OULUN SEUDUN
AMMATTIKORKEAKOULU

Arto Ruonala

Liskomannen — 2D-ammuskelupeli Unitylla

Liskomannen — 2D ammuskelupeli Unitylla

Arto Ruonala
Opinnäytetyö
Kevät 2015
Tietojenkäsittelyn koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä(t): Arto Ruonala
Opinnäytetyön nimi: Liskomannen 2D ammuskelupeli Unityllä
Työn ohjaaja(t): Matti Viitala
Työn valmistumislukukausi ja -vuosi: Kevät 2015 Sivumäärä: 23

Tämän opinnäytetyön tarkoituksena oli luoda yksinkertainen 2D ammuskelupeli. Peli toteutettiin käyttämällä Unity3D-pelinkehitystyökalua ja ohjelmointikielenä oli C#. Peli tehtiin omalähtöisenä projektina, joten toimeksiantajaa ei ole. Opinnäytetyön taustana oli oma mielenkiintoni pelikehitystä kohtaan ja halu luoda jotain itse tehtyä, jota voi esitellä vaikka tuleville työnantajille.

Opinnäytetyön teoriaosuudessa kerrotaan ohjelmointikielestä, suunnittelusta, toteutuksesta ja testauksesta. Toteutuksessa käydään läpi kuinka pelin ohjelmointipuoli luotiin pääpiirteittäin. Lisäksi kerrotaan Unity3D-pelinkehitystyökalun ominaisuuksista sekä 2D pelikehityksestä työkalulla joka on enemmän tarkoitettu 3D tuotantoon.

Mistään mullistavasta pelistä tässä ei ole kyse. Vain yhden opiskelijan halusta saada tehtyä jotain oman näköistä sekä kehittää ohjelmointitaitojaan.

Asiasanat: Kaksiulotteisuus, C#, Pelit, Ohjelmointi

ABSTRACT

Oulu University of Applied Sciences
Degree System of Information Technology

Author(s): Arto Ruonala
Title of thesis: Lizardman 2D shooting game with Unity
Supervisor(s): Matti Viitala
Term and year when the thesis was submitted: Spring 2015 Number of pages: 23

The purpose of this thesis was to create a simple 2D-shooting game. The game was developed using Unity3D game development tool and the programming language that was used is C#. The game was done as a self-started project, so my thesis has no external client. The background for this thesis was my own interest in game development and the wish to create something of my own to show my future employers.

The theoretical background of this thesis describes the programming language, planning, implementation and testing. Implementation describes how the game was programmed. The report also describes Unity3D game development tools features and 2D game development on a tool that is primarily meant for 3D game development.

This thesis is not about a ground-breaking game. It is about a student's wish to make something of his own and to improve his programming skills.

Keywords: Two-dimensionality, C#, Games, Programming

SISÄLLYS

1 JOHDANTO 6

2 TOIMEKSIANTAJA 7

3 YMPÄRISTÖ JA KIELI 7

3.1 Unity 7

3.2 C# 12

4 PROJEKTIN ETENEMINEN 13

4.1 Peli 13

4.2 Suunnittelu 13

4.3 Toteutus 14

4.4 Testaus 21

5 POHDINTA 22

LÄHTEET 23

1 JOHDANTO

Opinnäytetyön tarkoituksena oli luoda 2D side-scroller ammuskelupeli C# ohjelmointikielellä. Peli tehtiin Unity-pelinkehitystyökalulla. Unity työn teko hetkellä on versiossa 5.0.1.f1. Testauksen pelille tein iteratiivisella kehitystavalla, eli peliä testattiin aina kun siihen saatiin pienikään osa valmiiksi.

Vaikka Unity on pääsääntöisesti 3D-peleihin tarkoitettu, sillä pystyy tekemään myös 2D-pelejä ihan yhtä helposti. 2D-pelikehityksessä on paljon helpompi ja halvempi luoda kauniin näköinen peli verrattuna 3D-pelikehitykseen. 2D-peliä tehdessä ei tarvita suurta ryhmää luomaan pelin ulkonäkö näyttämään hyvältä ja valmiilta, eikä tarvita kalliita erillisiä ohjelmia kuten Maya. 2D-kehityksessä ei myöskään tarvitse miettiä yhteensopivuusongelmia yhtä paljon, koska 3D-pelit vaativat koneelta paljon enemmän tehoa kuin 2D-pelit. Yleensä 2D-tuotannosta on helpompi lähteä aloittelijan liikkeelle pelikehityksessä, koska siinä on vain kaksi ulottuvuutta käsiteltäväksi. Kameran kontrollointi on yksinkertaisempaa, tosin 2D-pelin kamera on hyvin yksinkertainen muutenkin, kun kamera vain seuraa pelaajaa sivulta toiselle. (Juuso Hietalahti 2015. Hakupäivä 8.4.2015)

