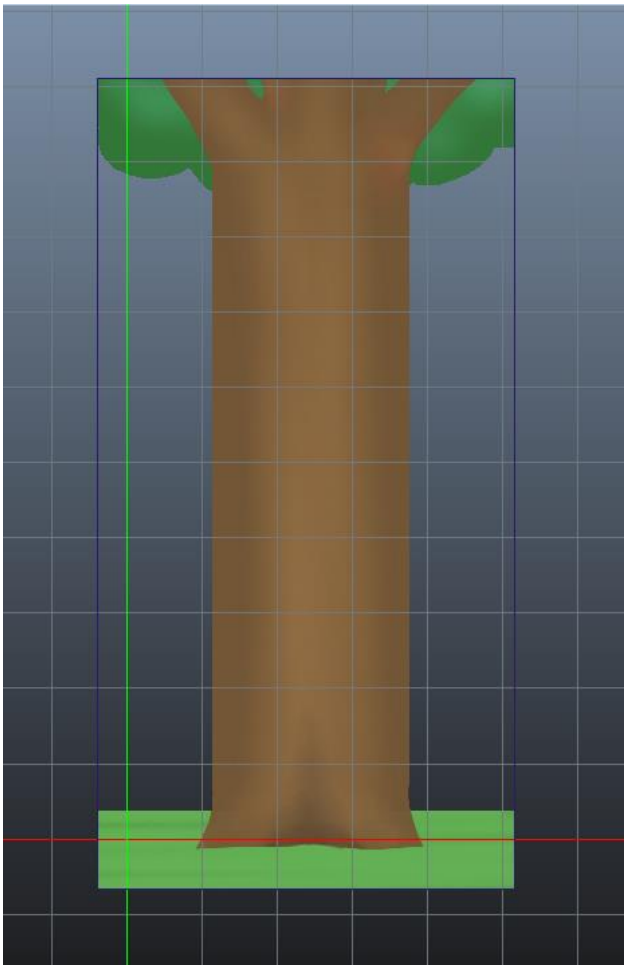
KUVIO 20. Background-peliobjektin arvot.

Tämän jälkeen lisätään tähän peliobjektiin Sprite-komponentti. Tämä tehdään samalla tavalla kuin muidenkin komponenttien lisäily, eli oikealla hiirennäppäimellä klikataan background-peliobjektia > *Add Component* > *Sprite*. Valitaan tämä juuri tehty komponentti ja laitetaan Properties-ikkunassa Image-valikosta background.atlas (KUVIO 21).



KUVIO 21. Background.atlas-tiedoston lisäys Sprite-komponenttiin.

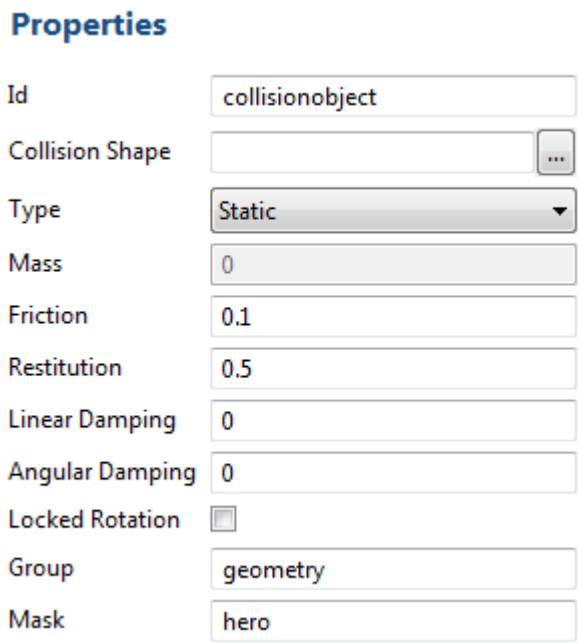
Kun tämä on tehty, päämuokkausnäkymä level.collectionin ollessa auki näyttää kuvion 22 kaltaiselta.



KUVIO 22. Level.collection, tausta lisätty.

Seuraavaksi luodaan toinen collection-tiedoston, joka toimii pelin tasanteena. Tämä tehdään kuvion 18 mukaisesti, mutta nimetään ground.collection. Tämä tiedosto eroaa level.collectionista sillä tavalla, että siihen ei lisätä mitään grafiikoita, mutta luodaan collisionobjektit. Ensimmäiseksi luodaan tälle kokoelmatiedostolle peliobjekti, joka nimetään ground0. Tämän objektin alle luodaan collisionobject, mikä tapahtuu samalla tavalla kuin Sprite-komponentin luonti. Tämän törmäyttimen arvoja muokataan

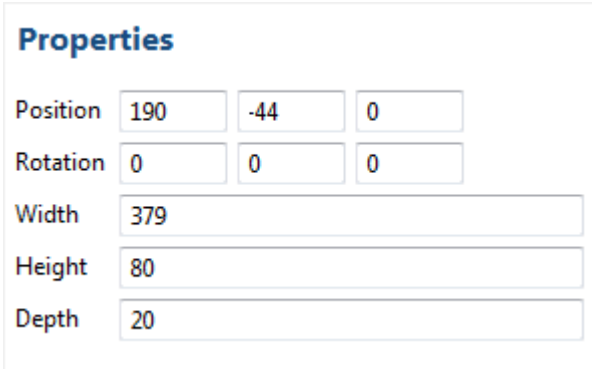
Properties-ikkunassa kuvion 23 mukaisesti, eli Type-arvo *static*, Group-arvo *geometry* ja Mask *hero*. Group ja Mask –arvot tulevat käyttöön myöhemmin pelihahmon skriptin kanssa.



KUVIO 23. Ground0-peliobjektin collisionobject-arvot.

