OULUN AMMATTIKORKEAKOULU

Teemu Tuhkanen

C#-OHJELMOINTIKIELI UNITY-PELIMOOTTORISSA JA OHJELMAVAATIMUSTEN TESTAUKSEN SEURANTA OHJELMALLISESTI

C#-OHJELMOINTIKIELI UNITY-PELIMOOTTORISSA JA OHJELMAVAATIMUSTEN TESTAUKSEN SEURANTA OHJELMALLISESTI

Teemu Tuhkanen
Opinnäytetyö
Kevät 2017
Tieto- ja viestintätekniikan
koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tieto- ja viestintätekniikan koulutusohjelma, Ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Teemu Tuhkanen
Opinnäytetyön nimi: C#-ohjelmointikieli Unity-pelimoottorissa ja ohjelmavaatimusten testauksen seuranta ohjelmallisesti
Työn ohjaaja: Veijo Väisänen
Työn valmistumislukukausi ja -vuosi: Kevät 2017 Sivumäärä: 66

Opinnäytetyö tehtiin kahdessa osassa osaopinnäytetyömallin mukaan. Opinnäytetyön ensimmäinen osa käsitteli peliohjelmointia C#-ohjelmointikielellä Unity-pelimoottorille. Tässä osassa käytiin läpi Unity-pelimoottorin perusteita, kuten Unityn omia funktioita, luokkia ja komponentteja. Aluksi nämä osa-alueet esitellään erikseen ja lopuksi nämä laitetaan käytäntöön pienen ja yksinkertaisen pelin muodossa.

Opinnäytetyön toinen osa tuli opinnäytetyön tilaajalta, Nokia Solutions and Networks Oy:lta. Tarkoituksena oli luoda mahdollisimman pitkälle automaattinen järjestelmä, jolla opinnäytetilaaja voisi seurata ohjelmistoille asetettujen vaatimusten täyttymistä. Tällä tavoin voidaan pitää huolen siitä, että jokainen vaatimus tulee testattua. Opinnäytetyönä tehty ohjelmistoa varten ohjelmistotestit tulivat merkitä vaatimuksille, joita testi testasi. Testien ajon jälkeen ohjelmisto käy läpi testilokit, joista se kerää yhteenvedon tekstimuotoisena tulosteena ja luo dynaamisesti verkkosivun.

Opinnäytetyön molemmat osa-alueet toivat hyvän mahdollisuuden tutustua erilaisiin puoliin ohjelmistokehityksessä. Tämä mahdollisti saamaan arvokasta kokemusta ohjelmoinnista, josta on varmasti hyötyä tulevaisuudessa

---

Asiasanat: ohjelmistotestaus, refaktorointi, peliohjelmointi, ohjelmistosuunnittelu

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Software Development

---

Author(s): Teemu Tuhkanen
Title of thesis: C#-programming language in Unity Game Engine and monitoring unit tests of software requirements programmatically.
Supervisor(s): Veijo Väisänen
Term and year when the thesis was submitted: Spring 2017 Number of pages: 66

This thesis was done in three parts. Thesis' first part covered game programming using C#-programming language in Unity game engine. This part went through the basics of Unity game engine, its own functions, classes and components. At first these sections were introduced separately and finally these are put in use in form of a small and simple game.

Topic of the second part of the thesis came from the subscriber, Nokia Solutions and Networks Oy. This thesis' purpose was to make a system that can be used to follow fulfillment of software requirements. Entire system was made to be as automatic as possible. All unit tests made for this system had to be marked for requirements, that unit test tested. After running these tests, the software goes through all test logs, where it collects a summary in a text form and creates dynamically a website.

Both parts of this thesis gave a good opportunity to explore different sides of software development. This provided to get valuable experience of programming, which will be beneficial.

Keywords: unit testing, refactoring, game programming, software design

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö on tehty kahdessa osassa. Tämän työn eri osat ovat valmistuneet vuosina 2016–2017. Työn osat käsittelevät ohjelmointia erilaisista näkökulmista ja eri lähtökohdista.

Ensimmäisessä osassa keskitytään Unity-pelimoottoriin ja C#-ohjelmointikieleen. Työ on pääosin tutkimustyötä ja tutustumista aiheeseen. Ennen opinnäytetyötä minulla oli kokemusta C#-ohjelmointikielestä, mutta hyvin vähän tietämystä miten Unityssa sitä hyödynnetään.

Toisessa osassa pääsin itse suunnittelemaan ja kehittämään toimivan tuotteen opinnäytetyön tilaajan käyttöön. Kyseessä oli työkalu ohjelmistotestaukseen, jonka tarkoitus olisi helpottaa opinnäytetilaajaa olemaan selvillä, mitä ohjelmistovaatimuksia on testattu ja kuinka monta kertaa. Opinnäytetyön kirjallisessa osiossa keskitytään kertomaan, kuinka projekti luotiin, tuoden esille koko ohjelmiston osa-alueet.

1 ENSIMMÄISEN OSAN ESITTELY

Ensimmäisessä osassa tutustutaan C#-ohjelmointikielen käyttöön Unity-pelimoottorissa. C#-kieli oli itselleni hyvä valinta, koska sitä kieltä käytetään myös muualla kuin peliohjelmoinnissa ja on syntaksiltaan samankaltainen Java-kielen kanssa.

Tämän osan tarkoituksena oli olla pieni pintaraapaisu Unity-pelimoottorille ohjelmoinnista. Tässä osassa käydään läpi osa Unityn peruskomponenteista ja miten niitä voidaan käyttää.

3 TOISEN OSAN ESITTELY

Opinnäytetyön toisen osan aihe tuli tilaajalta, joka tarvitsi ohjelmistotestaukseen työkalua. Opinnäytetyön tilaajalla oli tarve saada ohjelmistovaatimukset ja niitä varten tehdyt ohjelmistotestit kartoitettua. Oli tärkeää saada tietoa siitä, kuinka hyvin vaatimus on testattu.

Tilaajalla ei ollut varsinaista suunnitelmaa, miten vaatimukset ja testit voitaisiin nivoa yhteen, joten tämän suunnittelu jäi pitkälti minun tehtäväksi. Tämän ohjelmistokokonaisuuden sain suunniteltua ja luotua kokonaisuudessaan työrupeamani aikana. Ohjelma suunniteltiin alusta alkaen sillaiseksi, että ei tarvinnut lisätä työkalua käyttävien työmäärää liikaa. Työkalu toimii hyvin pitkälti automaattisesti. Kun Jenkins-ympäristössä testit on ajettu läpi, saadaan suoraan verkkosivuille näkymä, mikä on vaatimusten tilanne.

4 YHTEENVETO

Opinnäytetyö oli suuruudeltaan yhteensä 15 opintopistettä, joista 5 oli ensimmäisen osan osuus, ja toinen osa oli 10 opintopisteen suuruinen. Ohjelmointi oli opinnäytetyön kantava teema, vaikka aiheet olivat näennäisesti kaukana toisistaan. Ensimmäisessä osassa päästiin ottamaan pintaraapaisu ohjelmointiin tutustumalla aiheeseen, kun taas toisessa osassa tehtiin itsenäisesti kokonainen ja toimiva ohjelmistokokonaisuus.

Opinnäytetyön aiheet olivat erilaisia, mutta ensimmäisen osan opit ja taidot hyödyttivät myös toisen osan tekemistä. Molemmat aiheet antoivat mukavasti tietoa aiheista, joihin ei välttämättä olisi muutoin tullut tutustuttua.

LIITE 1

OULUN AMMATTIKORKEAKOULU

Teemu Tuhkanen

C#-OHJELMOINTIKIELI UNITY-PELIMOOTTORISSA

LIITE 1

C#-OHJELMOINTIKIELI UNITY-PELIMOOTTORISSA

Teemu Tuhkanen
Opinnäytetyö, osa 1
Kevät 2016
Tieto- ja viestintätekniikan
koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

1 JOHDANTO

Jo pidemmän aikaa peliteollisuudessa on enimmäkseen käytetty valmiita pelimoottoreita, jotta pelintekijöillä säästyisi aikaa ja rahaa. Pelimoottorin käyttäminen mahdollistaa myös hienompien pelien toteuttamisen vähemmällä osaamisella, kun peliä tehdessä ei välttämättä tarvitse kiinnittää huomiota monimutkaisempiin osiin, kuten valaistuksen luomiseen.

Yksi suosituimmista pelimoottoreista on tällä hetkellä Unity. Unity tarjoaa 3d-pelikehityksen lisäksi myös mahdollisuuden kehittää pelejä kaksiulotteisena. Pelimoottori tarjoaa mahdollisuuden julkaista pelit usealle eri alustalle, kuten PC:lle, Macille, Linuxille ja Playstation- ja Xbox-pelikonsoleille. Suurin osa peleistä kehitetään Googlen Android-järjestelmälle ja Applen IOS-järjestelmää käyttäville iPadille ja iPhonelle.

Unity on tällä hetkellä ladattavissa Windows- ja Mac OS X-tietokoneille (1.). Linuxille on myös ladattavissa koeversio Unitysta (2.).

Unityssa ohjelmointiin vaihtoehtoina ovat JavaScript- ja C#-ohjelmointikielet. Tässä opinnäytetyössä keskityn nimenomaan C#-kieleen ja sen hyödyntäminen Unityssa.

C# (C Sharp) on Microsoft-yhtiön kehittämä, vuonna 2000 julkistettu oliopohjainen ohjelmointikieli. Kielen tarkoitus on olla tehokas kuin C++-kieli, mutta helppokäyttöinen kuin Java.

Opinnäytetyön tarkoituksena on keskittyä Unityn C#-ohjelmointiin ja Unity-ohjelmakirjaston sisältämiin metodeihin esimerkkejä käyttäen. Työssä tehtiin myös yksinkertainen peli ja sen skriptejä käytetään esimerkkinä.

2 C#-OHJELMOINTIKIELI UNITY-YMPÄRISTÖSSÄ

Unity on C++:lla tehty pelimoottori, johon voi kirjoittaa skriptejä C#-kielellä. Skripti käännetään ajonaikaisesti (Just-In-Time compilation eli JIT), paitsi IOSin kohdalla. IOSilla skripti käännetään etukäteen Ahead-of-Time (AOT)-käännökseksi (4.).

2.1 C#-Skriptit

C#-skriptejä voi kirjoittaa Unityn MonoDevelop-ohjelmalla, tai Microsoftin Visual Studiolla. Visual Studio tarvitsee Unitylle ohjelmointia varten UnityVS-laajennoksen, koska ohjelmat kirjoitetaan Unityn sisäisessä virtuaaliympäristössä (4.).

Visual Studiolla kirjoitetut skriptit voidaan kääntää suoraan Visual Studiossa, mutta silti Unity kääntää peliin tuodut skripit omassa kääntäjässään. Unity Technologies kehottaakin tarkastelemaan käännöstä myös Unityn omassa konsolissa. Visual Studion C#-kääntäjässä on enemmän ominaisuuksia ja se tukee uudempia C#:n ominaisuuksia kuin Unityn kääntäjä. Tämän takia uusien C#-ominaisuuksien kohdalla skripti voi kääntyä Visual Studiossa, mutta ei Unityssa (4.).

2.2 Muuttujat ja tietotyypit

Kokemukseni perusteella Unityssa eniten käytettyjä muuttujia ovat int, bool ja float. Floatia suositellaan käytettäväksi Intin sijaan sen tarkkuuden vuoksi. Int-arvoja käytetään yksinkertaisten lukujen esittämiseen, jolloin tarkkuudella ei ole suurta merkitystä. Esimerkkinä pelihahmon energian esitys on tarpeen esittää vain kokonaislukujen 0–100 välillä.

2.2.1 Oliot

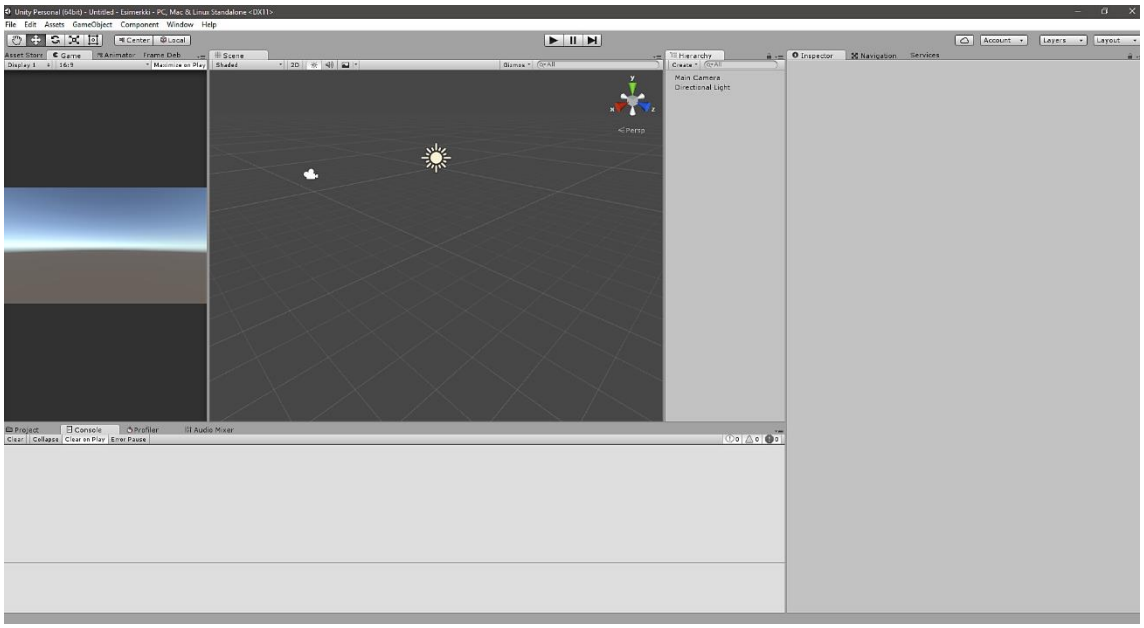
Koska C# on olio-ohjelmointikieli (OOP, Object-Oriented Programming) oliot ovat käytössä myös Unitylle ohjelmoinnissa. Oliot ovat kehitetty helpottamaan ja yksinkertaistamaan ohjelmointia.

2.2.2 Funktiot eli metodit

Olio-ohjelmointikielissä luokan sisäisiä funktioita kutsutaan metodeiksi, vaikka ne tarkoittavat melkein samaa asiaa. Metodit ovat aliohjelmia, joita pääohjelma tai toinen aliohjelma voi kutsua. Tätä ominaisuutta tarvitaan joko siihen, että ohjelmasta löytyy useampi samanlaista toimintoa tarvitsevaa kohtaa, tai yksinkertaisesti myös selkeyttämään ohjelman lukemista. (3, s. 20.)

