OULUN AMMATTIKORKEAKOULU

Janne Aikioniemi

MOBIILIPELIN ESITTELYVERSION SUUNNITTELU JA TOTEUTUS HAMSTERSCAPE-PELISARJAAN

MOBIILIPELIN ESITTELYVERSION SUUNNITTELU JA TOTEUTUS HAMSTERSCAPE-PELISARJAAN

Janne Aikioniemi
Opinnäytetyö
Kevät 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä: Janne Aikioniemi
Opinnäytetyön nimi: Mobiilipelin esittelyversion suunnittelu ja toteutus Hamsterscape-pelisarjaan
Työn ohjaaja: Pertti Heikkilä
Työn valmistumislukukausi ja -vuosi: 2015
Sivumäärä: 40

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa esittelyversio pelistä (demo) omaperäisellä idealla ja mekaniikalla Meizi Games Oy:lle. Meizi Games on julkaissut aikaisemmin Hamsterscape-pelisarjaa, jossa hamsteriatleetti kilpailee eri lajeissa. Peli tultaisiin mahdollisesti julkaisemaan Android- ja Windows Phone -alustoille jossain vaiheessa. Pelin aiheeksi muotoutui tukinheittopeli, jonka päähahmona toimi sarjan aiempien pelien mukaan hamsteri.

Työtä rajattiin siten, että esittelyversion tulisi sisältää vähintään toimiva pelimekaniikka ja tuloslistaus verkosta laitteelle. Pelikehityksessä käytettiin Unity3D:n testiversiota 4.6 b20 ja ohjelmointikielenä C#:a. Testiversioon oli julkaistu päivitys, joka toi uudet graafisen käyttöliittymän tekoon tarkoitetut työkalut. Työkalun käytön opettelu oli vahvin argumentti testiversion käyttöön.

Työn teoreettinen osuus käsittelee hyvin pintapuolisesti päätyökalun, Unity3D:n, ominaisuuksia, ulkoasua ja käyttöä. Kaikki informaatio on hankittu verkkolähteistä, joista suurin osa on Unity Technologiesin omaa dokumentaatiota.

Empiirinen osuus käsittelee projektin etenemistä prototyypin aloittamisesta lähes toimivan esittelyversion valmistumiseen. Pelisuunnittelua ei alkutiedoista poiketen juurikaan tarvinnut lopulta tehdä, koska idea oli tarpeeksi yksinkertainen, mutta jatkossa tämä pitäisi ottaa huomioon.

Toteutuksena saatiin valmiiksi toimiva esittelyversio pelistä. Peliin jäi kuitenkin ajan loppumisen vuoksi paljon virheitä ja tulee vaatimaan paljon jatkokehitystä, jotta siitä saataisiin julkaisukelpoinen.

Asiasanat: mobiilipelit, peliohjelmointi, Unity3D, Unity, Meizi Games

ABSTRACT

Oulu University of Applied Sciences
Bachelor's degree of information technologies, software development

Author: Janne Aikioniemi
Title of thesis: Designing and developing mobile game demo for Hamsterscape series
Supervisor: Pertti Heikkilä
Term and year when thesis was submitted: Spring 2015
Pages: 40

The goal of this thesis was to design and develop a game demo with distinctive game mechanics and original idea for Meizi Games Ltd. Meizi Games has already released games under the name of Hamsterscape where a hamster athlete competes in different sports. The main idea of the game took the form of log rolling which is done by the same athlete as in the previous games.

Thesis was limited so that the demo should include at least functional game mechanics and high score listing from network to mobile device. Development was done on Unity3D 4.6 beta version and the programming language used was C#. Unity had released new graphical interface creation tools and learning to use that was the main argument for using risky test version.

The theoretical part is about the Unity3D features, appearance and usage. All of the information is acquired from network sources. Most of the information is directly from Unity Technologies' web page and it's their own documentation. The empirical part deals with the progress of the development process.

By the end of the project, there was a functional game demo. There were also numerous bugs which need to be addressed if the game development continues and the plan is still to get the game released.

Keywords: mobile games, game development, Unity3D, Unity, Meizi Games

SISÄLLYS

SANASTO

Collider	Collider (törmäytin) määrittää peliobjektien fyysiset rajat
Git	Versiohallintaohjelmisto
Komponentti	Elementin osa, esimerkiksi skripti tai collider
Prefab	Uudelleenkäytettävä elementti
Repositorio	Paikka, jonne voidaan tallettaa dataa
Renderöinti	Digitaalisen tiedon muuntaminen näytölle sopivaan esitysmuotoon

1 JOHDANTO

Opiskeltuani viimeisen vuoden Oulu Game Labissa päätin, että opinnäytetyöni tulee käsittelemään pelikehitystä tavalla tai toisella. Päädyin kyselemään opinnäytetyöpaikkaa Meizi Gamesilta ja heiltä löytyi aiheita ja ideoita joka lähtöön. ”Meizien” Hamsterscape Olympics on tällä hetkellä sarja urheilupelejä, joissa päähenkilönä toimii hamsteri nimeltään Henry. Henry on kilpaillut tähän mennessä ainakin kolmiloikassa, juoksussa ja moukarinheitossa. Tämän opinnäytetyön myötä Henry pääsi kokeilemaan myös tukinheittoa.

Opinnäytetyössä suunniteltiin ja toteutettiin mobiilipeli, jonka mekaniikkaa on lainattu 80–90-lukujen urheilupeleistä ja sovitettu mobiililaitteen kosketusnäytölle. Pelikehitys tehtiin Unity3D:llä ja ohjelmointikielenä käytettiin C#:a, joista työryhmällä oli työn aloittamiseen mennessä jo jonkin verran aiempaa kokemusta. Tähän työhön valittiin käytettäväksi Unity3D:n testiversio 4.6b, koska siihen oli kehitetty pitkään odotettu graafisen käyttöliittymän tekoon tarkoitettu työkalu. Testiversion käytössä on tietysti aina riskinsä ja niiltä ei vältytty tässäkään tapauksessa.

Työssä esitellään aluksi Unity3D:tä yleisesti ja kerrotaan hiukan sen ominaisuuksista ja käyttöliittymästä. Tämän jälkeen käydään läpi projektin etenemistä lähes kronologisessa järjestyksessä. Lopuksi arvioidaan työn onnistumista ja vaihtoehtoisia ratkaisuja sekä esitetään joitain ajatuksia pelin jatkokehitystä ajatellen.

2 UNITY3D-PELINKEHITYSYMPÄRISTÖ

Unity3D (Unity) on Unity Technologies -nimisen yrityksen kehittämä järjestelmäriippumaton pelinkehitysympäristö. Se sisältää pelimoottorin ja kehitysympäristön. Unityä käytetään videopelien kehitykseen useille alustoille, kuten konsoleille (Xbox, PlayStation, Wii), mobiilialustoille (iOS, Android, Windows Phone), työpöytäympäristöihin (PC, Linux, OS X) ja web-sivuille. (1.)

Unityn kehitystyö aloitettiin vuonna 2001 ja sen ensimmäinen versio (Unity 1) julkistettiin Applen järjestämässä kehittäjäkonferenssissa (*WWDC*) vuonna 2005. Ohjelmiston kehitys jatkui ja vuonna 2009 siitä julkaistiin ilmainen versio. Samana vuonna yritys kasvoi kolminkertaiseksi. (2, linkit Milestones -> 2001, 2005, 2009.)

Vuonna 2010 julkaistiin *Unity Asset Store* ja Unityn kolmas versio. Asset Store on markkinapaikka ilmaisille ja maksullisille kolmannen osapuolen liitännäisille, ohjelmistokokonaisuuksille, kuville, 3D-malleille ja äänille, joita voidaan liittää Unityllä tehtäviin projekteihin. (2, linkki Milestones -> 2010.)

Vuonna 2013 julkistettiin Unityn 2D-työkalut sekä *Unity Cloud* -pilvipalvelut. Tällä hetkellä Unity on käytetyin pelinkehitysympäristö ja sillä on yli kolme miljoonaa rekisteröitynyttä käyttäjää. (2, linkki Milestones -> 2013.)

2.1 Unity-lisenssit ja käyttöehdot

Unitystä on tällä hetkellä saatavilla maksullinen (*Pro*) ja ilmainen versio. Näiden lisäksi mobiilialustoille on olemassa omat maksulliset versionsa. Ilmaisesta versiosta puuttuu sisäänrakennettuja ominaisuuksia, kuten *Profiler*, *SpritePacker*, *LOD*-tuki, audion suodatus, videon toisto ja suoratoisto. Ilmaisversiolla tehdyt pelit saa julkaista ilmaiseksi ilman lisenssimaksuja, mutta mikäli liikevaihto ylittää 100 000 dollaria vuodessa, täytyy yrityksen ostaa Pro-lisenssi. (3, linkki General.)

2.2 Unityn ominaisuuksia

Pelimoottorin skriptaus on rakennettu *Mono*-ohjelmistokehitysympäristön päälle. Ohjelmointikielinä Unityssä voidaan käyttää *C#:a*, *JavaScriptin* kaltaista *UnityScriptiä* ja *Boota*. Unity3D:n asennuspaketin mukana tulee *MonoDevelop*-ohjelmointiympäristö, joka tukee kaikkia edellä mainittuja kieliä, mutta myös muita ympäristöjä, kuten Microsoftin Visual Studiota, voi käyttää halutessaan. (1.)

Unityn grafiikkamoottori tukee *Direct3D*- ja *OpenGL*-rajapintoja. Avoimen lähteen *OpenGL*-rajapintaa voidaan käyttää lähes kaikille alustoille, mutta mobiilialustoille on suunniteltu myös oma rajapintansa *OpenGL ES*. *Direct3D*:tä käytetään Windows- ja Xbox-järjestelmille. (1) Unity tukee useita grafiikkaohjelmien tiedostotyyppejä, kuten *blend, max ja fbx*. Niitä pystyy käyttämään suoraan tai muuntamaan Unityn tukemaan muotoon. Suoraan käytettäviä tiedostomuotoja voidaan muokata ajonaikaisesti ilman Unityn sulkemista. (5.)

NVIDIA PhysX -fysiikkamoottori on Unityssä sisäänrakennettuna ja se mahdollistaa skaalautuvan fysiikkamallinnuksen. PhysX:n voi skaalata toimimaan niin kevyissä mobiilipeleissä kuin raskaammissa tietokonepeleissä, minkä vuoksi se soveltuu hyvin järjestelmäriippumattomaan pelikehitykseen. (4, linkki PhysX SDK.)