Työtä tehdessäni tutustuin paremmin Unity-pelinkehitystyökaluun sekä C#-ohjelmointikieleen. Suurin osa pelin grafiikasta on haettu OpenGameArt nimiseltä sivustolta ja äänet Freesound sivustolta

2 TOIMEKSIANTAJA

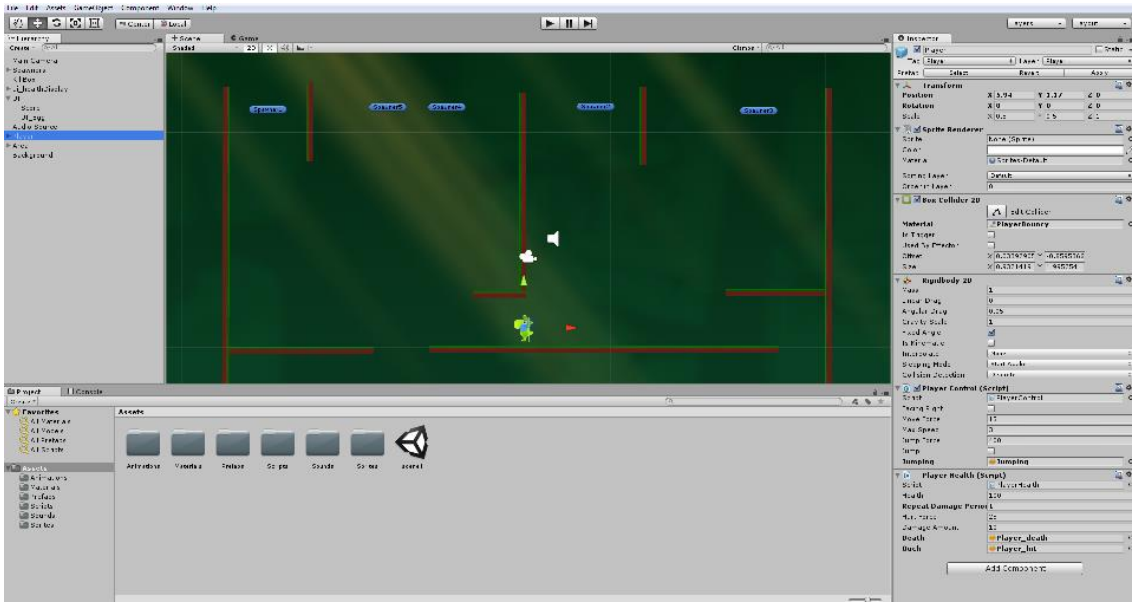
Toimeksiantajaa työlle ei ole. Alun perin tarkoitus oli tehdä työ Hawina Productions Oy:lle, mutta yhteistyö heidän kanssaan päättyi, koska he joutuivat lomauttamaan kaikki työntekijänsä. Tästä syystä tein työn omalähtöisenä projektina.