3 C#-OHJELMOINTI UNITYSSA

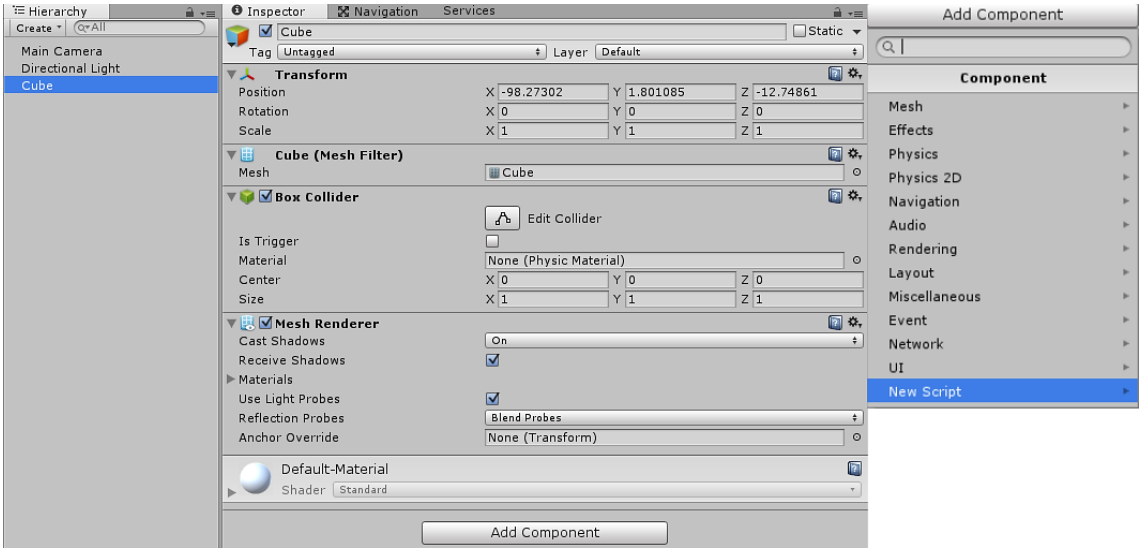
Unityssa käytettävät kielet ovat C# ja JavaScript. Unityn JavaScript ei oikeastaan ole puhdas JavaScript, vaan Unitya varten tehty versio. C# on puhtaasti sama kieli, jota käytetään laajemmalti muualla tietotekniikan sovelluksissa. Tämän takia ongelmatilanteissa C#-kieleen on helpompi löytää internetistä tai kirjallisuudesta apua. Lisäksi C#-ohjelmistokirjastot ovat laajat, joten niidenkin hyödyntäminen on mahdollista (3, s. 10.).



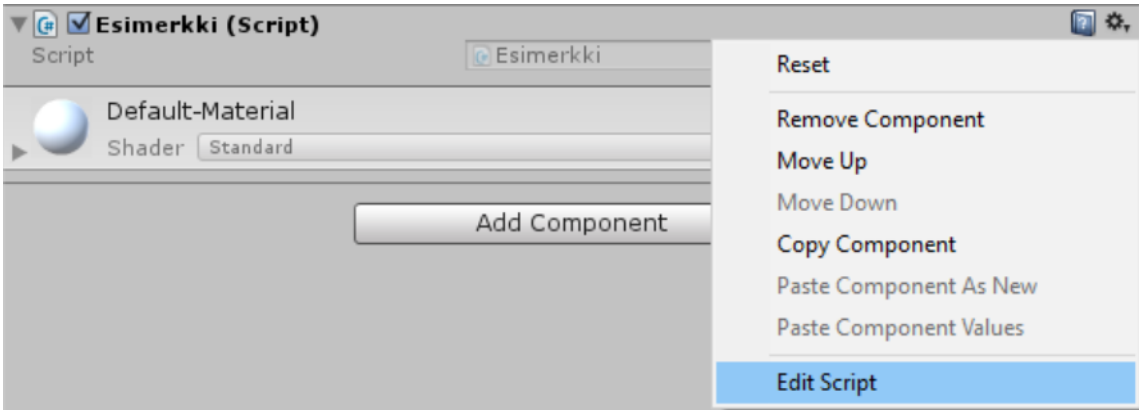
KUVA 1. Unityn perusnäkymä

3.1 Skriptin luominen

Unityssa skripti luodaan esimerkiksi klikkaamalla Hierarchy-välilehdessä olevaa peliobjektia. Inspector-välilehdessä Add Component-painiketta painamalla valitaan New Script ja nimetään se skriptin toimintoa parhaiten kuvaavalla nimellä.



KUVA 2. Skriptin luontia

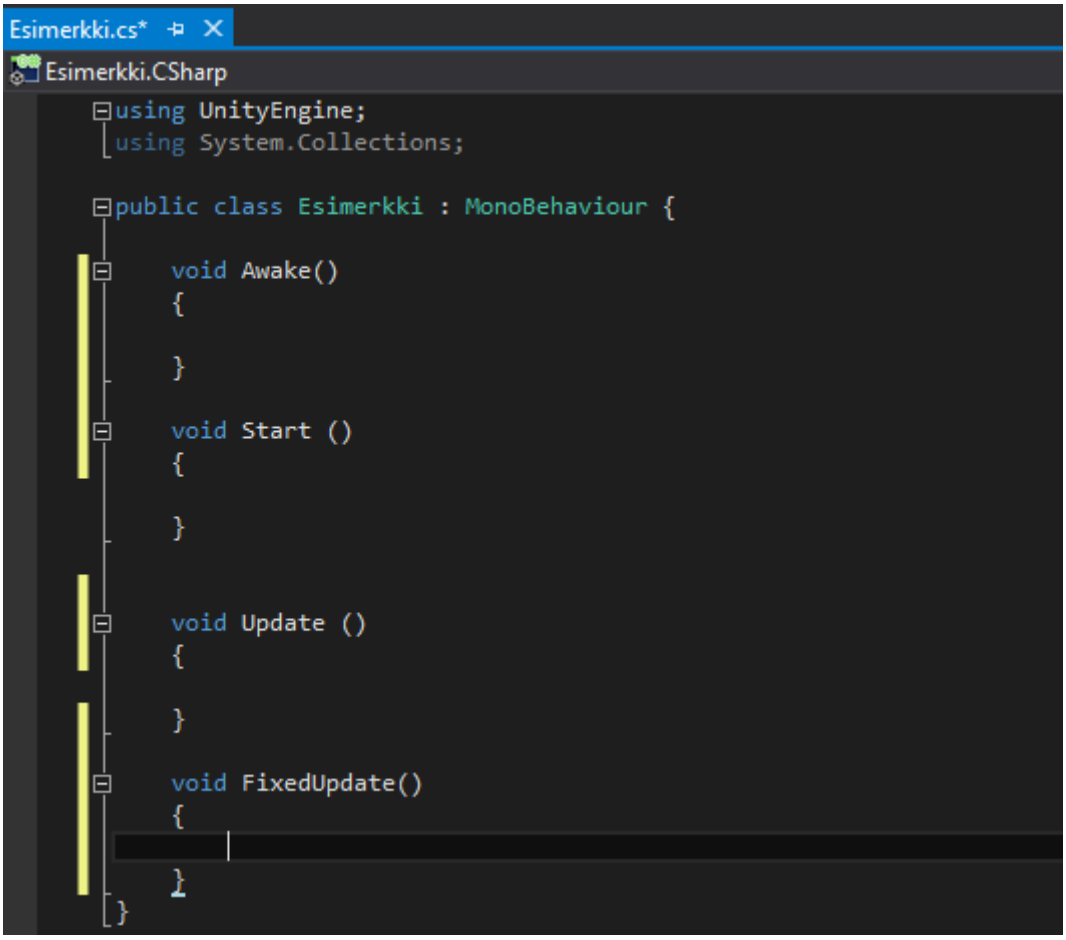


KUVA 3. Skriptin luontia

Script-komponentin oikean yläkulman hammasratasta painamalla avautuu valikko, josta valitaan Edit Script. Tämän jälkeen skripti avautuu editorissa.

3.2 Alustus ja ajaminen

Unityn skripteissä on omia metodeja, jotka kuuluvat olennaisesti ohjelman suoritukseen.



KUVA 4. Skriptin alustamiseen ja ajamiseen käytettävät metodit

3.2.1 Awake ja Start

Awake- ja Start-metodeja käytetään skriptin alustukseen. Awake-metodi kutsutaan ensin, jonka jälkeen kutsutaan Start-metodia. Awake-metodi ajetaan aina ohjelman käynnistyessä. Awake-metodia on hyvä käyttää, kun määritellään Esim. peliobjektien väliset yhteydet, koska se ajetaan aina ohjelman alussa, eikä vasta tarvittaessa toisin kuin Start-metodi. Start-metodi kutsutaan vasta, kun skripti on käytössä. Huomioitavaa on se, että näitä metodeja voi ajaa ohjelman aikana vain kerran (7.).

3.2.2 Update ja FixedUpdate

Update ja FixedUpdate ovat pelisilmukoita (Game loop). Update-silmukka päivitetään jokaisen ruudunpäivityksen yhteydessä, eli päivitystiheys voi vaihdella tietokoneen suorituskyvyn mukaan. FixedUpdate-silmukka päivittyy aina ajan mukaan ja on johdonmukainen, täten sitä suositellaan käytettäväksi rigidbodyn sisältävissä objekteissa (8).

3.3 Luokat ja tietueet Unityssa

3.3.1 GameObject

GameObject, eli peliobjekti-luokka on perusluokka kaikille Unityn sisäisille luokille. Peliobjekteilla on omia metodeita, jotka ovat käytössä kaikille sen perivillä luokilla (9.).

GetComponent-metodia käytetään, kun halutaan käyttää peliobjektin komponenttia (9.).

Tagit ovat hyödyllisiä, kun halutaan merkitä peliobjekti. Esim. Pelattava hahmo voidaan merkitä ”Player”-tagilla. Tagit voi myös itse nimetä haluamikseen, mutta Player-tag on valmiina valittavana (9.).

FindGameObjectWithTag-metodia käytetään, kun halutaan osoittaa skriptissä jokin peliobjekti tietyksi objektiksi, kuten pelaajaksi. Esim. Tällöin käytetään player = GameObject.FindGameObjectWithTag(”Player”) -metodia Awake-metodissa (9.).

3.3.2 Vector

Vektorit ovat suureita, joilla on suunta. Vektoreita käytetään Unityssa peliobjektien liikuttamiseen lähtöpaikasta haluttuun sijaintiin. Kaikki mitä pelialueelle mallinnetaan, ovat koordinaatistossa, jotka ovat x leveysakseli, y korkeusakseli ja z syvyysakseli. Piste 0,0,0 on origossa, eli koko pelialueen keskipisteessä (10.).

Vector2:ta käytetään esittämään kaksiulotteisia vektoreita ja sijainteja. Vector2 luo vektorin float-tyyppisistä x- ja y-komponenteista (11.).

Vector3:ta käytetään esittämään kolmiulotteisia vektoreita ja sijainteja (X, Y ja Z). Vector3 luo vektorin float-tyyppisistä x-, y- ja z-komponenteista (12.).

3.3.3 Transform

Jokaisella peliobjektilla on Transform-luokka. Sitä käytetään säilyttämään ja muokkaamaan peliobjektin sijaintia (Esim. this.transform.position) ja rotaatiota (Esim. this.transform.rotation) (13.).

3.3.4 Rigidbody

RigidBody, eli jäykkävartalo lisätään, jos halutaan laittaa peliobjektin liikkeet Unityn fysiikkamoottorin hallittavaksi. Rigidbodyn lisättyä, alkaa objektiin vaikuttaa painovoima ja se reagoi törmätessään toiseen objektiin (14.).

3.3.5 Collider

Collider eli vapaasti suomeksi käännettynä törmääjä on peleissä tapa estää ja rajoittaa peliobjektien liikkumista toisten objektien läpi. Se on perusluokka (Eng. Base Class) kaikille Collider-luokille. Kaikki colliderit, kuten esimerkkinä box, capsule, tai circle collider-luokat perivät tietyt ominaisuutensa collider-luokasta. 2D-, ja 3D-collidereita käytetään riippuen siitä, että tehdäänkö peli kolmi-, vai kaksiulotteisena (15.) (16.).

Collider voi toimia myös triggerinä eli laukaisijana (15.). Tämä tarkoittaa esimerkiksi, kun pelaaja kävelee triggerin yli, niin tämä laukaisee pelitason läpäisyn

4 PALASET YHTEEN

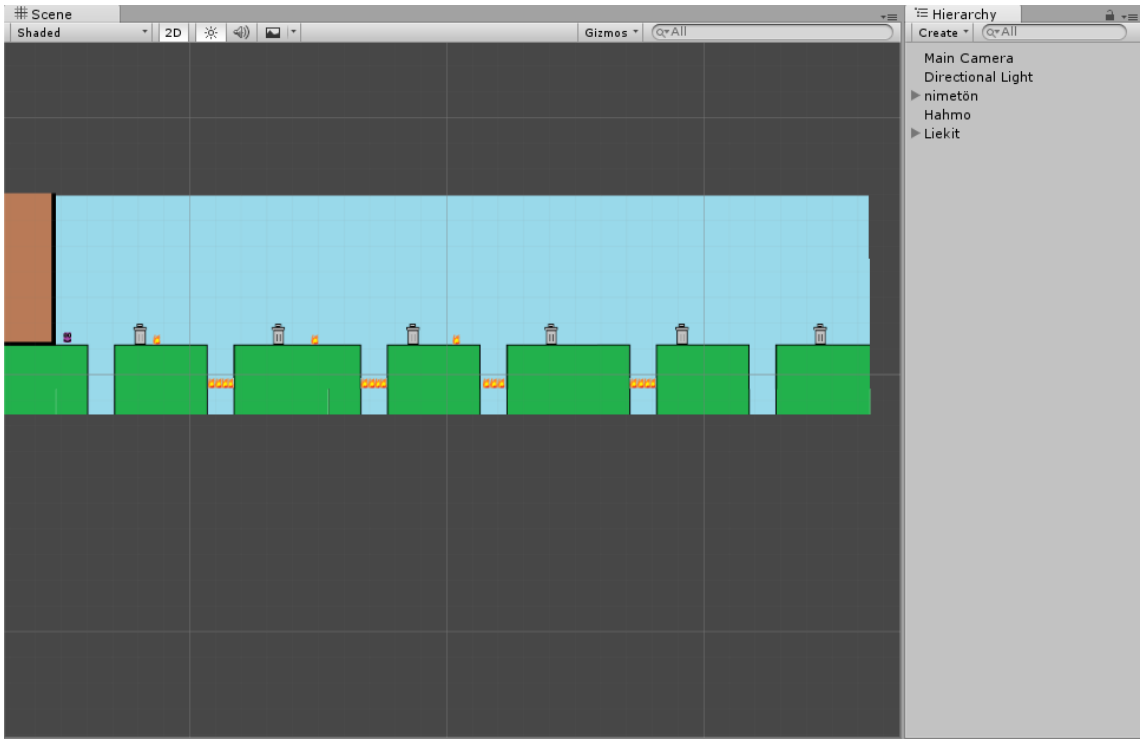
Tein yksinkertaisen pelin, jossa keskityttiin ohjelmointiin, joten muut kuin ohjelmointiin liittyvät osat oli jätetty vähemmälle huomiolle. Tarkoitus oli soveltaa oppimaansa ja avata Unitylle ohjelmointia. Peli on PC:lle tarkoitettu kaksiulotteinen tasohyppelypeli, jossa väistellään liekkejä hyppelemällä niiden yli.