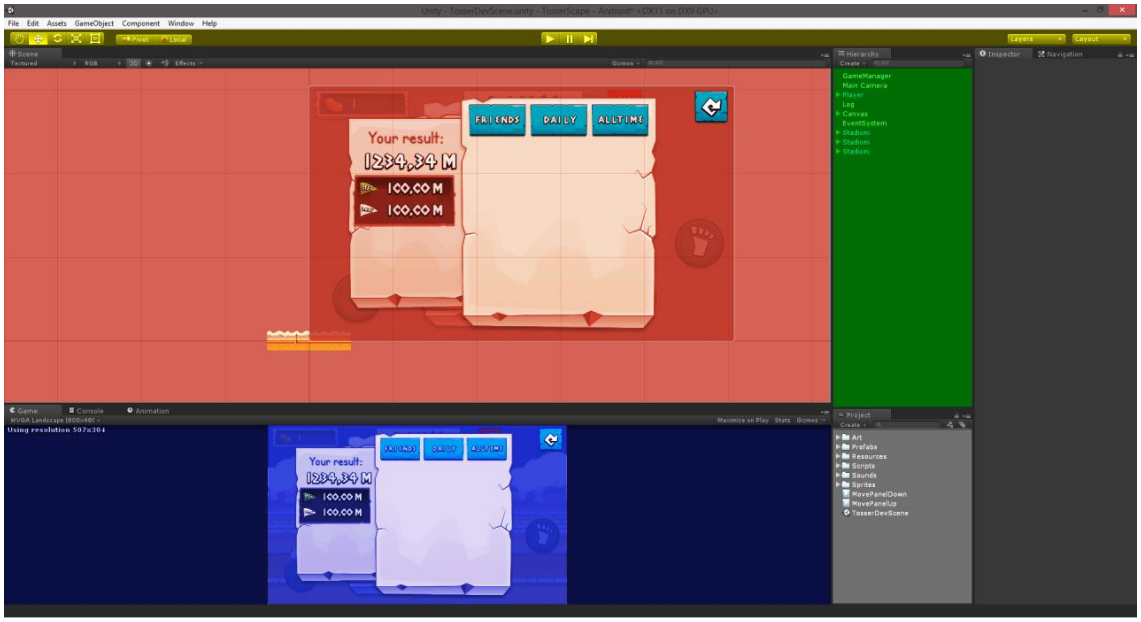
2.3 Unityn käyttöliittymä

Käyttöliittymä koostuu viidestä päänäkymästä: kohtausnäkymästä (*Scene view*), pelinäkymästä (*Game view*), projektinäkymästä (*Project panel*), hierarkianäkymästä (*Hierarchy panel*) ja tarkastelunäkymästä (*Inspector panel*). Päävalikon Window-valikosta saadaan näkyviin muitakin näkymiä, kuten konsoli- ja animaatioikkunat, joita projekteissa yleensä tarvitaan. Käyttöliittymän asettelu on muokattavissa käyttäjän tottumusten mukaiseksi siirtämällä tai lisäämällä ikkunoita haluttuihin paikkoihin tai käyttämällä Unityn valmiita asetelmia.

Kuvassa 1 käyttöliittymän osat on väritetty seuraavasti:

- työkalurivi keltaisella
- kohtausnäkymä punaisella

- pelinäkymä sinisellä
- hierarkia vihreällä
- projektinäkymä valkoisella
- tarkastelunäkymä tummanharmaalla.



KUVA 1. Unityn käyttöliittymä.

Kohtausnäkymä on eräänlainen interaktiivinen hiekkalaatikko, jossa objektit sijoitellaan halutuille paikoille. Objekteja pystyy siirtämään, skaalaamaan, kääntämään ja tarkastelemaan zoomaamalla sisään ja ulos. Kohtausnäkymän yläosassa on kontrollipalkki, josta voidaan ohjata kohtausnäkymää. Kohtausnäkymän palkin kautta voidaan ottaa valot ja äänet pois käytöstä tai vaihtaa 3D-näkymän ja 2D-näkymän välillä. (13.)

Pelinäkymä renderöidään pelin kameroiden kautta eli pelinäkymä näyttää sen, miltä peli tulisi näyttämään. Pelissä on siis oltava vähintään yksi kamera. Myös pelinäkymällä on oma kontrollipalkkinsa, josta voidaan ohjata näkymän ominaisuuksia. Pelinäkymässä voidaan näyttää statistiikkaa suorituskyvystä ja piirtokutsuista painamalla Stats-painiketta tai pakottaa resoluutio halutuille arvoille.

Projektinäkymässä hallitaan projektin tiedostoja. Näkymän voi jakaa kahteen osaan halutessaan, jolloin kansiorakenne ja tiedostot näkyvät eri osissa. Näkymän yläosassa on palkki, jonka avulla voidaan luoda sekä hakea tiedostoja ja kansioita.

Hierarkianäkymä sisältää kaikki nykyisessä kohtauksessa olevat objektit. Objekteja voidaan valita ja parentoida tässä näkymässä. Mikäli objekteja on paljon, voidaan tässäkin näkymässä käyttää kontrollipalkin hakutoimintoa.

Tarkastelunäkymä näyttää valitun objektin komponentit. Näkymän kautta voidaan muokata objektin toiminnallisuutta. Muokkaus onnistuu myös ajon aikana, mutta nämä muutokset eivät välttämättä ole pysyviä.

3 PROJEKTIN ETENEMINEN

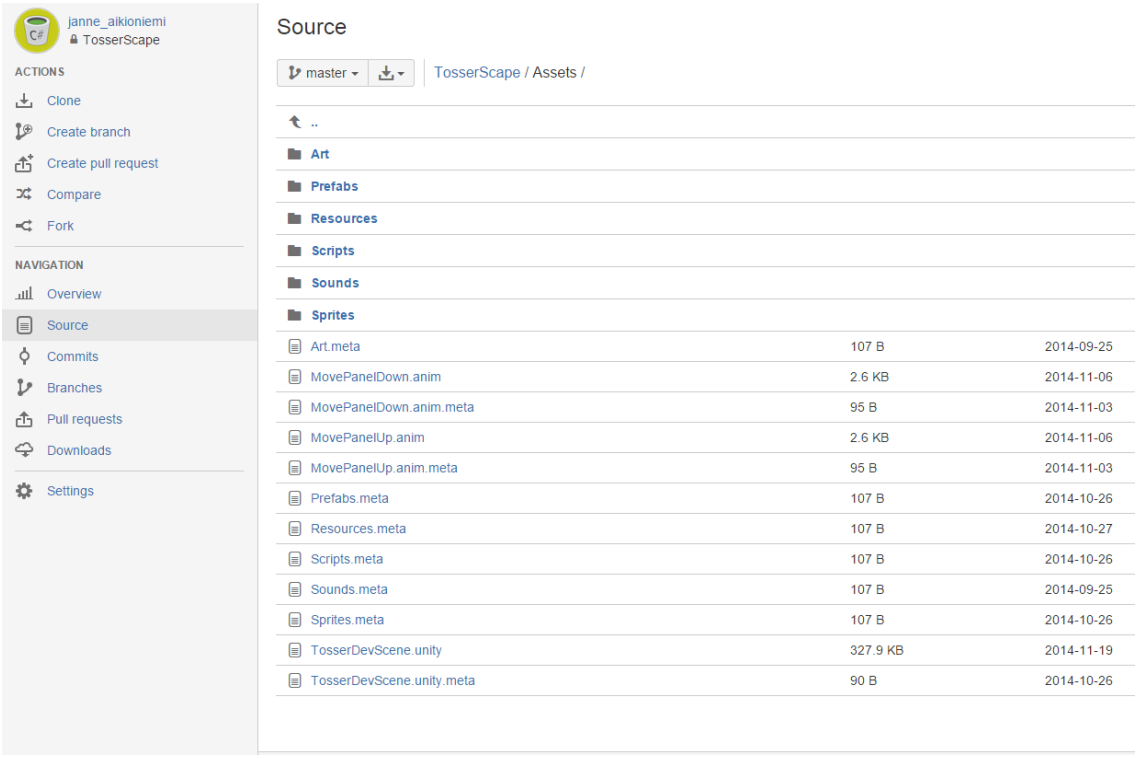
Projekti aloitettiin aloituspalaverilla 6.10.2014, jolloin sovittiin työn laajuudesta ja aikataulusta. Tavoiteaikataulu oli 6.10.–21.11.2014 ja työn tavoitteeksi muotoutui suunnitella ja toteuttaa Hamsterscape Olympics -pelisarjaan peli omaperäisellä idealla sekä yksinkertaisella pelimekaniikalla. Pelin ei tarvinnut olla valmis opinnäytetyön puitteissa, mutta siinä täytyi olla pelimekaniikka ja tuloslistaus verkosta laitteelle. (Lähtötietomuistio, liite 1.)

Lyhyen suunnittelupalaverin jälkeen ideaksi muodostui tukinheittopeli. Peli ei yrittäisi pitäytyä oikean tukinheiton säännöissä, vaan sitä yksinkertaistettaisiin niin, että tarkoituksena olisi saada pelihahmolle mahdollisimman kova juoksuvauhti ja heittää tukki, ennen kuin se kaatuu tai lipeää otteesta. Pisteet laskettaisiin pelkästään heiton pituuden mukaan. Aikaisempien pelien mukaan heiton pituuteen vaikuttaisivat lisäksi Ateenan tuulen suunta ja nopeus, jotka päivitetään internetistä tietyin väliajoin, sekä ajan ja onnistumisten myötä kerääntyvät voimapähkinät.

Aluksi mietittiin pelin tekemistä Triple Jump -pelin pohjalle, jolloin siihen tarvitsisi toteuttaa pelkästään pelattava osuus. Tämä vaikutti kuitenkin liian helpolta ratkaisulta, joten projekti aloitettiin puhtaalta pöydältä. Ajatus Unityn testiversion käyttämisestä otettiin positiivisesti vastaan, joten se valjastettiin käyttöön sen enempää aiheeseen syventymättä.

