SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

SERIAL SNACKER

Diabetesaiheinen mobiilipeli Androidille

TEKIJÄ: Gedi Skog

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala			
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma			
Työn tekijä(t) Gedi Skog			
Työn nimi Serial Snacker			
Päiväys	27.3.2017	Sivumäärä/Liitteet	31/0
Ohjaaja(t) Lehtori Jussi Koistinen, Lehtori Sami Lahti			
Toimeksiantaja/Yhteistyökumppani(t) Ines Skog, yrittäjä			

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda diabetesaiheinen mobiilipeli, jonka avulla diabetesta sairastavat, 10 - 17-vuotiaat nuoret voivat opetella hallitsemaan oman sairautensa hoitoa. Pelin tarkoitus oli opastaa laskemaan ravinnosta saatavia hiilihydraatteja ja pistämään omaan insuliiniherkkyyteen sopiva annos insuliinia. Pelin tuli olla informatiivinen, hauska, mielenkiintoinen ja houkutteleva. Mobiilialustan vuoksi peli haluttiin pitää yksinkertaisena. Lisäksi 2D-grafiikoita pidettiin alustalle sopivina. Informatiivisuuden ja hauskuuden yhdistäminen teki suunnittelusta haastavan.

Peli toteutettiin Unitylla Android-käyttöjärjestelmälle. Unity valittiin, koska se oli entuudestaan tuttu. Grafiikoiden luomiseen käytettiin sähköisiä piirustustyökaluja Paint X Litea ja MS Paintia. Toteutuksen suurimpia kokonaisuuksia olivat pelaajahahmon ja vihollisten luominen sekä tason rakentaminen. Hahmojen luomisen vaiheita olivat piirtäminen, animoiminen ja ohjelmoiminen. Tason rakentamisessa hyödynnettiin automatisointia, mutta viimeistely tehtiin käsin. Pelille annettiin nimeksi Serial Snacker.

Toteutuksen aikana tavoite muuttui. Pelistä haluttiin monimutkaisempi versio tietokoneelle; pelin laajentamisen katsottiin parantavan sitä. Samalla siirryttiin piirretyistä 2D-grafiikoista 3D-grafiikoihin. Unreal Enginellä toteutettiin demo, joka hyödynsi aiemmin luotuja grafiikoita. Tähän vaiheeseen valittiin Unreal, koska sitä pidettiin kiinnostavana vaihtoehtona Unitylle. 3D-mallinnukseen käytettiin Blenderiä.

Lopputuotteena syntyi kaksi demoa: yhden tason käsittävä mobiilipelialku sekä konseptia esittelevä, tietokoneella pyöritettävä pieni taso. Mobiilipeli vastasi alun perin asetettuja tavoitteita muuten kuin laajudeltaan. Syynä suppeuteen oli tavoitteiden muuttuminen. Toinen demo jäi huomattavan suppeaksi teknisten ongelmien vuoksi. Unrealin havaittiin vaativan kehityskoneelta huomattavasti enemmän kuin Unityn. Kehityskoneen suorituskyky ei riittänyt laajempaan toteutukseen. Pelin idea vaikutti lupaavalta, ja täysi versio halutaan toteuttaa.

Avainsanat
Unity, Unreal Engine, diabetes, videopelit, pelinkehitys, mobiilipelit, Android

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Gedi Skog			
Title of Thesis Serial Snacker – Diabetes Mobile Game			
Date	March 27, 2017	Pages/Appendices	31/0
Supervisor(s) Mr. Jussi Koistinen, Lecturer; Mr. Sami Lahti, Lecturer			
Client Organisation /Partners Ms. Ines Skog, Entrepreneur			

Abstract

The original purpose of this thesis was to create a mobile game aimed towards the 10 - 17 year old youngsters freshly diagnosed with diabetes. The game was meant to guide the player to estimate the carbon hydrates consumed during a meal and to decide the amount of insulin to be injected based on their insulin sensitivity. The game needed to be both fun and informative which was found challenging in the designing phase. It also had to be kept simple due to the mobile platform. The game would be made in 2D for it was regarded the best solution for this platform.

The game was made with Unity which the author was already familiar with. Drawing tools Paint X Lite and MS Paint were used to create graphics. The target platform was Android. The game has a player character and enemies which were made by drawing, animating and programming. The level uses an algorithm to build itself automatically. The game was given the name Serial Snacker.

However, the goal was later changed as the client desired a pc game instead of the planned mobile game. The platform was switched because the game needed to be more complex which meant that mobile was no longer a viable option. It also felt natural to switch to 3D graphics. The new game was designed and a small demo was made with Unreal Engine. Unreal was a new experience and it was chosen for this phase because it was thought to be an interesting alternative to Unity. Blender was used to create a 3D model.

As a result of this thesis, two demos were made. The first one was a one-level game for Android and the other was a piece of a level meant to be run on pc. The Android demo met the original goals except for the amount of content designed. It was left short after the idea for a new game was conceived. The pc demo did not reach its goals due to performance issues on the computer used for development. The idea, after all, was found solid and the game is wished to be realized later.

Keywords
Unity, Unreal Engine, Diabetes, Video games, Game design, Mobile games, Android

SISÄLTÖ

TERMIT JA LYHENTEET

2,5D-peli	Peli, jonka grafiikat ovat 2D:n ja 3D:n välimaastossa. Esimerkiksi peli, joka on toteutettu 3D:nä, mutta jossa näkymä ja liike on rajattu 2D:hen.
Tekstuuri	Kuvatiedosto, jolla esitetään esim. puu- tai kivipintaa (Geig, 2013-10-07).
Sprite	2D-bittikartta, joka piirretään kohteeseen ilman lisäkäsittelyä (Microsoft, 2017).
Shader	Ohjelman sisältämät valmiit ohjeet materiaalin kuvantamiseksi (esim. kiilto, heijastus, rosoisuus) (Geig, 2013-10-07).
Materiaali	Resurssi, jolla annetaan esineelle sen pinta. Unityssa tehty lisäämällä tekstuuriin shader. (Geig, 2013-10-07.)
Rigidbody	Unityn komponentti, joka mahdollistaa olion käyttäytymisen fysiikan lakien mukaisesti (Unity documentation a, 2017-03-08).
Trigger	Tässä: Boolean, jonka erityispiirre on palautuminen välittömästi epätodeksi, kun se asetetaan todeksi (Unity documentation b, 2017-03-08).
HUD	Heads-up display, pelaajalle tärkeää tietoa esittävä käyttöliittymän osa (Wilson, 2006-02-03).

1 JOHDANTO

Opinnäytetyön alkuperäinen aihe on mobiilialustainen peli, jota voidaan hyödyntää diabeteksen hoidossa. Työ pitää sisällään pelin aiheen rajauksen, ominaisuuksien suunnittelun ja demon toteutuksen. Lisäksi mukana on eri tekniikalla toteutettu pienempi demo, joka vastaa muuttuneita vaatimuksia sekä suunnitelmia pelin lopullisesta muodosta.

Pelejä pidetään yleensä leluina. Suuri yleisö on alkanut vasta vähitellen ymmärtää pelien potentiaalin niin taidemuotona kuin vaikkapa opetuksessa käytettävänä työkaluna. Pelillisyyden hyödyntämisestä julkaistaan jatkuvasti uusia tutkimuksia. Kuopiossa järjestetään kahdesti vuodessa Health Game Jam, jossa osallistujat luovat 48 tunnissa jollakin tapaa terveyteen liittyvän pelin tai sovelluksen. Tällaiset tapahtumat poikivat lukuisia esimerkkejä pelien hyödyntämisestä niin sairauksien kuin yleisterveyden tapauksessa, mutta markkinoilla terveyspelien osuus on pieni. Ne eivät myöskään saa osakseen erityisen paljon huomiota.

Työn tilaaja sairastaa tyypin 1 diabetesta ja seuraa tarkasti hoidon ja tutkimuksen kehittymistä niin maailmalla kuin Suomessa. Tyypin 2 diabetes yleistyy kaikkialla epäterveellisten elintapojen vuoksi, mutta Suomessa myös tyyppi 1 on yleinen. Sitä sairastaa yhteensä 50 000 suomalaista (Diabeteskeskus, 2016). Tyyppiin 1 sairastutaan tavallisesti nuorena. Tilaajan mielestä erityisesti sairastuneilla lapsilla ja nuorilla olisi käyttöä pelille, joka opettaisi taudin kanssa elämistä hauskalla tavalla. Sairastuminen on kuitenkin iso ja pelottavakin asia.

Tyypin 2 diabeteksen hoidossa tärkeintä ovat terveelliset elämäntavat ja painonhallinta, joiden avuksi on saatavilla runsaasti erilaisia sovelluksia. Siksi opinnäytetyössä otetaan huomioon vain tyyppi 1. Tyypin 1 diabetesta hoidetaan pitkin päivää otettavilla insuliinipistoksilla, joten peli keskittyisi erityisesti tähän puoleen. Pelin on tarkoitus olla yhtä aikaa hauska ja opettavainen, ja se olisi suunnattu 10 - 17-vuotiaille nuorille, joiden diabetes on vasta todettu. Tarkempia vaatimuksia esimerkiksi pelin tarinasta, mekaniikasta tai genrestä ei esitetty, vaan ne keksittiin suunnitteluvaiheessa.

2 DIABETES

Diabetes on ryhmä sairauksia, joissa keho ei pysty hallitsemaan verensokerin tasoa normaalisti. Nämä sairaudet jaotellaan lisäksi tarkemmin eri ryhmiin, joista yleisimmin käytetyt ovat tyypin 1 ja tyypin 2 diabetes. (Diabeteskeskus, 2016.) Tässä opinnäytteessä keskitytään tyyppiin 1, koska tähän tarpeeseen vastavaa peliä tai sovellusta ei ole ainakaan yleisessä levityksessä. Tyypin 2 tärkein ehkäisy- ja hoitomuoto on hyvien elämäntapojen noudattaminen (Diabeteskeskus, 2016). Tähän kannustavia sovelluksia on olemassa jo runsaasti.

2.1 Tyypin 1 diabetes

Tyypin 1 diabetekseen ei nykytietojen mukaan ole olemassa parannuskeinoa, vaan hoidon tarkoitus on pitää oireita kurissa, jotta sairastunut voi elää hyvää elämää. Hoitona käytetään lisäinsuliinia, jota annostellaan joko pistoksina tai pumpulla. Pistoksia otetaan päivässä useita: Pitkävaikutteisen perusinsuliinin lisäksi pistetään lyhytvaikutteista ateriainsuulinia. (Diabeteskeskus, 2016.)

Kerralla pistettävän insuliinin määrä on yksilöllistä. Ateriainsuliinin annosta määritettäessä tärkeimmät tekijät ovat aterialla nautittujen hiilihydraattien määrä sekä henkilön insuliiniherkkyys. Muutkin tekijät kuitenkin vaikuttavat. Esimerkiksi ravinnon laatu vaikuttaa hiilihydraattien imeytyvyyteen ja sitä myötä näiden aiheuttamaan verensokerin nousuun. Lisäksi koholla olevaa verensokeria korjataan aterioiden yhteydessä, mikä tarkoittaa suurempaa ateriainsuliiniannosta. Myös perusinsuliinin tarve vaihtelee yksittäiselläkin henkilöllä päivittäin muun muassa stressin ja liikunnan määrän mukaan. (Diabeteskeskus, 2016.)

Hyvän hoitotasapainon saavuttaminen on elintärkeää, sillä huonosti hoidetulla diabeteksella on vakavat seuraukset (Diabeteskeskus, 2016). Diabetekseen vaikuttavat kuitenkin monet seikat, eikä sen hoito ole lainkaan niin yksinkertaista kuin ensin voisi kuvitella. Tyyppiin 1 sairastuu paljon lapsia, Suomessa suhteessa enemmän kuin missään muualla. Siksi olisi hyvä saada erilaisia, erityisesti lapsille ja nuorille sopivia, sovelluksia, joiden avulla voisi opetella oman sairauden hallintaa. Näiden sovellusten haaste on sairauden monimuotoisuus ja sen vaikutusten ilmeneminen eri tavalla eri yksilöillä.