4.1 Peliympäristön ja hahmojen luominen

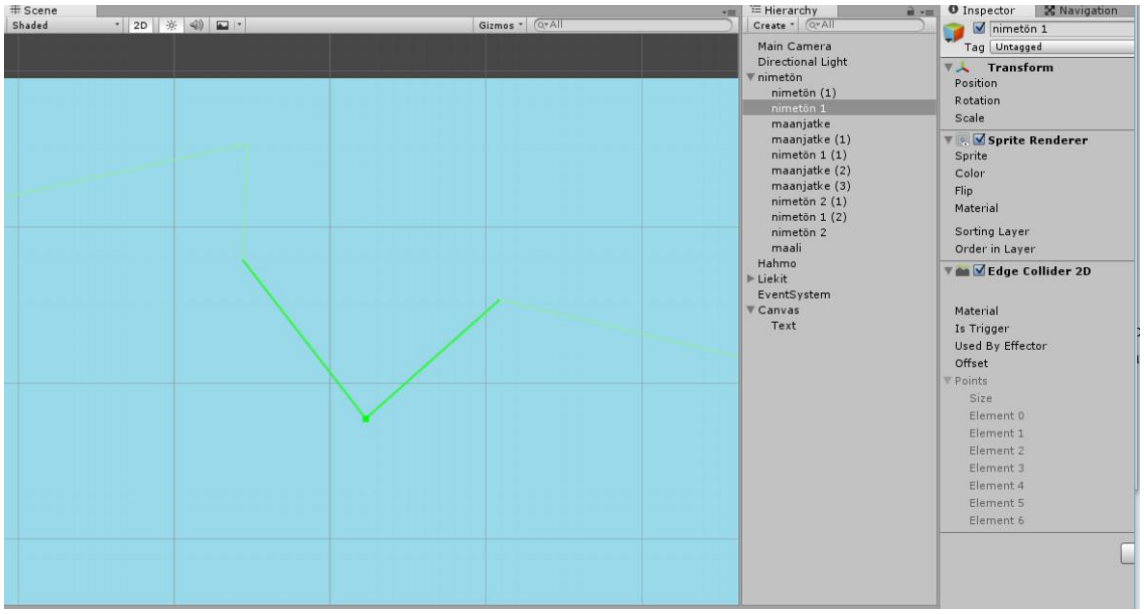
Peliympäristö luodaan Sceneen, eli peliobjektit ja komponentit sisältävään alueeseen. Pelissä käytetään png-tiedostoina kuvia, jotka muutetaan editorissa tekstuurista spriteksi, jotta niitä voidaan käyttää peliobjekteina. Peliobjekteja ovat kameran lisäksi: tausta, pelihahmot ja liekit.

Hahmoille lisätään Rigidbody 2D- ja Box Collider 2D-komponentit. Pelin taustaan lisätään Edge Collider-komponentti (Kuva 6.). Tämän käyttäminen mahdollistaa pelin kävelypintojen ja pelin rajojen määrittämisen vapaalla kädellä. Edge Colliderilla voi kattaa tässä tapauksessa koko pelialueen vain yhtä törmääjää hyödyntäen.

Pelihahmolle lisätään inspector-välilehdeltä löytyvältä tag-valikosta ”Player”-tag, jotta myöhemmin skripteissä käytettävä GetComponent.FindGameObjectWithTag(”Player”) toimisi.



KUVA 5. Peliympäristön luontia



KUVA 6. Edge Colliderin muokkausta.

4.2 Pelihahmon liikuttaminen

Pelihahmon on tarkoitus liikkua pelialueella sivuttaisessa suunnassa ja hyppimään pystysuunnassa. Pelihahmo kiihdyttää vauhtiaan sivusuunnassa niin kauan, kunnes tulee vastaan jokin este. Pelihahmo pystyy hypähtämään ilmassa hidastaakseen putoamistaan. Pelin painovoima kuitenkin aiheuttaa sen, että pelihahmo ei voi estää putoamistaan kokonaan.

4.2.1 Alustus

Alustuksessa määritetään pelihahmon liikuttamiseen tarvittavat muuttujat, luokat ja tietueet. Unityssa julkisiksi (public) määritetyt muuttujat voidaan muokata editorissa, joten sopivien arvojen laittaminen onnistuu ilman skripti-tiedoston avaamista.

```
public float speed = 0.1f;
public float jumpforce = 100f;
public float jumpTime = 1f;
private float timer;
public bool isDead = false;
private Rigidbody2D playerRig;
private Vector2 movement;
private Vector2 jumping;
private BoxCollider2D box;


void Awake () {
    playerRig = GetComponent<Rigidbody2D>();
    box = GetComponent<BoxCollider2D>();
}

void Start()
{
    timer = jumpTime;
}
```

KUVA 7. Skriptin alustaminen pelihahmon liikuttamiselle

4.2.2 Liikuttaminen

FixedUpdate-metodissa jokaisen päivityksen yhteydessä ohjelma tarkistaa, onko boolean isDead tosi, vai epätosi. Jos isDead on epätosi, niin pelihahmo voi liikkua ja hyppiä. Mutta jos isDead on tosi, niin ohjelma toteuttaa pelihahmon

kuolemiseen liittyvät toimenpiteet. Pelihahmon kuoleman kohdatessa pelihahmo-objekti kutsuu kahdesti jump-metodia, jotta hahmo hypähtää ilmaan ja tämän jälkeen muuttaa colliderin triggeriksi, jotta objekti voi pudota ulos pelialueelta.

```
void Update(){
    timer += Time.deltaTime;
}

void FixedUpdate () {
    if (!isDead)
    {
        float x = Input.GetAxisRaw("Horizontal");
        move(x);

        if (Input.GetKeyDown("w") && timer >= jumpTime)
        {
            jump();
        }
    }

    if (isDead)
    {
        jump();
        jump();
        box.isTrigger = true;
        GameOverText.text = "GAME OVER";
    }
}

void move(float x)
{
    movement.Set(x, 0);
    movement = movement.normalized;
    playerRig.AddForce(movement * speed / Time.deltaTime);
}

void jump()
{
    if (!box.isTrigger)
    {
        timer = 0;
        jumping.Set(0, jumpforce);
        playerRig.AddForce(jumping);
    }
}
```

KUVA 8. Skripti pelihahmon liikuttamiselle

Float x = Input.GetAxisRaw(”Horizontal”) toimii siten, että Unityn editorissa määritetyt Horizontal-napit, joko nostattavat x:n arvoa tai sitten laskevat sitä. Esim. nuolinäppäimien oikeanuoli nostaa, kun taas vasennuoli laskee x:n arvoa.

Muuttujan x:n arvo viedään move-metodiin, jossa se määritetään movement-nimisen Vector2:n x-arvoksi.

Hyppäämiseen käytetyssä jump-metodissa käytetään vastaavasti jumpforcen arvoa jumping-nimiseen Vector2:een, jotta saadaan voimaa y-akselille. Hahmon RigidBodyn sisältämä painovoima hoitaa pelihahmon tipahtamisen. Samalla pyörivä timer-ajastin laskee aikaa edellisestä hypystä ja jumpTime määrittää, milloin hyppy onnistuu seuraavan kerran.

4.3 HUD

HUD, eli head-up-display tarvitaan peliin näyttämään pelaajalle ”GAME OVER”-teksti. Peliin lisätään Canvas-objekti, jonka sisälle lisätään vielä Text-objekti.

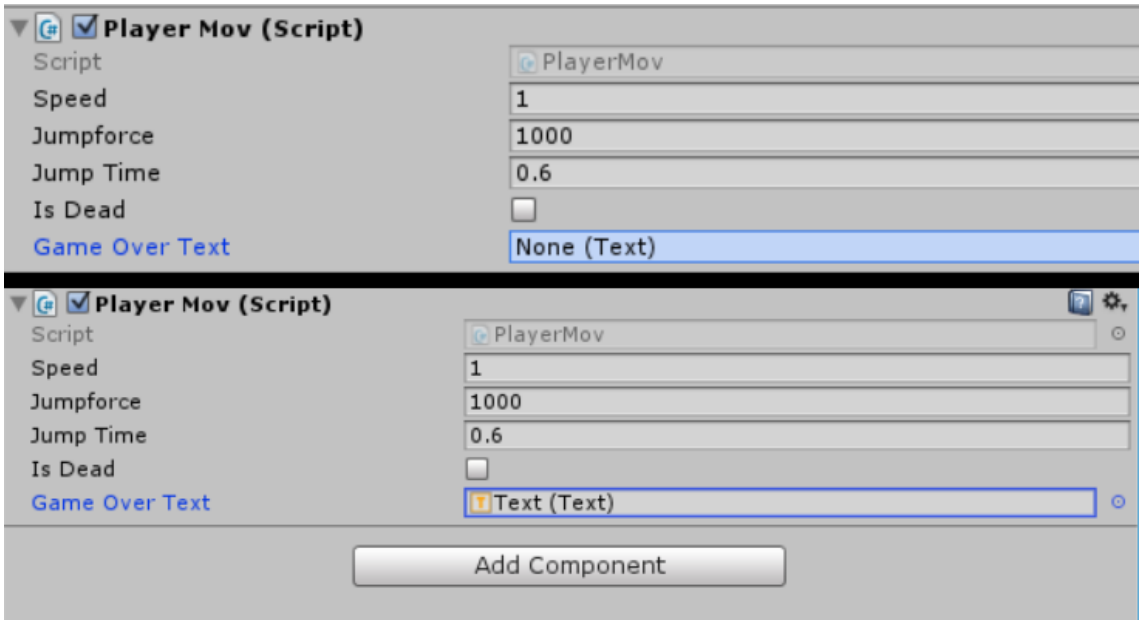
Jotta nämä toimisivat itse pelissä, on lisättävä Text-objektia käyttävään ”PlayerMov”-skriptiin using unityEngine.UI-tieto ja esiteltävä Public Text GameOverText (Kuva 9.). Tämän jälkeen Unityn editorissa raahataan hiirellä Text-objekti pelihahmon PlayerMov-skripti Game Over Text-kohdan päälle. Näin Unity luo näiden väliset yhteydet (Kuva 10.).

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class PlayerMov : MonoBehaviour {

    public Text GameOverText;
```

KUVA 9. Lisättävät tiedot skriptiin



KUVA 10. Objektien välisten yhteyksien luominen.

4.4 Pelikameran skripti

Pelissä käytettävän kameran (Camera) tehtävänä on määrittää, mikä osa peli-alueesta näytetään.

```
public float smoothing = 4;
Vector3 offset;
PlayerMov playmov;
GameObject player;


void Awake()
{
    player = GameObject.FindGameObjectWithTag("Player");
    playmov = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMov>();
}
void Start()
{
    offset = transform.position - player.transform.position;
}

void FixedUpdate()
{
    if (!playmov.isDead)
    {
        Vector3 targetCamPos = player.transform.position + offset;
        transform.position = Vector3.Lerp(transform.position, targetCamPos, smoothing * Time.deltaTime);
    }
}
```

KUVA 11. Kameralle skripti

Ohjelma tarkistaa päivityksen yhteydessä, onko isDead-arvo epätosi. Jos se on epätosi, kamera seuraa pelihahmoa smoothing-arvon mukaisella vauhdilla.

4.5 Pelihahmon vahingoittuminen ja kuoleminen

Kun pelihahmon etäisyys liekistä on pienempi tai yhtä suuri kuin 0,5, pelihahmon isDead-arvo muuttuu todeksi, joka tarkoittaa pelihahmon kuolemista.

Pelihahmon kuollessa hahmo hypähtää ilmaan ja tipahtaa kentästä ulos. Samalla näyttöön tulee teksti ”GAME OVER” ilmoittamaan pelin päättymisestä.

```
GameObject player;
PlayerMov playmov;

void Awake()
{
    player = GameObject.FindGameObjectWithTag("Player");
    playmov = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMov>();

}

void Update()
{
    if (Vector2.Distance(this.transform.position, player.transform.position) <= 0.5f)
    {
        playmov.isDead = true;
    }
}
```

KUVA 12. Pelihahmo ottaa vahinkoa ja kuolee