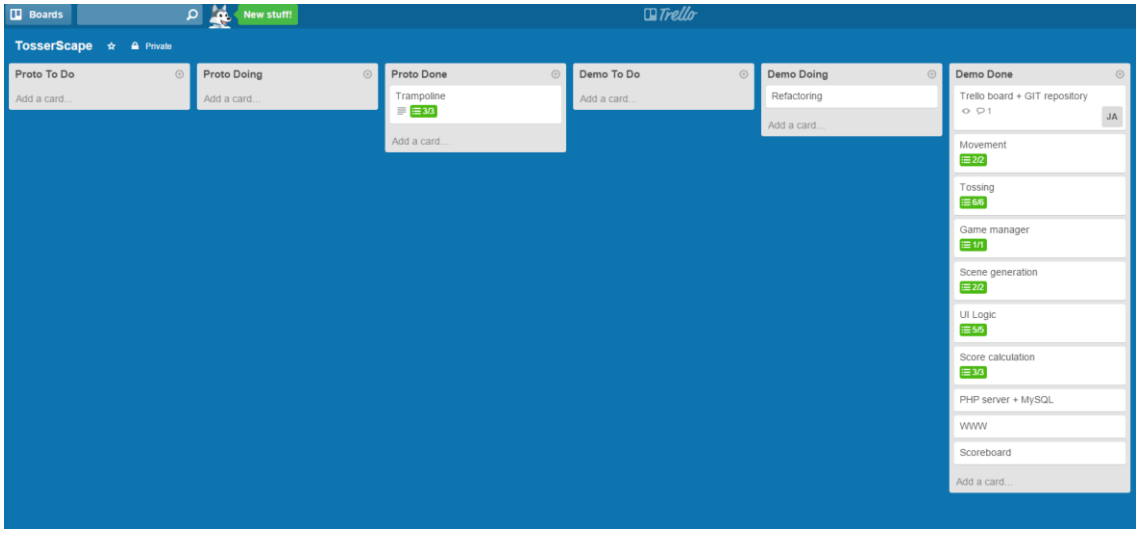
3.1 Projektin aloitus

Ensimmäisenä luotiin uusi repositorio BitBucket-palveluun. BitBucket tarjoaa ilmaisia ja maksullisia versiohallintapalveluita verkon kautta projekteille, jotka käyttävät joko Mercurialia tai Gitiä (12). Tässä tapauksessa käytettiin Git-versiohallintaa. Repositorioon ladattiin Triple Jumpista viimeisin versio, mutta työn alettua siitä jätettiin vain muutamia skriptejä sekä grafiikkaa. Kuvassa 2 näkyvä kansiorakenne on projektin loppuvaiheelta, joten se ei kuvaa tarkalleen alkutilannetta.



KUVA 2. BitBucket-palvelun sisältönäkymä.

Seuraavaksi luotiin Trello-projektinhallintapalveluun (kuva 3) uusi projekti. Tätä käytettiin enemmän ideapankkina kuin varsinaiseen projektinhallintaan.



KUVA 3. Trellon projektinäkymä.

Ennen prototyypin aloittamista vielä mietittiin, kuinka pelistä saataisiin mielenkiintoisempi. Ajatus pelkästä tukinheitosta, ilman jotain hauskaa koukkua, ei kuulostanutkaan tarpeeksi hyvältä. Kaikenlaisia ideoita heiteltiin puolin ja toisin, mutta lopuksi päädyttiin lisäämään peliin trampoliini. Tämän myötä peli muuttuisi niin, että pelaaja heittää tukin ja jatkaa juoksemista trampoliinia samalla työntäen. Mikäli pelaaja saa trampoliinin tukin alle, tukki lähtee uuteen ilmalentoon. Pelaaja voisi käyttää trampoliinia tähän tarkoitukseen kolme kertaa, jonka jälkeen se hajoaisi ja tukin osuessa maahan mitattaisiin lopputulos.

3.2 Prototyyppi

Varsinainen kehitystyö aloitettiin prototyypin tekemisellä. Yksinkertaisella prototyypillä haluttiin testata, toimiiko peli-idea, muutetaanko ideaa vai otetaanko työn alle jokin muista ideoista. Prototyyppiin oli tarkoitus tehdä peliympäristö, hahmon ohjaus ja tukin heittäminen käyttäen fysiikkamoottoria sekä trampoliinin perustoiminnallisuus. Myös graafisen käyttöliittymän tekoa alettaisiin opetella.

Peliympäristö siirrettäisiin Triple Jumpista suoraan ja siitä karsittaisiin tässä vaiheessa ylimääräiset skriptit ja peliobjektit pois. Hahmon ohjaus perustuisi vanhojen urheilupelien ohjausmekaniikkaan, joka sovitettaisiin kosketusnäytölle. Vanhoissa, esimerkiksi Commodore 64:n aikaisissa, urheilupeleissä ohjaimen sauvaa heilutettiin rytmikkäästi edestakaisin tai ohjaimen painikkeita naputettiin mahdollisimman nopeasti ja tämän jälkeen ajoitettiin jokin tietty toiminto oikea-aikaisesti. Tässä fyysisen ohjaimen korvaisi kosketusnäytölle määritetyt painikkeet.

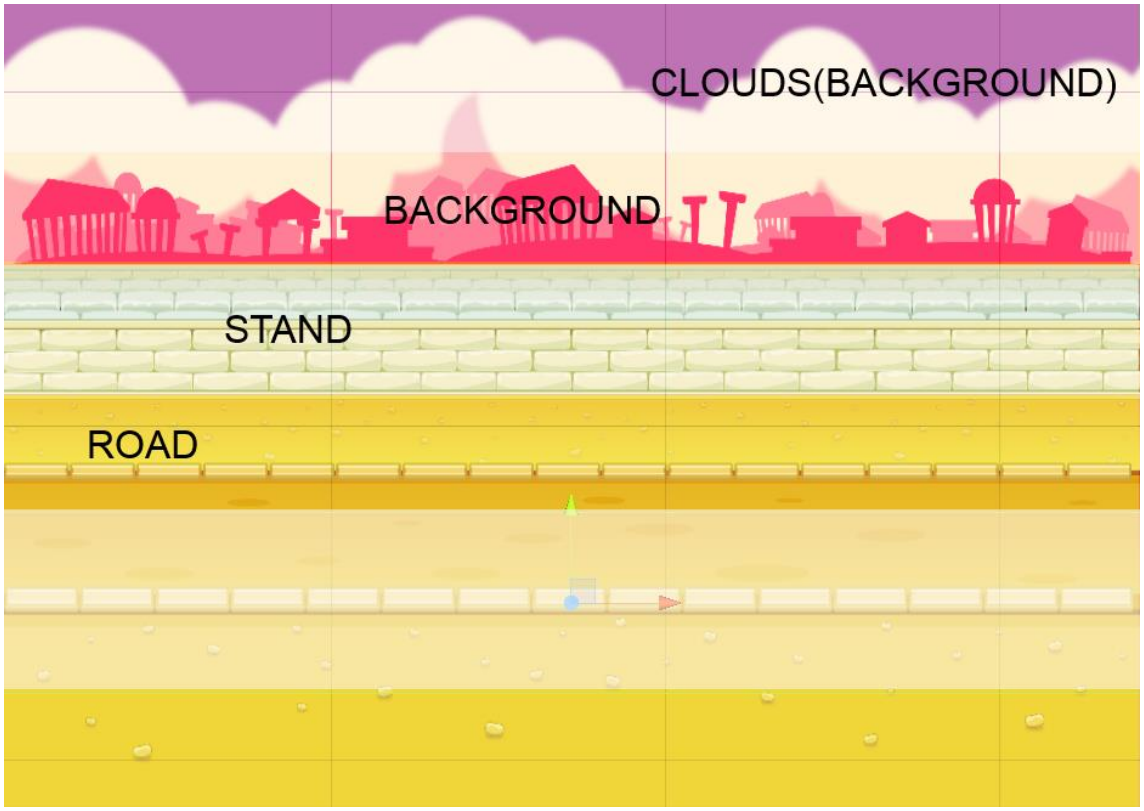
Trampoliinille tehtäisiin yksinkertainen toiminnallisuus, joka reagoisi tukin osumaan ja tukille arvottaisiin uusi, eteenpäin suuntaava lähtökulma ja voima.

3.2.1 Pelin ympäristö

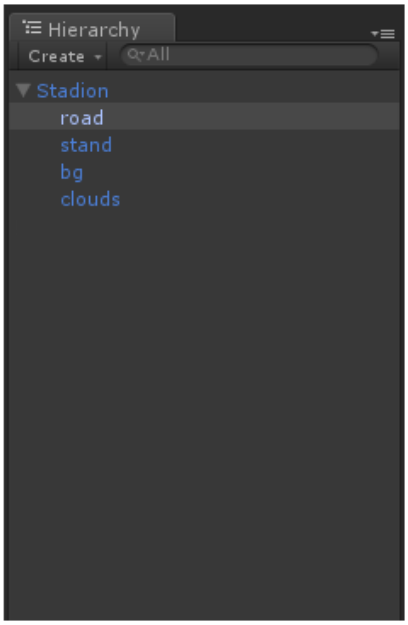
Pelissä käytetään kolmea identtistä stadion-objektia, joiden sisältöä siirretään sen mukaan, millä kohdalla kamera on menossa. Kameran näkyvistä poistunut

osa siirretään kameran edelle x-akselin suunnassa. Näin pelaaja voi periaatteessa juosta loputtomasti eteenpäin. Stadion-objekti pystyttiin siirtämään suoraan Triple Jumpista, koska siitä oli tehty *prefab*. Prefabit ovat uudelleen käytettäviä objekteja. Kaikkien prefabista luotujen objektien ominaisuudet voidaan muuttaa yhdellä kertaa muokkaamalla itse prefabia. Näin jokaisen objektin ominaisuuksia ei tarvitse erikseen säätää.

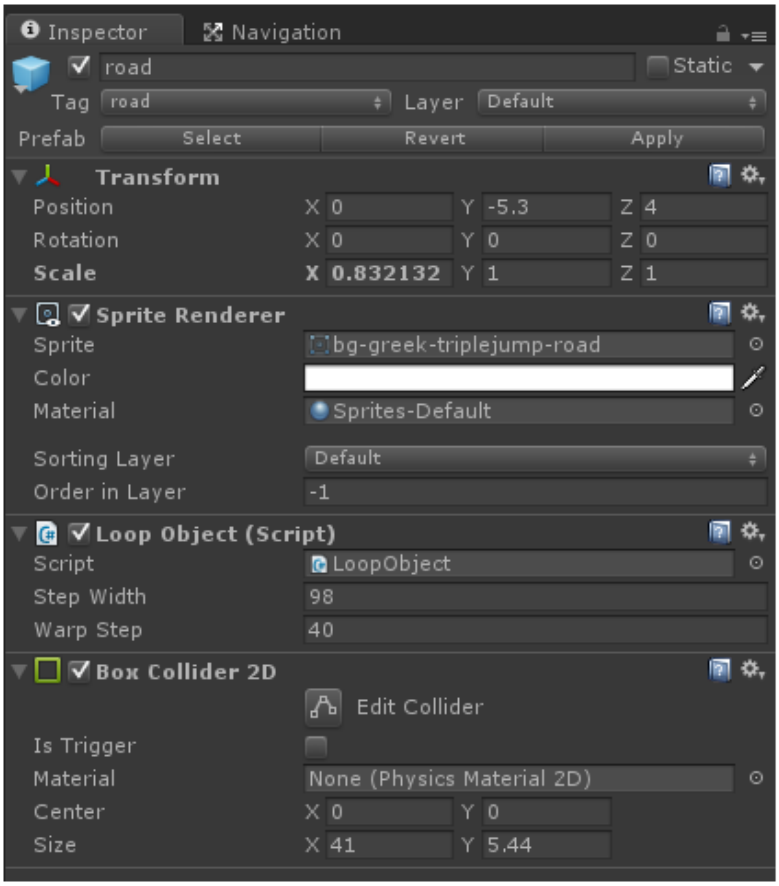
Stadion-prefab muodostuu neljästä objektista: road, stand, bg ja clouds (kuva 4 ja kuva 5). Jokaisessa objektissa on *Sprite Renderer* -komponentti sekä *LoopObject*-skripti. Road-objektissa on lisäksi *Box Collider 2D* -komponentti, joka estää muita *colliderin* omaavia objekteja menemästä siitä läpi (kuva 6).