3 YMPÄRISTÖ JA KIELI

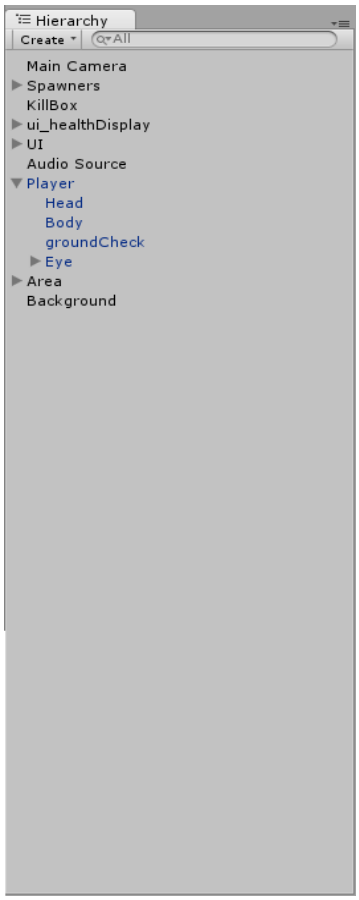
3.1 Unity

Unity on alustariippumaton 3D-pelimoottori, jolla projekti voidaan kääntää useille eri alustoille. Tällä hetkellä tuetut alustat ovat iOS, Android, Windows Phone 8, Blackberry 10, Windows, Mac, Linux, Web-selaimet, Playstation 3, Xbox360 ja Wii U. Unitystä on olemassa ilmainen sekä maksullinen Pro-versio. Tätä kirjoittaessa Unity Pron -lisenssi maksaa 1500$ tai 75$/kk. Pro-versiota voi halutessaan kokeilla ilmaiseksi 30 päivän ajan. Se tarjoaa paljon lisäominaisuuksia, joita ei ole ilmaisessa versiossa. (Unity Technologies 2014a. Hakupäivä 7.5.2014)

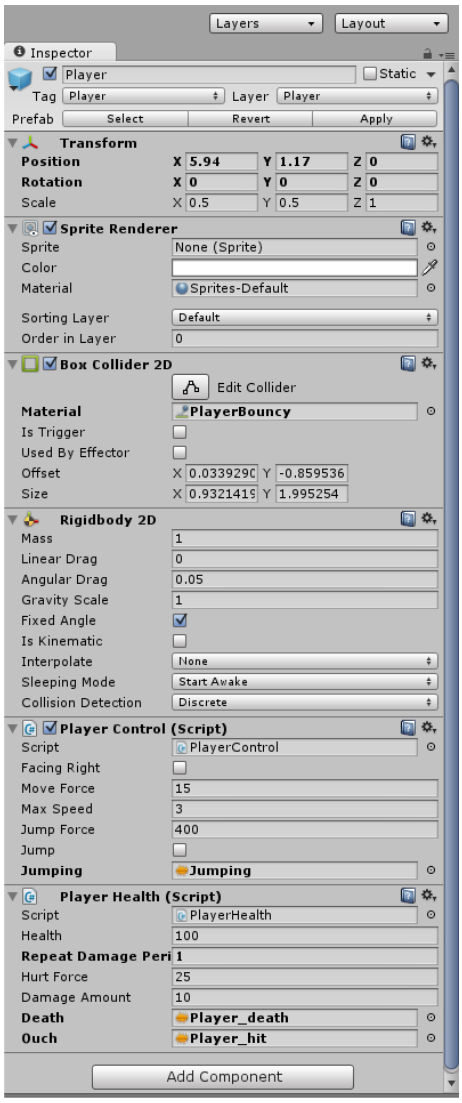
Unity on visuaalinen työkalu, jossa on paljon erilaisia näkymiä pelin tekemistä varten. Joitakin näkymistä ei paljon käytetä. Unityn oletus script editor on MonoDevelop, joka on ollut oletuksena versiosta 3.4 lähtien. Muitakin script editoreita voi käyttää, mutta unity automaattisesti aukaisee scriptit MonoDevelop editorilla. Tästä syystä scriptit täytyy käyttäjän halutessa avata erikseen toisella editorilla. Viisi eri näkymää on käytössä lähes koko ajan. Projektiselaimessa (kuvio 5) näkyy kaikki projektiin ladatut resurssit (assetit), kuten 3D-mallit, koodit ja tekstuurit. Inspector-ikkunassa (kuvio 3) voi katsoa ja säätää scenen peliobjektien sekä projektiin lisättyjen resurssien ominaisuuksia. Pelinäkymä toimii esikatseluikkunana ja näyttää miten peli toimii laitteelle käännettynä. Scene-ikkuna (kuvio 1) on ns. hiekkalaatikko eli alue, jonka objektit voivat olla yhteydessä toisten alueella olevien objektien kanssa, mutta eivät pysty ottamaan yhteyttä alueen ulkopuolella oleviin objekteihin. Hierarkia-ikkuna (kuvio 2) näyttää kaikki scenen käyttämät peliobjektit ja niiden hierarkian, eli mitkä peliobjektit ovat liitettynä toisen peliobjektin alle, kuviossa 2 on hyvä esimerkki tästä Player peliobjektissa. (Unity Technologies 2014a. Hakupäivä 7.5.2014)



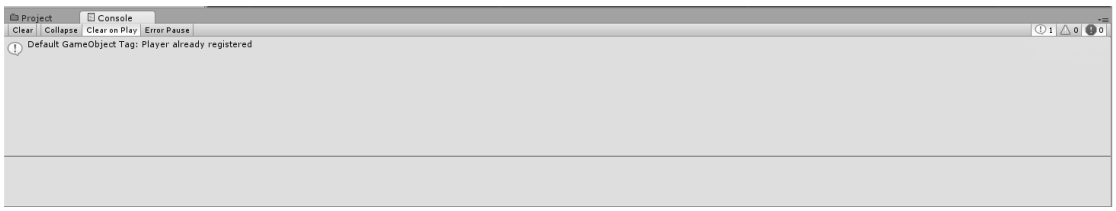
KUVIO 1. Unityn käyttöliittymä.