```
    if (isDead)
    {
        jump();
        jump();
        box.isTrigger = true;
        GameOverText.text = "GAME OVER";
    }
}
```

KUVA 13. Pelihahmon kuolema aiheuttaa muutamia jälkiseuraamuksia

5 POHDINTA

Tässä opinnäytetyössä tutustuin C#-ohjelmointikieleen Unity-pelimoottorin työkaluna. Tavoitteena oli oppia ymmärtämään skriptaustyyppistä ohjelmointia ja käyttämään Unitya. Opinnäytetyötä varten tehtyä esimerkkipeliä oli yksinkertainen ohjelmoida, kun oli tutustunut Unityyn ja C#-kieleen valmiiksi. Vaikkei peli ollut opinnäytetyön tarkoitus, niin se syntyi ajatuksesta, että jos pelimoottorille ohjelmoimisesta kirjoittaa, niin täytyy olla myös tehtynä peli.

C# on helposti omaksuttavissa oleva kieli varsinkin, jos omaa ohjelmointitaustaa muista ohjelmointikielistä. Mielestäni C-kieleen viittaavasta nimestään huolimatta siinä on enemmän yhtäläisyyksiä Java-kielen, kuin C:n tai C++-kielen kanssa.

C# ja Unity luovat hyvän yhdistelmän pelinkehittämistä varten. Unitylla pelejä tekemällä pääsee mielekkäällä tavalla harjoittelemaan ohjelmointia. Ohjelmointitaidot karttuvat, mutta myös antaa harhakuvan ohjelmoinnin helppoudesta, koska teknisiin seikkoihin ei tarvitse kovin paljoa kiinnittää huomiota.

LÄHTEET

1. System Requirements. Unity Technologies. Hakupäivä 14.4.2016. Saatavissa: http://unity3d.com/unity/system-requirements

2. Bard, Na'tosha 2015. UNITY COMES TO LINUX: EXPERIMENTAL BUILD NOW AVAILABLE. Unity Technologies. Hakupäivä 14.4.2016 Saatavissa: http://blogs.unity3d.com/2015/08/26/unity-comes-to-linux-experimental-build-now-available/

3. Norton, Terry 2013. Learning C# by Developing Games with Unity 3D Beginner's Guide. Packt Publishing Ltd. Hakupäivä 04.2.2016. Saatavissa: http://site.ebrary.com/lib/oamk/reader.action?docID=10772045

4. Tuliper, Adam 2014. Unity: Developing Your First Game with Unity and C#. MSDN Magazine Blog. Hakupäivä 8.4.2016. Saatavissa: https://msdn.microsoft.com/en-us/magazine/dn759441.aspx

5. Unity Manual. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/Manual/VisualStudioIntegration.html

6. 2016. Unity Manual. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/Manual/index.html

7. Unity Tutorials. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: https://unity3d.com/learn/tutorials/modules/beginner/scripting/awake-and-start

8. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: https://unity3d.com/learn/tutorials/modules/beginner/scripting/update-and-fixedupdate

9. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/GameObject.html

10. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://unity3d.com/learn/tutorials/modules/beginner/scripting/vector-maths-dot-cross-products

11. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/Vector2.html

12. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/Vector3.html

13. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/Transform.html

14. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/Rigidbody.html

15. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/Collider.html

16. Unity scripting reference. Unity Technologies. Hakupäivä 7.4.2016. Saatavissa: http://docs.unity3d.com/ScriptReference/Collider2D.html

Teemu Tuhkanen

OHJELMAVAATIMUSTEN TESTAUKSEN SEURANTA OHJELMALLISESTI

LIITE 2

OHJELMAVAATIMUSTEN TESTAUKSEN SEURANTA OHJELMALLISESTI

Teemu Tuhkanen
Opinnäytetyö, osa 2
Kevät 2017
Tieto- ja viestintätekniikan koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

SANASTO

GCC	GNU-projektin luoma kääntäjä. Tällä käännetään mm. C-kielistä ohjelmaa
Komentorivi, command line interface, CLI	Ihmisen ja tietokoneen väliseen kommunikointiin käytettävä tapa. Tyypillisesti komentoriviltä ajetaan komentotulkkia, jolle annetaan käskyjä, jotka tulkki käsittelee.
Lista, List	Muuttuja, joka sisältää järjestyksessä listan alkioita. Listassa olevat alkiot voivat olla esimerkiksi olioita tai lukuja.
Luokka, Class	Olion tyyppi, joka sisältää joukon loogisesti yhteenkuuluvaa tietoa ja toiminnollisuutta.
Muodostin, Constructor	Oliota luotaessa automaattisesti suoritettu funktio.
Ohjelmistokehys, Framework	Ohjelmistotuote, joka muodostaa rungon sen päälle rakennettavalle tietokoneohjelmalle
Olio, Object	Luokan ilmentymä
Otsikkotiedosto, Header file	C- ja C++-kielissä ohjelmakoodiin sisällytettävä tiedosto, jonka tiedostomuoto on tavallisesti .h
Skripti, Script	Tulkattava tietokoneohjelma
Testitapaus, Test case	Yksittäinen ohjelmistotesti

1 JOHDANTO

Ohjelmistotestauksessa voidaan testata ohjelmistolle asetettuja vaatimuksia. Hankaluuksia voi ilmetä, kun pitäisi tietää, mitä vaatimusta on testattu ja minkälaisilla testeillä. Jos seurantaa ei tehdä, vaatimukset voidaan helposti joko testata puutteellisesti tai työmäärien kasvaessa jättää kokonaan testaamatta.

Tämän opinnäytetyön tarkoituksena oli saada tilaajalle (Nokia Solutions and Networks Oy) työkalu, jolla voidaan mahdollisimman automatisoidusti seurata ohjelmistovaatimusten täyttymistä. Työkalun tulee palvella niin suunnittelijoita kuin ohjelmoijia työssään.

Tämä työkalu koostuu pelkistettynä kolmesta osasta: ohjelmistotestien merkinnästä, testilokien läpikäymisestä ja siitä saadun tiedon näyttämisestä sekä verkkosivuille että puhtaassa tekstimuodossa. Ohjelmistotestien merkitsemistä lukuun ottamatta koko prosessi hoidetaan Jenkins-automatisointia hyödyntäen. Työkalun avulla saadaan mahdollisimman tarkka kuva, kuinka hyvin vaatimukset on täytetty.

2 OHJELMISTOTESTAUS

Ohjelmistotestauksen tavoitteena on paikallistaa ohjelmiston viat ja häiriöt. Ohjelmille ja funktioille tehdään testejä erilaisia testiohjelmistoja käyttäen. Testatessa ohjelman toimivuuden voi kuitenkin varmistaa ainoastaan tietynlaisissa ympäristöissä, eikä kaikkia mahdollisia tilanteita voida näin testata.

2.1 Yksikkötestaus

Yksikkötestit ovat pieniä testejä, joita ohjelmoijat ajavat varmistaakseen kirjoittamiensa ohjelmien toimivuuden. Yksikkötestaukset ovat tärkeitä työkaluja, varsinkin kun ne ovat automatisoituja ja ajetaan joka kerta muutoksien tai lisäyksien tullessa ohjelmaan. Tällä tavoin huomataan, rikkooko muutos olemassa olevia ohjelmia.

2.1.1 Unity ja Python unit testing framework

Unity on ohjelmistotestaukseen tarkoitettu täysin C-kielellä kirjoitettu ohjelmistokehys (engl. Framework). Sitä voidaan käyttää 8-bittisistä mikrokontrollereista aina 64-bittisiin prosessoreihin. (1.)

Python unit testing framework on Python-versio JUnitista. Se on ohjelmistokehys Pythonille ja kuuluu Pythonin standardikirjastoon. (2.)

2.2 Test-driven development eli TDD

Test-driven development eli suomeksi testivetoinen kehitys on ohjelmistotuotannon tapa, jossa ohjelmistotestit suoritetaan, ennen kuin ohjelmistoa on edes kirjoitettuna. Siitä luonnollisesti seuraa testin epäonnistuminen, jonka jälkeen ohjelma kirjoitetaan siten, että testistä saisi hyväksytyn tuloksen.

Tavallisesti testivetoisen kehityksen työvaiheet menevät näin:

1. Kirjoitetaan testi. Tuloksena epäonnistuminen.
2. Kirjoitetaan ohjelma, jolla saadaan testi läpi.
3. Siivotaan ohjelma kaikesta ylimääräisestä. Säilytetään vain tarvittavat osat.

Vaikka maallikon silmissä voi tuntua oudolta kirjoittaa testi ainoastaan sitä varten, että se ei tulisi menemään läpi, se tuo paljon hyötyjä kehittäjän näkökulmasta. Tällä tavoin ohjelmat saadaan selkeiksi, ne tekevät mitä niiden kuuluukin tehdä ja kehittäjät saavat paremman kuvan ohjelmastaan. Jos ohjelma tekee sen mitä pitää, eikä mitään muuta, ohjelma sisältää vähemmän koodia ja yksinkertaisemman rakenteen. (3, s.195—196.)

2.3 Vaatimusten seuraamisen tarve

Opinnäytetyön tilaajan täytyi noudattaa standardia, joka vaatii, että ohjelmistovaatimusten tulisi olla kartoitettavissa koko prosessin ajalta. Myös tieto siitä, montaako vaatimusta on testattu, oli oltava saatavilla. Tätä varten ei ollut työkalua, vaan vaihtoehtona oli vain manuaalisesti käydä läpi, miten vaatimukset oli testattu.

3 PYTHON JA JAVASCRIPT

Python ja JavaScript olivat tämän opinnäytetyön pääasialliset ohjelmointikielet. Molemmat kielet ovat skriptauskieliä (engl. scripting language) ja olio-ohjelmointikieliä (Engl. object-oriented programming language).

Skriptauskielillä tarkoitetaan yleensä tulkattavia ohjelmointikieliä. Kun käännettävät kielet (esim. C-kieli) käännetään konekielelle ja vasta sitten ajetaan, voidaan tulkattavat kielet ajaa suoraan ja ohjelmointikielellä kirjoitetut tulkataan konekielelle käskyjen tullessa vastaan. Skriptauskielillä ja ohjelmointikielillä ei ole muuta eroa kuin suoritustapa, eli kyse onkin vain siitä, että on haluttu erottaa kielet suoritustavan perusteella.

3.1 Python

Python on usein komentoriviltä ajettava korkean tason ohjelmointikieli. Sen syntaksi on yksinkertainen ja helppo omaksua. Se on monipuolinen ohjelmointikieli, jossa on pyritty yksinkertaiseen ja havainnolliseen rakenteeseen. Se ottaa vaikutteita ja välillä lainaa suoraan ohjelmointikieli Perlistä. Python-ohjelman ajamiseen tarvitaan Python-tulkki, joka lukee Python-kielisen koodin ja ajaa suoraan sen käskyt. (4.)

Python-ohjelmointikielestä on pyritty tekemään ohjelman rakenteesta mahdollisimman yksinkertainen. Sen kirjastot ovat laajat ja niitä hyödyntämällä saatiin tämän projektin vaatimat tehtävät suoritettua. Sen etu on tulkkaus, koska ohjelma on välittömästi valmis ajettavaksi kirjoittamisen jälkeen, ilman tarvetta kääntämiselle.

Tässä ohjelmassa Pythonilla kirjoitettiin verkkosivuja lukuun ottamatta työkalun kaikki muut osat. Python-kielinen ohjelma hoitaa testilokien lukemisesta JavaScript-listan luomiseen.

3.2 JavaScript

JavaScript on Netscapen kehittämä ohjelmointikieli, jolla saadaan lisättyä www-sivuille lisättyä dynaamisia toimintoja. JavaScript on tarkoitettu lähinnä Web-ohjelmointikieleksi tuottamaan Web-sivuille toiminallisuuksia. JavaScriptin skriptit suorittaa verkkosivuja käyttävä tietokone (client-side scripting), täten se ei rasita palvelinta.

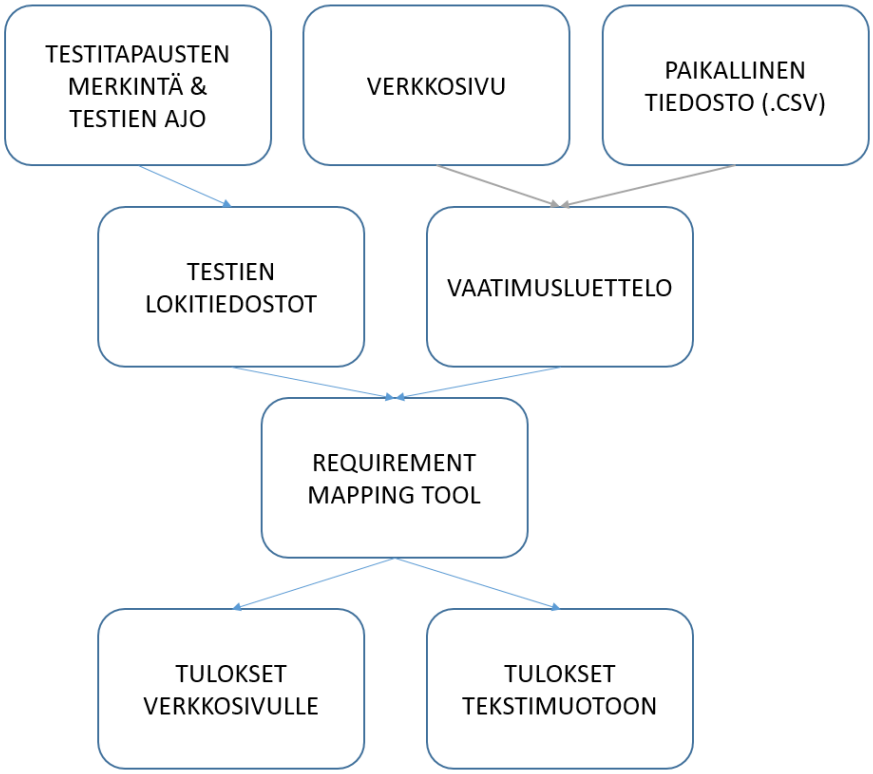
Tässä projektissa verkkosivu luotiin suurimmaksi osaksi hyödyntämällä JavaScriptiä, joten verkkosivun toiminnot ja näkymät syntyvät dynaamisesti vaatimusten määrän ja niiden sisältämän informaation mukaan. HTML-kieli pyrittiin pitämään minimissään, koska vaatimuksien määrää ei voi ennalta tietää, joten verkkosivun näkymän tuli syntyä vaatimuksien mukaan.

4 OHJELMISTON TOTEUTUS

Vaatimusten täyttymistä seuraava työkalu täytyi luoda tyhjästä. Työkalun tuli pystyä lukemaan vaatimusten nimet ja tiedot vaatimusluettelosta ilman, että käyttäjän tarvitsisi niitä syöttää. Näiden vaatimusten perusteella ohjelma tarkistaa, että testilokista löytyy vaatimuksille merkattuja testitapauksia.

4.1 Yleiskuva

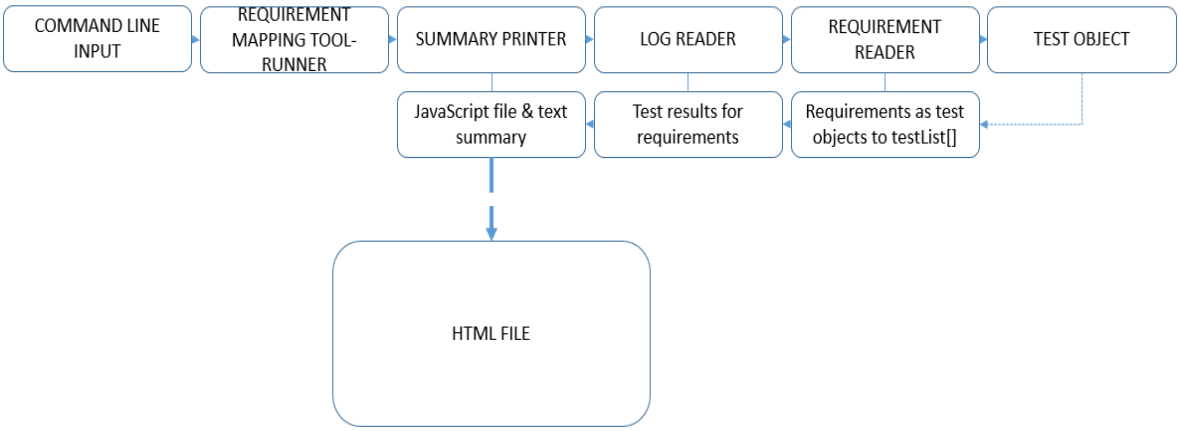
Valmiin työkalun käyttäminen alkaa testausympäristössä testitapausten merkitsemisestä vaatimukselle, jonka jälkeen testit ajetaan omissa ympäristöissään. Testeistä saadut tapahtumalokit syötetään komentoriviparametreina työkalulle yhdessä vaatimusluettelon kanssa. Tästä saadaan ulos tekstitiedosto ja JavaScript-tiedosto, jota työkalun verkkosivu lukee. (Kuva 1.)



KUVA 1. Työkalun tapahtumat

Ohjelmiston rakenne koostuu useasta osasta. Työkalun ajamisesta vastaava Python-tiedosto käynnistetään komentoriviltä, jolloin sille annetaan parametreina testilokit ja vaatimusluettelo. Tämä python tiedosto käynnistää työkalun tulostuksista vastaavan summary printer -olion.

Summary printer -olio luo lokitiedoston lukemisesta vastaavan log reader -olion, joka taas luo vaatimusluetteloa lukevan requirement reader–olion. Requirement reader –olio luo listan testiolioista, jotka palautetaan aiempien olioiden läpi takaisin summary printer–oliolle. Summary printer tulostaa tekstitiedoston ja JavaScript-tiedoston, joita työkalun erillinen HTML-tiedosto voi lukea. (Kuva 2.)



KUVA 2. Ohjelmiston rakenne

4.2 Testiolio

Testiolio oli työn eniten hyödynnetty olio. Sitä käytettiin sekä otsikkotiedoston generointiin että tietojen käsittelyyn. Olion tarvitsemat muuttujat asetettiin olion muodostimessa. Muut osat asetettiin olion muuttujaa kutsumalla.

Olio sisälsi mm. vaatimuksen nimen, kuvauksen, ja lisäinformaation. Lisäksi luonnollisesti olio sisälsi tiedot testausten määrästä, montako oli päässyt ja moniko ei ollut päässyt läpi, ja montako kertaa mikä oli hypätty yli. Lisänä saatiin vastaavat tulokset kokonaismääräisenä, eli jos alivaatimusta oli testattu, se lisättiin myös päävaatimuksen kokonaistuloksiin, mutta ei suoraan päävaatimuksen vaatimustuloksiin. (Kuva 3.)