KUVA 4. Pelin ympäristö.



KUVA 5. Stadion-prefab hierarkianäkymässä.



KUVA 6. Road-objektin komponentit tarkastelunäkymässä.

2D-kuvan sisältö haetaan bittikarttakuvasta ja määritetään tietylle *Sprite*-luokalle (6). Sprite-luokkaa voidaan tämän jälkeen käyttää Sprite Renderer -komponentissa, joka on tarkoitettu 2D-kuvan renderöintiin (7). Tässä tapauksessa kaikki kuvassa 4 näkyvä grafiikka on alun perin tallennettu yhteen bittikarttakuvaan ja Unity3D:ssä pilkottu Spriteiksi. Mikäli pelin grafiikkaa saadaan tallennettua yhteen ja samaan bittikarttaan, bittikarttaa kutsutaan *atlakseksi* (8). Atlaksen käyttö vaikuttaa suorituskykyyn vähentämällä piirtokutsuja (draw calls). Mikäli bittikarttakuva on pakattavissa, eli sen resoluutio on kahden potenssissa (esim. 1024 x 1024), ohjelmistopaketin kokoa saadaan pienennettyä, mikä on tärkeää varsinkin mobiililaitteiden ohjelmistoja suunniteltaessa.

LoopObject-skripti (kuva 7) on vastuussa peliobjektien siirrosta. Skriptille on määritetty kaksi float-tyyppistä muuttujaa *StepWidth* ja *WarpStep*. Mikäli objektin, esimerkiksi road, paikka on x-akselilla pienempi kuin kameran paikka vähennettynä WarpStep-muuttujan arvolla, siirretään objekti muuttujan StepWidth arvon verran oikealle. Tässä tapauksessa toimivat arvot olivat kuten kuvassa 6.

```
using UnityEngine;
using System.Collections;

public class LoopObject : MonoBehaviour {

    private GameObject MyGameObject;
    private Transform MyTransform;

    private Transform MainCamera;
    public float StepWidth;
    public float WarpStep;

    void Start () {
        MainCamera = Camera.main.transform;
        MyGameObject = gameObject;
        MyTransform = transform;
    }

    void Update () {
        if(MyTransform.position.x < (MainCamera.position.x - WarpStep)) {
            Vector3 newPosition = new Vector3(MyTransform.position.x + StepWidth, MyTransform.position.y, MyTransform.position.z);
            MyTransform.position = newPosition;
        }
    }
}
```

KUVA 7. LoopObject-skripti.

3.2.2 Kamera

Pelissä käytetään ortografista kameraa. Kameran ominaisuudet siirrettiin Triple Jumpista ja muokattiin peliin sopivaksi. Unity3D:n lisäämien peruskomponenttien lisäksi kameraan lisättiin *SmoothCamera2D*-skripti, jonka tarkoituksena on sujuvoittaa kameran liikettä. Kamera seuraa tukkia, ikään kuin tukki vetäisi sitä perässään kuminauhalla.

SmoothCamera2D-skripti (kuva 8) käyttää Vector3-luokan metodia SmoothDamp, joka toimii liikuttamisen vaimentajana. Se tarvitsee paikan laskemiseen objektin nykyisen paikan ja seuraavan paikan koordinaatin x-akselilla, nopeuden sekä vaimennukseen kuluvan ajan. (9.)

```
public class SmoothCamera2D : MonoBehaviour
{
    public float DampTime = 0.15f;
    public Transform Target;

    private Vector3 Velocity = Vector3.zero;
    private Transform MyTransform;

    void Start()
    {
        MyTransform = transform;
    }

    void FixedUpdate ()
    {
        if (Target)
        {
            Vector3 point = camera.WorldToViewportPoint(Target.position);
            Vector3 delta = Target.position - camera.ViewportToWorldPoint(new Vector3(0.35f, 0.15f, point.z));
            delta.y = 0f;
            Vector3 destination = MyTransform.position + delta;
            MyTransform.position = Vector3.SmoothDamp(MyTransform.position, destination, ref Velocity, DampTime);
        }
    }
}
```

KUVA 8. SmoothCamera2D-skripti.

3.2.3 Pelihahmo

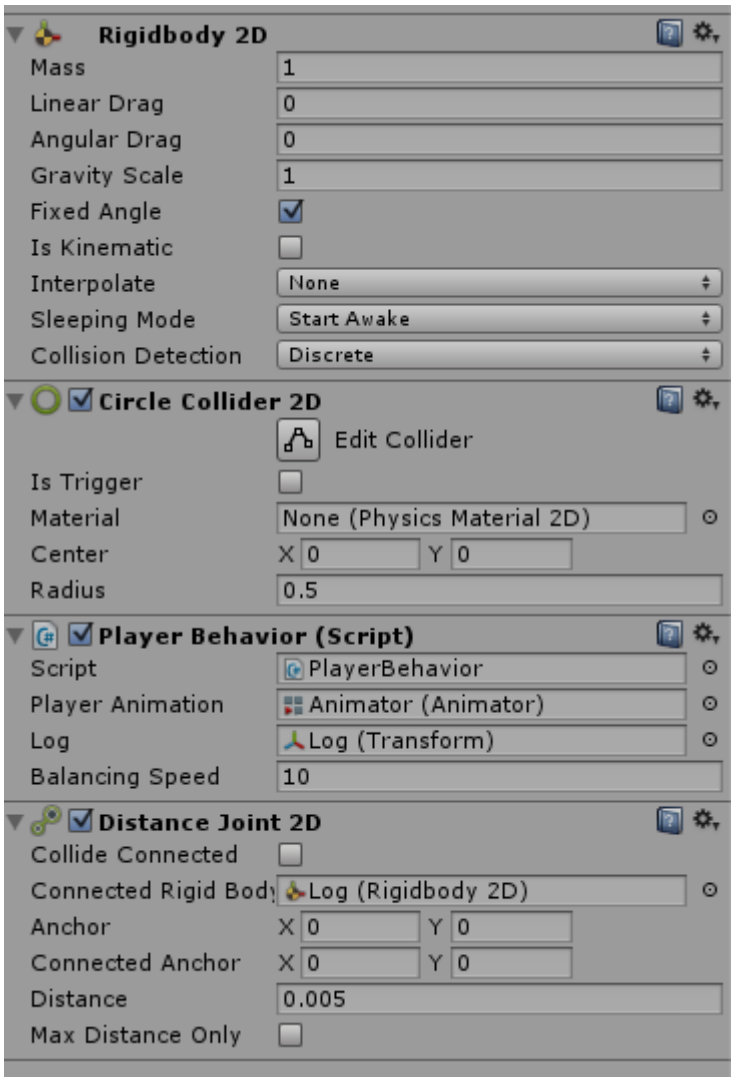
Pelihahmo koostuu Unityssä *Rigidbody 2D*-, *Circle Collider 2D*- sekä *Distance Joint 2D* -komponenteista ja *PlayerBehavior*-skriptistä. Lisäksi hahmolla on lapsiobjekti, jossa on määritetty animaatiosiirtymät *Animator*-komponentilla. Pelihahmon toiminnallisuutta tehtäessä animaatioihin ei tarvinnut kiinnittää huomiota.

Rigidbody 2D -komponentti toimii samalla periaatteella kuin Rigidbody-komponentti, joten se mahdollistaa Unityn fysiikkamoottorin käytön peliobjektissa. Objekti, jolla on kyseinen komponentti, reagoi painovoimaan ja muihin voimiin sekä törmäyksiin mahdollisimman luonnollisella tavalla, fysiikkamoottorin rajoitukset huomioon ottaen. On hyvä huomata, että käytettäessä fysiikkamoottoria objektien koolla (scale) on merkitystä, mikäli haluaa pelifysiikasta luonnollisemman oloisen. Yksi Unityn määrittämä yksikkö (world unit) vastaa yhtä metriä, joten esimerkiksi ihminen voisi olla pituudeltaan kaksi yksikköä korkea. (10.)

Circle Collider 2D -komponentti (Kuvassa 9 sinisellä) antaa peliobjektille sen fyysisen muodon. Tässä tapauksessa se on ympyrä, koska törmäyksen tarkistusta pelihahmossa tarvitaan pelkästään siihen, että pelihahmo pysyisi pelikentällä eikä tippuisi road-objektista läpi.

Distance Joint 2D (kuvassa 9 vaalea piste sinisen renkaan sisällä) pitää kaksi Rigidbody 2D:n sisältävää peliobjektia määritetyn vakioarvon päässä toisistaan. Tässä tapauksessa toinen objekti on tukki ja sen etäisyys mahdollisimman pieni (kuva 10, Distance Joint 2D).

KUVA 9. Pelihahmo.



KUVA 10. Pelihahmon komponentit tarkastelunäkymässä.

3.2.4 Hahmon liikuttaminen

Pelihahmoa liikutetaan fysiikalla. Rigidbody 2D:lle lähetetään välitön voimaimpulssi kutsumalla Rigidbody 2D -luokan metodia AddForce. AddForce-metodille annetaan parametrina voima sekä voiman käyttötapa. Voima annetaan Vector2-tyyppisenä ja käyttötapa ForceMode2D-tyyppisenä. ForceMode2D:ssä on kaksi tapaa, joilla voimaa voidaan lisätä objektille: *Force* tai *Impulse.* Force-tapa lisää voimaa aikaa myöten ja Impulse välittömästi. (11) Kuvassa 11 on Move-metodin toteutus.

```
public void Move()
{
    _myRigidbody.AddForce(new Vector2(3.5f - _myRigidbody.velocity.x * 0.12f, 0.0f), ForceMode2D.Impulse);
}
```

KUVA 11. Move-metodi voiman lisäystä varten.