2.2 Pelin hyödyntäminen diabeteksen hoidossa

Diabetesta siis hoidetaan insuliinipistoksilla, ja pistettävän insuliinin määrä riippuu suurelta osin nautittujen hiilihydraattien määrästä. Siksi on tärkeää seurata hiilihydraattien määrää joka aterialla. Tähän tarkoitukseen on olemassa erilaisia taulukoita, joiden avulla hiilihydraattimäärä voidaan laskea. Prosessi nopeutuu, jos tärkeimpien ruokien sisältämät hiilihydraattimäärät opetellaan ulkoa. Taulukoiden ulkoa opettelu on kuitenkin tylsää, eikä motivaatiota moiseen välttämättä löydy.

Videopelin avulla opetteleminen olisi mielekkäämpää. Peliä voisi pelata vaikka päivittäin, ja hiilihydratit jäisivät vähitellen mieleen. Ateriainsuliinin määrän laskeminen helpottuisi ja hyvän hoitotasapainon saavuttaminen olisi helpompaa.

Kuten aiemmin todettiin, verensokerin tasoon ja päivittäiseen insuliinintarpeeseen vaikuttaa moni muukin seikka. Hiilihydraattien laskeminen on kuitenkin konkreettinen asia, joka toimii samalla tavalla kaikille potilaille. Siksi se on luonteva valinta pelin mekaniikaksi.

3 SERIAL SNACKER

Tässä luvussa esitellään Serial Snacker -pelin suunnittelu ja toteutus käytettyine työkaluineen. Suunnitteluvaihe oli pitkä, koska vaatimukset olivat tulkinnanvaraisia. Toteutusvaihe jakautui kahteen osaan kahden eri demon vuoksi.

3.1 Suunnitteluprosessi

Vaatimusmäärittelyvaiheessa kohderyhmäksi valikoituivat 10 - 17 -vuotiaat nuoret, joiden diabetes on vasta todettu. Lisäksi todettiin, että paras alusta pelille on mobiili, jotta peli tavoittaisi mahdollisimman monta kohderyhmään kuuluvaa. Samalla pelin pitäisi sopia myös laajemmalle yleisölle. Ihannetapauksessa myös muun ikäiset ja henkilöt, jotka eivät sairasta diabetesta, haluaisivat pelata peliä. Pelin tulisi keskittyä hiilihydraattien laskemiseen ja ottaa samalla huomioon käyttäjän insuliiniherkkyys. Pelin tarkoitus olisi opettaa nuorille itsenäisyyttä ja vastuuta oman taudin hoidosta. Näiden lisäksi pelin pitäisi olla informatiivinen, hauska, mielenkiintoinen ja houkutteleva.

Mobiilialustan vuoksi peli oli pidettävä yksinkertaisena; supersuositut Angry Birds, Pokémon GO ja Candy Crash Saga toimivat kaikki vain yhdellä sormella. Pienessä ruudussa käyttöliittymän piti olla minimalistinen ja huomioon tuli ottaa myös puhelinten rajallinen prosessointiteho.

Ikäryhmälle sopiva peli olisi sävyltään kepeä ja iloinen, mutta ei liian lapsellinen. Kirkkaat värit ja selkeä, omintakeinen tyyli uppoavat kaikenikäisiin. Aihe saisi olla helposti lähestyttävä. Diabetesta ei myöskään saisi tunkea peliin liian vahvasti, muuten lopputuloksesta tulisi teennäinen.

Suurin haaste oli yhdistää hauskuus ja opettavaisuus. Monet opetuskäyttöön tarkoitetut pelit tulkitsevat aihettaan hyvin kirjaimellisesti. Paras tapa toteuttaa diabetespeli olisi vahingossa oppiminen: Peliä ei pelattaisi siksi, että halutaan oppia jotakin, vaan koska peli on hauska. Oppiminen tapahtuisi huomaamatta, kun diabetekseen liittyviä elementtejä toistettaisiin riittävästi olematta kuitenkaan liian päällekäyvä.

Pelistä päätettiin tehdä yhdellä sormella pelattava 2D-peli, jossa seikkailisi supersankariviittaan pukeutunut poika. Pelissä olisi useita tasoja, joissa olisi tavoitteena poimia erilaisia ruokia. Lisäksi pelaajan pitäisi taistella vihollisia vastaan. Peli toteutettaisiin entuudestaan tutulla Unitylla (ks. luku 3.2.1). Ensimmäinen versio päätettiin tehdä Androidille.

Mobiilipelidemon valmistuttua todettiin, että peli voisi olla monimutkaisempikin. Myös tilaaja oli tätä mieltä. Tällaista peliä ei kuitenkaan ollut mielekästä toteuttaa puhelimelle. Syntyi päätös tehdä pelistä uusi versio tietokoneelle. Samalla päätettiin kokeilla pelin tekemistä Unreal Enginellä (ks. luku 3.2.2). Tietokoneen suurempi näyttö mahdollistaa monimutkaisemman käyttöliittymän, näppäimistö laajemman skaalan erilaisia ohjausvaihtoehtoja ja tehokkaampi

prosessori sekä suurempi muisti kehittyneemmän pelin. Pelistä tulisikin 2,5D-tasohyppely, jonka grafiikat olisi toteutettu 3D:nä.

3.2 Käytetyt menetelmät

Tässä luvussa esitellään toteutuksessa käytetyt ohjelmat sekä testauslaitteet. Tärkeimmät ohjelmat olivat pelien toteutukseen käytettävät Unity ja Unreal Engine.

3.2.1 Unity

Unity on sekä pelimoottori että pelien tekemiseen käytettävä ohjelma. Se tarjoaa graafisen käyttöliittymän, jossa voi rakentaa tasoja, luoda animaatioita ja hallita projektin resursseja. Ohjelmointi tapahtuu C#:lla. Unity on monen aloittelevan pelinkehittäjän valinta, sillä se on helppokäyttöinen, netistä löytyy runsaasti materiaalia avuksi ja siitä on saatavilla täysin ilmainen versio. Unity oli luonteva valinta opinnäytetyön tekemiseen, koska se oli entuudestaan tuttu.

3.2.2 Unreal Engine

Unreal Engine on Epic Gamesin tarjoama pelimoottori ja pelinkehitystyökalu. Unityn lailla siinäkin on monipuolinen graafinen käyttöliittymä. Ohjelmointikielenä on kuitenkin C++. Ohjelmointi on lisäksi mahdollista jättää kokonaan pois rakentamalla toimintalogiikka vuokaavioiden avulla graafisessa käyttöliittymässä. Unreal oli uusi tuttavuus, mutta se kiehtoi kehittyneempien ominaisuuksiensa vuoksi. Esimerkiksi materiaalien luonti vastaa 3D-mallinnustyökaluissa käytettäviä. Unityn ja Unrealin materiaalinmuokkaimia on verrattu kuvissa 1 ja 2. Oli yllättävää huomata, että Unreal vaatii huomattavasti enemmän tehoa kehityskoneelta.

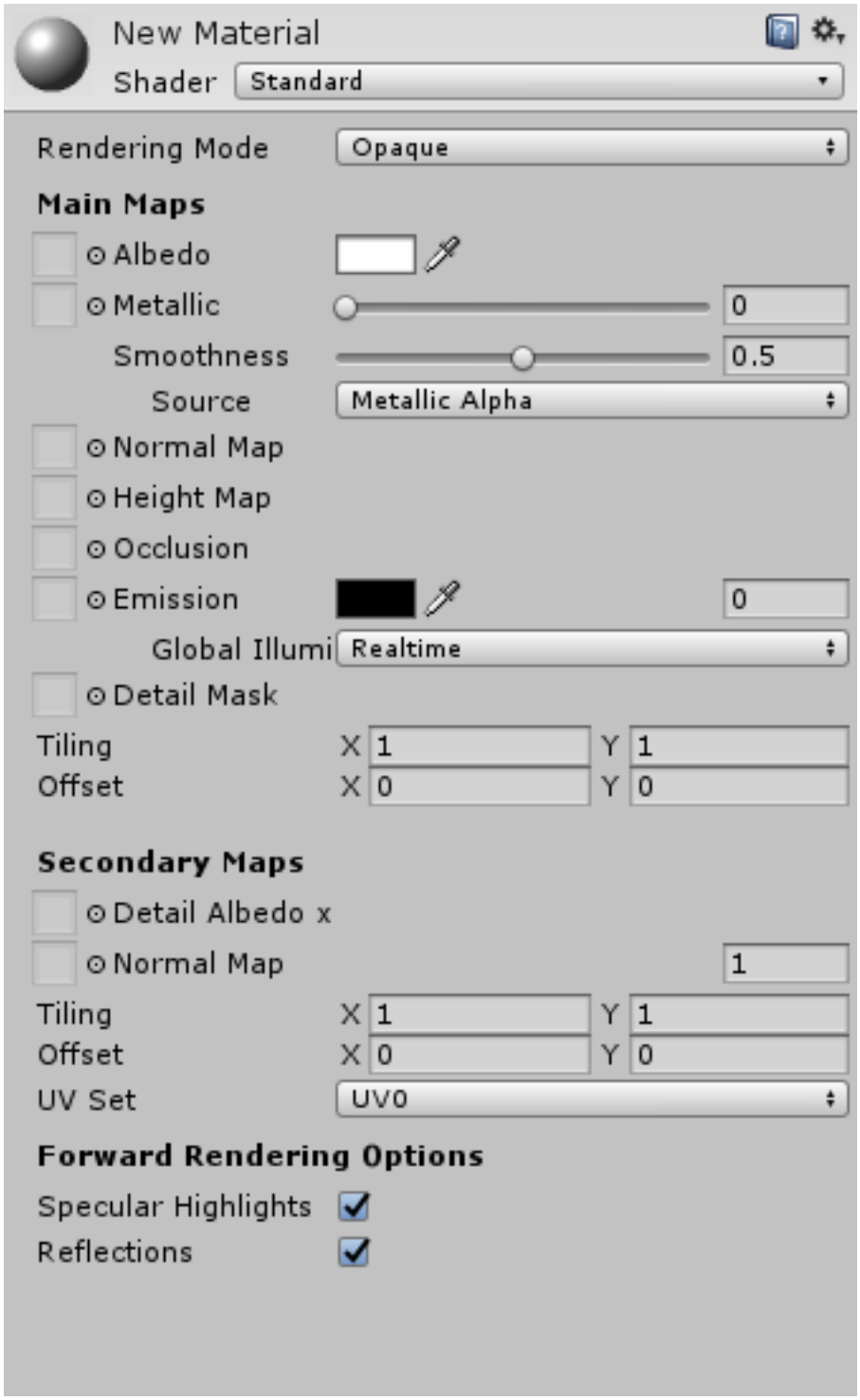
3.2.3 Muut työkalut

Versionhallinta ja varmuuskopioiminen tapahtuivat GitLabilla. Ensimmäistä demoa tehtiin kahdella eri koneella, ja GitLabin avulla tiedostoja sai siirreltyä kätevästi koneelta toiselle. Myös GitLab oli entuudestaan tuntematon.