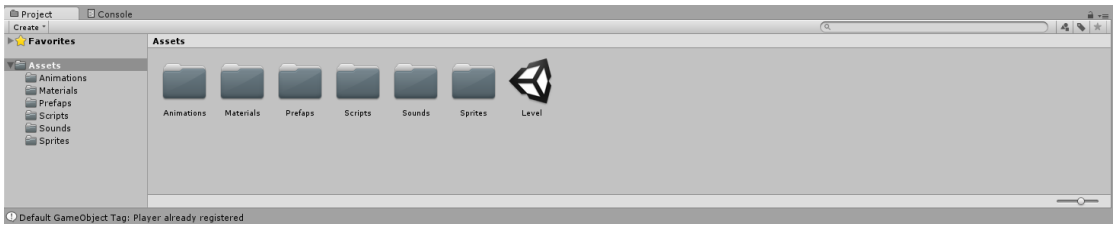
KUVIO 2. Unityn hierarkia ikkuna



KUVIO 3. Unityn peliobjektin inspector ikkuna



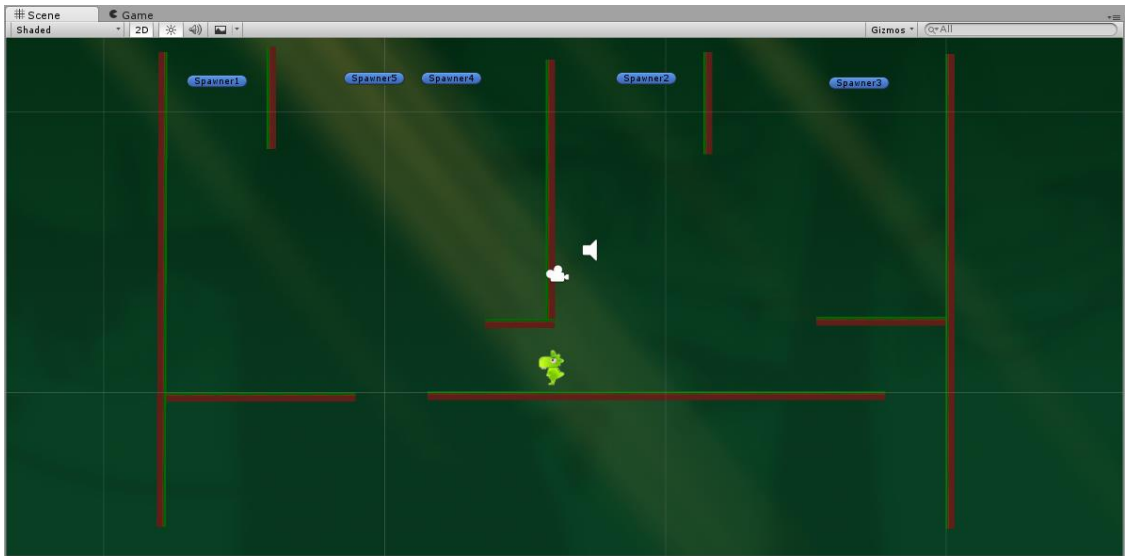
KUVIO 4. Unityn consoli ikkuna



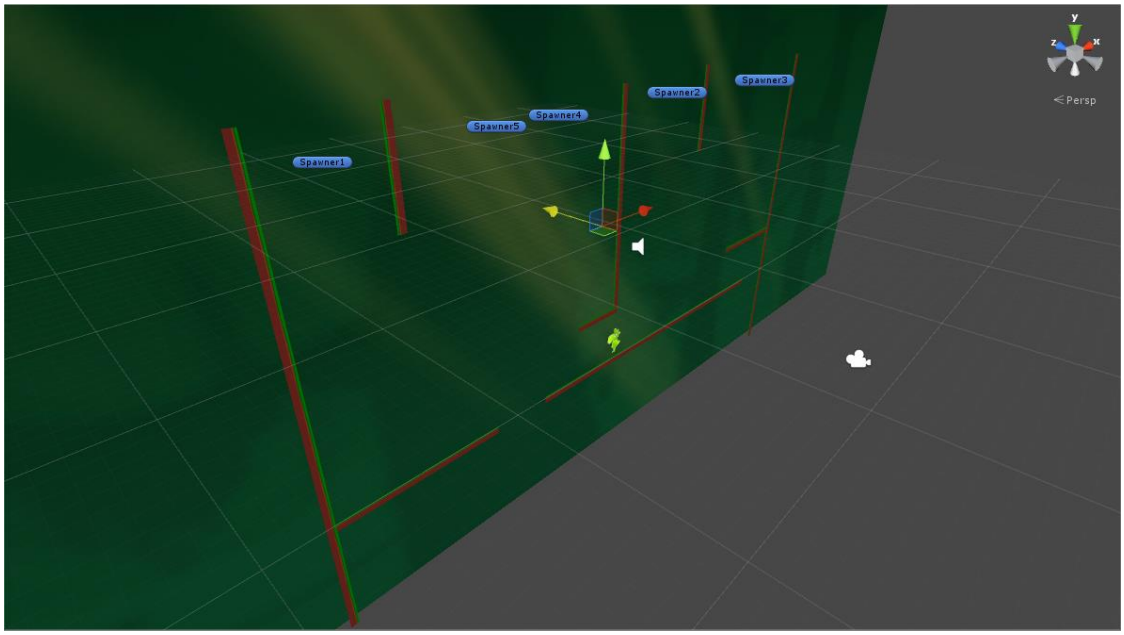
KUVIO 5. Unityn projektin assetit ikkuna

Vaikka Unityllä tehdään pääsääntöisesti 3D-pelejä, pystyy sillä tekemään erittäin hyvin 2D-pelejä. Unityn versiossa 4.3 tuotiin paljon 2D-työkaluja käyttäjien saataville. Huomattavin ero 2D-pelin luomisessa Unityllä on selkeästi sen 2D-katselumoodi, joka näkyy kuviossa 6. Vaikka kyseessä onkin 2D-projekti, pystyy käyttäjä laittamaan 3D-katselumoodin (kuvio 7) päälle halutessaan. 3D-katselumoodilla näkee millä tasoille (layer) käyttäjä on laittanut objekteja vaikka ne näkyvät 2D-katselumoodissa päällekkäin. Kun 2D-katselumoodi on päällä, käyttäjälle annetaan 2D-katselukulma scenestä. Tämä helpottaa objektien asettelua sekä auttaa visualisoimaan pelialueen. 2D graafisia objekteja kutsutaan spriteiksi. Vaikka spritet ovat käytännössä aivan tavallisia tekstuureita, niiden käsittelyssä on omat erikoiset tekniikat, joilla niistä saadaan tehokkaampia ja helpompia käyttää tuotannon aikana. Unityssä on sisään