3.2.5 Tukki

Pelaajahahmo ”kantaa” tukkia, joka alussa pysyy suhteellisen hyvin hallussa, mutta nopeuden myötä se alkaa lipsua otteesta ja kaatua eteenpäin. Tukki koostuu Sprite Renderer-, Rigidbody 2D- ja Box Collider 2D -komponenteista sekä *LogBehavior*-skriptistä.

LogBehavior-skriptissä tarkastellaan ja ohjaillaan tukin käyttäytymistä. Mikäli pelaaja on lähtenyt liikkeelle ja on saavuttanut tietyn nopeuden, tukki alkaa lisätä vääntövoimaa eteenpäin pelaajan nopeuden ja asetetun kertoimen mukaisesti. Pituuden laskenta pitäisi aloittaa vasta heiton tapahtuessa, mutta tässä tapauksessa sitä aletaan laskea noin yhden metrin jälkeen, jotta pelaaja saisi edes jonkinlaisen tuloksen. Skriptissä havaitaan tukin törmäys maahan, jolloin lopetetaan pituuden mittaus ja ilmoitetaan *EventManager*-skriptille, että tulostaulu voidaan näyttää.

3.2.6 Tukin toiminta

Ennen kuin tukki on heitetty, se lukitaan Rigidbody 2D:n Fixed Angle -parametrillä. Tämän vuoksi tukki pysyy pystyasennossa siihen asti, kun on tarpeen alkaa lisätä voimia. Mikäli pelaaja ylittää asetetun nopeuden, tukilta poistetaan lukitus ja sille aletaan lisätä vääntöä Rigidbody-luokan AddTorque-metodilla (kuva 12). Vääntövoimaan vaikuttavat pelaajan nopeus sekä erikseen säädettävä kerroin. Vääntöä jatketaan, kunnes tukki osuu maahan tai pelaaja heittää tukin. Mikäli tukkia ei heitetä ja se osuu maahan, tapahtuu automaattinen heitto ja tulosikkuna näytetään.

Tässä vaiheessa ongelmana oli se, että mikäli pelaaja saa pidettyä nopeuden alle asetetun arvon, hän pystyy juoksemaan loputtomasti tukkia kantaen ja saavuttamaan näin parhaan tuloksen. Tätä korjattiin ensiksi poistamalla etäisyyden laskenta käytöstä ennen heittoa, ja lopuksi päädyttiin antamaan tukille vääntövoimaa tietyn matkan jälkeen riippumatta pelaajan nopeudesta.

Tukin heittäminen tapahtui testiversiossa välilyöntinäppäintä painamalla ja graafisen käyttöliittymän kautta se onnistuu heittopainikkeesta. Ohjelmallisesti näppäimen tai painikkeen painaminen aiheuttaa sen, että PlayerBehavior-skriptissä katkaistaan pelaajan ja tukin välinen side (kuva 13, TossLog) ja kutsutaan LogBehavior-skriptin metodia Toss. Toss-metodille annetaan parametrina pelaajan senhetkinen nopeus x-akselilla ja nopeus lisätään tukin voimaksi (kuva 13, Toss).

LogBehavior-skripti laskee tukin etäisyyttä aloituspaikasta nykyiseen paikkaan. Käyttäjälle näkyvä tulos saavutetaan, kun nykyisestä paikasta vähennetään aloituspaikka, tämä tulos kerrotaan kymmenellä ja pyöristetään lähimpään kokonaislukuun, joka vielä jaetaan sadalla (kuva 14). Tällä saavutetaan järkevissä rajoissa oleva tulos, mikäli yleensä käy järkeen, että hamsteri heittää tukkia.

```
void FixedUpdate()
{
    if (addTorque && !playerTossed && !hasTouchedGround)
    {
        _myRigidbody.AddTorque(-(_torque * _torqueMultiplier));
    }
```

KUVA 12. Vääntövoiman lisääminen.

```
public void TossLog()
{
    _playerTossed = true;
    DistanceJoint.enabled = false;
    logBehavior.Toss(_myRigidbody.velocity.x);
}


public void Toss(float velocity)
{
    if (!playerTossed)
    {
        _myRigidbody.gravityScale = 1;
        _myRigidbody.fixedAngle = false;
        _extraVelocity = velocity;
        _myCollider.enabled = true;

        isTossed = true;
        playerTossed = true;
    }
}
```

KUVA 13. Tukin heittämiseen liittyvät metodit.

```
private float CalculateDistance()
{
    _currentPosX = _myTransform.position.x;
    return Mathf.Round((_currentPosX - _startPosX) * 10) / 100;
}
```

KUVA 14. Etäisyyden mittaaminen.

3.2.7 Trampoliini

Trampoliinin tarkoituksena on pompauttaa tukki uudelleen ilmaan, ja pelaajan pitää pystyä työntämään sitä pelihahmolla eteenpäin. Trampoliini kestää tietyn määrän osumia ja hajoaa, jolloin pelaaja ei pysty enää työntämään sitä. Tarkoitus olisi lopullisessa versiossa se, että trampoliini muuttuu raskaammaksi jokaisen osuman myötä.

Trampoliini toteutettiin mahdollisimman yksinkertaisesti, jotta päästäisiin testaamaan, miten se sopii peliin. Trampoliini-objektiin lisättiin Sprite-, Box Collider 2D-

sekä Rigidbody 2D -komponentit. Lisäksi tarvittiin skripti, joka ohjaa trampoliinin toimintaa.

TrampolineBehavior-skripti laskee tukin osumat trampoliiniin, ja kolmannen osuman tullessa trampoliini pysähtyy ja sitä ei voida enää työntää. Osumat lasketaan siten, että OnCollisionEnter-metodissa lisätään hitCounter-muuttujan arvoa yhdellä aina, kun tukki-objekti osuu. Update-silmukassa seurataan osumia, ja mikäli kolmen osuman ehto täyttyy, kutsutaan TrampolineBreakdown-metodia, joka asettaa rigidbodyn massan niin suureksi, että pelaajan on käytännössä mahdotonta jatkaa trampoliinin työntämistä. Lopullisessa versiossa massaa lisätään jokaisella osumalla, joten TrampolineBreakdown-metodia ei luultavasti enää tarvittaisi.

Tukin osuessa trampoliiniin LogBehavior-skriptissä arvotaan uusi voima ja kulma, mihin tukki seuraavaksi lähtee. Tätä helpotettiin pelaajan kannalta siten, että tukin uusi lähtökulma on aina eteenpäin.

3.2.8 Valmis prototyyppi

Prototyypistä saatiin tehtyä suhteellisen hyvin toimiva paketti. Sitä testattiin Unityn editorissa, Windows Phone 8 -puhelimella ja muutamalla Android-tabletilla. Toisella Android-tabletilla pelin suoritus oli jähmeää ja tukki käyttäytyi aiempaan verrattuna oudosti. Asiaa selviteltiin jonkin aikaa ja lopulta päädyttiin kokeilemaan Timescale-arvon pienentämistä kyseiselle laitteelle, ja näin pelaamisesta saatiin siedettävämpää.

Trampoliini päätettiin jättää pelistä pois ainakin toistaiseksi. Se ei tuntunut oikein sopivan peliin, ja lopulliseen viilaamiseen kuluisi varmasti kohtuuttoman paljon aikaa. Yleinen mielipide oli, että peli sopisi hyvin Hamsterscape-pelisarjaan ja sen kehitystä kannattaisi jatkaa, joten pelistä alettiin tehdä laajempaa esittelyversiota.

3.3 Esittelyversio

Projekti siivottiin ennen kuin esittelyversion tekeminen voitiin aloittaa. Ylimääräisistä skripteistä, objekteista, prefabeista ja trampoliinista piti päästä eroon. Ohjelmakoodit käytiin läpi ja kaikki ylimääräinen koodi poistettiin. Siivoamisen jälkeen peliä vielä testattiin ja se toimi niin kuin aikaisemminkin.

Ensimmäisenä peliin oli saatava jonkinlainen käyttöliittymä. Käyttöliittymän asettelun pitäisi näyttää samalta kuin aikaisemmissa peleissä. Tehtäisiin ns. splash-ikkuna, joka näytetään ennen päävalikkoa ja sitä painamalla päästäisiin peliin. Mikäli peliä pelataan ensimmäisen kerran, näytettäisiin profiili-ikkuna. Päävalikko näyttäisi samalta kuin Triple Jumpissa, mutta toiminnallisuuksia ei tarvinnut alkaa toteuttamaan kuin pakollisin osin.

Peliin tarvittaisiin pelaajaprofiili, johon tallennetaan pelaajan syöttämä pelaajanimi ja maa, tulokset, kerääntyneiden pähkinöiden määrä sekä laitteen yksilöllinen tunniste. Käyttöliittymään pitäisi tehdä pelaajaa varten ikkuna, missä pystyisi syöttämään nimen sekä valitsemaan maan lippujen joukosta. Kun pelaajaprofiili on saatu toimimaan lokaalisti, se siirrettäisiin tietokantaan. Tietokantaa varten tehtäisiin PHP:llä palvelinohjelma, jolla kyselyt tehtäisiin.

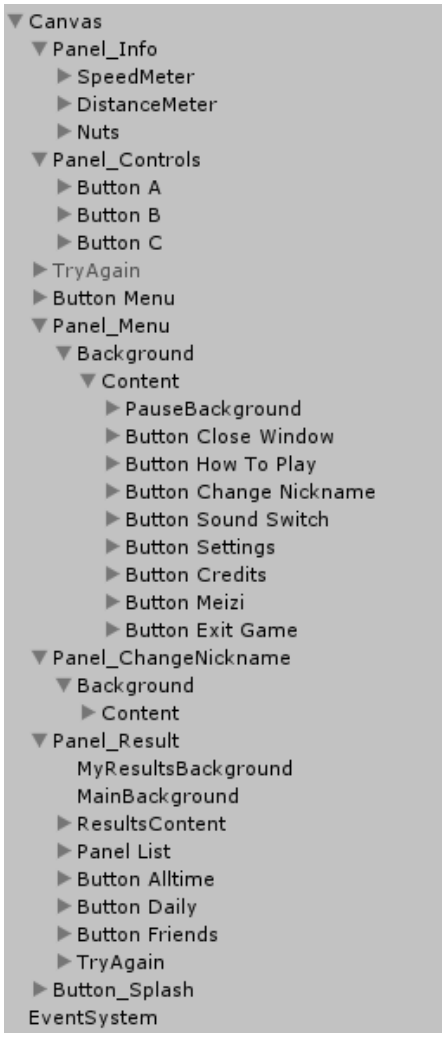
3.3.1 Käyttöliittymä

Käyttöliittymän juuriobjekti Canvas sekä tapahtumankäsittelijä Event System lisättiin hierarkiaan. Kaikkien UI-elementtien täytyy olla Canvas-objektin lapsiobjekteja. Mikäli Canvas-objektia ei ole projektin hierarkiassa, se luodaan automaattisesti, kun lisätään uusi käyttöliittymäelementti. Canvasin lapsiobjektit piirretään ruudulle hierarkianäkymän mukaan ylhäältä alas, eli alempana hierarkiassa oleva objekti piirretään edellisen päälle.