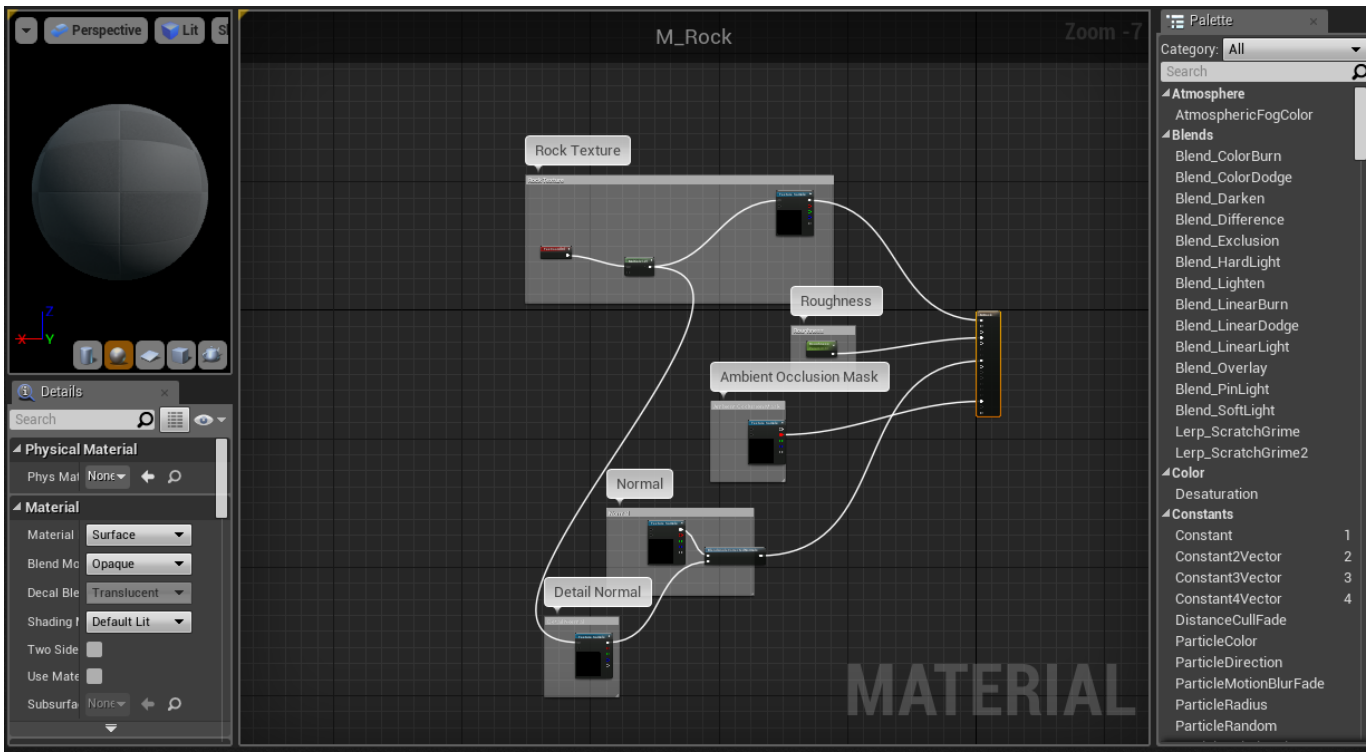
Ensimmäisen demon grafiikat tehtiin piirtämällä PNG-kuvia pääosin Paint X Litella, joka on Macille saatava ilmainen, MS Paintia hienostuneempi piirustusohjelma. Käytössä oli laaja valikoima erilaisia pensseleitä ja kuviin oli mahdollista luoda läpinäkyvä tausta. Käytössä oli lisäksi MS Paint, jolla kuvat lopuksi muokattiin.

Toiseen demoon tehtiin 3D-malli Blenderillä. Blender on ilmainen avoimen lähdekoodin 3D-mallinnusohjelma, jolla voi luoda mitä vain teknisistä malleista animaatioelokuviin.

Ensimmäistä demoa testattiin kehityksen eri vaiheissa Huawein Y625-U51 -puhelimella, jossa on Android 4.4.2 Jelly Bean.



KUVA 1: Unityn materiaalimuokkain. Käyttäjä valitsee valmiin shaderin ja syöttää tarvittavat parametrit.



KUVA 2: Unrealin materiaalinmuokkain. Käyttäjä lisää solmuja, joilla asetetaan materiaalille parametreja.

3.3 Mobiilipelin toteutus Unitylla

Pelin toteutus aloitettiin kirjoittamalla luokat pelin tallentamista ja lataamista sekä lokalisaatiota varten. Peliin haluttiin valmius kääntää se monelle kielelle. Tuetuille kielille kirjoitettaisiin kullekin oma tekstitiedostonsa, jossa pelissä näytettävä fraasi on yhdistetty englanninkieliseen avaimeen. Pelin käynnistyksen yhteydessä luetaan valitun kielen mukainen tiedosto ja puretaan rivit avain-fraasi -pareiksi. Avaimet ovat joka tiedostossa samat, joten niihin voidaan viitata muussa koodissa välittämättä valitusta kielestä. Avainten hyödyntämistä on esitelty kuvassa 6. Lisäksi laadittiin luokkia ylätason pelinhallintaan sekä tasojen ja tasovalikoiden hallintaan. Pelissä pitää jatkuvasti tietää esimerkiksi valittu insuliiniherkkyys sekä tasot, jotka pelaaja on jo suorittanut. Tason sisällä taas on tiedettävä, miten paljon hiilihydraatteja on kerätty ja paljonko pelaajalla on terveyttä jäljellä. Pelikertojen välillä tarvittava tieto tallennetaan omaan luokkaansa, joka serialisoidaan ja tallennetaan tiedostoon. Pelin käynnistyessä tiedosto etsitään ja ladataan. Serialisoitu data puretaan luokaksi, jota voidaan helposti käsitellä muussa koodissa. Serialisoitava luokka on esitetty kuvassa 3 ja tiedostojen käsittelystä vastaava luokka kuvissa 4 ja 5.

GameController

```
using System.Runtime.Serialization;
using System;
using System.Collections;

[Serializable()]
public class GameController : ISerializable
{
    //serializable class for saving the game information

    public int score; //player's score
    public double insulinSensit; //player's insuline sensitivity
    public string lang; //selected language
    public int theme; //current theme
    public ArrayList levels; //passed levels
    public bool soundsOn; //are the game's sounds on

    //default constructor; used for new game
    public GameController()
    {
        score = 0;
        insulinSensit = 1.0;
        lang = "FIN";
        theme = 1;
        levels = new ArrayList();
        soundsOn = true;
    }

    //constructor for loading game; deserialization
    public GameController(SerializationInfo info, StreamingContext ctxt)
    {
        score = (int)info.GetValue("score", typeof(int));
        insulinSensit = (double)info.GetValue("insulinSensit", typeof(double));
        lang = (string)info.GetValue("lang", typeof(string));
        theme = (int)info.GetValue("theme", typeof(int));
        levels = (ArrayList)info.GetValue("levels", typeof(ArrayList));
        soundsOn = (bool)info.GetValue("sound", typeof(bool));
    }

    //serialization
    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("score", score);
        info.AddValue("insulinSensit", insulinSensit);
        info.AddValue("lang", lang);
        info.AddValue("theme", theme);
        info.AddValue("levels", levels);
        info.AddValue("sound", soundsOn);
    }
}
```

KUVA 3: Serialisoitava GameController-luokka.

FileController LoadLanguage()

```
using UnityEngine;
using System.Collections;
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public static class FileController {

    //static class for handling the IO processes
    //also holds the gamecontroller object

    public static GameController gamecontroller;

    public static bool FileExists()
    {
        return File.Exists(Application.persistentDataPath + "/savedgame.diab");
    }

    public static string NewGame()
    {
        try
        {
            gamecontroller = new GameController();
            return "OK";
        }

        catch (Exception e)
        {
            return e.ToString();
        }
    }

    public static string LoadGame()
    {
        try
        {
            Stream stream = File.Open(Application.persistentDataPath + "/savedgame.diab", FileMode.Open);
            BinaryFormatter bformatter = new BinaryFormatter();

            gamecontroller = (GameController)bformatter.Deserialize(stream);
            stream.Close();
            return "OK";
        }

        catch (Exception e)
        {
            return e.ToString();
        }
    }
```

KUVA 4: FileController-luokka.

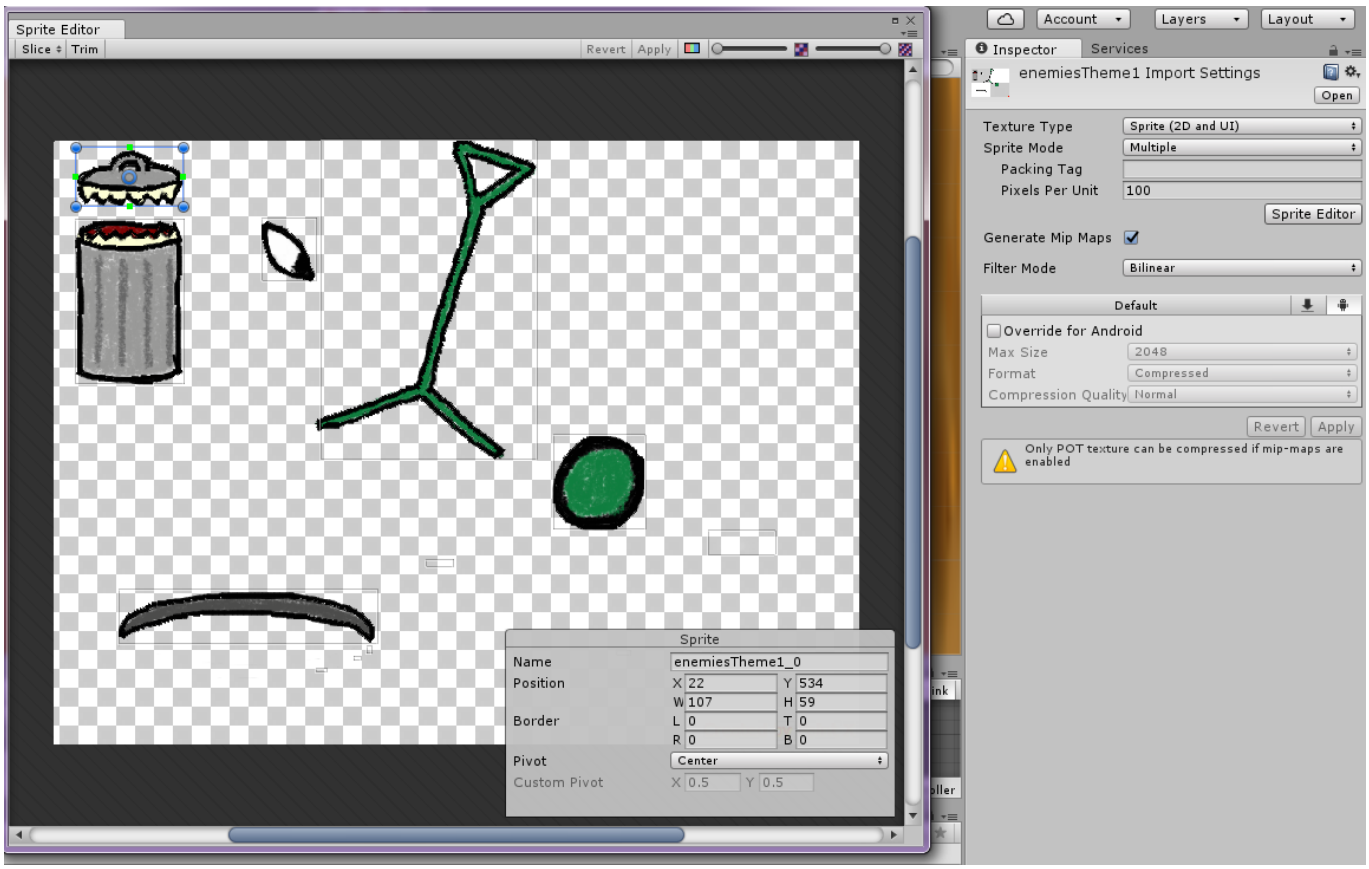
```
    public static string SaveGame()
    {
        try
        {
            Stream stream = File.Open(Application.persistentDataPath + "/savedgame.diab", FileMode.Create);
            BinaryFormatter bformatter = new BinaryFormatter();

            bformatter.Serialize(stream, gamecontroller);
            stream.Close();
            return "OK";
        }

        catch (Exception e)
        {
            return e.ToString();
        }
    }

    public static string LoadLanguage()
    {
        try
        {
            TextAsset txt = (TextAsset)Resources.Load(gamecontroller.lang, typeof(TextAsset));
            string filetext = txt.text;
            LanguageController.ProcessLanguageFileText(filetext);
            Resources.UnloadUnusedAssets();
            return "OK";
        }

        catch (Exception e)
        {
            return e.ToString();
        }
    }
}
```

KUVA 5: FileController-luokka, jatkoa.