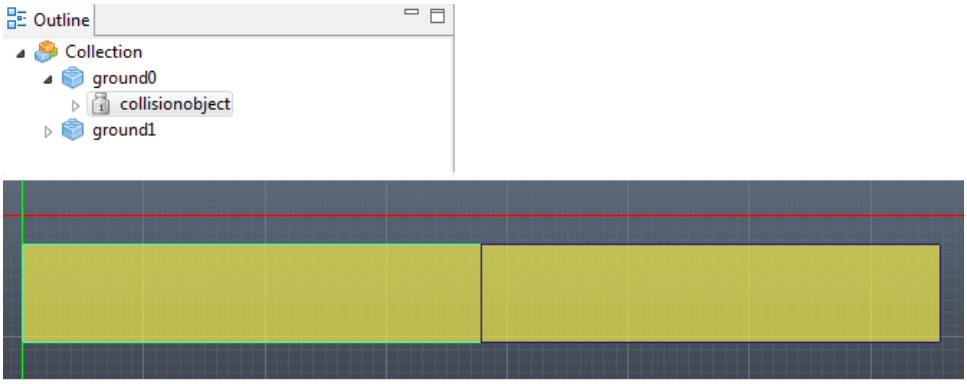
Seuraavaksi luodaan collisionobjectin alle Shape-komponentti, jolla määritämme tämän törmäyttimen muodon. Klikataan hiiren oikealla näppäimellä *collisionobject* > *Add Shape.* Valitaan näistä vaihtoehdoista Box Shape, joka on suorakulmion muotoinen. Muista vaihtoehdoista Sphere on ympyrän ja Capsule kapselin muotoinen.

Kun suorakulmio on luotu, on aika muokata sen pituutta ja leveyttä. Objektin sijaintia on myös siirretty niin, että collisionobject alkaa ground0-tiedoston kulmasta. Näitä arvoja voi muokata joko Properties-ikkunan kautta, tai käyttämällä pikanäppäinkomentoja päämuokkainnäkymässä. W-näppäin antaa mahdollisuuden muokata valitun objektin sijaintia, E-näppäin mahdollistaa valitun objektin kääntämisen, ja R-näppäin on käytettävissä objektin muodon muokkaamiseen pituus- ja leveyssuunnassa. Kuviossa 24 näkyy tähän projektiin luotujen muotojen arvot.



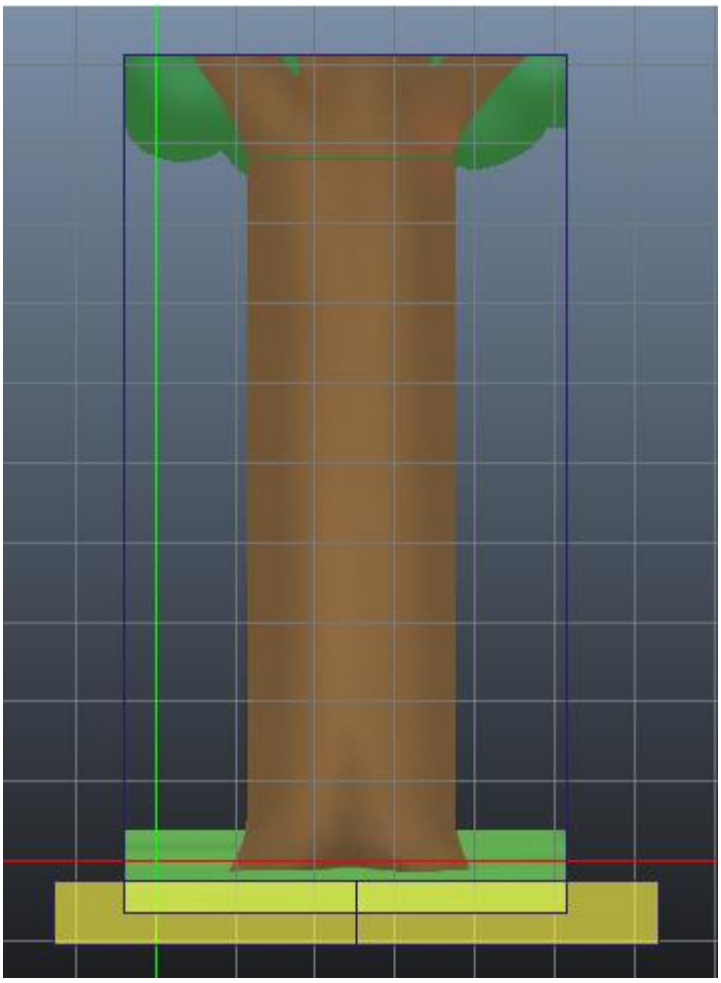
KUVIO 24. Collisionobjectin Box Shape-komponentin arvot.

Kun tämä on tehty, tehdään ground0-peliobjektista kopio, joka nimeytyy automaattisesti ground1:ksi. Ground.collection-tiedosto näyttää Outlinessa ja päämuokkausnäkymässä lopulta kuvion 25 mukaisesti.



KUVIO 25. Ground.collection Outline-näkymä ja päämuokkausnäkymä.