Käyttöliittymässä on tärkeää, että sen pystyy skaalaamaan eri näytöille sopivaksi. Unityssä tämä pystyttiin toteuttamaan lisäämällä Canvasille Canvas Scaler -komponentti. Referenssiresoluutioksi valittiin 800 x 480, josta käyttöliittymää skaalataan suuremmaksi, mikäli resoluutiota kasvatetaan.

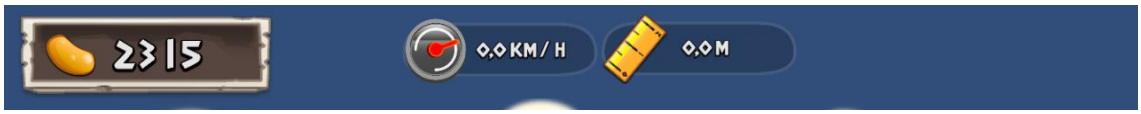
Unityn työkalut mahdollistavat elementin ankkuroinnin parent-objektiin. Tässä työssä on pyritty ankkuroimaan kaikki elementit oikeaan yläkulmaan, jotta ne pysyisivät samoilla paikoilla resoluutiosta huolimatta ja elementtien venymistä ei tapahtuisi.

Pelin käyttöliittymän osat on pyritty järjestelemään paneeleihin (panel), joita voidaan kutsua ikkunoiksi. Paneelit on muista elementeistä poiketen ankkuroitu siten, että ne täyttävät aina koko ruudun. Ylimpänä hierarkiassa ovat pelin kontrollit ja mittarit, koska näiden täytyy jäädä valikoiden taakse ja niitä ei pitäisi pystyä käyttämään valikkotilassa. Näiden jälkeen ovat järjestyksessään päävalikko, profiilinimen valinta, tulosikkuna ja lopuksi splash-ikkuna (kuva 15). Splash-ikkuna piirretään päällimmäiseksi, kun peli käynnistetään, ja se täyttää koko ruudun (kuva 20).



KUVA 15. Canvas-hierarkia.

Infoikkunassa (kuva 16) näytetään pähkinöiden määrä, juoksunopeus sekä heiton pituus. Jokaiselle on määritetty taustakuva juuriobjektin (hierarkiassa SpeedMeter, DistanceMeter, Nuts) Image-komponenttiin. Juuriobjektit on ankkuroitu ikkunan oikeaan yläreunaan, jotta ne säilyttäisivät paikkansa resoluution vaihtuessa. Juuriobjektien sisään on lisätty Image-objektit sekä tekstiobjektit. Tekstiobjekteille on määritetty alkuarvot ja niitä päivitetään EventManager-skriptissä.

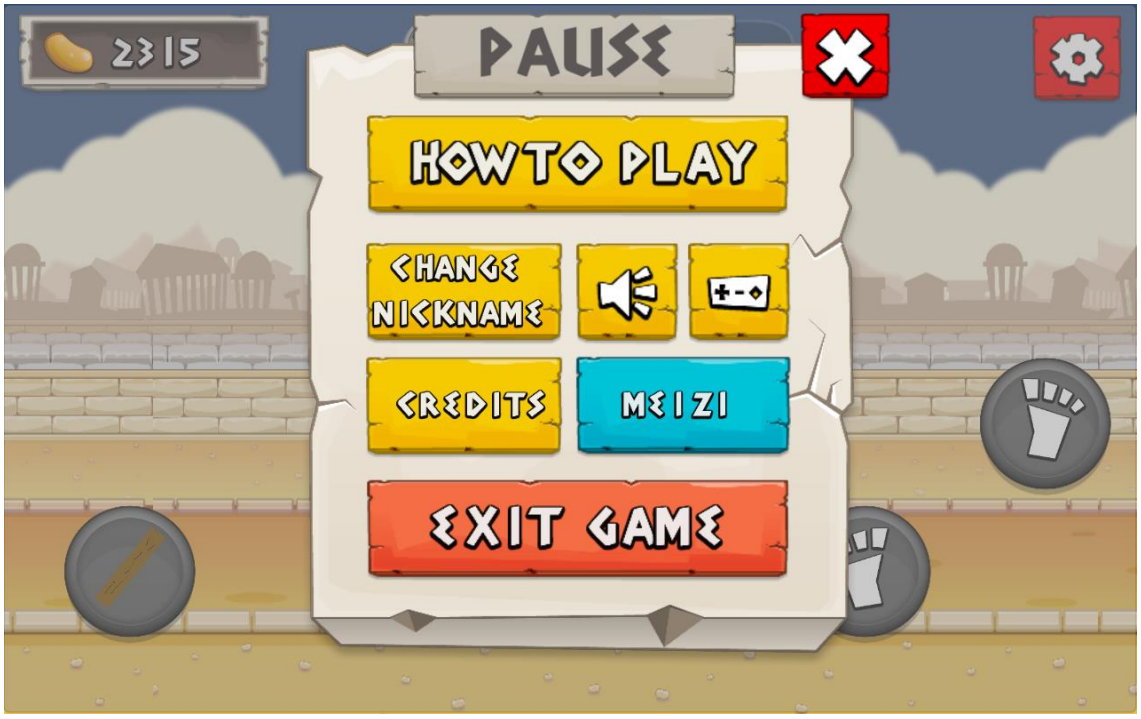


KUVA 16. Infoikkuna.

Kontrolli-ikkunassa (kuva 17) on kolme painiketta. Oikealla puolella olevilla painikkeilla ohjataan pelihahmoa ja vasemmalla puolella olevalla heitetään tukki. Painikkeille on määritetty Image-komponenttiin taustakuva. Juuriobjektin sisään on lisätty Image-objekti, jossa on painikkeen ikoni. Painikkeiden OnClick-tapahtumissa kutsutaan määrättyjä metodeita (Move ja TossLog) EventManager-skriptistä.

KUVA 17. Kontrolli-ikkuna.

Päävalikko (kuva 19) piirretään aiempien ikkunoiden päälle ja takana olevia painikkeita ei voi käyttää. Juuriobjektissa (Panel_Menu) on Image-komponentti, joka sävyttää taustaa hieman harmaaksi. Valikon taustakuvaa (hierarkiassa Background) jouduttiin kääntämään 90 astetta, joten taustakuvan alle jouduttiin tekemään uusi objekti (content), jolla kääntö korjattiin. Content-objektin alle tehtiin päävalikon sisältö, joka koostuu pääasiassa painikkeista. Painikkeiden toiminnallisuutta ohjataan aiempien ikkunoiden tapaan EventManager-skriptin kautta.



KUVA 19. Päävalikko.

Splash-ikkuna on käytännössä koko näytön peittävä painike. Alalaidassa vilkkuva teksti ”tap to start” on toteutettu BlinkText-skriptissä. Painikkeen OnClick-tapahtumassa kutsutaan EventManagerin SplashClicked-metodia, joka ohjaa joko profiilin luontiin tai suoraan peliin.

KUVA 20. Splash-ikkuna

Käyttöliittymän tekoon tarkoitettujen työkalujen, menetelmien ja komponenttien opettelu oli paikoitellen haastavaa, koska dokumentaatiota oli tarjolla hyvin vähän. Suuri osa ajasta meni erilaisiin kokeiluihin ja toimivien menetelmien keksimiseen. Käyttöliittymä alkoi kuitenkin olla tulostaulukkoa vaille valmis ja työlle varattu aika vähissä.

3.3.2 Pelaajaprofiili

Pelaajaprofiiliin tallennetaan pelaajan syöttämä nimi ja maa, laitteen yksilöllinen tunniste, heittotulokset ja pähkinöiden määrä. Heittotulokset on jaettu kahteen osaan, paras tulos sekä päivän paras tulos.

Lokaaliin tallennukseen käytetään *PlayerPrefs*-ominaisuutta. PlayerPrefs mahdollistaa helpon tavan tallentaa ja hakea pelissä tarvittavaa tietoa, mutta toisaalta käyttäjä voi myös helposti muokata sitä pelin ulkopuolelta ja näin huijata pelissä. Tuloksia ei tämän vuoksi tallenneta lokaalisti kuin testausta varten. Pähkinöiden

määrä tallennetaan laitteelle, mutta koska nekin vaikuttavat tulokseen, ne siirretään verkkoon myöhemmin.

Käyttöliittymään lisättiin ikkuna, johon pelaaja voi kirjoittaa pelaajanimensä sekä valita kotimaataan kuvaavan lippunsa. Pelaajanimen syöttöä varten lisättiin syöttökenttä. Lipun valintaa varten tehtiin parent-objekti FlagSelectionBackground, johon lisättiin Image-komponentti taustakuvan näyttämistä varten ja FlagSelector-skripti. Parent-objektin lapsiobjekteina olivat Image-komponentin sisältävä Flag-objekti sekä Text-komponentin Country-objekti. Lisäksi ikkunaan lisättiin viisi painiketta, nuolet vasemmalle ja oikealle, hyväksymispainike, ikkunan sulkemispainike sekä syöttökentän tyhjentämiseen tarkoitettu painike (kuva 21).

KUVA 21. Nimen syöttö ja lipun valinta.

FlagSelector-skripti lataa resursseista Flags-kuvan, joka on pilkottu spriteiksi ja nämä lisätään listaan. Spritet on nimetty maan mukaan, joten lippu ja maan nimi on helppo hakea listasta. Mikäli profiilia ei ole luotu, käytetään listan ensimmäistä lippua (alkio 0). Nuolipainikkeiden OnClick-tapahtumat on liitetty FlagSelector-luokan metodeihin NextFlag ja PreviousFlag. Metodeilla määritetään, mikä listan alkio näytetään. Listasta haettu sprite näytetään Flag-objektin Image-komponentissa ja spriten nimi näytetään Country-objektin Text-komponentissa.