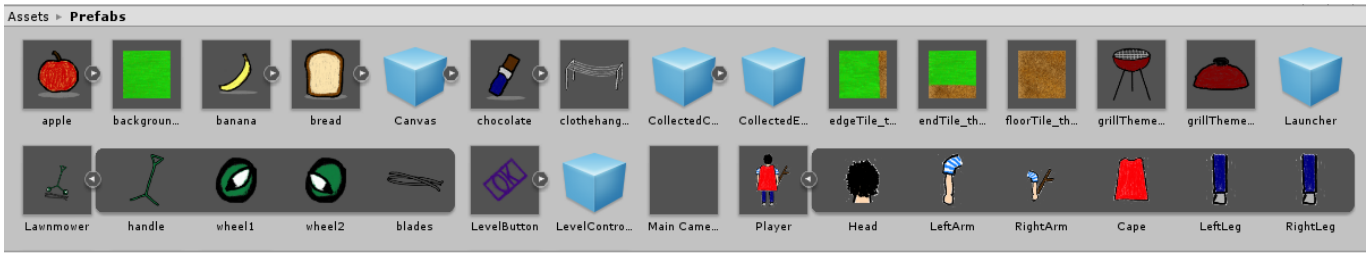
```
tipText.text = LanguageController.UItexts["InsulinTip1"] + FileController.gamecontroller.insulinSensit + LanguageController.UItexts["InsulinTip
nextText.text = LanguageController.UItexts["Next"];
placeholder.text = LanguageController.UItexts["InsulinInputPlaceholder"];
helpButtonText.text = LanguageController.UItexts["InsulinHelpButton"];
youCollectedText.text = LanguageController.UItexts["YouCollected"];
totalText.text = LanguageController.UItexts["CarbTotal"];
```

KUVA 6: Esimerkki dynaamisista tekstikentistä. Teksti haetaan muuttujasta avainten avulla.

Tukitoimintojen toteutuksen jälkeen piirrettiin pelaajahahmo. Piirtäminen tapahtui siis Paint X Litella Macillä. Hahmo piirrettiin paloissa, eli yhden kokonaista hahmoa esittävän kuvan sijaan piirrettiin kaikki kehon osat erikseen. Jalkoja ja käsiä sekä viittoja tehtiin erilaisia, jotta animaatioista saataisiin monipuolisempia. Tiedosto ladattiin GitLabiin, josta se edelleen ladattiin varsinaiselle työkoneelle. Unityyn tuomisen jälkeen kuvan tyyppi piti vielä määritellä. 2D-grafiikoissa kaikki kuvat määritellään yleensä spriteiksi, jotka ovat joko yksi- tai monikuvaisia. Pelaajahahmo on monikuvainen, koska hahmon osat on piirretty erillisinä paloina. Sprite-muokkauksessa valittiin, mitkä osat tiedostossa ovat erillisiä kuvia. Esimerkiksi yksi jalka on yksi kuva ja pää toinen kuva. Kuville asetettiin koko ja napapiste. Esimerkki sprite-muokkauksesta on esitetty kuvassa 7, jossa käsitellään pelin muita hahmoja. Tässä vaiheessa Unityssa oli siis sprite-tyyppinen tiedosto, joka sisälsi useita kuvia. Kuvassa 8 näytetään, miltä monesta osasta koostuvat hahmot näyttävät Unityssa.



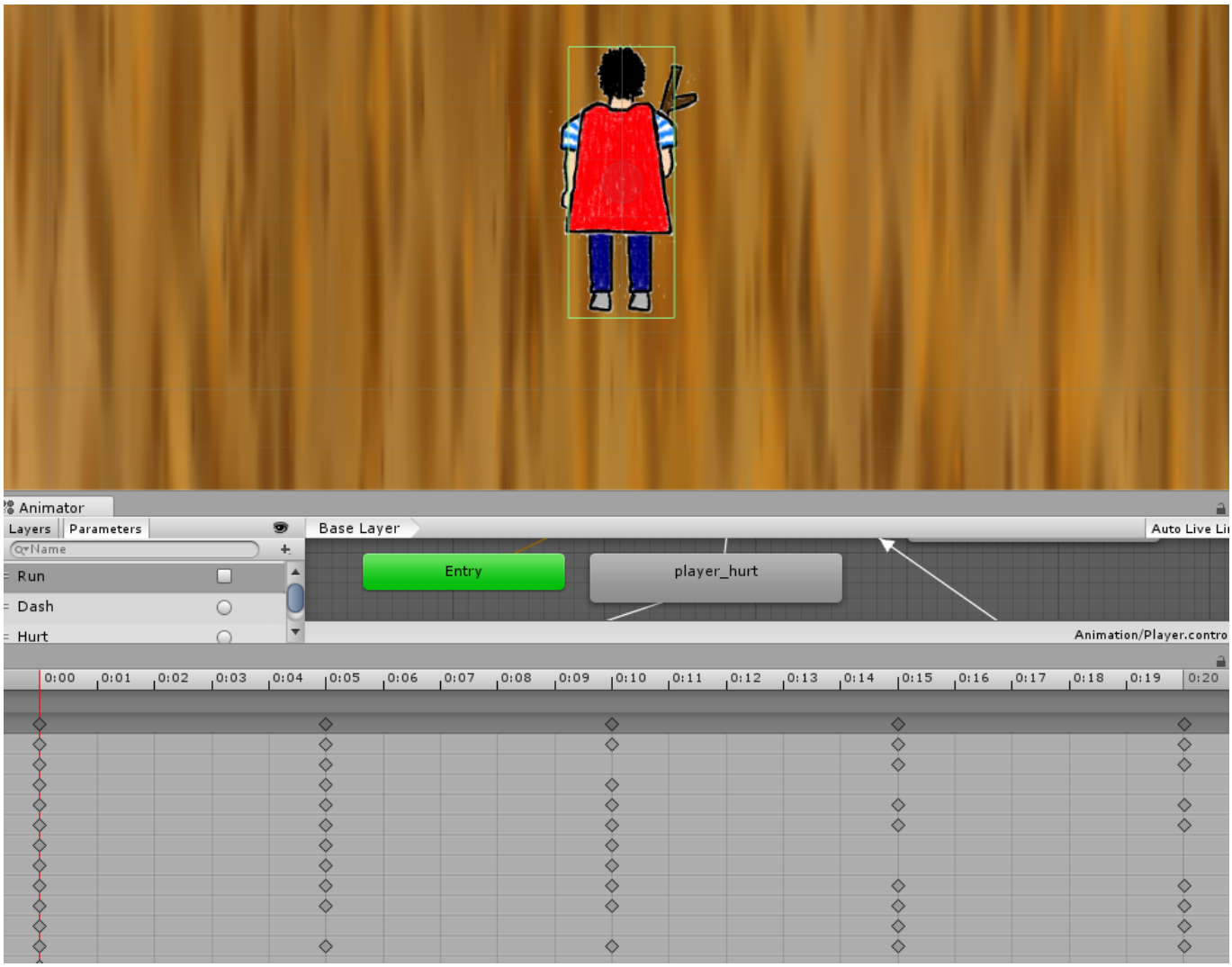
KUVA 7: Esimerkki sprite-muokkauksesta. Yksi tiedosto voi sisältää useita kuvia.



KUVA 8: Prefab-kansio. Pelin hahmot muodostuvat useasta osasta.

Seuraavaksi hahmo animoitiin. Pelihahmoilla on tavallisesti useita erilaisia animaatioita: yksi joka toiminnolle sekä joutilas-animaatio, jota käytetään, kun pelaaja ei anna mitään komentoja. Animaatioita voi toteuttaa 2D-pelissä kahdella eri tavalla. Sprite sheet -tekniikka toimii kuten klassiset piirrosanimaatiot. Liikkeen eri vaiheista piirretään kuvia, joiden välillä hahmon asento muuttuu hieman. Kun kuvia toistetaan suurella nopeudella, hahmo näyttää liikkuvan. Tuloksena on kaunis animaatio, mutta tekniikka vaatii taidokasta piirtämistä. Helpompi vaihtoehto on stop motion -animointia muistuttava tekniikka, jossa hahmon osia liikutellaan vähän kerrallaan ja kunkin muutoksen jälkeen uusi asento tallennetaan. Myös 3D-pelissä animoidaan tällä tekniikalla. Unityssa valitaan olion ominaisarvoja, joita muokataan animaation aikana. Muutokset tallennetaan avainkuviin (key frame). Nukkeanimaatioista poiketen kuvien välillä tapahtuvien muutosten ei tarvitse olla erityisen hienovaraisia. Unity osaa lisätä puuttuvat kohdat kahden määrätyn avainkuvan väliin, jolloin animaatiosta tulee sujuva. Esimerkiksi kävelyanimaatiossa

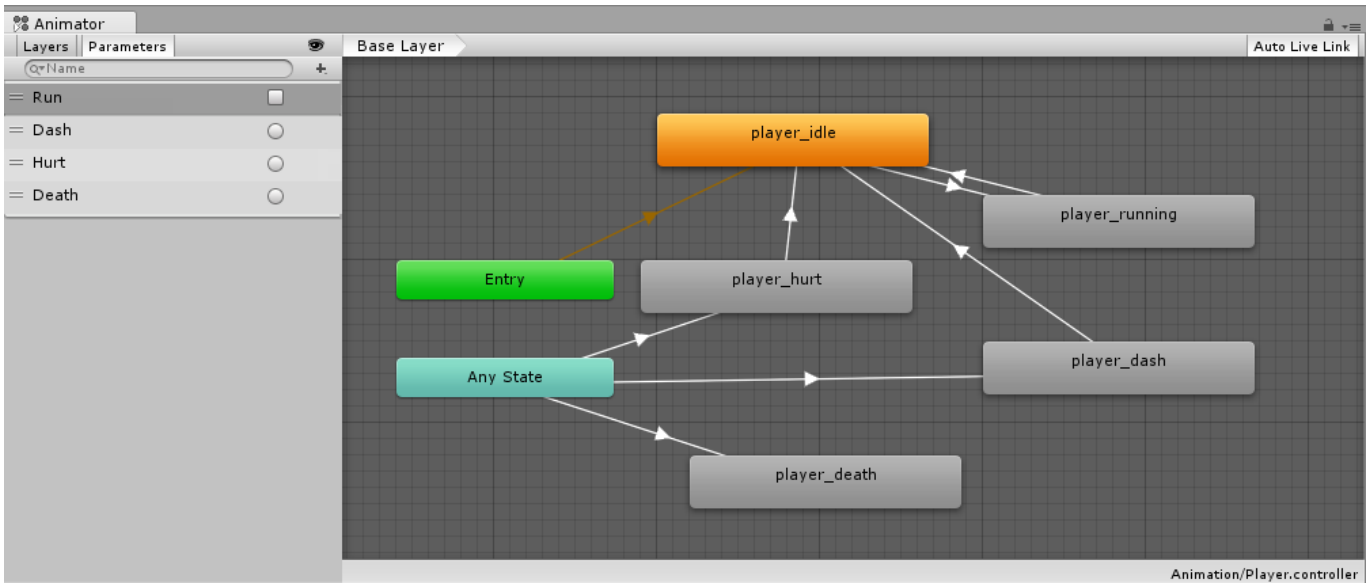
riittää tehdä viisi avainkuvaa. Kolmessa kuvassa hahmon molemmat jalat ovat maassa ja kahdessa toinen jalka on ilmassa. Animaation ensimmäisen ja viimeisen avainkuvan tulee olla identtiset, jotta animaatio toistuu saumattomasti kehässä. Animointia esitellään kuvassa 9.



KUVA 8: Esimerkki animaatioleikkeen luonnista. Valitun olion komponentteja muokataan ja muutokset tallennetaan avainkuviin. Yksi vinoneliö kuvaa yhtä muokattua komponenttia. Allekkain olevat vinoneliöt kuuluvat samaan avainkuvaan.