```
class Test(object):
        def __init__(self, requirement, description, additionalinformation):
            self.requirement = requirement
            self.tested = 0
            self.passed = 0
            self.failed = 0
            self.ignored = 0
            self.numbofsubs = 0
            self.subreqs = []
            self.testcases = []
            self.description = description
            self.additionalinformation = additionalinformation
            self.unsorted = []
            self.ownTests = False
            self.Parent = ''
            self.totaltested = 0
            self.totalpassed = 0
            self.totalfailed = 0
            self.totalignored = 0
            self.allpartstested = False
```

Kuva 3. Testiolio-luokka ja muodostin

Kuten moni muukin asia, olioiden luonti pythonissa on tehty yksinkertaiseksi. Ensin määritellään class eli luokka ja nimetään luokka halutulla tavalla. Esimerkiksi oliota luodessa muodostimelle täytyy antaa vaatimuksen nimi, kuvaus ja vaatimuksen perustelu tai lisäinformaatio. Vaatimuksen nimeä lukuun ottamatta muodostimelle annetut tiedot voidaan antaa tyhjänä. Kun luokka on saatu luotua, voidaan varsinaisessa käytössä luoda uusi olio. Esimerkiksi testi = Test(”Vaatimus1”, ”Kuvaus”, ”Perustelu”).

4.3 Vaatimusluettelon noutaminen

Ohjelmistovaatimukset toimitetaan vaatimusluettelona Excel-muodossa, jossa käydään läpi mitä ohjelmilta vaaditaan. Ohjelman tuli hyödyntää vaatimusluetteloa siten, että luodaan lista, johon luodaan vaatimusluettelon mukaisesti vaatimuksen tietoja kantavia testiolioita.

Koska tavoitteena oli tehdä mahdollisimman automatisoitu järjestelmä, vaatimukset piti pystyä hakemaan luettelosta ilman ylimääräistä käsityötä. Tätä varten kehitin kaksi tapaa: paikallisen CSV-tiedoston ja HTML-tiedoston lukeminen.

Ensimmäisenä tein paikallisen tiedoston käsittelyyn käytettävän skriptin, joka oli vaatimusluettelon alkuperäisen Excel-tiedoston CSV-muotoon tallennettu versio. Tämän käyttäminen huomattiin haastavaksi, kun todettiin, että vaatimusluettelo saattaa

muuttua ajan myötä. Tämä luonnollisesti tuotti ongelman, missä säilytetään vaatimusluettelo ja kuinka varmistuttaisiin, että kaikilla käyttäjillä olisi sama versio siitä.

Toinen vaihtoehto ratkaisi edellisen ongelman. Ajatuksena oli luoda Nokian wiki-sivulle taulukko, jonka olisi tarkoitus pysyä virallisena versiona. Tarvittaessa taulukkoa voidaan taulukkotyökalujen avulla kätevästi muokata. Kaikki vaatimuslistaa käyttävät ominaisuudet pystyivät täten ottamaan suoran yhteyden verkkosivuille ja saamaan viimeisimmät versiot vaatimuksista.

4.3.1 Vaatimusluettelo CSV-tiedostosta

CSV-tiedostossa (Comma-separated values) taulukkorakenteen kentät ovat eroteltu toisistaan puolipisteellä. Puhtaasti tekstimuodossa teksti näkyy näin: ”*Vaatimus;Kuvaus;Perustelu;”.* Esimerkiksi Microsoft Excel osaa avata tiedoston normaalissa taulukkonäkymässä. (Kuva 5.)

	A	B	C
1	#REQUIREMENT NAME	#DESCRIPTION	#RATIONALE
2	REQUIREMENT1	DESCRIPTION FOR REQUIREMENT1	OPTIONAL RATIONALE IF NEEDED
3	REQUIREMENT2	DESCRIPTION FOR REQUIREMENT2	
4	REQUIREMENT3	DESCRIPTION FOR REQUIREMENT3	

KUVA 5. Excel-taulukkolaskentaohjelman näkymä CSV-tiedostosta

CSV:n käyttö oli perusteltua, koska puolipisteellä erotetut tekstikentät oli helppo käydä läpi ohjelmallisesti ja vaatimuksia pystyi lisäämään tai muokkaamaan Excelissä. Tämän takia tämä oli ensimmäinen versio käytetystä taulukkomuodosta.

CSV-tiedostosta olennaisten tietojen poimiminen onnistui siten, että python-skripti kävi tiedostoa läpi rivi kerrallaan ja jakoi tiedot puolipisteen perusteella. Jokainen rivi jaettiin split-funktiolla, jossa merkkijonon puolipisteellä jaetut tiedot tallennetaan listan alkioiksi. Tätä keinoa hyödyntämällä saatiin kaivettua vaatimusluettelosta ohjelman tarvitsemat tiedot.

Kuvan 6 osoittamalla tavalla rivin vaatimustiedot tallentuvat listaan. Listan tiedot ovat käytettävissä viittaamalla listan alkioon. Ensimmäinen tieto tallentuu 0-alkioon, toinen

1-alkioon jne. Esimerkiksi testioliota luodessa voidaan muodostimelle syöttää *test = Test(list[0], list[1], list[2]).*

```
list = 'Vaatimus1; Kuvaus; Perustelu;'.split(';')
```

KUVA 6. Split-funktiossa jaetaan rivin tiedot suluissa määritetyn puolipisteen perusteella

4.3.2 Vaatimusluettelon noutaminen verkkosivuilta

Vaatimusluettelon hakemiseen verkkosivuilta varten tarvitaan python-skripti, joka osaa ottaa yhteyden verkkosivustoon ja hakea sieltä sivun lähdekoodin ja käsitellä ja tallentaa HTML-taulukkoformaattiin tallennetun tiedon listaan. Lisäksi se, että vaatimukset verkkosivuilla ovat salasanan takana, oli otettava huomioon skriptiä kirjoittaessa.

Verkkosivuilta datan noutamiseen käytettiin Curlia. Curl on komentorivityökalu, jolla voidaan siirtää dataa palvelimelta koneelle käyttäen yhtä lukuisista tuetuista protokollista (HTTP, HTTPS POP3, POP3S…). Tätä hyödyntämällä verkkosivun HTML-koodi saatiin haettua ja talletettua listaan python-skriptissä.

Pythonin os-kirjastoa käyttämällä voidaan syöttää käyttäjän tietokoneen komentoriville komentoja. Tätä hyödyntämällä saatiin listaan talletettua komentoriville annetun komennon jälkeiset tulokset. Lista tulee määrittää os.popen(”curl –u +”Käyttäjänimi” + ”verkkosivun osoite”). Tämän jälkeen ohjelma kysyy käyttäjän salasanaa. Ilman salasanan kysymistä ohjelman komento menee näin ”curl –u ”käyttäjänimi:salana””. (6.)

Verkkosivujen taulukkoformaatissa <td>-merkintää käytetään taulukon kenttien kuvantamiseen. Kun oli päätetty, että tietojen näyttämiseen tarvitaan kolme kenttää (kolme <td>-merkintää), ohjelmalle voitiin määrittää järjestysnumeron mukaan ylhäältä alaspäin, mikä kenttä kuuluu mihin kohtaan.

Ohjelma käy läpi HTML-sivua rivi riviltä. Kun <tr>-merkintä tulee vastaan, ohjelma tietää, että seuraavien kolmen <td>- ja </td>-merkinnän väliltä löytyy vaatimuksen nimi ja sitä kuvaavat tiedot.

Ohjelma poimii testilistaan tiedot ja aloittaa tietojen siistinnän. Siistimisen tarkoituksena on poistaa kaikki HTML-merkinnät tallennettavista tiedoista. Yksinkertaisuudessaan ohjelmalle on määritelty, mitä merkintöjä ei haluta mukaan, ja ohjelma poistaa ne.

```
<table>
<tr>
    <td>REQUIREMENT NAME</td>
    <td>DESCRIPTION</td>
    <td>RATIONALE</td>
</tr>
</table>
```

KUVA 4. Käytetty HTML-taulukkoformaatti

4.4 Vaatimukset ja alivaatimukset

Koska yksittäinen vaatimus saattoi olla vaikeasti testattava yhdellä testillä, saatiin ajatus jakaa päätason vaatimus alivaatimuksiksi. Myös alivaatimuksilla tuli olla mahdollisuus omiin alivaatimuksiinsa. Tämä ratkaisu mahdollisti vaatimuksen testauksen antamatta kuvaa siitä, että koko vaatimus olisi kerralla testattu. Vaatimukset ja alivaatimukset erotellaan alaviivalla ja nimetään järjestysnumerolla. Esim. Vaatimus_1_1.

4.4.1 Testioliot omaan listaan

Kun python-skripti käy läpi vaatimusluetteloa, se luo jokaisen vaatimuksen kohdalla testilistaan testiolion. Vaatimusluetteloa käsittelevä skripti poimii luettelosta vaatimuksen nimen, kuvauksen ja mahdollisen lisäinformaation. Nämä tiedot ohjelma antaa oliolle muodostimessa ja tallettaa listaan.

4.4.2 Alitason vaatimukset ylätason vaatimuksen sisälle

Kun ohjelma käy vaatimusluetteloa läpi, aloittaa ohjelma ensimmäisen löytyvän vaatimuksen tarkastelun. Tarkastelussa ohjelma laskee, montako alaviivaa vaatimukselta löytyy. Tästä ohjelma voi päätellä, että jos alaviivoja on saman nimen omaavalla vaatimuksella yksi enempi, tämän täytyy olla vaatimuksen alivaatimus.

Kun päävaatimustasot on tunnistettu, ohjelma kerää ensiksi päätason vaatimukset listaan, jonka jälkeen testilistan lukeminen aloitetaan. Ensiksi katsotaan, löytyykö saman nimen omaavia vaatimuksia, joissa merkkejä on yksi enemmän. Ohjelma syöttää löydetyn alitasovaatimuksen ylätasovaatimuksen sisälle omaan listaan.

Tällä menetelmällä oletetaan vahvasti, että vaatimukset nimetään saman käytännön mukaisesti ja vaatimustaso on alaviivalla eroteltu. Alaviivalla erottelu voidaan pienellä työllä muuttaa melkeinpä miksi tahansa merkiksi.

4.5 Testien yhdistäminen vaatimukseen

Kun vaatimusluettelo oli saatu haettua, oli mietittävä, miten työkalu tietää, mikä testitapaus testaa mitäkin vaatimusta. Tätä varten päätettiin, että testiohjelmistossa tulostetaan vaatimukselle merkintä ennen testitapauksen tulostusta.

Kun työkalu käy testilokia läpi ja kun riviltä löytyy vaatimusmerkintä, työkalu tietää, että seuraava testitapaus kuuluu merkitylle vaatimukselle. Yhdelle testitapaukselle voidaan kuitenkin merkata enemmän kuin yksi vaatimus, joten testiloki luetaan useaan otteeseen läpi. Jokaisen vaatimuksen kohdalla työkalu siis lukee testilokin läpi ja etsii tälle merkattuja testitapauksia.

4.5.1 Testitapausten merkitsemiseen aputiedosto

Työn helpottamiseksi kehitettiin vaatimusluetteloa lukeva ja valmiin aputiedoston luova skripti. Yksinkertaisuudessaan aputiedosto sisältäisi vaatimukset testilokia lukevan olion ymmärtämässä muodossa. Lisäksi tiedostosta löytyy vaatimuksen kuvaukset ja testilokiin tulostamiseen oma funktio.

Pelkkää vaatimuksen nimeä ei voinut käyttää merkintänä, koska testilokissa saattaisi olla vaatimuksista maininta, joten se sotkisi testin lopputuloksen. Merkintänä käytettiin merkkijonoa, joka tuskin löytyisi testilokista muuten kuin vaatimusmerkintänä. Päätettiin, että vaatimuksen nimet kirjoitettaisiin isoilla kirjaimilla ja alkuun lisättäisiin ”REQ_”- ja loppuun % -merkinnät. Tällä tavoin minimoitiin mahdollisuus siihen, että lopputulos olisi vääristynyt. (Kuva 7.)