Syöttökenttä tehtiin siten, että parent-objektiksi luotiin NickNameInput-objekti, johon lisättiin Image-komponentti taustakuvaa varten. Tämän lapsiobjektiksi lisättiin InputField. InputFieldin lapsiobjektina oleva NameText-objekti liitettiin InputFieldin Text Component -kenttään. Näin InputField komponentti voi tunnistaa NameText-objektista tapahtumat, esimerkiksi tekstin vaihtumisen tai syötön lopettamisen.

Kun ikkunaan on syötetty nimi ja valittu maa, profiili luodaan painamalla hyväksymispainiketta oikeasta alakulmasta. Hyväksymispainikkeen OnClick-tapahtuma on liitetty EventManager-skriptin ChangeNickApplyClicked-metodiin. Metodissa kutsutaan profiilin tallennusfunktiota SaveProfile, mikäli aiempaa profiilia ei löytynyt tai sitä ei ole ladattu. Mikäli aiempi profiili löydetään, kutsutaan päivitysfunktiota UpdateProfile.

SaveProfile tallentaa ensiksi nimen, pähkinät ja maa-indeksin lokaalisti. Tämän jälkeen kutsutaan verkkoon tallentamiseen tarkoitettua metodia SaveOnline. SaveOnline aloittaa WWWBehavior-luokan metodin, PostNewProfile, rinnakkaisen suorittamisen, jotta pelin ja käyttöliittymän suoritus ei keskeytyisi. WWWBehavior-luokka hoitaa profiilitietojen lähettämisen edelleen PHP-palvelinohjelmalle.

Tässä vaiheessa ilmeni vakavia ongelmia. Kun profiili-ikkunan deaktivoi hierarkiassa, Unity kaatui ja scene-tiedostoa ei saanut enää avattua. Versionhallinnasta löytyi edellinen toimiva versio, mutta kaikki tähän asti tehdyt muutokset oli menetetty. Profiili-ikkuna tehtiin uusiksi ja jälleen Unity kaatui. Tämän jälkeen päätettiin tehdä ikkuna erillisessä scenessä osa kerrallaan. Ilmeni, että syöttökenttä kaatoi Unityn. Internetistä löytyi ohjeita, miten ongelmaa voisi yrittää kiertää ja miten

menetetyt tiedot voisi saada takaisin (14). Lopulta vain Unityn päivitys seuraavaan testiversioon auttoi ja profiilin luominen saatiin onnistumaan.

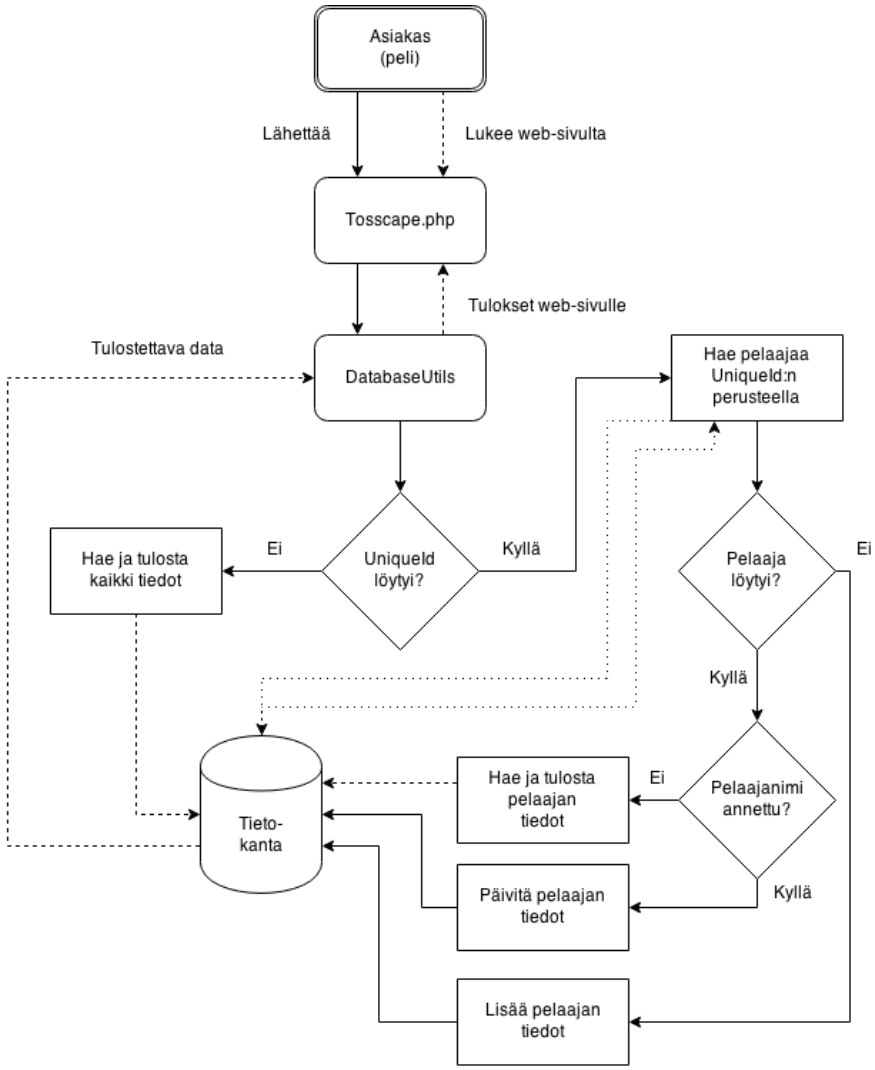
3.3.3 PHP-palvelinohjelma

Työssä käytettiin OAMK:n tietokantapalvelinta sekä www-palvelinta. Www-palvelimelle asennettiin phpMyAdmin nopeuttamaan tietokannan luontia ja manuaalista käsittelyä. Www-palvelimelle luotiin kaksi PHP-tiedostoa, *Tosscape.php* ja *DatabaseUtils.php*. Tosscape.php sijoitettiin public_html-hakemistoon, jotta siihen pystyisi ottamaan yhteyttä verkosta. DatabaseUtils.php sijoitettiin siten, että siihen ei pääsisi käsiksi muuten kuin palvelimelta. Tietokantapalvelimelle luotiin tietokanta, joka sisältää yhden taulun tarvittavine kenttineen.

DatabaseUtils.php sisältää *DatabaseUtils*-luokan, joka on palvelinohjelman pääluokka. Luokassa määritellään MySQL-tietokantaa varten tarvittavat yhteystiedot ja palvelimen salausavain, ja siihen on toteutettu palvelinohjelman toiminnallisuudelle tärkeät funktiot. DatabaseUtils-luokka mm. päättelee, millaisen kyselyn asiakas haluaa tehdä tietokantaan.

Tietokantaa pyrittiin toistaiseksi suojaamaan käyttämällä salausavainta, joka on määritetty kiinteäksi sekä asiakkaan että palvelimen puolella. Salausavaimen tarkoituksena oli, että tietokantaa ei pääsisi helposti muokkaamaan esimerkiksi web-selaimella. Lisäksi pyrittiin suojautumaan SQL-injektiolta käyttämällä valmisteltuja lauseita. (15.)

Kuvan 22 mukaisesti pelistä lähetetään Tosscape-sivulle tietoja. Sivu kutsuu DatabaseUtils-luokan metodia *CheckHash*. CheckHash purkaa ja tarkastaa MD5-koosteena tulevan salausavaimen. Mikäli avaimet täsmäävät, Tosscape-sivu kutsuu metodia DetectAction.