Yksittäisiä animaatioleikkeitä hallitaan animaatio-ohjaimella. Ohjain siirtyy animaatiosta toiseen tiettyjen ehtojen mukaan ja häivyttää tarvittaessa leikkeiden rajat, jolloin liikkeet näyttävät luonnollisemmilta. Serial Snackerin pelaajahahmolla on viisi erilaista animaatiota: joutilas, juokseminen, lyöminen, vahingon ottaminen ja kuoleminen. Animaatioiden hallinnasta vastaava ohjain on esitetty kuvassa 10. Joutilas-animaatio on hahmon oletusanimaatio, joka käynnistyy, kun olio aktivoidaan suorituksen aikana. Tästä animaatiosta voidaan siirtyä juoksuanimaatioon ja takaisin. Kuvan 10 vasemmassa reunassa on lista parametreista, joiden avulla siirtymiä hallitaan. Run-parametri on boolean-tyyppinen parametri, jonka ollessa tosi näytetään juoksuanimaatio. Kun parametri palautetaan epätosi-tilaan, siirrytään takaisin oletusanimaatioon. Muihin animaatioihin voidaan siirtyä mistä tahansa tilasta. Näitä siirtymiä hallitaan trigger-tyyppisillä parametreilla. Lyönnistä ja vahingon ottamisesta siirrytään oletusanimaatioon, kuolemasta ei

siirrytä mihinkään. Näissä tapauksissa siirtymä ei tapahdu minkään parametrin vaikutuksesta, vaan leikkeen päättyessä.



KUVA 9: Esimerkki animaatio-ohjaimesta. Oranssilla merkitty leike on oletusleike. Muihin leikkeisiin siirrytään vasemmassa reunassa listattujen parametrien mukaan.

Animoimisen jälkeen hahmolle kirjoitettiin oma luokka, joka vastaa pelaajan komentojen toteuttamisesta. Luokka kuuntelee pelaajan tekemisiä ja liikuttaa hahmoa sekä käynnistää animaatioleikkeitä niiden mukaisesti. Luokan ensimmäisessä versiossa kaikki tuetut komennot olivat näppäinkomentoja, koska peliä testattiin aluksi vain tietokoneella. Ensimmäisten testien tarkoituksena oli hienosäätää animaatioita. Tietokoneella testaaminen oli nopeampaa kuin puhelimella testaaminen, joten näppäinohjaus pidettiin mukana koko kehityskaaren ajan pieniä korjauksia silmällä pitäen. Kun animaatioihin oltiin tyytyväisiä, lisättiin kosketustunnistus ja siirryttiin testaamaan varsinaista hahmon ohjaamista puhelimella. Hahmon ohjaaminen kosketusnäytöllä ei juuri eroa näppäimillä ohjaamisesta. Liikkumiseen ja animaatioiden käynnistymiseen tarvittava koodi on kummassakin tapauksessa sama, erona on vain syötteen tunnistus. Kun ohjataan näppäimistöllä, kuunnellaan tiettyjen näppäimien painalluksia. Kun ohjataan kosketuksella, lasketaan kosketusten määrää sekä tarkastellaan kosketuksen siirtymistä edellisestä mittauskohdasta. Tässä työssä oltiin kiinnostuneita vain yhdestä kosketuksesta, koska peliä pelataan yhdellä sormella. Näppäinkomentojen seuraamista on esitelty kuvassa 11, kosketuksen seuraamista puolestaan kuvassa 12.

```
# if UNITY_EDITOR

          if (hit)
           {
               transform.Translate(0F, forwSpeed * -1 * Time.deltaTime, 0F);
           }

           else if (Input.GetKey(KeyCode.W) && !dashing)
           {
               RunForward();
           }

           else if ((Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D)) && !dashing)
           {
               animator.SetBool("Run", true);
               transform.Translate(Input.GetAxis("Horizontal") * sideSpeed * Time.deltaTime, forwSpeed * forwCapper * Time.deltaTime, 0f);
           }

          else if (dashing)
          {
              Dash();
          }

          else if (dashReady)
          {
              animator.SetTrigger("Dash");
              dashing = true;
              Dash();
              Invoke("StopDash", dashTime);
          }

           else if (ready)
           {
               Idle();
           }

# endif

#if UNITY_ANDROID && !UNITY_EDITOR

        //dash on release if near enemy

        if (hit)
        {
            transform.Translate(0F, forwSpeed * -1 * Time.deltaTime, 0F);
        }

        else if (Input.touchCount > 0)
```

KUVA 10: Pelaajahahmon liikkumisesta vastaava koodin osa.

```
else if (Input.touchCount > 0)
{
    if (Input.GetTouch(0).phase == TouchPhase.Moved)
    {
        Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;

         animator.SetBool("Run", true);
         transform.Translate(touchDeltaPosition.x * sideSpeed * Time.deltaTime, forwSpeed * forwCapper * Time.deltaTime, 0f);

    }

    else
    {
        RunForward();
    }
```

KUVA 11: Pelaajahahmon liikkumisesta vastaava koodin osa, jatkoa.

Kun ohjattava hahmo oli valmis, piti rakentaa ympäristö. Suunnitteluvaiheessa oli päätetty, että pelissä olisi erilaisia teemoja, joiden mukaan koristelu, viholliset ja ruuat vaihtelisivat. Ensimmäiseksi teemaksi valittiin takapiha. Piha on hiekkapohjainen ja sitä reunustaa nurmikko. Pihalla on vaja, roskapönttöjä, pyykkinaru, ruohonleikkuri, grilli, kiviä ja marjapensaita. Pihalla seikkaillessaan sankari syö tavallisia välipaloja, kuten suklaata, jogurttia ja banaaneja. Pelissä osa pihan esineistä on liikkumattomia esteitä, osa taas keinoälyn ohjaamia vihollisia.

Ensimmäinen vaihe ympäristön rakentamisessa oli vihollisten luominen. Vihollisiksi valittiin roskapönttö ja käsikäyttöinen ruohonleikkuri. Nämä luotiin samalla tavalla kuin pelaajahahmo – hahmot piirrettiin osissa, tuotiin Unityyn ja animoitiin liikuttelemalla osia. Molemmille vihollisille laadittiin omat animaatio-ohjaimensa ja ohjaavat luokkansa. Pelaajahahmosta poiketen viholliset eivät kuuntele pelaajan ohjausta, mutta sen sijaan ne seuraavat hahmon sijaintia tasolla. Kun hahmo tulee riittävän lähelle, viholliset aktivoituvat. Vielä lähemmäs tultaessa ne pyrkivät hyökkäämään ja vahingoittamaan pelaajahahmoa. Jos pelaaja onnistuu lyömään vihollista, se kuolee. Pelaaja voi myös vain ohittaa vihollisen, jolloin se siirtyy inaktiiviseen tilaan.

Seuraavaksi luotiin itse taso. Ympäristö tehtiin asettamalla kuva pihasta hahmojen taakse. Todellisuudessa kuva on kaukana hahmoista, jotta se ei häiritse pelin toimintoja, mutta ortografinen kamera poistaa syvyysnäkymän kokonaan, jolloin hahmot näyttävät olevan aivan taustan päällä. Lisäksi kuva on rakennettu useasta pienestä kuvasta, jotta vältytään suuren zoomauksen aiheuttamalta huonolta kuvanlaadulta. Näitä pieniä kuvia kutsutaan jatkoissa laatoiksi. Laattoja piirrettiin useita erilaisia: osa oli kokonaan hiekkaa, osa kokonaan nurmikkoa ja osassa oli molempia. Reunat tehtiin nurmikkolaatoista ja keskus hiekkalaatoista. Välissä oli sekoitelaattoja. Yhden tason rakentamiseen tarvittiin satoja laattoja. Työtä nopeutettiin kirjoittamalla algoritmi, joka sijoittaa laatat taustalle automaattisesti, kun taso ladataan. Algoritmi tarkistaa, mihin kohtaan tasoa laatta on tulossa, ja valitsee oikeanlaisen laatan. Lisäksi puolet hiekan reunaan tulevista laatoista pitää kääntää eri asentoon, koska ne ovat vastakkaisella puolella kuin miten ne on piirretty. Laatojen automaattisesta asettelusta vastaava algoritmi on esitetty kuvissa 13 ja 14.

```
void Start ()
{
    Renderer tileRenderer = floorTile.GetComponent<Renderer>();
    Renderer floorRenderer = gameObject.GetComponent<Renderer>();
    tileX = tileRenderer.bounds.size.x;
    tileY = tileRenderer.bounds.size.y;
    floorX = floorRenderer.bounds.size.x;
    floorY = floorRenderer.bounds.size.y;
    xcount = Mathf.CeilToInt(floorX / tileX);
    ycount = Mathf.CeilToInt(floorY / tileY);
    zeroPoint = floorRenderer.bounds.center;
    zeroPoint = new Vector3(zeroPoint.x - floorX / 2, zeroPoint.y - floorY / 2, 0);
    PlaceTiles();
}
```

KUVA 12: Laattojensijoitusalgoritmi.

```
for (int i = 0; i < ycount; i++)
{
    for (int j = 0; j < xcount; j++)
    {
        //left edge
        if (j == 0 && i != 0 && i != ycount - 1)
        {
            tile = edgeTile;
            spawnrotation = Quaternion.identity;
        }

        //right edge
        else if (j == xcount - 1 && i != 0 && i != ycount - 1)
        {
            tile = edgeTile;
            spawnrotation = Quaternion.Euler(0, 0, 180f);
        }

        //start
        else if (i == 0 && j != 0 && j != xcount - 1)
        {
            tile = endTile;
            spawnrotation = Quaternion.Euler(0, 0, 180f);
        }

        //end
        else if (i == ycount - 1 && j != 0 && j != xcount - 1)
        {
            tile = endTile;
            spawnrotation = Quaternion.identity;
        }
```

KUVA 13: Laattojensijoitusalgoritmi, jatkoa.

Pelaajalla oli ympäristö, jossa liikkua ja vihollisia, joita vastaan taistella. Seuraavaksi lisättiin ruokia, joita pelaaja voi kerätä. Ruuiksi valittiin banaani, omena, suklaapatukka, maustettu jogurtti ja paahtoleipä. Ruuat piirrettiin samaan tiedostoon. Lisäksi niille piirrettiin pieni varjo, jotta ne näyttäisivät leijuvan vähän maan yläpuolella. Myös ruuille lisättiin yksinkertainen animaatio vahvistamaan kuvaa leijuvasta esineestä. Ruokiin liitettiin koodia, joka havaitsee pelaajahahmon kosketuksen ja kerryttää kerättyjen hiilihydraattien määrää.

Kamera asetettiin siten, että kuvan rajat ovat tason reunoissa. Kamera liikkuu pelaajahahmon mukana ainoastaan ruudulla ylöspäin, kohti tason loppua. Jos pelaaja siis ohjaa hahmoa riittävästi sivulle, hahmo voi kävellä pois kuvasta. Tämä estettiin koodilla, joka havaitsee hahmon lähestyvän reunaa ja työntää tämän takaisin tulosuuntaansa.

Kaikki pelaajahahmon kanssa kosketuksissa olevat hahmot ja esineet huolehtivat tämän kosketuksen seurauksista. Joissakin tapauksissa tämä tarkoittaa pelaajahamon luokassa olevien julkisten funktioiden kutsumista. Esimerkiksi vahingon ottaminen vihollisen koskemisesta aiheuttaa hahmon siirtymisen taaksepäin ja animaation käynnistymisen, molemmat toimintoja, jotka haluttiin pitää muiden vastaavien toimintojen kanssa tämän hahmon omassa luokassa. Tässä tapauksessa voisi kuvitella olevan viisaampaa, että pelaajahahmo itse havaitsisi kosketuksen vihollisen kanssa. Tämä ei kuitenkaan ole Unityn tapauksessa mielekästä. Unityssä kosketus on helpointa havaita törmäyksenä. Törmäyksessä yhden olion rigidbody-komponentti menee sisään toisen olion collider-komponenttiin. Toisin sanoen toisella oliolla on pakko olla rigidbody ja toisella collider. Se olio, jolla on collider, havaitsee törmäyksen. Rigidbody on samalla komponentti, jonka avulla olio altistetaan fysikaalisille voimille. Koska pelaajahahmoon kohdistetaan pelin aikana voimia, tulee sillä olla rigidbody. Toisella osapuolella, siis vihollisilla, ruuilla tai tason reunalla, tulee olla collider. Siksi juuri vihollinen havaitsee kosketuksen pelaajahahmon kanssa ja aiheuttaa vaikutukset molempiin osapuoliin.