```
#define REQ_VAATIMUS1 "Vaatimus1:n kuvaus"
#define TAG "%"

#ifdef REQUIREMENT_MAPPER
#define TEST_FOR_REQUIREMENT(x) print_requirement_tag(#x, x)
#else
#define TEST_FOR_REQUIREMENT(x)
#endif

#ifdef REQUIREMENT_MAPPER
static void print_requirement_tag(const char* requirement_id, const char* requirement_desc)
{
    (void)requirement_desc;
    printf("%s%s\n", requirement_id, TAG);
}
#endif

#ifdef __cplusplus
}
#endif

#endif
```

KUVA 7. Testilokiin vaatimusmerkinnän tulostukseen käytettävä otsikkotiedosto

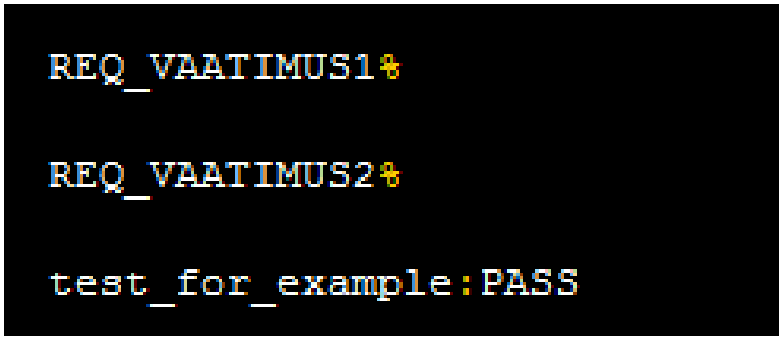
4.5.2 Vaatimusmerkinnän tulostus

Unityn ajettavaan testiohjelmaan sisällytetään generoitu otsikkotiedosto, jonka jälkeen otsikkotiedoston makroja voidaan käyttää. Ennen jokaista testiä merkataan TEST_FOR_REQUIREMENT(vaatimusmakron nimi). (Kuva 8.)

```
void test_for_example()
{
    TEST_FOR_REQUIREMENT(REQ_VAATIMUS1);
    TEST_FOR_REQUIREMENT(REQ_VAATIMUS2);
```

KUVA 8. Testifunktio merkataan testaamaan kahta eri vaatimusta

Kun testejä ruvetaan ajamaan, ohjelma tulostaa vaatimusmerkinnät otsikkotiedoston sisältämän funktion mukaisesti (kuva 7). Ohjelma tulostaa vaatimuksen nimellä varustetun makron sisällön sijaan makron nimen. Tämä ratkaisu perustui siihen, että vaatimuksen nimi ei olisi tarpeeksi kuvaava, että käyttäjä osaisi nimen perusteella lisätä vaatimukset testitapauksille. Useassa editorissa, kuten Eclipsessa, voidaan tarkistaa makron sisältö asettamalla hiiren kursori makron ylle, jonka jälkeen editori näyttää, mitä makro pitää sisällään. Makron sisältö toimii tässä tapauksessa vain vaatimuksen kuvauksena. Testilokiin vaatimusmerkinnät tulostuvat ennen testitapausta. (Kuva 9.)



KUVA 9. Testimerkinnät lokitiedostossa

4.6 Testilokien käsittely

Koska tässä tapauksessa ohjelmistotestit ajetaan automaattisesti ja testien tulostukset ovat konsolitulosteita, tapahtumat kerätään tapahtumalokiin, josta tulokset olisi tarkoitus poimia. Tulosten keräämistä varten luotiin python-skripti, joka kävi lokitiedoston läpi ja tallensi vaatimuksille merkatut testit ja tulokset.

Skriptin tarkoituksena oli kerätä testiolioille tulokset läpikäymällä lokitiedostoa, jossa testit olivat. Tätä varten skripti kirjoitettiin python-kielellä, koska sen laajat ohjelmakirjastot mahdollistivat nopean ja tehokkaan tavan poimia oleelliset tiedot tekstitiedostosta. Kokonaisuus rakentui useasta kohdasta, joissa käytettiin useita eri olioita, jotka oli tarkoitettu tekemään oman osansa kokonaisuudesta.

Ohjelma kerää ensiksi päävaatimukset testiolioina testilistaan, jonka jälkeen se aloittaa listan läpikäynnin jokaisen päävaatimuksen mukaisesti. Ohjelma tarkistaa, löytyykö vaatimusluettelosta vaatimusta, jossa on sama nimi, mutta alaviivoja yksi enemmän, kuin päävaatimuksessa. Jos sellainen vaatimus löytyy, se lisätään testiolion sisältämään alivaatimuslistaan. Tämän jälkeen alivaatimuslistalle tehdään samalla tavalla, eli etsitään, löytyykö alivaatimukselle alivaatimuksia.

Kun ohjelma löytää vaatimusmerkinnän, se osaa etsiä seuraavaa testitulosta, joka tulee vastaan. Se etsii niin kauan testitulosta, kunnes löytää sen tai lokitiedosto on käyty loppuun.

Esimerkiksi Unity-testauksessa vaatimusmerkinnän löytymisen jälkeen ohjelma etsii rivi kerrallaan merkinnästä eteenpäin tuleeko vastaan ”:PASS”, ”:FAIL” tai ”:IGNORE”-teksti. Tällä tavalla ohjelma tunnistaa, että kyseessä on testitulos ja se liittyy merkattuun vaatimukseen.

Vaatimuksen ja tuloksen löytämiseen tarvitaan useita silmuikoita (kuva 10). Ulommainen for-silmukka kävi vaatimuksia sisältävän testilistan oliot yksitellen läpi. Sisempi luki testilokia rivi kerrallaan. Jokaisen testiolion kohdalla ohjelma aloittaa for-silmukassa lokitiedoston läpikäymisen alusta loppuun ja kerää olion sisältämän vaatimuksen nimi-tiedon mukaiset merkinnät. Merkinnän tullessa vastaan käynnistyy while-silmukassa etsi-toiminto, jonka tarkoituksena oli löytää vaatimukselle testi ja tämän tulos. Tätä while-silmukkaa tarvitaan, koska testi saattoi olla, vaikkapa kahdeksan rivin päästä merkinnästä, riippuen montako merkintää oli yhdellä testillä. Eli merkinnän tultua while-silmukka jatkaa samalta riviltä eteenpäin, kunnes testi tulee vastaan tai tekstitiedosto päättyy. Jos tekstitiedosto päättyy eikä testiä löydy, jättää ohjelma merkinnän huomioimatta.

```
for res in resfile:
    if self.REQADD+Test.requirement.upper()+self.tag in res or Test.requirement+self.tag in res:
        search = True
        j = i
        res = string.replace(res, '\n', '').replace('\r', '')
        if self.ERROR == "ERROR":
            res = self.removetags(res);
            search = self.testFinder(Test, res, search)
        try:
            while search:
                ress = resfile[j]
                ress = string.replace(ress, '\n', '').replace('\r', '')
                if self.ERROR == "ERROR":
                    ress = self.removetags(ress);
                search = self.testFinder(Test, ress, search)
                j += 1
```

KUVA 10. Sisäkkäisiä silmukoita

Kun ohjelma on löytänyt vaatimusmerkinnän ja sille testitapauksen, lisätään testiolio testitulos. Ohjelma tunnistaa testitapauksesta, että onko se päässyt läpi, vai ei. Nämä tulokset lisätään kasvattamalla tuloksen mukaista arvoa ja testien määrää yhdellä kokonaisluvulla. Samalla talletetaan testitapaus testiolion testcases-nimiseen listaan.

4.7 Vaatimustilanteen näyttäminen tekstitiedostossa

Tietojen käsittelyn jälkeen Python-skripti tuottaa tekstitiedoston, jossa on näkyvillä ensinnä testaamattomat vaatimukset, joiden perässä yksityiskohtainen tuloste vaatimuksista. Tämä järjestely nähtiin tarpeelliseksi, koska opinnäytetyön tilaajan kannalta tärkeimpiä tietoja on se, mikä vaatimus on vielä testaamatta.

Testaamattomien vaatimusten jälkeen näytetään testatut vaatimukset. Tuloste näyttää testitulosten lisäksi vaatimuksen kuvauksen ja jokaisen sille osoitetun testin. Lisäksi näytetään myös lyhempi tiivistelmä testien tuloksista. Kaikki alivaatimukset esitetään päävaatimuksen sisällä, jotta epäselvyyksiltä vältyttäisiin.

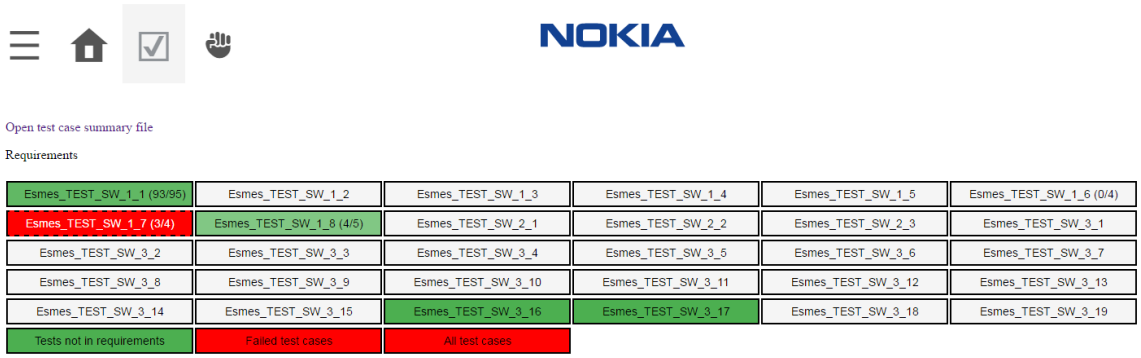
4.8 Vaatimustilanteen näyttäminen verkkosivuilla

Helppokäyttöisyyden ja luettavuuden vuoksi työkalulle lisättiin HTML-tiedosto, josta näkisi nopealla silmäyksellä, että missä tilanteessa vaatimukset ovat. Tämä mahdol-

listi JavaScriptin käytön, joka toi mukanaan verkkosivulle toiminnollisuuksia. Esimerkiksi näkymän vaihtaminen, kuten esimerkiksi käyttäjälle huomionarvoisten asioiden näyttäminen, kuten vaatimuksella on testitapaus, joka ei ole mennyt läpi.

Verkkosivut oletuksena näyttävät perusnäkymän, jossa päävaatimukset on listattu painikkeina. Painikkeita klikkaamalla saadaan tarkempaa tietoa kustakin vaatimuksesta. Vasemmalla yläkulmassa olevista symboleista näkymää voidaan muuttaa (kuva 12). Näkymät voidaan muuttaa näyttämään pelkästään vaatimukset (pää-, ja alivaatimukset), joita ei ole vielä testattu. Vaihtoehtona on myös kaikkien vaatimusten näyttäminen.

Hamburger-painikkeesta avautuu näkymä (kuvan 12, kolme päällekkäistä viivaa, vasen yläkulma), josta voidaan seurata päätason vaatimuksen edistymispalkeista, kuinka pitkälti vaatimus on testattu. Toinen vaihtoehto on näyttää ns. vaatimuspuu. Siinä näkee puurakenteessa pää-, ja alivaatimukset ja sen, mitkä niistä on testattu. (Kuva 12.)



KUVA 12. Verkkosivujen perusnäkymä.

4.8.1 JavaScript-lista

Python-skripti tulostaa testresults.js-tiedoston, jossa luodaan JavaScript-versio testioliosta ja python-listaa vastaava JavaScript-lista testiolioista. Tätä listaa hyödynnetään verkkosivujen kaikissa toiminnollisuuksissa.

4.8.2 CSS

CSS, eli Cascading Style Sheets on verkkosivuille tarkoitettu tyyliohje, jota verkkoselain noudattaa. CSS-tyyliohjeita käytetään verkkosivuilla näyttämään HTML-sivun rakenne halutulla tavalla. Verkkosivuille oli hyvä lisätä ulkonäöllisiä määrittelyjä, jotta verkkosivujen käyttäminen olisi miellyttävämpää. (Kuva 13.)