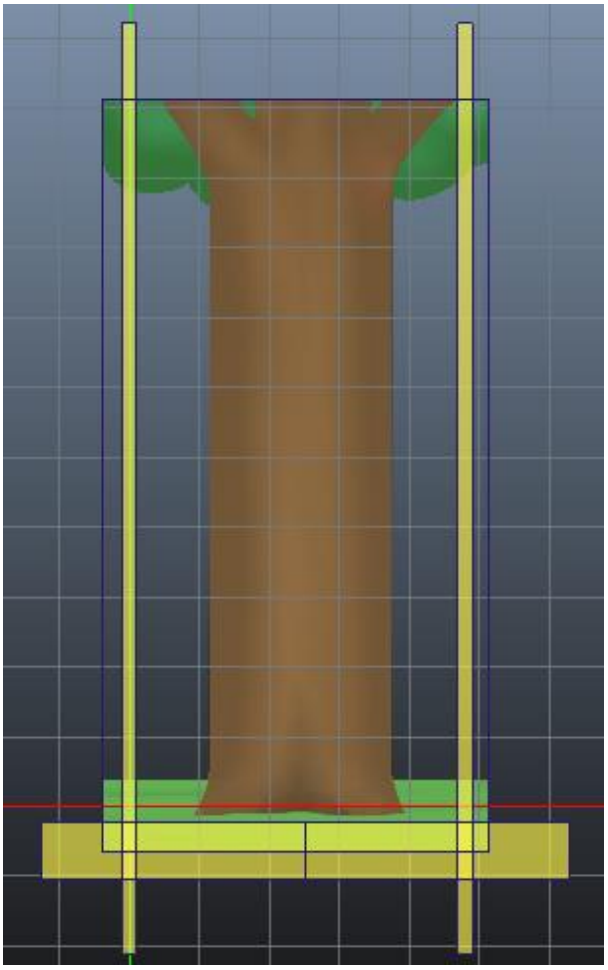
Seuraavaksi lisätään tämä ground.collection-tiedosto level.collection-tiedostoon. Jälleen kerran, klikataan hiiren oikealla näppäimellä *Collection* > *Add Collection From File.* Level.collection näyttää päämuokkausnäkymässä kuvion 26 kaltaiselta tämän vaiheen jälkeen.



KUVIO 26. Level.Collection, tausta ja ground.collection lisättynä.

Kenttä ei ole kuitenkaan vielä ihan valmis. Tämän hetkinen kenttä ei sisällä pelihahmon liikkumista estäviä rajoja, jolloin pelihahmo pystyy hyppäämään peliruudun ulkopuolelle. Yksinkertaisin ja helpoin tapa rajata tämä on luoda vielä kaksi collisionobject-komponenttia. Nämä luodaan tällä kertaa level.collection-tiedoston alle, eli ei tehdä uutta tiedostoa kuten tasanteen kanssa.

Aloitetaan luomalla peliobjekti level.collectionin hierarkiaan Outline-ikkunassa. Annetaan tiedoston nimeksi Properties-ikkunassa *leftwall*. Lisätään collisionobject, Box Shape ja muokataan sen muotoa ja sijaintia niin, että se luo kentälle selvät rajat. Kopioidaan leftwall-objekti, ja siirretään sen kopio toiselle puolelle kenttää. Uudelleen nimetään tämä objekti *rightwall*. Tämän jälkeen level.collection on valmis, ja näyttää suunnilleen kuvion 27 tapaiselta.



KUVIO 27. Level.collection, valmiiksi tehtynä.

Nyt voidaan lisätä tämä valmis kokoelma pääkokoelmatiedostoon eli main.collection auki, ja Outline-ikkunassa *Collection* > *Add Collection from File.* Seuraava vaihe on pelihahmon tekeminen.

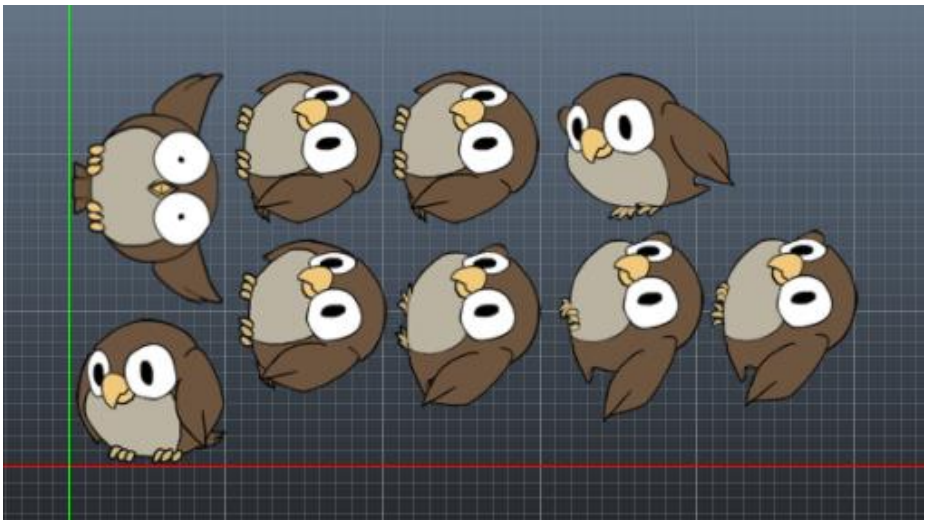
4.6 Pelihahmon luonti

Pelihahmon luomiseen tarvitsee Defoldissa ottaa huomioon kolme eri osaa: spritesheet, skripti ja colliderit. Nämä kaikki osat laitetaan kentän luonnissa selitetyn Game Objectin sisään. Ensimmäinen vaihe hahmon luonnissa on spritesheetin luominen atlasiin.

4.6.1 Kuvasarja

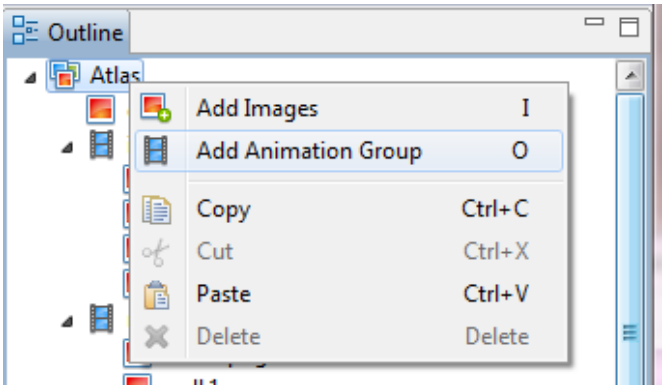
Spritesheetillä tarkoitetaan kuvasarjaa, jossa näkyy pelihahmon eri toimintojen animaatiot kuva kuvalta. Defoldissa kuvasarjan rakentaminen tapahtuu atlas-tiedostossa, joka luodaan tavalliseen tapaan klikkaamalla owl-kansiota oikealla hiiren näppäimellä, ja valitsemalla *New* > *Atlas File*. Tässä tapauksessa atlas-tiedosto on nimetty owl.atlas.