KUVA 22. Palvelinohjelman toiminta.

```
public function CheckHash($hash)
{
    $realhash = md5($this->secretKey);

    if($realhash == $hash)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

KUVA 23. CheckHash-metodi.

DetectAction-metodi yrittää tunnistaa, minkä kyselyn asiakas haluaa tehdä. Aluksi metodi tarkastaa uniqueid-kentän. Jos kentässä on jokin arvo, etsitään sen perusteella tietokannasta pelaajan tietoja. Mikäli pelaajan tietoja ei löytynyt, lisätään uusi pelaaja saaduilla arvoilla. Jos pelaajan tiedot löytyvät ja pelaajanimeä ei ole asiakkaalta saatu, tulostetaan pelaajan tiedot. Mikäli pelaajanimi on annettu, päivitetään kyseisen pelaajan tiedot. Jos Uniqueid-kenttä on tyhjä, tulostetaan kaikkien pelaajien tiedot tietyssä formaatissa web-sivulle. Asiakasohjelmassa tiedot luetaan ja parsitaan käytettäväksi pelissä.

3.3.4 Tulostaulukko

Peliin haluttiin tulostaulukko (kuva 24), jotta pelaajat voisivat kilpailla sijoituksista ja näin palata pelin pariin mahdollisimman useasti. Se näytetään jokaisen heiton jälkeen pienellä viiveellä ja se koostuu sivupaneelista ja pääikkunasta. Sivupaneelissa näytetään edellisen heiton tulos, päivän paras tulos sekä kaikkien aikojen paras tulos. Pääikkunassa näytetään listaus kaikkien pelaajien tuloksista. Pääikkunaan on tarkoitus toteuttaa kaverien tuloksien listaaminen sekä päivän parhaiden tuloksien listaaminen. Tällä hetkellä toiminnassa on pelkästään parhaiden tuloksien listaaminen.

KUVA 24. Tulosikkuna.

Listaukseen on käytetty *ScrollView*-komponenttia, jolla saadaan aikaiseksi vieritettävä ikkuna. Vieritys on rajoitettu pystysuuntaan ja aluetta on rajattu *Mask*-komponentilla, jotta listauksen rivit eivät näkyisi ikkunan ulkopuolella. ScrollView-komponentille on määritetty content-objekti, mihin kaikki listan alkiot lisätään, kun ne luodaan. Sisältö-objektille on lisätty *Vertical Layout Group* -komponentti, jotta ne saadaan asemoitua automaattisesti. Lisäksi sillä on *Scale Content* -komponentti, joka skaalaa kaikki listan alkiot tiettyyn kokoon leveyssuunnassa.

Listan alkiot ovat *ScoreItem*-prefabeja, jotka GameManager-skripti lisää sisältö-objektin lapsiobjekteiksi. Alkioiden tiedot saadaan WWWBehavior-skriptistä. Kun WWWBehaviorin metodi lähettää pelkän salausavaimen Tosscape-sivulle, se saa vastauksena kaikkien pelaajien tulokset. Vastaus tallennetaan string-muodossa ja parsitaan GameManager-skriptissä ProcessAllScoresString-metodilla. Metodissa jokaisesta pelaajasta tehdään Dictionary-tyyppinen kirjasto ja näistä jokainen lisätään vielä erilliseen kirjastoon.

GameManager-skriptin metodi PopulateListOfScores luo peliobjektin ScoreItem-prefabista ja tämän jälkeen käy kirjaston läpi ja hakee tiedoista maan, pelaajanimen sekä parhaan tuloksen. ScoreItemHandler-skriptin SetValues-metodia kutsumalla lisätään lista-alkiossa näytettävät tiedot. PopulateListOfScores-metodi poistaa vanhat objektit ja luo uudet joka kerta, kun ikkuna avataan. Tässä olisi voinut käyttää uudelleen jo luotuja prefabeja, mutta koska suorituskykyongelmia ei ollut ja työlle varattu aika oli jo periaatteessa loppunut, se pyrittiin vain kasaamaan nopeasti.

Tulosikkuna saatiin toimimaan suurelta osin, mutta sinne jäi niin paljon virheitä, että niitä ei ehditty enää korjaamaan. Tietojen lataaminen ja parsiminen toimivat, mutta tiedot päivittyvät vasta tulosikkunan näyttämisen jälkeen. Siten vasta seuraavalla näyttökerralla nähdään edellisen heiton tulos, mikäli se sattui olemaan aikaisempaa parempi tulos.

4 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa Hamsterscape Olympics -pelisarjaan peli omaperäisellä idealla ja yksinkertaisella pelimekaniikalla. Pelin ei ollut tarkoitus olla valmis opinnäytetyölle varatussa ajassa, mutta siinä täytyi olla toimiva ja yksinkertainen pelimekaniikka ja tuloslistaus verkosta laitteelle. Helpoin ja nopein tapa olisi ollut käyttää opinnäytetyön tilaajan aiempien pelien pohjaa ja toteuttaa vain pelattava osuus sekä tuloslistaus. Tavoitteena oli kuitenkin aiempien vaatimusten lisäksi oppia käyttämään Unityn graafisen käyttöliittymän tekoon tarkoitettuja työkaluja ja menetelmiä, joten valmista koodipohjaa ei haluttu käyttää. Tilaajan mielestä testiversion käyttö ei ollut ongelma, mutta kaikki olivat yhtä mieltä siitä, että siinä on omat riskinsä ja tukea ei välttämättä paljoa ole saatavilla.

Peli haluttiin saada lähes julkaisukuntoon opinnäytetyön puitteissa ja prototyypin valmistuessa kaikki näyttikin vielä hyvältä, mutta myöhemmin ongelmia alkoi ilmetä ja ne hidastivat työntekoa. Työssä olisi pitänyt olla ainakin yksi ylimääräinen ohjelmoija, jonka kanssa jakaa tehtävät. Tilaajan työntekijät pyrkivät auttamaan missä pystyivät, mutta heilläkään ei ollut kokemusta uudesta Unitystä.

Tuloslistauksen tekeminen ja varsinkin WWW-luokan käyttö oli ihan uutta asiaa, vaikka perusteet olivatkin tiedossa. Työlle varattua aikaa oli tässä vaiheessa jäljellä noin 30 tuntia, vaikka sitä olisi tarvinnut vielä vähintään kaksinkertaisen määrän. Aikataulutusta ja tehtävien tärkeysjärjestystä olisi syytä miettiä tulevaisuudessa tarkemmin.

Jatkokehitystä ajatellen palvelin ja tietokantaa käsittelevä luokka olisi tehtävä uusiksi kokonaisuudessaan. Tietoturvaa pitäisi parantaa ja turhaa verkkoliikennettä olisi rajoitettava. Palvelimen uusiminen tarkoittaisi luonnollisesti sitä, että tietojen lataaminen ja parsiminen jouduttaisiin myös tekemään uusiksi. Helpoin tapa tähän olisi käyttää JSON:a, jonka parsiminen Unity3D:n puolella on helppoa Unityn wikistä löytyvän SimpleJSON-luokan avulla (16).

Pelimekaniikkaa, sääntöjä ja rajoituksia pitäisi miettiä tarkemmin. Pelimekaniikka sinällään toimii, mutta vaatii viilausta. Säännöt ja rajoitukset ovat sitä vastoin ihan

alkutekijöissään. Pähkinöille pitäisi keksiä parempi käyttötarkoitus ja rajoituksia pitäisi kehittää huijareiden varalta. Lisäksi olisi hyvä pohtia, onko peliä järkevää julkaista omana pelinään vai olisiko se parempi sisällyttää suurempaan kokonaisuuteen yhtenä lajina.

Aiempi kokemus Unity3D:n käytöstä helpotti työn suorittamista. Testiversion työkaluista ei kuitenkaan ollut tarpeeksi dokumentaatiota tarjolla, mikä aiheutti sen, että välillä oli suuria vaikeuksia saada haluamiaan asioita aikaiseksi. Nyt, kun dokumentaatiota on tarjolla, voi huomata, että monet asiat olisi voinut tehdä paljon helpommin. Tosin moni asia on myös muuttunut sitten testiversion.

Ongelmia aiheuttivat myös testiversion virheet, jotka kyllä korjautuivat myöhemmissä versioissa, mutta aiheuttivat turhaa työtä siihen asti. Vaikka testiversion tiedettiin olevan riski ja sitä päätettiin käyttää, nyt peli tehtäisiin varmasti vakaammalla alustalla. Harmittavista ohjelmavirheistä ja projektin suunnittelun puutteista huolimatta pelistä saatiin suhteellisen hyvin toimiva esittelyversio.

LÄHTEET

1. Unity (game engine). 2014. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Unity_(game_engine). Hakupäivä 12.10.2014.

2. Company facts. 2014. Unity Technologies. Saatavissa: http://unity3d.com/public-relations. Hakupäivä 12.10.2014.

3. License comparisons. 2014. Unity Technologies. Saatavissa: http://unity3d.com/unity/licenses. Hakupäivä 12.10.2014

4. GameWorks PhysX Overview. 2014. NVIDIA Corporation. Saatavissa: https://developer.nvidia.com/gameworks-physx-overview. Hakupäivä 1.12.2014.

5. How do I import Models from my 3D app? 2014. Unity Technologies. Saatavissa: http://docs.unity3d.com/Manual/HOWTO-importObject.html. Hakupäivä 1.12.2014.

6. Sprite. 2014. Unity Scripting Reference. Unity Technologies. Saatavissa: http://docs.unity3d.com/ScriptReference/Sprite.html. Hakupäivä 1.12.2014.

7. SpriteRenderer. 2014. Unity Scripting Reference. Unity Technologies. Saatavissa: http://docs.unity3d.com/ScriptReference/SpriteRenderer.html. Hakupäivä 1.12.2014.

8. Texture atlas. 2014. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Texture_atlas. Hakupäivä 1.12.2014.

9. Mathf – SmoothDamp. 2014. Unity Scripting Reference. Unity Technologies. Saatavissa: http://docs.unity3d.com/ScriptReference/Mathf.SmoothDamp.html. Hakupäivä 1.12.2014.

10. Rigidbody. 2014. Unity Scripting Reference. Unity Technologies. Saatavissa: http://docs.unity3d.com/ScriptReference/Rigidbody.html. Hakupäivä 1.12.2014.

11. ForceMode2D. 2014. Unity Scripting Reference. Unity Technologies. Saatavissa: http://docs.unity3d.com/ScriptReference/ForceMode2D.html. Hakupäivä 13.12.2014.

12. BitBucket. 2014. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Bitbucket. Hakupäivä 13.12.2014.

13. Scene View. 2015. Unity Manual. Unity Technologies. Saatavissa: http://docs.unity3d.com/Manual/SceneView.html. Hakupäivä 3.1.2015.

14. Unity crashes when I disable a gameobject with inputfield con 4.6 b20. Ringhino 2014. Unity Community Support. Saatavissa: http://forum.unity3d.com/threads/unity-crashes-when-i-disable-a-gameobject-with-inputfield-con-4-6-b20.271692/. Hakupäivä 11.1.2015.

15. How can I prevent SQL-injection in PHP? Johnson, A. 2014. Stack overflow. http://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php. Hakupäivä 21.2.2015.

16. SimpleJSON. 2014. Unify community. http://wiki.unity3d.com/index.php/SimpleJSON. Hakupäivä 13.4.2014.

Liite 1. Lähtötietomuistio

LÄHTÖTIETOMUISTIO

Työn tiedot	Tekijä[1] Janne Aikioniemi,	Tilaaja[2] Meizi Games Oy
	Tilaajan yhdyshenkilö ja yhteystiedot[3] Sami Koski, Meizi Games Oy. Torikatu 19, 90100 Oulu.	
	Työn nimi[4] **Hamsterscape: Caber Toss (työnimi)**	
	Työn kuvaus[5] Meizi Games Oy on heinäkuussa 2014 perustettu start-up pelifirma. Firmalla on tällä hetkellä kaksi Hamsterscape -teemalla toteutettua urheilupeliä tarjolla mobiilimarketeissa. Tämä opinnäytetyö sisältää urheilupelisarjan pelin suunnittelun ja ohjelmoinnin vähintään testattavan mekaniikkademon tasolle.	
	Työn tavoitteet[6] Tämän työn tavoitteena on suunnitella ja toteuttaa Hamsterscape Olympics -pelisarjan peli. Peli tulee olla samalla Hamsterscape -teemalla, mutta omaperäisellä idealla ja pelimekaniikalla. Pelin ei tarvitse olla opinnäytetyön puitteissa valmis, mutta tulisi sisältää pelimekaniikan ja highscore -listojen toteutuksen. Opinnäytetyön pohjalta tullaan mahdollisesti julkaisemaan seuraava urheilusarjan peli.	
	Tavoiteaikataulut[7] 6.10.2014 - 21.11.2014	
	Päiväys ja allekirjoitukset[8] 29/syyskuu/2014 Janne Aikioniemi Tekijän allekirjoitus	29/syyskuu/2014 Sami Koski Tilaajan allekirjoitus

1. Tekijän nimi, puhelinnumero ja sähköpostiosoite.
2. Työn teettävän yrityksen virallinen nimi.
3. Sen henkilön nimi ja yhteystiedot, joka yrityksessä valvoo työn suoritusta.
4. Työn nimi voi olla tässä vaiheessa työnimi, jota myöhemmin tarkennetaan.
5. Työ kuvataan lyhyesti. Siinä esitetään muun muassa työn tausta, lähtötilanne ja työssä ratkaistavat ongelmat.
6. Esitetään lyhyesti ja selvästi työn tavoitteet.
7. Esitetään projektin tavoiteaikataulu. Silloin, kun työllä on välitavoitteita, myös ne merkitään aikatauluun. Tavoiteaikataulun ja oppilaitoksen yleisaikataulun perusteella tekijä laatii oman aikataulunsa.
8. Lähtötietomuistio päivätään ja sen allekirjoittavat tekijä ja tilaajan yhdyshenkilö.