```
.button {
    background-color: white;
    border: none;
    color: white;
    padding: 6px 20px;
    padding-left: 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 14px;
    margin: 2px 1px;
    -webkit-transition-duration: 0.4s; /* Safari */
    transition-duration: 0.4s;
    cursor: pointer;
    layout:fixed;
    word-wrap:break-word;
}
```

KUVA 13. Button-, eli painikeluokan tyyliohjeita, joita verkkosivu noudattaa

4.8.3 Vaatimuspainikkeet

HTML-objekteista katsottiin parhaimmaksi käyttää painikkeita, joissa lukee vaatimuksen nimi. Tällä tavoin vaatimukset voidaan listata verkkosivulle mahdollisimman tiiviisti. Vaatimuksen tarkemmat tiedot saatiin näkyviin klikkaamalla vaatimus auki.

Koska vaatimukset saattoivat sisältää alivaatimuksia, katsottiin luontevaksi, että vaatimusta klikkaamalla pääsee katsomaan, mitä alivaatimuksia vaatimus sisältää (kuva 14).

KUVA 14. Vaatimuspainikkeita

Vaatimuspainiketta klikatessa verkkosivu avaa vaatimuksen tarkemmat tiedot ja vaatimuksen mahdolliset alivaatimukset. Ohjelma tarkistaa, että löytyykö testilistasta vaatimusta, joille on merkattu parent-arvoksi vaatimuksen nimi, ja listaa ne päävaatimuslistauksen alle pystysuoraan listaan.

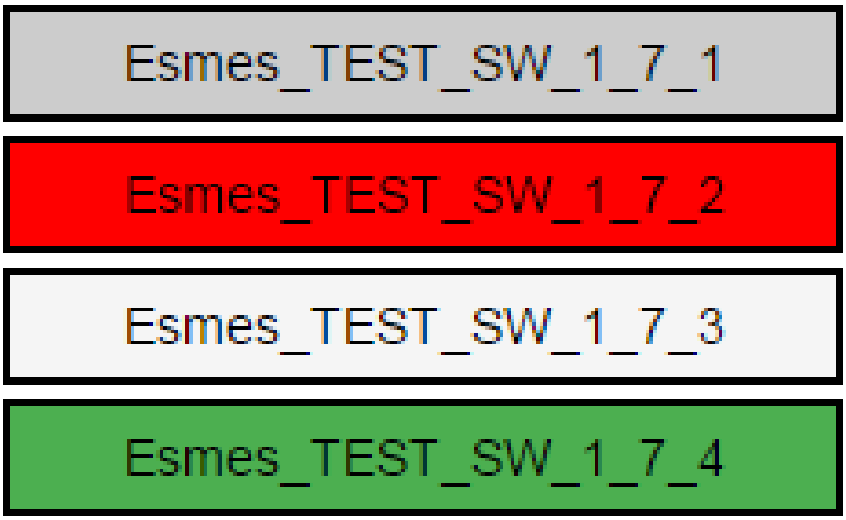
Värikoodaus oli pieni, mutta tarpeellinen lisä helpottamaan nopealla vilkaisulla kuvan saamista siitä, missä tilanteessa vaatimukset ovat. Luonnolliset värit olivat vihreä, punainen, harmaa ja valkoinen. (Kuva 15.)

- Harmaa. Vaatimuksella on testejä, mutta ainoastaan ylihypättyjä testejä. (Tulos: ”IGNORE”)

- Punainen. Jos vaatimuksella tai yhdelläkään alivaatimuksella on hylätty testitulos, painikkeen väri muuttuu punaiseksi.

- Valkoinen. Vaatimuksella ei ole yhtään testiä.

- Vihreä. Vaatimuspainike saa vihreän värin, jos ohjelmaa testattu ja joukossa ei ole yhtään hylättyä testiä. Tämän lisäksi vaatimuspainikkeen läpinäkyvyys viittaa siihen, kuinka hyvin vaatimusta on testattu. Jos vaatimus on suoraan testattu tai kaikkien alivaatimukset on testattu, painikkeen väri on täysin vihreä eikä yhtään läpinäkyvä. Jos esimerkiksi vaatimuksen yhtä alivaatimusta kymmenestä alivaatimuksesta on testattu, väri on vihreä, mutta hyvin haalean läpinäkyvä.

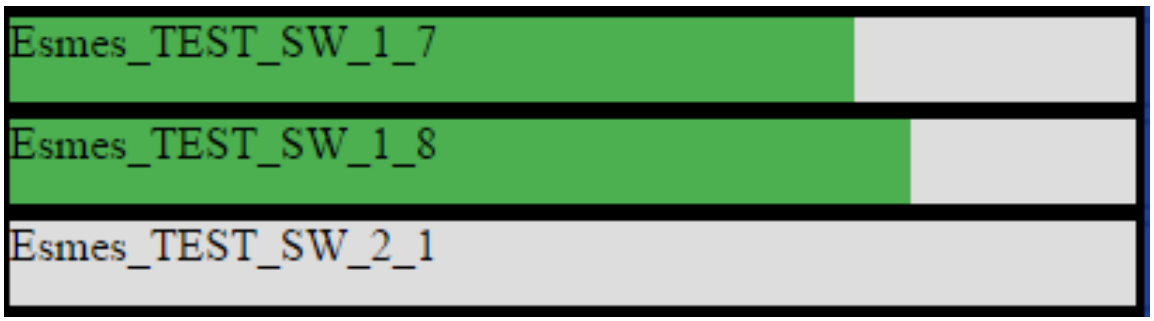


KUVA 15. Vaatimusten värikoodaus

4.8.4 Vaatimuksen edistyspalkki

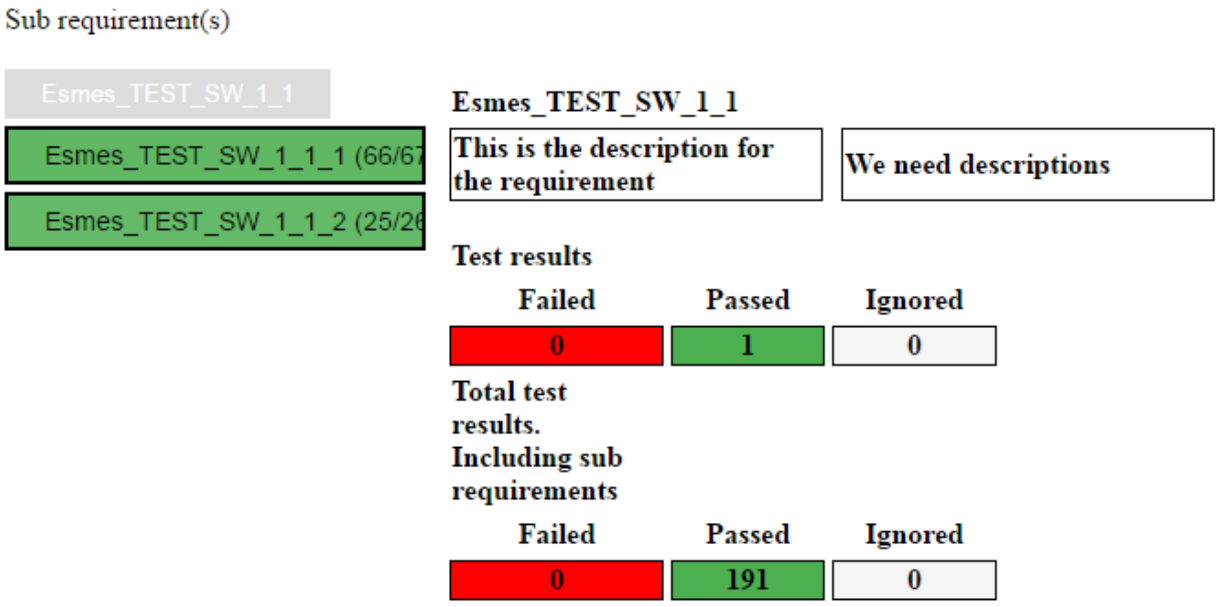
Päävaatimustasosta haluttiin myös selkeä visuaalinen näkymä, jossa näkisi vaatimuksen edistymistilanteen. Tätä varten kehitettiin vaatimuspalkkinäkymä.

Näkymässä näkee kasvavalla vihreällä tai punaisella palkilla harmaata taustaa vasten, että kuinka pitkälti vaatimus on testattu. Tämä toteutettiin siten, että ohjelma laskee, montako prosenttia vaatimuksen osista on testattu ja edistyspalkki täyttää leveyssuunnassa sillä prosenttiluvulla harmaan taustan. (Kuva 16.)



KUVA 16. Vaatimusten täyttymistä kuvaavat edistymispalkit

Vaatimuspainiketta painettaessa tulee näkyviin yksi tai kaksi näkymää testien tuloksista. Jos vaatimusta on suoraan testattu, näkyy vaatimuskohtaiset tulokset. Jos vaatimuksen alivaatimusta on testattu, tulee näkyville totaalitestauksen tulokset. (Kuva 17.)



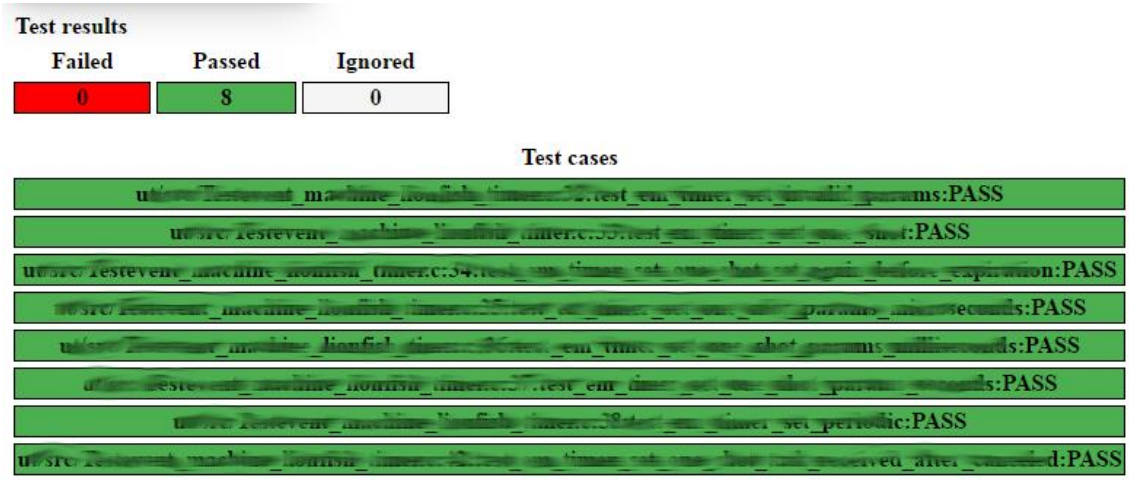
KUVA 17. Testitulokset

4.8.5 Testitapausten näyttäminen

Aina kun vaatimuspainiketta on painettu, ohjelma tarkastaa, että onko vaatimuksella testejä. Jos testejä löytyy, ohjelma tulostaa löydetyt testit ja tulokset taulukkoon.

Testit näkyvät vain vaatimuksen kohdalla, jolla on omia testejä, eli vaatimuksen alivaatimuksen testitapaukset näkyvät vain alivaatimuksen kohdalla. Päävaatimustasolla näkee vaatimuksen totaalitulokset numeraalisena esityksenä. Testikertoja esittävää totaalitulosta klikkaamalla aukeaa ponnahdusikkuna, jossa listataan kaikki testit, jotka kuuluvat tälle vaatimukselle.

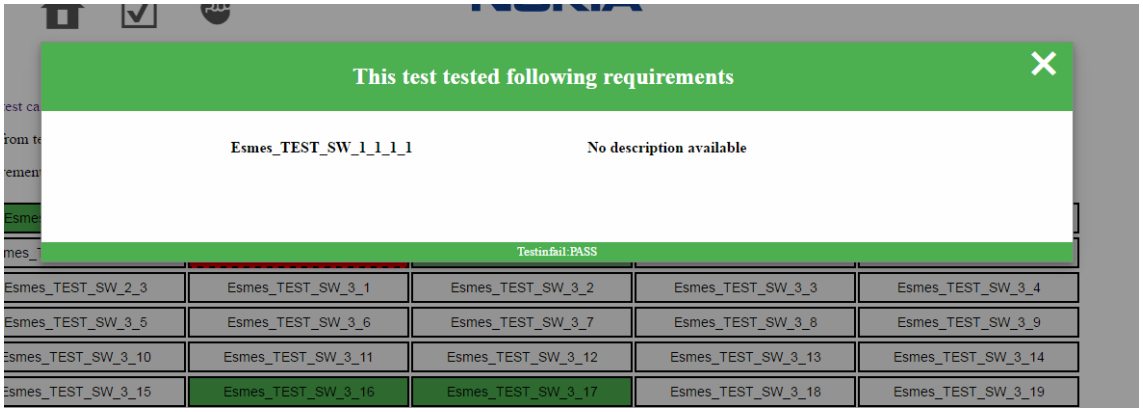
Taulukko, jossa testit näytetään CSS-tyyliohjeita hyväksikäyttäen, on muokattu näyttämään enemmän painikkeilta kuin varsinaiselta taulukolta. Tämä johtuu siitä, että käyttäjä hoksaisi myös painaa niitä saadakseen lisäinformaatiota. (Kuva 18.)



KUVA 18. Testitapaukset lueteltuina. Testitapaukset ovat oikeita, joten niiden sisältö on sensuroitu

Testitapausta klikkaamalla aukeaa näkymä, joka näyttää mitä kaikkia vaatimuksia testi on myös testannut. Listan vaatimuksia klikkaamalla pääsee avaamaan kyseisen vaatimuksen.

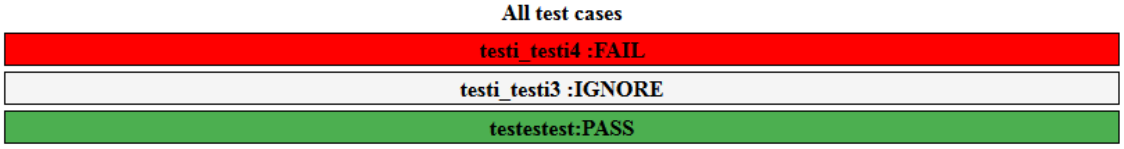
Tämä on toteutettu siten, että ohjelma tarkistaa testitapauksen nimen ja käy testilistan jokaisen testiolion testilistan läpi, että löytyykö sieltä sama testi. Jos löytyy, ohjelma lisää vaatimuksen listaan, joka lopuksi palautetaan ja tulostetaan. (Kuva 19.)



KUVA 19. Ponnahdusikkuna näyttää kaikki vaatimukset, joita testi testasi.

Jokainen listattava testitapaus käy läpi kyselyn, jossa tarkistetaan mikä on testin tulos. Jos testi ei ole päässyt läpi, se tulostetaan punaiseen laatikkoon, ylihypätty tai

huomiotta jätetty testi (”ignore”) tulostetaan valkoiseen laatikkoon ja läpi mennyt testi tulostetaan vihreään laatikkoon. (Kuva 20.)



KUVA 20. Testitapausten värit kertovat jo itsessään tuloksen.

4.8.6 Vaatimuspuu

Verkkosivulla katsottiin hyödylliseksi, että tekstimuotoisen version tapainen vaatimuspuu käytössä. Tämä auttaisi kokonaisuuden hahmottamisessa. Kuten muutkin osat verkkosivuilta, tämäkin toteutettiin puhtaasti dynaamisesti, eli ilman vaatimuksia ei syntyisi edes puun runkoa.

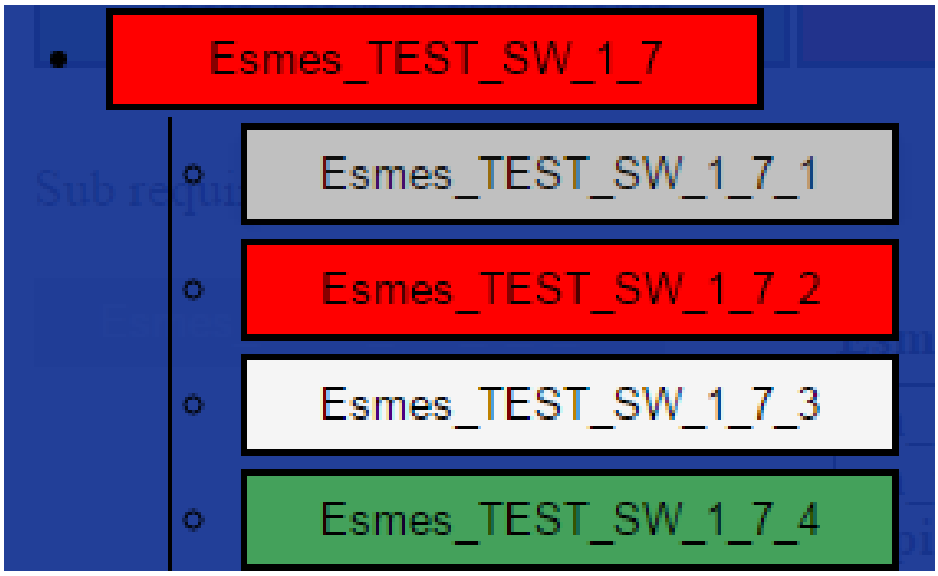
Tämä ratkaisu toteutettiin luomalla dynaamisesti HTML-lista. Listaan lisätään päävaatimukset, joiden sisälle omalle oksalle tulevat alivaatimukset. Alivaatimuksen alivaatimukset jatkavat tämän kaavan noudattamista.

Ensiksi luotiin lista hyödyntämällä JavaScriptin document luokan createElement()-funktiota hyödyntäen. Tällä funktiolla voidaan luoda HTML-objekteja dynaamisesti JavaScriptin kautta. Ohjelma luo ensiksi ul-elementin eli listan, johon ohjelma rupeaa lisäämään li-elementtejä eli listan osia. Jokaisen li-elementin kohdalla, ohjelma tarkistaa, että jos vaatimuksella on alivaatimuksia, li-elementti ottaa lapsielementikseen uuden ul-elementin, johon li-elementit lisätään. Tätä samaa jatketaan, kunnes vaatimuksella ei ole alivaatimuksia. (Kuvat 21, 22.)