Kun atlas-tiedosto on luotu, voidaan sinne lisätä pelihahmon kuvasarja. Tässä tapauksessa pöllön animaatio kootaan useammasta eri kuvasta. Kuvat lisätään Outlinen kautta, kuten kentän luomiskappaleessa näytettiin. Atlasiin lisätyt kuvat näkyvät päämuokkausnäkymässä kuvion 28 tapaisesti, mahdollisimman tiiviissä paketissa.



KUVIO 28. Atlas-tiedostoon lisätyt kuvat ovat mahdollisimman tiiviisti pakattu.

Koska pöllölle kuitenkin tulee animaatioita, lisätään atlasiin vielä jokaiselle animaatiolle oma anim group. Nämä animaatioryhmät nimetään selkeästi, sillä niitä käytetään pelihahmon skriptissä myöhemmin. Anim group luodaan Outlinessa, kun oikealla hiiren painikkeella klikkaa atlasia (KUVIO 29).



KUVIO 29. Animaatioryhmän lisäys atlasiin.

Kun animaatioryhmä on luotu, atlasissa olevat kuvat voi raahata oman ryhmän alle. Tässä projektissa on animaatioryhmät nimetty seuraavasti: idle, run, fall ja jump eli suomennettuina toimeton, juoksu, tippuminen ja hyppäys. Tämän projektin tippumisen ja hyppäämisen animaatiot eivät sisällä juoksuanimaatiosta poikkeavaa kuvasarjaa, mutta ryhmät pidetään mukana myöhemmin selitettävän animaationvaihdoskoodinpätkän vuoksi.

4.6.2 Koodaus

Seuraavaksi luodaan pelihahmolle skripti, joka nimetään yksinkertaisesti player.script. **Skripti** on tässä tapauksessa Lua-koodikieltä sisältävä tiedosto, jolla pyritään luomaan hahmon liikkuvuuden rajat, komponenttien liitokset ja collider-säännöt. Sillä voi luoda myös pelin muiden objektien fysiikanlakeja ja muita sääntöjä. Tämän projektin pelihahmoskripti on kirjoitettu Defoldin sivuilta löytyvän tutoriaalin mukaisesti, ja kyseinen skripti käydään läpi osioittain.

Player.script alkaa pelihahmon fysiikanlakien numeerisella määrittämisellä (KUVIO 30). Tässä tapauksessa on määritetty, millä tahdilla pelihahmon nopeus kiihtyy vasemmalle tai oikealle mennessä, miten vauhti kiihtyy ilmassa ollessa ja mikä on maksiminopeus. Lisäksi on määritetty painovoiman arvo, jolla määritetään pelihahmon ilmasta takaisin tasanteelle tipahtamisen nopeus.

```
-- the acceleration to move right/left
local move_acceleration = 3500
-- acceleration factor to use when air-borne
local air_acceleration_factor = 0.8
-- max speed right/left
local max_speed = 400
-- gravity pulling the player down in pixel units
local gravity = -1600
-- take-off speed when jumping in pixel units
local jump_takeoff_speed = 800
-- time within a double tap must occur to be considered a jump
-- (only used for mouse/touch controls)
local touch_jump_timeout = 0.2
```

KUVIO 30. Player.script, hahmon fysiikkalakien määrittelyosio.

Seuraava osio skriptissä (KUVIO 31) on pelkästään skriptin suorituskyvyn tehostamista varten. Määritellään etukäteen skriptissä käytettävien muuttujien arvot, jotka tässä tapauksessa ovat kiinnikkeitä erinäisiin osiin pelimoottorissa. Esim. Local input_left = hash("left") viittaa game.input_binding tiedosta esiintyvään input-luokkaan left, ja local anim_run = hash("run") viittaa tässä kappaleessa aiemmin luotuun animaatioryhmän nimeen.

```
-- pre-hashing ids improves performance
local msg_contact_point_response = hash("contact_point_response")
local msg_animation_done = hash("animation_done")
local group_geometry = hash("geometry")
local input_left = hash("left")
local input_right = hash("right")
local input_jump = hash("jump")
local input_touch = hash("touch")
local anim_run = hash("run")
local anim_idle = hash("idle")
local anim_jump = hash("jump")
local anim_fall = hash("fall")
```

KUVIO 31. Player.script, käytettävien ominaisuuksien ennakkomäärittelyosio.

Function init (self) -komentokoodinpätkä määrittää pelihahmon fysiikkan lait (KUVIO 32). Ensimmäinen koodirivi antaa käsitellä komentoja skriptissä. Sitten määritetään pelaajan vauhti. Seuraava rivi tarkistaa hahmon tasanteella olemisen. Loput koodeista määrittävät liikkumiskomennon, pelin käynnistyessä hahmon suuntauksen, käynnissä olevan animaation ja hiirellä tai kosketusnäytöllä hypyn ajastuksen.

```
function init(self)
    -- this lets us handle input in this script
    msg.post(".", "acquire_input_focus")

    -- initial player velocity
    self.velocity = vmath.vector3(0, 0, 0)
    -- support variable to keep track of collisions and separation
    self.correction = vmath.vector3()
    -- if the player stands on ground or not
    self.ground_contact = false
    -- movement input in the range [-1,1]
    self.move_input = 0
    -- character direction, for anim selection (-1 or 1)
    self.direction = 1
    -- the currently playing animation
    self.anim = nil
    -- timer that controls the jump-window when using mouse/touch
    self.touch_jump_timer = 0
end
```

KUVIO 32. Player.script, function init(self).