rakennettu sprite editor joka antaa käyttäjän ottaa haluamansa sprite grafiikan suuremmasta kuvasta. Tämä antaa mahdollisuuden muokata kuvankäsittely ohjelmalla useampaa sprite tekstuuria kerralla. Tätä voi käyttää hyödyksi, jos haluaa esimerkiksi pitää hahmon jokaisen ruumiinosan erillisenä elementtinä yhdessä kuvassa. 2D-objektit renderöidään käyttämällä Sprite Renderer komponenttia, kun 3D-objektit renderöidään käyttämällä Mesh Renderer komponenttia. (Unity Technologies 2014b. Hakupäivä 31.3.2015)

Spritet ovat bittikarttagrafiikkaa joka on tarkoitettu olemaan osa suurempaa kokonaisuutta. 1980-luvulla sekä ison osan 1990- lukua, spritet olivat standardi tapa integroida grafiikkaa peleihin. Graafikot loivat pieniä 2D- kuvia jotka esittivät hahmoja sekä muita objekteja peleissä. Kehittäjät viittasivat näihin spriteihin lähdekoodissa ja asettivat niille ominaisuuksia, kuten milloin spritet tuodaan esille ja miten ne reagoivat toisiin spriteihin. Vaikka spritejen käyttö on vähentynyt moderneissa videopeleissä, käytetään niitä edelleen, esimerkiksi spritejä käytetään usein lisäämään painikkeita, symboleja sekä muita käyttöliittymän osia. (TechTerms 2015. Hakupäivä 27.4.2015)