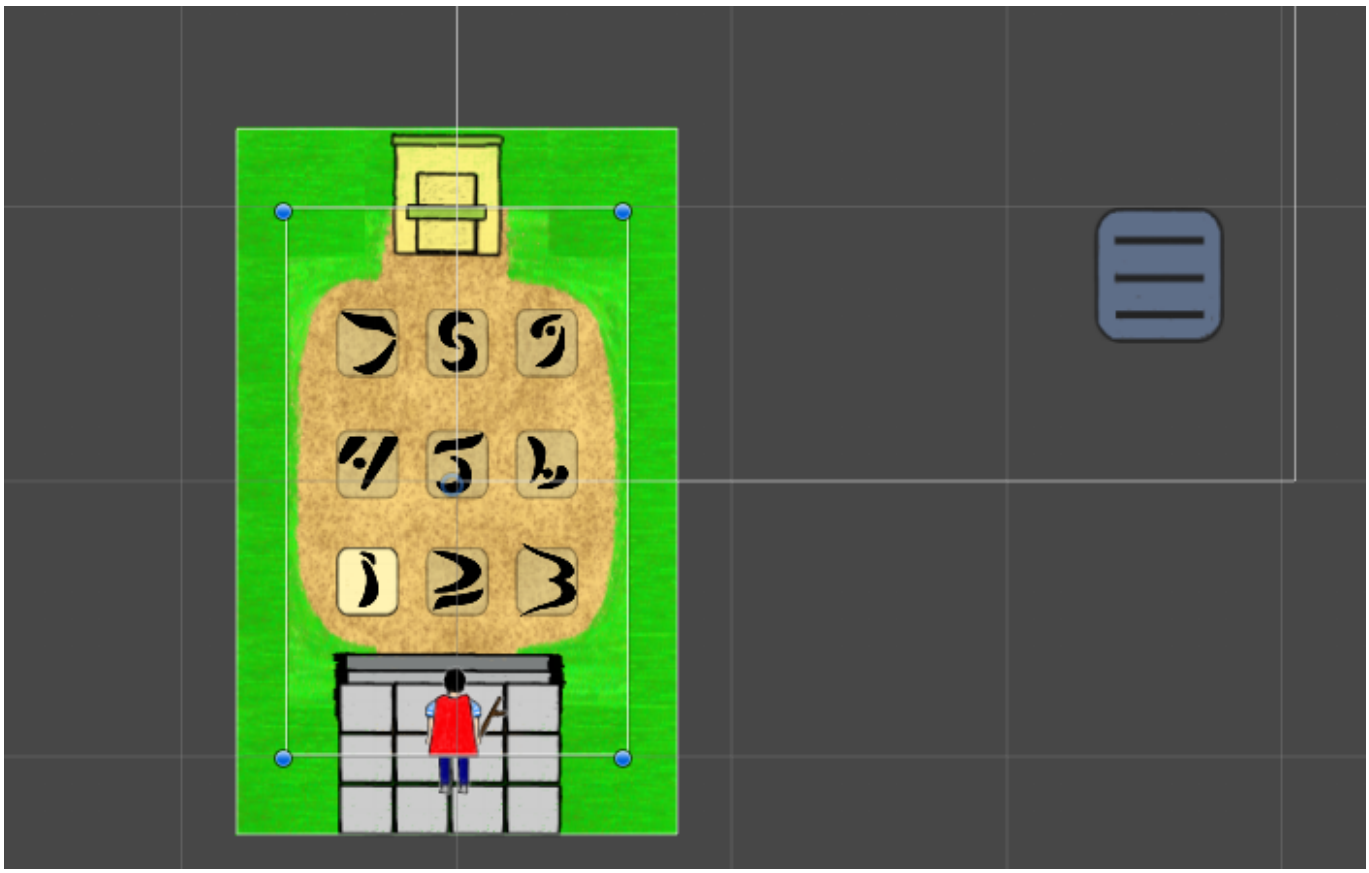
Tässä vaiheessa kehitystä pelissä oli siis taso, jossa oli vihollisia ja ruokia. Lisäksi pelissä oli tietysti myös ohjattava hahmo. Taso oli kuitenkin edelleen kovin tyhjän tuntuinen ja pelistä puuttui palaute pelaajan suuntaan. Seuraavaksi piirrettiin siis kiviä, grilli ja pyykkinaru, joita aseteltiin tasolle esteiksi. Hahmo ei voi kulkea näiden esineiden läpi vaan ne pitää kiertää. Lisäksi piirrettiin marjapensaita, joita asetettiin tason loppuun. Kun pelaaja saapuu tälle alueelle, kamera pysähtyy ja peli ottaa hahmon hallintaansa. Hahmo kävelee automaattisesti pensaiden sekaan ja pois kuvasta, jolloin taso päättyy.

Pelaajalle pitää kertoa, miten paljon terveyttä hänellä on jäljellä ja kuinka paljon hän on kerännyt hiilihydraatteja. Ruudun yläreunaan lisättiin mittari terveydelle ja tekstikenttä hiilihydraateille. Myös muuta palautetta lisättiin. Vahinkoa ottaessaan pelaajahahmo välähtää punaisena. Hahmon kaikille osille annetaan punainen päällys, joka häivytetään koodissa kuvan 15 mukaisesti. Kun ruoka poimitaan, sen sisältämä hiilihydraattimäärä ilmestyy hetkeksi ruudulle. Pelaajahahmon perään piirtyy valkoinen viiva, kun vihollisen lyöminen on mahdollista.

```
IEnumerator ColorFade()
{
    while (rends[0].color != normalColor)
    {
        foreach (SpriteRenderer rend in rends)
        {
            rend.color = Color.Lerp(rend.color, normalColor, fadeDelay * Time.deltaTime);
        }

        yield return null;
    }
}
```

KUVA 14: Pelaajahahmon punaisen välähdyksen häivytys.

Pelissä oli tarkoitus olla useita samaan teemaan kuuluvia tasoja, joten peliin lisättiin teeman mukainen tasovalikko. Tasovalikossa ei tarvinnut huolehtia huonosta kuvanlaadusta, koska kameraa ei zoomattu yhteen alueeseen kuvassa. Siksi taustaksi piirrettiin yksi koko pihaa esittävä kuva. Keskelle ruutua lisättiin nappeja, joita painamalla pääsee pelaamaan tiettyä tasoa. Ruudun yläreunaan lisättiin vaja, jonka ovi on kiinni. Ideana oli, että kun kaikki teeman tasot olisi suoritettu, vajan ovi aukeaisi ja sisällä odottaisi klassinen pomotappelu. Tasovalikko on esitetty kuvassa 16.

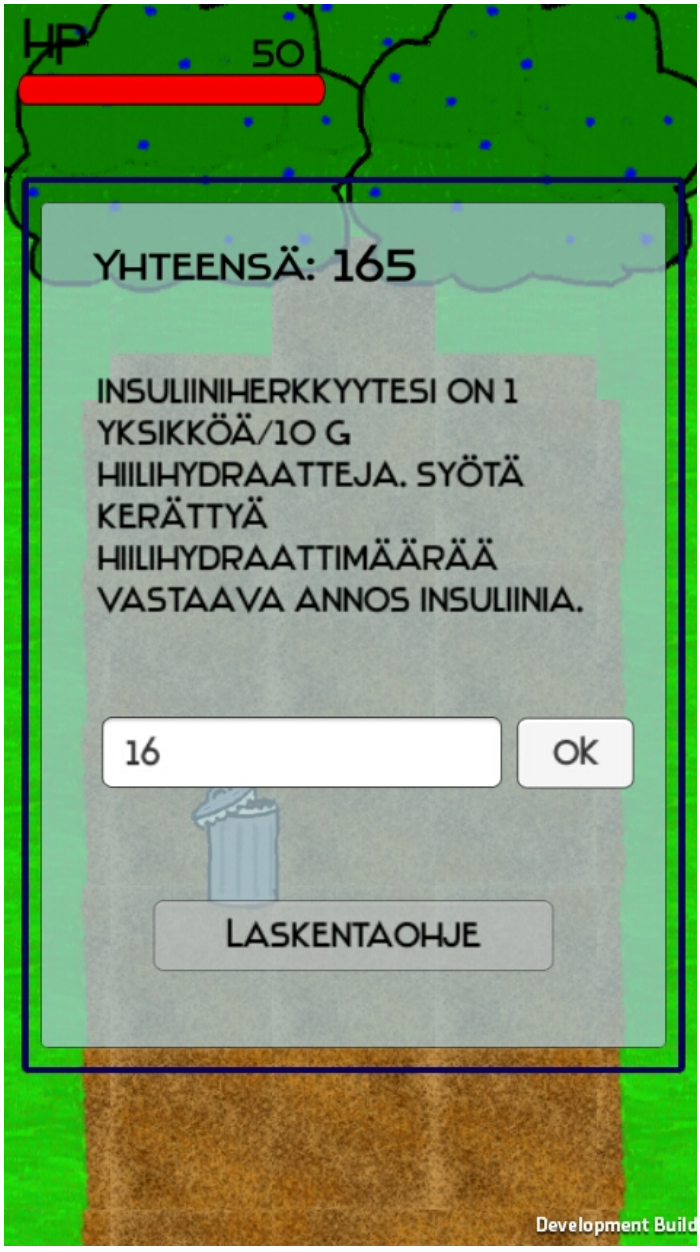


KUVA 15: Tasovalikko.

Suurin osa pelin komponenteista oli tässä vaiheessa kasassa. Pelille keksittiin nimi ja laadittiin splash screen. Tämän pelin splash screen on yksinkertainen sinitaustainen näyttö, jossa lukee pelin nimi. Kielitiedoston lataaminen ja mahdollisen tallennuksen etsiminen tapahtuvat tämän näytön taustalla. Peliin lisättiin siirtymät. Ajastetulta splash screeniltä siirrytään tasovalikkoon. Tasovalikosta pääsee tasolle napista painamalla. Tasolta palataan tasovalikkoon kolmella tavalla: pelaamalla taso loppuun, kuolemalla tai keskeyttämällä. Kun tason pelaa loppuun, lisätään napin päälle tasovalikossa leimaa imitoiva merkki. Näin pelaaja näkee, mitkä tasot on suoritettu.

Viimeiseksi tason loppuun lisättiin hiilihydraattiyhteenveto. Se aukeaa ruudulle, kun pelaajahahmo on kadonnut marjapensaiden sekaan. Yhteenveto koostuu kahdesta näytöstä. Ensimmäisessä listataan kaikki tason ruuat hiilihydraattimäärineen, kuten kuvassa 17. Lisäksi näytetään, kuinka monta kappaletta kutakin ruokaa pelaaja keräsi sekä kokonaishiilihydraattimäärä. Toisella näytöllä näytetään kokonaishiilihydraatit sekä pelaajan insuliiniherkkyys kuvan 18 mukaisesti. Pelaajaa pyydetään syöttämään näiden perusteella oikea insuliinin määrä. Hiilihydraattiyhteenveto toteutettiin animoimalla käyttöliittymäelementtejä, joita voi Unityssa animoida samalla tavalla kuin muitakin olioita.