Kuviossa 33 esiintyvä koodinpätkä määrittää animaatiolle säännön. Tämän mukaan peli ei saa pyörittää animaatiota, joka on jo sillä hetkellä käytössä. Sitten koodissa kerrotaan hahmon spriten pyörittävän animaation ja muistavan mikä animaatio pyörii sillä hetkellä.

```
local function play_animation(self, anim)
    -- only play animations which are not already playing
    if self.anim ~= anim then
        -- tell the sprite to play the animation
        msg.post("#sprite", "play_animation", {id = anim})
        -- remember which animation is playing
        self.anim = anim
    end
end
```

KUVIO 33. Player.script, animaation pyörimismäärittelyfunktio.

Kuvion 34 koodinpätkä jatkaa animaation määrittämistä, mutta tällä kertaa laitetaan animaation vaihtumiselle säännöt sen mukaan, mihin suuntaan hahmo kulkee. Eli, jos pelaaja ei liiku x-akselilla ollenkaan, pysyy animaatio idle-animaatiossa. Muutoin hahmon animaatio vaihtuu juoksuksi. Jos taas hahmon korkeus on nouseva, animaatio vaihtuisi hyppäämisen kuvasarjaksi. Muuten animaatio muuttuu tippumisen kuvasarjaksi.

```
local function update_animations(self)
    -- make sure the player character faces the right way
    sprite.set_hflip("#sprite", self.direction < 0)
    -- make sure the right animation is playing
    if self.ground_contact then
        if self.velocity.x == 0 then
            play_animation(self, anim_idle)
        else
            play_animation(self, anim_run)
        end
    else
        if self.velocity.y > 0 then
            play_animation(self, anim_jump)
        else
            play_animation(self, anim_fall)
        end
    end
end
```

KUVIO 34. Player.script, animaation muuttumissääntöjen määrittäminen.

Kuvion 35 komentokoodi on jokaisessa ruudunpäivityksessä tapahtuvaa toimintaa ja se määrittää pelihahmon nopeuden muuttumisen lait. Ensin määritellään napin painalluksesta määräytyvä nopeus, jolle annetaan ehtoja nopeuden muuttumiselle. Nopeus laitetaan muuttumaan riippuen siitä, vaihtaa-ko hahmo suuntaa, vai onko hän ilmassa. Sitten skripti laskee nopeuden kyseisellä ruudulla, ja tallen-taa sen tiedon. Tämän jälkeen tallennettua nopeutta käytetään hyväksi uuden nopeuden laskentaan, joka lopuksi päivitetään hahmon nopeudeksi.

```
function update(self, dt)
    -- determine the target speed based on input
    local target_speed = self.move_input * max_speed
    -- calculate the difference between our current speed and the target speed
    local speed_diff = target_speed - self.velocity.x
    -- the complete acceleration to integrate over this frame
    local acceleration = vmath.vector3(0, gravity, 0)
    if speed_diff ~= 0 then
        -- set the acceleration to work in the direction of the difference
        if speed_diff < 0 then
            acceleration.x = -move_acceleration
        else
            acceleration.x = move_acceleration
        end
        -- decrease the acceleration when air-borne to give a slower feel
        if not self.ground_contact then
            acceleration.x = air_acceleration_factor * acceleration.x
        end
    end
    -- calculate the velocity change this frame (dv is short for delta-velocity)
    local dv = acceleration * dt
    -- check if dv exceeds the intended speed difference, clamp it in that case
    if math.abs(dv.x) > math.abs(speed_diff) then
        dv.x = speed_diff
    end
    -- save the current velocity for later use
    -- (self.velocity, which right now is the velocity used the previous frame)
    local v0 = self.velocity
    -- calculate the new velocity by adding the velocity change
    self.velocity = self.velocity + dv
    -- calculate the translation this frame by integrating the velocity
    local dp = (v0 + self.velocity) * dt * 0.5
    -- apply it to the player character
    go.set_position(go.get_position() + dp)

    -- update the jump timer
    if self.touch_jump_timer > 0 then
        self.touch_jump_timer = self.touch_jump_timer - dt
    end

    update_animations(self)

    -- reset volatile state
    self.correction = vmath.vector3()
    self.move_input = 0
    self.ground_contact = false

end
```

KUVIO 35. Player.script, hahmon nopeuden muuttumislain määritys.

Kuvio 36 sisältää skriptinpätkän, jossa määritellään tarkasti, miten pelin fysiikka toimii, kun pelihahmon ja tasanteen collider eli törmäytin kohtaavat toisensa.

```
local function handle_geometry_contact(self, normal, distance)
    -- project the correction vector onto the contact normal
    -- (the correction vector is the 0-vector for the first contact point)
    local proj = vmath.dot(self.correction, normal)
    -- calculate the compensation we need to make for this contact point
    local comp = (distance - proj) * normal
    -- add it to the correction vector
    self.correction = self.correction + comp
    -- apply the compensation to the player character
    go.set_position(go.get_position() + comp)
    -- check if the normal points enough up to consider the player standing on the ground
    -- (0.7 is roughly equal to 45 degrees deviation from pure vertical direction)
    if normal.y > 0.7 then
        self.ground_contact = true
    end
    -- project the velocity onto the normal
    proj = vmath.dot(self.velocity, normal)
    -- if the projection is negative, it means that some of the velocity points towards the contact point
    if proj < 0 then
        -- remove that component in that case
        self.velocity = self.velocity - proj * normal
    end
end
```

KUVIO 36. Player.script, hahmon fysiikan lakeja.