```
<ul>
    <li> Vaatimus1
        <ul>
            <li>Vaatimus1_1</li>
            <li>Vaatimus1_2</li>
            <li>Vaatimus1_3
                <ul>
                    <li>Vaatimus1_3_1</li>
                    <li>Vaatimus1_3_2</li>
                    <li>Vaatimus1_3_3</li>
                </ul>
            </li>
            <li>Vaatimus1_4</li>
        </ul>
    </li>
   <li> Vaatimus2</li>
</ul>
```

- Vaatimus1
  - Vaatimus1_1
  - Vaatimus1_2
  - Vaatimus1_3
    - Vaatimus1_3_1
    - Vaatimus1_3_2
    - Vaatimus1_3_3
  - Vaatimus1_4
- Vaatimus2

KUVA 21. Havainnollistava kuva, mitä ohjelma tekee dynaamisesti



KUVA 22. Vaatimuspuun osa verkkosivuilla

4.9 Refaktorointi

Ohjelmakoodin tarkastelussa huomattiin tarve tehdä ohjelman rakenteesta ja funktioista selkeämpiä. Vaikka ohjelma toimisi moitteetta, on hyvä, että tarvittaessa toinen ohjelmoija pystyisi vaivatta lukemaan ja tekemään koodiin lisäyksiä tai muutoksia.

Refaktoroinnilla (Engl. Code refactoring) tarkoitetaan prosessia, jossa ohjelman rakennetta muutetaan muuttamatta toiminnollisuutta.

Refaktorointia käytetään parantamaan esimerkiksi ohjelman luettavuutta, mikä auttaa ohjelman ylläpitoa. Sitä tarvitaan myös tekemään ohjelmasta yksinkertaisemman, poistamalla redundanssia, eli informaation toistuvuutta. Refaktoroinnilla ei ole tarkoitus korjata ohjelmointivirheitä. Perusajatus refaktoroinnissa on se, että tunnistetaan alueet, missä ohjelma voisi olla yksinkertaisempi. (1, s. 21.)

4.9.2 Refaktorointi opinnäytetyössä

Refaktoroinnissa funktiot muokattiin siten, että ne olisivat mahdollisimman lyhyitä. Lisäksi tarkasteltiin, että ohjelmasta ei löytynyt samoja asioita tekeviä funktioita. Muuttujat ja funktiot pyrittiin muuttamaan sellaisiksi, että niiden käyttötarkoitus selviäisi pelkän nimen perusteella.

Python-skripti jaoteltiin eri osiin, eli moduuleihin, jotka hoitavat kaikki vaatimusten lukemisesta, tiedon kaivamiseen ja JavaScript-listan generointiin. Nämä resurssitiedostot olivat olioita, joiden tarkoitus oli hoitaa oma osansa kokonaisuudessa. Tarkoitus oli, että esimerkiksi lokitiedostoa lukeva olio tekisi vain sen asian, eikä muuta. Tällä tavoin ohjelman osia voitiin käyttää monipuolisemmin ja mahdollisesti myös muualla.

Kun esimerkiksi testiolio-luokka ja vaatimusluettelon lukemisesta vastaava requirementReader-luokka tehtiin omiksi moduuleiksi, sitä voitiin käyttää useammassa tapauksessa. Python-skripteista näitä moduuleita käytti testilokin-lukija, että otsikkotiedoston luova generaattori.

Verkkosivuja ohjaava JavaScript-skripti ja ulkonäöstä vastaava CSS-määrittely sisällytettiin yhteen HTML-sivuun pelkästään jo sen takia, että verkkosivujen käyttöönotto olisi mutkatonta missä ympäristöstä tahansa. Pelkästään HTML-sivu ja testresults.js-niminen skriptitiedosto riittäisivät informaation lukemiseen.

Verkkosivujen JavaScript kävi läpi refaktoroinnin. Refaktoroinninnissa keskityin pilkkomaan funktioita pienempiin osiin. Pyrin siihen, että yhden funktion sisältöä pystyisi

tarkastelemaan siten, että se näkyisi yhdellä silmäyksellä eikä tekstieditorilla tarvitsisi selata sivua eteenpäin.

4.10 Testaus

Ohjelmatestauksen työkalu ei voi itse olla testaamaton ohjelma. Tätä varten ohjelmalle määriteltiin testejä, joilla voitiin testata, että tulokset ovat oikeita eivätkä myöskään väärin lisätyt vaatimusmerkinnät kaada ohjelmaa.

Testit rakennettiin siten, että tehtiin oma testiloki, johon oli lisätty tietty määrä vaatimusmerkintöjä ja testitapauksia. Pythonin unit testillä tehtiin muutamia testifunktioita, joiden tehtävänä oli tarkistaa, että vaatimuksille löytyi oikea määrä testejä.

Testejä ajettiin myös muilla tavoin. Esimerkiksi ohjelmalle merkattiin vaatimusmerkintä testilokin loppuun, joka tarkoitti sitä, että ohjelma ei löydä merkinnälle testitulosta. Tämä johti ohjelman kaatumiseen. Ohjelmaa suojattiin poikkeuskäsittelijää hyödyntäen. Siinä koetetaan ajaa (try) ohjelmaa ja virheen sattuessa ohjelma ei kaadu, vaan hyppää virheenkäsittelijän (except) hoidettavaksi, joka yleensä, tilanteesta riippuen jättää yksinkertaisesti sen virheen huomiotta ja jatkaa toimintaansa. Virhe ei jää käyttäjältä huomiotta, koska ohjelma tulostaa virheilmoituksen, kuten vaikka tilanteessa jossa vaatimusmerkinnälle ei löydy testitulosta. Virheen tullessa virheilmoituksessa ilmoitetaan, että kyseiselle vaatimukselle ei löytynyt tulosta. Samalla ilmoittaen testilokista rivin, josta merkintä löytyy

Valmiiden testien tekeminen oli tärkeää, koska ohjelmiston toimivuus tulisi varmistaa, kun ohjelmistoon tehdään muutoksia, tai lisäyksiä. Jos muutosten jälkeen testit antavat saman tuloksen, kuin ennen muutoksia, ohjelma ei mennyt rikki.

4.11 Runner-tiedoston käyttäminen

Jotta työkalua voisi käyttää komentoriviltä käsin, oli työkalulle luotava runner-tiedosto. Runner-tiedoston tehtävänä oli ottaa komentoriviltä vastaan käynnistyskäskyn lisäksi, käytettävät testilokit ja vaatimusluettelo.

Runner-tiedoston komentoriviparametrit merkitään tiedostonimen eteen --log tai --req. Esimerkiksi --log-merkinnän tullessa vastaan Runner olettaa seuraavan parametrin sisältävän käytettävän lokitiedoston nimen. Lokitiedoston (--log-merkintä) kohdalla ohjelma tarkistaa, että onko sen nimistä tiedostoa olemassa. Jos tiedostoa ei ole olemassa, ohjelma palauttaa käyttäjälle virheilmoituksen ilmoittamalla, että syötetyn nimen mukaista tiedostoa ei ole olemassa. Vaatimusluettelon (--req-merkintä) kohdalla ohjelma tarkastaa myös, onko vaatimus annettu paikallisen tiedoston mukaan. Jos ei ole ohjelma tarkastaa, että onko kyseessä verkkosivu. (Kuva 23.)

```
python reqmapper_run.py --log esimerkkitestiloki1.log
--req [vaatimusluettelon verkkosivu / paikallinen tiedosto]
```

KUVA 23. Runner-tiedoston ajaminen komentoriviltä käyttäen pythonia

Nämä saadut tiedot syötetään summaryprinter-olion muodostimelle, joka aloittaa varsinaisesti työkalun toiminnan.

4.12 Jenkins-automatisointi

Koska koko projektin kantava teema oli automatisointi, tarvittiin järjestelmä, joka hoitaa testilokien käsittelyt automaattisesti ja sitä kautta käyttäjä saisi viimeisimmät tulokset käyttöönsä. Tässä vaiheessa tuli käyttöön Jenkins-automatisointi.

Jenkins on Java-pohjainen jatkuvan integroinnin työkalu. Jenkins hoitaa testien ajamisen, kääntämisen ynnä muut automatisointia vaativat operaatiot. Vuonna 2004 Kohsuke Kawaguchi aloitti Jenkins-projektin. Se on avoimen lähdekoodin yhteisö, joka toimii voittoa tavoittelemattoman Software in the Public Interest-järjestön alla. (6)

Jenkinsiä käytettäessä Jenkinsille luodaan job, eli ajettava tehtävä, jota Jenkins hallinnoi. Jobille määritellään komentorivikäskyt, jotka se suorittaa peräkkäin. Esimerkiksi Jenkins ensinnä hakee versiohallinnasta jobia varten tarvittavat tiedostot. Versiohallintaa käytetään hyväksi, jotta aina olisi viimeisin versio ajossa mukana. Kun

tarvittavat tiedostot on haettu, Jenkins aloittaa GCC-kääntäjälle komentorivikäskyjen antamisen. Testiloki tallennetaan versiohallintaan.

Toinen Jenkinsin job pollaa (tarkistaa toistuvasti) viiden minuutin välein, että onko versiohallintaan tehty muutoksia. Jos muutoksia havaitaan, Jenkins hakee viimeisimmät lokitiedostot ja aloittaa kääntämisen. Tässä jobissa Jenkins hakee testilokit ja kaikki työkalun ohjelmistot versiohallinnasta. Tämän jälkeen Jenkins syöttää työkalulle testilokit automaattisesti. Jenkins-job näyttää etusivullaan testitulokset verkko- ja tekstiversioina

5 POHDINTA

Työn tavoitteena oli saada ohjelmistovaatimukset ja testitapaukset yhteen ja siinä onnistuin mielestäni hyvin. Työn minimivaatimuksena voitiin pitää sitä, mistä saisi pelkän tekstipohjaisen raportin ulos. Tämä vaatimus ylitettiin, koska lisäksi rakensin dynaamisen verkkosivun, jonka sisältö mukautui ohjelmistovaatimusten mukaan.

Tekstiraportin teko olikin suhteellisen helppo, kunhan oppi käyttämään Python-kieltä sujuvasti. Verkkoversio lisättiin, koska aikaa oli paljon ja koettiin, että olisi helppokäyttöisempi ja informatiivisempi.

Työssä käytettiin kahta ohjelmointikieltä. Minulla ei ollut aiemmin juuri kokemusta näistä ohjelmointikielistä. Python oli syntaksinsa takia helposti omaksuttava, eikä se JavaScriptikään osoittautunut juuri sen hankalammaksi. Kielten maailmaan pääsin helposti, mutta suurimmat haasteet liittyivät näin suuren projektin hallitsemiseen itsenäisesti.

Työ tuli valmiiksi, tai niin valmiiksi kuin sen voi tehdä. Aina tulee uusia ideoita, mutta tällä hetkellä työtä on ruvettu pikkuhiljaa ottamaan käyttöön ja mahdollisesti sitä tullaan käyttämään laajemmaltikin.

Tämän työn sain tehdä hyvin pitkälti omalla tavallani. Sain myös oppia asioita kantapään kautta, mikä ei missään nimessä ole huono asia, koska ne asiat ovat iskostuneet päähäni, enkä toista kertaa samoja virheitä tekisi.

Melkeinpä hyödyllisimpiä oppimiani asioita oli se, että ohjelmasta kannattaa tehdä alusta alkaen siisti ja helppolukuinen. Tämä voi kuulostaa itsestään selvältä, mutta on vaikea toteuttaa, jos ohjelman rakentaminen lähtee käyntiin suoraan kirjoittamalla koodia. Jos pyrkii saamaan nopeasti toimivan prototyypin ohjelmasta, täytyy palata takaisin ja selkeyttää ohjelmarakennetta. Tämä on työlästä, eikä välttämättä miellyttävintä, mutta todella opettavaista. Tuntuu, että jokaisella kerralla ohjelmaa tarvitsee aina vain vähemmän siistimistä

Funktiot tai metodit eivät saisi olla liian pitkiä. Pitkät funktiot olisi jo pelkästään luettavuuden takia pätkittävä lyhemmiksi funktioksi, joita toinen funktio esittää sisällään.

Mielestäni nimeäminen olisi tärkeä asia sisäistää. Jos muuttujat ja funktiot nimetään hyvin ja kuvaavasti, ohjelman kommentointi tulee melkein tarpeettomaksi. Usein, varsinkin opiskelija unohtaa, että samaa koodia saattaa lukea ja käyttää joku muukin. Tällöin ohjelman tulisi olla selkeää luettavaa, eikä ohjelmoijan tarvitsisi liikaa kuluttaa aikaa koettaessaan ymmärtää koodia. Mielestäni hyväksi ohjelmoijaksi ei tee pelkästään se, että tietokone ymmärtää koodia, vaan se, että toinen ihminen osaa myös lukea sitä.

Kirjoittamassani ohjelmakoodissa näkee hyvin kehittymiseni ohjelmoinnissa. Vaikka ohjelmaa on muokattu ja kirjoiteltu ympäriinsä, näkee missä jotkin koodit ovat tehty turhankin vaikealla tavalla, kun helpompi keino olisi olemassa. Myös on nähtävillä, missä helpommat ja tehokkaammat keinot on otettu käyttöön.

Jatkokehityksenä näkisin ohjelman tekemisen vieläkin tiiviimmäksi ja tehokkaammaksi refaktoroimalla. Lisäksi ohjelman helppokäyttöisyyttä voisi lisätä. Projektin Jenkins-sivuilla voisi suoraan nähdä, että kannattaako vaatimuksien tilaa tarkastella tarkemmin.

LÄHTEET

1. VanderVoord Mark, Karlesky Mike, Williams Greg. 2015. UNITY UNIT TESTING FOR C (ESPECIALLY EMBEDDED SOFTWARE).
   Saatavissa: http://www.throwtheswitch.org/unity. Hakupäivä 16.4.2017.
2. unittest — Unit testing framework. Python Software Foundation.
   Saatavissa: https://docs.python.org/2/library/unittest.html. Hakupäivä 23.9.2016
3. Dooley John. 2011. Software Development and Professional Practice.
4. Korpela Jukka. 2006. Perustietoa Python-ohjelmointikielestä.
   Saatavissa: https://www.cs.tut.fi/~jkorpela/python/. Hakupäivä 21.9.2016
5. About Jenkins. CloudBees.
   Saatavissa: https://www.cloudbees.com/jenkins/about. Hakupäivä 15.4.2017.
6. Miscellaneous operating system interfaces. Python Software Foundation.
   Saatavissa: https://docs.python.org/2/library/os.html. Hakupäivä 16.4.2017.