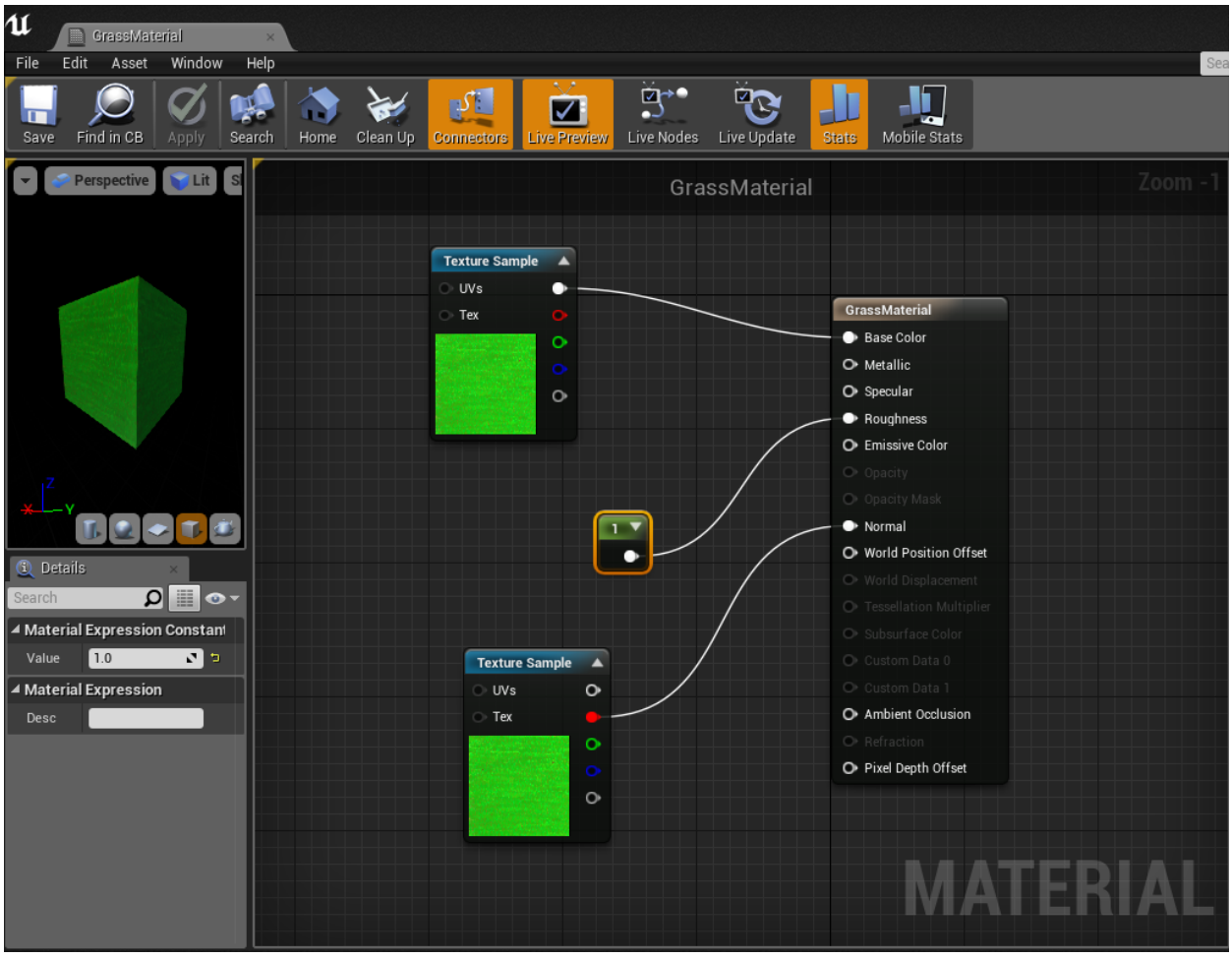
KUVA 16: Hiilihydraattiyhteenveto, näyttö 1



KUVA 17: Hiilihydraattiyhteenveto, näyttö 2.

3.4 Demon toteutus Unreal Enginellä

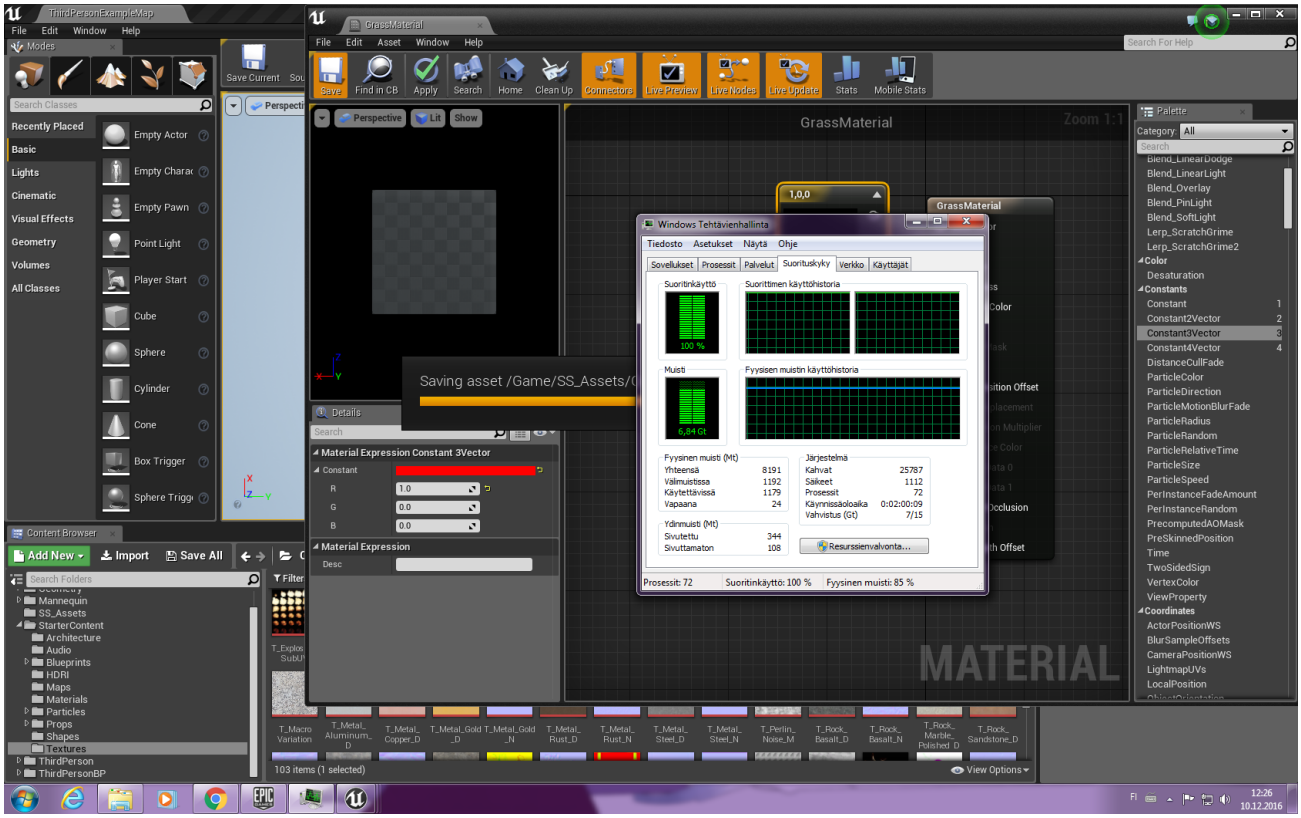
Unrealilla toteutettavan demon oli tarkoitus olla mahdollisimman samankaltainen kuin edellinen demo. Tasosta pyrittiin siis tekemään samannäköinen hiekkapiha, jonka reunoilla on nurmikkoa. 3D-ympäristön vuoksi kaikkia resursseja ei voinut vain siirtää uudelle pelimoottorille. Kierrätettyjä resursseja ovat siis vain ruohon ja hiekan tekstuurit. Edellistä demoa varten piirretyt kuvat tuotiin Unrealiin yksinkertaisina kuvatiedostoina. Niiden pohjalta luotiin hiekka- ja ruohomateriaalit. Unrealissa on hienostunut materiaalinmuokkauskomponentti, jolla tämä tehtiin. Unrealin materiaalit muodostuvat solmuista, joita yhdistelemällä saadaan aikaiseksi hyvin monimuotoisia materiaaleja. Tekstuuri on yksi solmu. Demon materiaaleihin lisättiin kaksi muuta solmua: normaalikartta ja rosoisuus. Normaalikartta antaa materiaalille 3D-pinnan, esimerkiksi puulle syyt. Rosoisuus määrittää materiaalin kiiltävyyden. Hiekasta ja ruohosta tehtiin kiillottomia ja niiden pintaan tehtiin pientä rakennetta. Ruohomateriaalin luominen on esitetty kuvassa 19.



KUVA 18: Ruohomateriaalin luominen.

Tason lattia rakennettiin asettamalla laattoja peliavaruuteen. Reunojen laatat päällystettiin ruohomateriaalilla ja keskialueen laatat hiekkamateriaalilla. Ruohoalueille lisättiin Unrealin omia pensasresursseja. Näinkin pienen ja yksinkertaisen tason rakentamiseen kului paljon aikaa. Pelkkä laattojen asettelu oli hidasta, koska Unreal jumiutui jatkuvasti. Sama ongelma ilmeni materiaalien luomisessa. Solmujen lisääminen ja päivittäminen vei tietokoneen suorituskykynsä

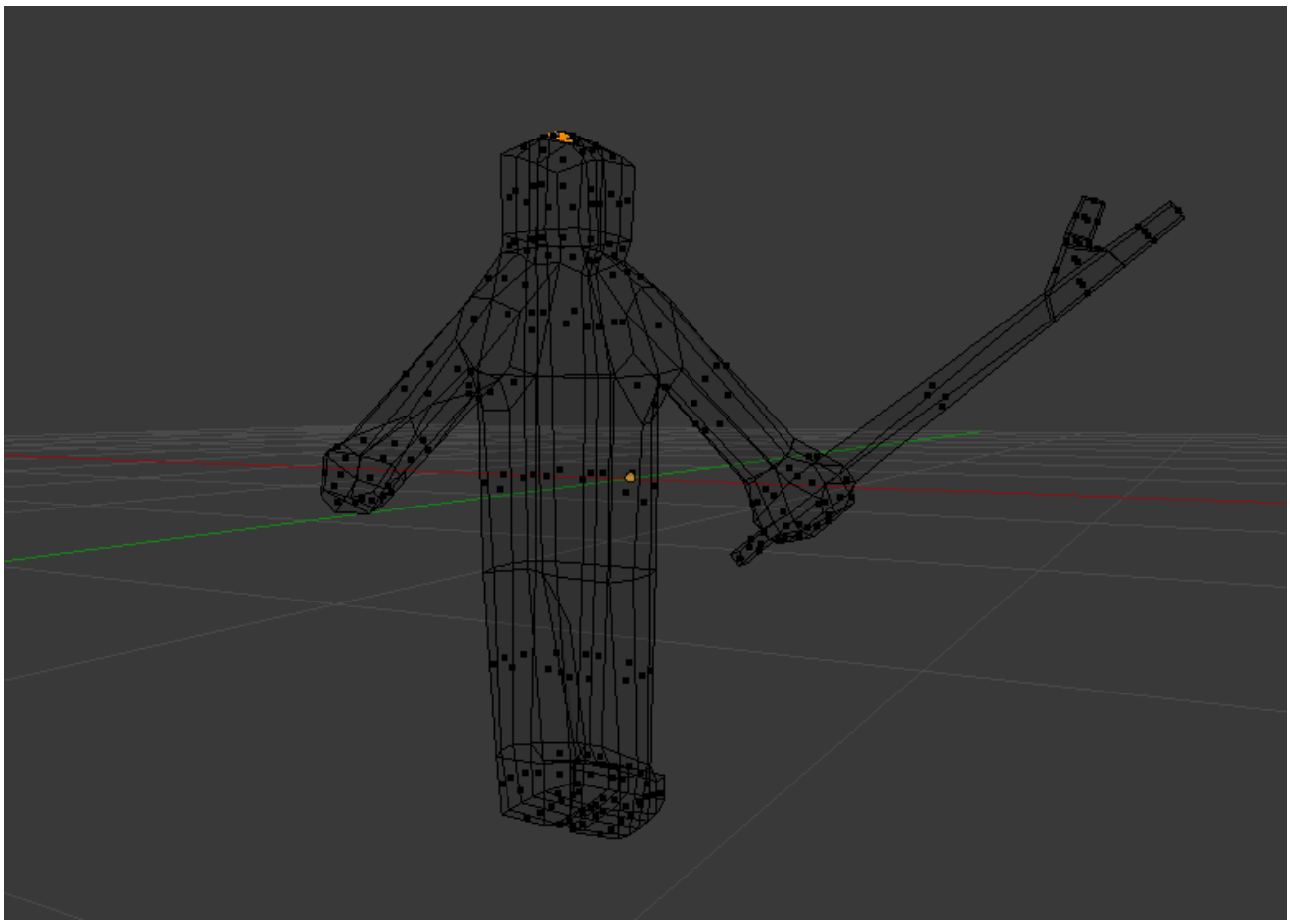
äärirajoille. Tilanne on kuvattu kuvassa 20. Näytönohjain kaatui useaan otteeseen kaataen samalla Unrealin, mikä johti muutosten menetykseen. Unreal ei myöskään suostunut lähtemään joka kaatumisen jälkeen käyntiin ilman koko koneen uudelleenkäynnistystä. Jumiutuminen taas kesti joissakin tapauksissa tunteja.