Seuraava skriptinosa (KUVIO 37) laittaa edellä tehdyn skriptinpätkän toimintaan, kun törmäyttimet kohtaavat toisensa. Jos pelihahmon kohtaa törmäyttimen, jonka maskinimi on geometry, se toteuttaa aiemmin selitetyn funktion.

```
function on_message(self, message_id, message, sender)
    -- check if we received a contact point message
    if message_id == msg_contact_point_response then
        -- check that the object is something we consider geometry
        if message.group == group_geometry then
            handle_geometry_contact(self, message.normal, message.distance)
        end
    end
end
```

KUVIO 37. Player.script, pelihahmon ja tasanteen kohtaamisen määrittely

Seuraava osa skriptistä (KUVIO 38) määrittää hahmon hyppäämisen rajat. Jos hahmolle haluaa kaksoishypyn, function jump(self) voidaan jatkaa elseif-lauseella. Tässä projektissa ei kuitenkaan kaksoishypyllle koettu tarvetta. Abort_jump funktio määrittää sen, missä vaiheessa hyppääminen pysäytetään.

```
local function jump(self)
    -- only allow jump from ground
    -- (extend this with a counter to do things like double-jumps)
    if self.ground_contact then
        -- set take-off speed
        self.velocity.y = jump_takeoff_speed
        -- play animation
        play_animation(self, anim_jump)
    end
end

local function abort_jump(self)
    -- cut the jump short if we are still going up
    if self.velocity.y > 0 then
        -- scale down the upwards speed
        self.velocity.y = self.velocity.y * 0.5
    end
end
```

KUVIO 38. Player.script, hypyn fysiikan määritys.

Kuvion 39 skriptiosa määrittää, mitä tapahtuu, kun painetaan tiettyjä komentoja. Ensimmäiseksi määritellään mitä tapahtuu pelissä, kun painaa vasemmalle, oikealle tai hyppäämisen komentoa, jotka on määritetty game.input_binding-tiedostossa. Sitten määritetään kosketusnäytöillä, tai tässä demotapauksessa hiiren näpäytyksillä tapahtuvat komennot. Hahmo liikkuu hiiren kursorin mukaisesti, kun hiiren vasenta näppäintä pidetään pohjassa. Hyppääminen toteutetaan kaksoisnäpäytyksellä.

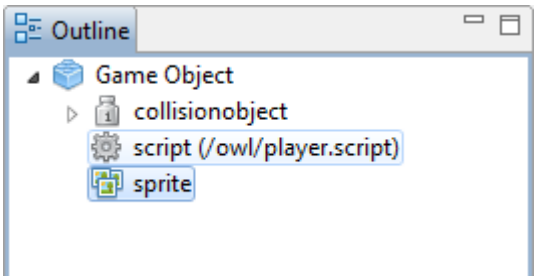
```
function on_input(self, action_id, action)
    if action_id == input_left then
        self.move_input = -action.value
        self.direction = -1
    elseif action_id == input_right then
        self.move_input = action.value
        self.direction = 1
    elseif action_id == input_jump then
        if action.pressed then
            jump(self)
        elseif action.released then
            abort_jump(self)
        end
    elseif action_id == input_touch then
        -- move towards the touch-point
        local diff = action.x - go.get_position().x
        -- update direction
        if diff > 0 then
            self.direction = 1
        else
            self.direction = -1
        end
        -- only give input when far away (more than 10 pixels)
        if math.abs(diff) > 10 then
            -- slow down when less than 100 pixels away
            self.move_input = diff / 100
            -- clamp input to [-1,1]
            self.move_input = math.min(1, math.max(-1, self.move_input))
        end
        if action.released then
            -- start timing the last release to see if we are about to jump
            self.touch_jump_timer = touch_jump_timeout
        elseif action.pressed then
            -- jump on double tap
            if self.touch_jump_timer > 0 then
                jump(self)
            end
        end
    end
end
```

KUVIO 39. Player.script, näppäinkomentojen määritys

Ja näin pelihahmon skripti on valmiiksi koodattu. Seuraavaksi ruvetaan kokoamaan pelihahmon osia yhteen pakettiin.

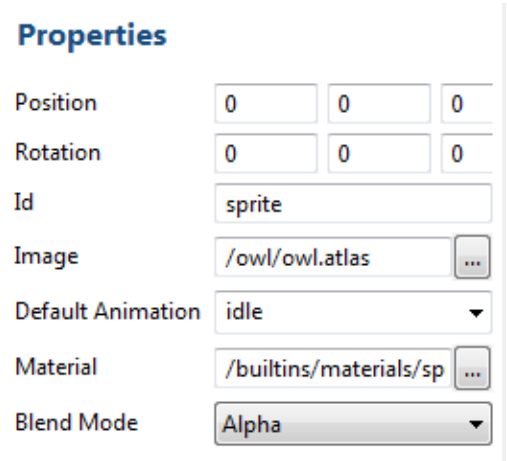
4.6.3 Hahmon osien koonti

Seuraavaksi luodaan itse pelihahmo-objekti ja lisätään juuri tehty owl.atlas sekä player.script siihen. Tämä tapahtuu samalla tavalla kuin kentän luontikappaleessa. Peliobjektin nimeksi laitetaan tässä projektissa hero.go. Tämän peliobjektin alle lisätään aiemmin tehty skripti, collisionobject ja sprite. Collisionobject ja sprite lisätään yksitellen klikkaamalla oikealla hiiren näppäimellä Game Objektista valitsemalla A*dd component.* Skripti lisätään puolestaan *Add component from file* –kohdasta. Peliobjektin rakenne näyttää Outlinessa kuvion 40 mukaisesti.



KUVIO 40. Hero.go-peliobjektin rakenne

Kuten kentän luonti-kappaleessa, myös pelihahmon sprite on se, mihin atlas liitetään. Valitaan sprite ja properties-ikkunassa (KUVIO 41) vaihdetaan sen kuvaksi owl.atlas. Default animation on tässä tapauksessa idle.

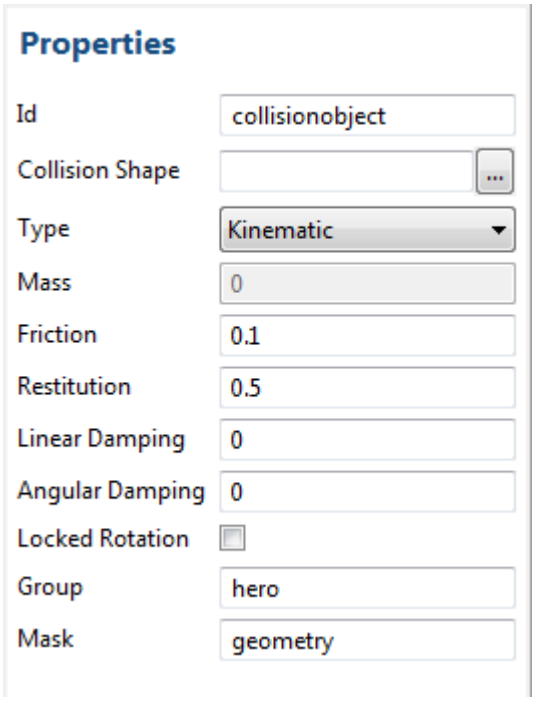


KUVIO 41. Pelihahmon spriten tiedot.

Collisionobjectin alle luodaan neliön muotoinen törmäytin (KUVIO 42) entiseen tapaan klikkaamalla oikealla hiirennäppäimellä *collisionobject* > *add shape.* Tämän muodon koon voi muuttaa samalla tavalla kuin tasanteen luomisessa.

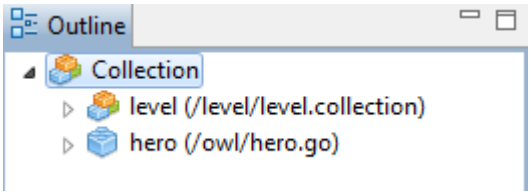
KUVIO 42. Pöllön colllisionobjectin muoto.

Seuraavaksi määritetään tämän collisionobjectin tiedot. Tyypiksi laitetaan kinematic, ryhmäksi hero ja maskiksi laitetaan tasanteen collisionobjectin ryhmän nimi eli geometry (KUVIO 43).



KUVIO 43. Pelihahmon collisionobjectin tiedot.

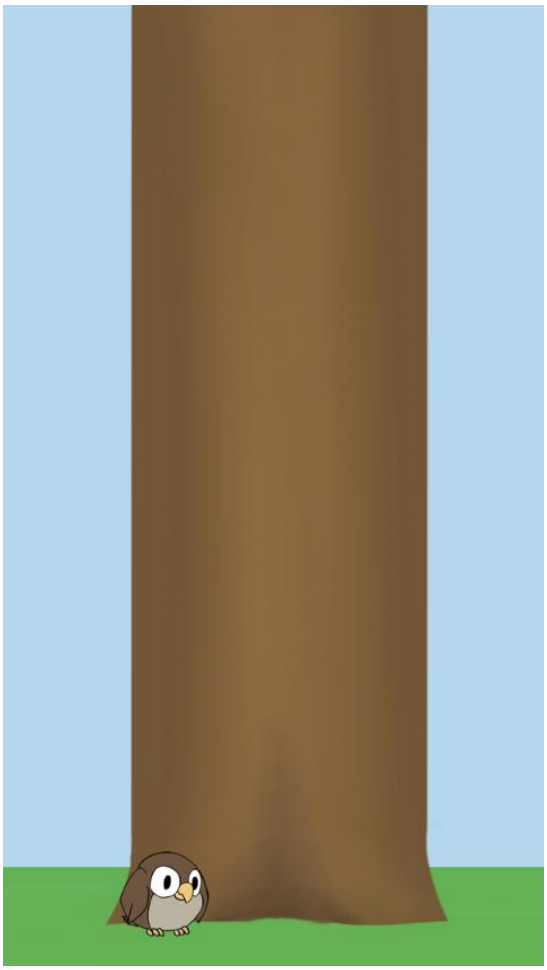
Tämän vaiheen jälkeen pelihahmo on valmis laitettavaksi pääpelinäkymään. Avataan main.collection, ja Outline-ikkunassa lisätään collectioniin oikealla hiirennäppäimellä *Add Game Object File* > *hero.go.* Tämän jälkeen main.collectionin sisältö näyttää kuvion 44 mukaiselta.



KUVIO 44. Main.collection hero.go objektin lisääminen jälkeen.

5 TULOKSET

Peliprojekti on nyt valmis opinnäytetyötä varten. Siihen sisältyi kentän ja pelihahmon kokoamiset. Pelidemo jäi ajan puutteen myötä hyvin yksinkertaiseksi, mutta on sopiva alku laajentamiselle. Pelihahmolla, joka on pöllönpoikanen, pystyy liikkumaan ja hyppimään alatasanteella. Hahmolla pystyy liikkumaan ainoastaan ruudun rajoissa. Pelidemo ei sisällä GUI-elementtejä eikä myöskään ääniä laitettu toimintaan (KUVIO 45).



KUVIO 45. Valmis pelidemotuotos.

Työskentely Defoldin kanssa oli haastavaa. Alussa läpi käytyihin pelimoottoreihin verrattuna Defold on hyvin uusi pelimoottori markkinoilla, eli materiaalia ei löytynyt paljoa. Myös käyttöliittymä on muihin verrattuna kankeampi, koska Defoldissa ei pysty tekemään kenttiä ja muita osia raahaus ja pudotus -

tekniikalla. Se ei myöskään omista minkäänlaista lisäosakauppaa, joten kaikki koodit ja mekaniikkaosat pitää tehdä itse, tai etsiä ladattavana. Näiden puutteiden takia isompien pelien teko tuntuu hankalalta Defoldilla. Pelimoottori tuntuu sointuvan kuitenkin hyvin yksinkertaisten mobiilipelien tekoon. Ja koska se on täysin ilmainen, ei ole tarvetta miettiä eri versioiden eroavaisuuksia.