KUVA 19: Ruohomateriaalin luominen, teknisiä ongelmia. Prosessorin käyttöaste on 100 % ja ohjelma jumiutuu.

Kun jonkinlainen taso oli lopulta tehty, piti luoda pelaajahahmo. Alkuperäinen hahmo on kuvatiedosto, joten sitä ei voitu tässä tapauksessa hyödyntää. Hahmon luontiin käytettiin siis Blenderiä. Blenderissä otetaan jokin aloitusmuoto, josta aletaan muokata haluttua hahmoa manipuloimalla muodon kärkiä. Kaikki aloitusmuodot ovat yksinkertaisia geometrisiä muotoja lukuun ottamatta apinan päätä, joka Blenderistä myös löytyy. Aloitusmuodoksi valittiin kuutio. Ensin kuutio venytettiin korkeaksi suorakaiteeksi, johon lisättiin kärkiä pitkin pituutta. Osaa kärjistä vedettiin sitten ulospäin, jolloin syntyi kädet ja jalat. Tästä eteenpäin muokkaus oli hienovaraisempaa, kun alun perin kantikkaaseen muotoon piti saada ihmismäistä pyöreyttä.

Kun yksinkertainen ihmismalli oli valmis, lisättiin hahmon käteen keppi venyttämällä kädessä olleita pintoja ulospäin. Valmis muoto on esitetty kuvassa 21. Sitten mallin eri alueet väritettiin, jotta hahmolle saatiin hiukset, paita, housut, kengät ja keppi. Valmis malli tuotiin Unrealiin ja asetettiin seisomaan tason reunaan. Viitta tehtiin erikseen. Se on yksinkertainen, nelikulmainen ja jäykkä kappale, sillä kunnollisen kankaan mallintaminen on erittäin vaikeaa jopa kokeineille mallintajille ja suorastaan mahdotonta aloittelijoille.



KUVA 20: Pelaajahahmon luominen Blenderissä.

Lopuksi tason valoja säädettiin ja kamera aseteltiin pelaajahahmon taakse. Toiveissa oli, että hahmoon voitaisiin liittää yksinkertaista koodia, joka mahdollistaisi tasolla liikkumisen. Se olisi kuitenkin ollut turhaa, koska kone ei jaksanut kunnolla pyörittää sitäkään vähää, mitä oli saatu aikaiseksi.

4 TULOKSET

Tässä luvussa esitellään opinnäytteen tulokset. Opinnäytetyössä luotiin kaksi erilaista pelidemoa eri alustoille käyttäen erilaisia tekniikoita.

4.1 Mobiilipeli

Tuloksena syntyi 2D-mobiilipeli, jossa kamera on hahmon takana ylhäällä. Kuvassa 22 näkyy pelaajahahmo, vihollisena toimiva roskapönttö, kerättävä banaani sekä HUD. Hahmoa liikutetaan yhdellä sormella. Ideana on kulkea tason päästä päähän keräten tasolle ripoteltuja ruokia, joiden sisältämä hiilihydraattimäärä ilmoitetaan pelaajalle keräämisen yhteydessä. Tasolla on myös vihollisia ja esteitä, joiden ohi tulee päästä. Tason lopussa näytetään pelaajalle yhteenveto kaikista kerätyistä ruuista ja niiden hiilihydraattimääristä. Kokonaishiilihydraattien pohjalta pelaajan tulee syöttää oikea insullinimäärä perustuen pelaajan insuliiniherkkyyteen.

Pelissä on valmius useaan eri tasoon sekä eriteemaisiin tasovalikkoihin. Lisäksi pohjustettiin mahdollisuutta kääntää peli usealle eri kielelle. Tasoja ei tehty enempää, koska yhdestäkin tasosta saa pelin idean hyvin esille, ja haluttiin siirtyä tekemään monimutkaisempaa tuotetta. Alkuperäinen ajatus oli kuitenkin ollut tehdä valmiimpi ja laajempi peli.

Tilaaja oli tyytyväinen peliin. Kiitosta tuli erityisesti ulkoasusta ja värien käytöstä sekä tason lopun hiilihydraattiyhteenvedosta, jota pidettiin selkeänä ja tarkoitukseen sopivana. Hiilihydraattiyhteenveto esitellään kuvissa 17 ja 18. Pelitekniikka selvisi testauksessa lyhyen selostuksen jälkeen, ja peliä aiemmin näkemättömätkin osasivat sitä pelata. Peli vaikuttaa stabiililta, mutta siinä on joitakin pieniä vikoja. Kokonaisuudessaan peliä pidettiin hyvänä pohjana tulevalle.

KUVA 21: Kuvakaappaus Serial Snacker -mobiilipelistä.

4.2 Tietokonedemo

Unrealilla toteutettu tietokoneelle tarkoitettu demo jäi tyngäksi. Se sisältää pienen tason, jossa on alkuperäisillä tekstuureilla toteutettu maisema sekä uusi 3D-malli. Kuvassa 23 näkyy uusi pelaajahahmo sekä pieni taso. Lähinnä demosta saa jotakin ajatusta siitä, miltä uusi tuote voisi näyttää. Siitä puuttuu paljon elementtejä eikä sitä voi pelata. Taustalla oleva idea vaikuttaa kuitenkin lupaavalta. Tarkoituksena on toteuttaa suunniteltu peli, kun käytettävissä on tehokkaampi tietokone. Unrealissa olevan kameravian vuoksi lopullinen toteutus saattaa sittenkin tapahtua Unitylla.

KUVA 22: Kuvakaappaus Serial Snackerin 3D-tietokonepeliversiosta.

5 YHTEENVETO

Opinnäytetyön aiheena oli diabetespeli, joka opettaisi nuoria potilaita hoitamaan tautiaan itsenäisesti. Alkuperäinen tavoite oli luoda mobiilipeli. Työn tilaaja antoi pelille joitakin reunaehtoja, mutta muuten tekijällä oli vapaat kädet. Suunnitteluun käytettiin paljon aikaa. Haastavaa oli yhdistää hauskuus ja opettavaisuus. Ideoita peliin oli runsaasti, mutta mobiilialusta itsessään oli haaste. Tietokonepelejä on olemassa visuaalisista novelleista avoimen maailman roolipeleihin, mutta mobiilipelien maailmassa valikoima on rajallinen. Pieni näyttö ja nappuloiden puute rajoittavat ohjaamista eivätkä grafiikat voi koskaan olla yhtä hyviä kuin tietokoneissa ja konsoleissa. Pelistä oli tehtävä yksinkertainen. Työn edetessä tavoite muuttui, kun päätettiin tehdä demo monimutkaisemmasta tietokonepelistä. Peli-ideaan ei kuitenkaan tehty muutoksia tässä vaiheessa, vaan tyydyttiin vaihtamaan grafiikat 3D:hen.

Opinnäytteen tuloksena syntyi kaksi pelidemoa, joista toinen on tehty Unitylla mobiilialustalle ja toinen Unrealilla tietokoneelle. Ensin mainittu käsittää splash screenin, tasovalikon ja yhden tason, toinen yhden tason, jota ei kuitenkaan voi pelata. Unity-demo päätettiin jättää suunniteltua vajaammaksi, koska toiveissa oli monimutkaisempi peli. Unreal-demon toteutusta vaikeutti käytössä olleen tietokoneen heikko suorituskyky, joka ei ollut riittävä massiiviselle ohjelmalle.

Toimeksiantaja oli tyytyväinen tulokseen. Mobiilipeli vastasi hyvin alussa asetettuja vaatimuksia. Koska havaittiin, että mobiilialusta asettaa omat rajoituksensa, mietittiin, tekisikö monimutkaisempi malli pelistä paremman. Tietokoneelle suunnatussa pelissä on mahdollista lisätä hahmon hallittavuutta ja tuoda siten peliin uutta sisältöä. Samalla voitaisiin käsitellä muita diabetekseen liittyviä elementtejä hiilihydraattien laskemisen lisäksi. Kenties voitaisiin jotenkin huomioida fyysisen rasituksen tai ympäristömuuttujien vaikutukset verensokeriin. Näiden lisäelementtien ongelma on, että vaikutukset ovat yksilöllisiä eivätkä samalla tavalla mitattavissa kuin jonkin ruuan sisältämä hiilihydraattimäärä. Uusia ominaisuuksia kehitellessä on myös pidettävä huolta, ettei pelin varsinainen idea, diabetes, häviä kokonaan.

Uusi kohdealusta avaa mahdollisuudet toisenlaiseen peliin. Pelattavalle hahmolle keksittiin jonkinlainen taustatarina jo opinnäytetyön aikana, mutta alun perin suunnitellun pelin pääpaino oli pelimekaniikassa. Uudessa versiossa on tilaa paremmin kehitetylle tarinalle ja usealle eri hahmolle. Peli keskittyy kuitenkin jatkossakin enemmän hauskuuteen kuin tarinaan. Uudet mahdollisuudet tuovat myös uusia haasteita. Yksinkertaiset piirrokset toimivat 2D-pelissä, mutta 3D-ympäristö vaatii taidokasta mallinnusta. Mobiilipelejä pelataan usein ilman ääniä, mutta tietokonepelissä musiikki on tärkeä tekijä. Hahmot ja musiikki voitaisiin ostaa ulkopuolisilta, mikä säästäisi aikaa ja takaisi paremman laadun. Investointi voisi kuitenkin olla liian suuri. Pienissä peliprojekteissa grafiikat ovat usein kallein osa. Järkevintä olisikin ehkä käyttää ilmaisia valmiita grafiikoita tilapäisesti, jotta peliä voitaisiin esitellä ja kenties saada sille rahoitusta esimerkiksi Diabetesliitolta. Hahmojen ja musiikin valinta ovat kuitenkin vielä matkan päässä. Ensimmäinen

vaihe on laajentaa peli-ideaa ja suunnitella pelin kokonaisuus. Tasojärjestelmä ja ruokien poimiminen ovat lopullisessa muodossa mukana. Itse tasoihin voidaan nyt lisätä esimerkiksi hyppimis- ja kiipeilyosuuksia, joiden toteuttaminen mobiiliversiossa olisi tuntunut pakotetulta.

LÄHTEET

DIABETESTIETOA 2016. [Verkkojulkaisu]. Tampere: Diabeteskeskus. [Viitattu 2016-6-25.] Saatavissa: http://www.diabetes.fi/diabetestietoa

GEIG, Mike 2013-10-07. Models and Materials [luennon nauhoitus]. Saatavissa: https://unity3d.com/learn/tutorials/topics/graphics/models-and-materials

MICROSOFT 2017. What Is a Sprite? [Viitattu 2017-03-27.] Saatavissa: https://msdn.microsoft.com/en-us/library/bb203919.aspx

UNITY DOCUMENTATION a 2017-03-08. Rigidbody. [Viitattu 2017-03-22.] Saatavissa: https://docs.unity3d.com/Manual/class-Rigidbody.html

UNITY DOCUMENTATION b 2017-03-08. Animation Parameters. [Viitattu 2017-03-22.] Saatavissa: https://docs.unity3d.com/Manual/AnimationParameters.html

WILSON, Greg 2006-02-03. Off With Their HUDs!: Rethinking the Heads-Up Display in Console Game Design. [Viitattu 2017-03-22.] Saatavissa: http://www.gamasutra.com/view/feature/2538/off_with_their_huds_rethinking_.php