Jatkoa varten tätä pelidemoa pystyisi kehittämään jopa mobiilikauppoihin asti. Seuraavia vaiheita olisi satunnaisesti ilmestyvien tasopalikoiden ilmestyminen ruudun reunoille. Tämä elementti loisi jonkinmoista uudelleenpelattavuutta, koska taso ei olisi aina täsmälleen samanlainen. Lisäksi tehtäisiin ääniefektejä, taustamusiikkia ja toteutettaisiin pelikonseptin alkuperäinen idea: loputtomasti jatkuva ja ajastettu tasohyppely. Lisäksi pelin graafista puolta päivitettäisiin miellyttävämpään muotoon.

Defoldin dokumentaatio oli omasta mielestäni vähän häkellyttävää luettavaa. Varsinkin hahmon koodia kirjoittaessa tuli vastaan ei-toimivia skriptinosia itse manuaalissa, jolloin piti alkaa etsiä valmiista tutoriaalipaketeista lisätietoa. Hakukone kuitenkin helpotti tiettyjen termien etsimistä tästä massiivisesta nettimanuaalista.

Loppujen lopuksi koen Defoldin aika kivana pelimoottorina, ja kunhan sitä kehitetään vielä enemmän käyttäjäystävällisemmäksi käyttöliittymältään ja mahdollisesti ladattavien lisäosien kautta, uskon että siitä on kilpailemaan muiden pelimoottorien rinnalla. Tällä hetkellä se kuitenkin vajoaa sinne hankalalle polulle. Pelimoottoria kuitenkin päivitellään tiuhaan tahtiin, joten uskon sen kehittyvän hyvää vauhtia.

6 POHDINTA

Tämän opinnäytetyön tavoite oli yksinkertainen: toteuttaa teknillinen pelidemo tutustuen samalla Defold-pelimoottoriin. Tarkoituksena oli saada aikaiseksi pelattava kenttä, josta näkisi mihin pelimoottori pystyy. Lisäksi tutkittiin muita tunnettuja 2D-peleille tarkoitettuja pelimoottoreita. Opinnäytetyöllä ei ollut työnantajaa, ja sitä tehtiin yksin omaan tahtiin.

Kuvittelin tämän aiheen helpoksi ja nopeaksi, mutta erinäisistä syistä opinnäytetyön tekoaika venyi pitkäksi. Alkuperäinen kolmen kuukauden aikataulu muuttui noin yhdeksän kuukauden tahdiksi. Lopputulos on kuitenkin ihan hyväksyttävä, koska sain tehtyä juuri sen, mitä halusinkin: opin käyttämään Defoldi-pelimoottoria perustasolla ja tiedän pelimoottorin perusasiat hyvin. Opin myös miten helppo Lua-koodikieli on oppia aiempien koodikokemusten avulla. Samalla tykästyin tähän koodikieleen hyvin paljon.

Isoin ongelma tämän opinnäytetyön kanssa oli kuitenkin aikataulutus. Opinnäytetyön alussa onnistuin pysymään aikataulussa, mutta pitkällä ajalla aloin lipsua siitä. Loppua kohden sain kuitenkin itselleni aikaiseksi sopivan aikataulun, josta onnistuin pitämään aika hyvin kiinni. Äänet jäivät uupumaan, mikä sinänsä harmittaa itseäni, mutta niistä ei kuitenkaan olisi tullut niin olennainen osa peliä että se haittaisi lopputulosta.

Tulevaisuutta ajatellen, uskon jatkavani tämän projektin parissa, vaikka se olisikin vain harrastuksena. Tätä projektia oli kaikista vastoinkäymisistä huolimatta oikein mukava työstää, enkä näe mitään syytä, joka estäisi minua jatkamassa tämän kanssa. Uskon myös jatkavani muidenkin pelimoottorien tutkimista tarkemmin.

LÄHTEET

Clickteam 2016. Versions Comparison. Viitattu 27.12.2016, www.clickteam.com/compare-versions.

King 2016. Defold: Free 2D Game Engine for 2D games. Viitattu 25.5.2016, http://www.defold.com/.

King 2016. Defold game development manual – Defold game development tutorial – Getting Started. Viitattu 10.8.2016, http://www.defold.com/tutorials/getting-started/.

King 2017. Defold game development Manual. Building Blocks, Physics ja 2D Graphics. Viitattu 22.2.2017, http://www.defold.com/manuals/introduction/.

King.com 2016. Viitattu 15.6.2016, http://company.king.com/.

Made With Unity 2017. Viitattu 30.1.2017, https://madewith.unity.com/en/games.

PCGamer 31.7.2014. No coding required: How new designers are using GameMaker to create indie smash hits. Viitattu 7.12.2016, http://www.pcgamer.com/no-coding-required-how-new-designers-are-using-gamemaker-to-create-indie-smash-hits/.

PUC-Rio 2016. Lua: about. Viitattu 27.5.2016, https://www.lua.org/about.html.

Steam 2016. Clickteam Fusion 2.5 Steamissä. Viitattu 13.12.2016, http://store.steampowered.com/app/248170/?l=finnish.

Unity Store 2017. Welcome to Unity. Viitattu 30.1.2017, https://store.unity.com/.

Wikipedia 2017. Unity. Viitattu 12.1.2017, https://fi.wikipedia.org/wiki/Unity_(pelimoottori).

Wikipedia 2016. GameMaker: Studio. Viitattu 3.12.2016,
https://en.wikipedia.org/wiki/GameMaker:_Studio.

Yoyo Games 2016. Individuals, Indies and Studios. Viitattu 3.12.2016,
https://www.yoyogames.com/get.