KUVIO 6. Peli 2D-moodissa



KUVIO 7. Peli 3D-moodissa

KUVIO 8. esimerkki 3D-mallista

KUVIO 8. 2D-spritesheet

Ennen Unityn 4.3 Versiota 2D-pelien pelien luominen Unityllä oli todella työlästä. Unitylle oli käyttäjien tekemiä omia 2D-työkaluja ladattavana Unityn asset storesta. Suurin osa näistä lisäosista olivat maksullisia. Tietoa siitä miten 2D-pelejä tehtiin ennen Unityn 4.3, on hyvin huonosti saatavilla. Onnistuin löytämään oppaan, josta selviää, miten on ollut mahdollista tehdä 2D-pelejä ennen kyseistä päivitystä. Siinä käytettiin hyödyksi Orthello 2D Framework lisäosaa spritejen hallintaan. ITween lisäosaa käytettiin animaatioiden tekoon hahmoille. A* Pathfinding Project lisäosan ilmaista versiota sekä erillistä TexturePacker ohjelmaa spritesheettien tekoon.

3.2 C#

C# on samankaltainen Java-ohjelmointikielen syntaksin kanssa; se on olioperustainen ohjelmointikieli. Kielen on kehittänyt Microsoft, ja kielen kehitys aloitettiin vuonna 1997. C# kutsutaan myös nimellä C sharp. C# on saatavilla useille eri alustoille mm. Windows, UNIX, Linus ja Solaris. C# käyttää Microsoftin luomaa .NET framework ympäristöä.
(Hernando Cadet Technology 2013, hakupäivä 25.3.2015)

C# tarkoitettiin olemaan yksinkertainen, moderni, yleiskäyttöinen ja olioperustainen ohjelmointikieli. Kielen kehittäjät halusivat kielen ja sen implementaatioiden antavan tukea ohjelmistokehitykselle tekemällä asioita helpommaksi ilman erillisiä prosesseja, kuten käyttää alustamattomia muuttujia sekä automaattisen ”roskien keräyksen”. Roskien keräämisellä tässä tarkoitetaan automaattista turhien muuttujien, joita ohjelma ei enää käytä, poistamista muistista.
(Hernando Cadet Technology 2013, hakupäivä 25.3.2015)

4 PROJEKTIN ETENEMINEN

Opinnäytetyö on jaettu kolmeen eri vaiheeseen suunnitteluun, toteutukseen sekä testaukseen. Aikataulun takia testaus osa projektista jää vähemmälle huomiolle muihin nähden.

4.1 Peli

Peli on 2d-side-scroller peli, jossa pelaajan on tarkoitus mahdollisimman paljon kerätä pisteitä tuhoamalla taivaalta tippuvia vihollisia. Pelaajalla on arsenaalissaan perusase, joka on pelaajalla aina käytettävissä pelin alusta alkaen sekä vihollisten pudottamia avustustavaroita, joita pelaaja voi nostaa. Jokaisesta vihollisesta, jonka pelaaja tuhoaa, pelaaja saa viisi pistettä. Pelin tarkoituksena on kerätä mahdollisimman paljon pisteitä, eli selviytyä mahdollisimman pitkään. Pelaaja liikuttaa hahmoaan käyttämällä hiirtä ja näppäimistöä. Hahmo liikkuu vasemmalle painamalle näppäimistön A-painiketta, oikealle painamalla D-painiketta. Hahmo hyppii painamalla välilyöntiä. Ampuminen tapahtuu hiirellä. Hahmo tähtää sinne mihin hiiri osoittaa, ja ampuu normaalia asettaan hiiren vasemmalla näppäimellä. Hiiren oikealla näppäimellä hahmo ampuu erikoisaseen, joita löytyy vihollisten pudottamana.

4.2 Suunnittelu

Tiukan aikataulun vuoksi pelille ei tehty mitään suunnitteludokumenttia. Pelistä ei tullut niin laajaa, että se olisi sitä oikeastaan vaatinutkaan. Tietenkin olisi ollut hyötyä, jos peliä olisi suunniteltu ensin huolella ja sitten alkanut työstämään.

Pelin suunnitteluvaiheessa ajatukset kulkivat todella paljon pelin yleisissä mekaniikoissa, kuten kuinka laaja pelistä pitäisi tehdä, kuinka paljon vihollisia pelin alussa pitäisi tulla, paljonko pisteitä jokaisesta vihollisesta tulisi saada. Mietinnässä oli myös, miten muut samankaltaiset pelit ovat erottuneet toisistaan ja mitä niistä voisin ottaa peliin mukaan. Ensimmäisenä mieleen tuli ns. po-

wer-upit eli erilaiset nostettavat tavarat, joista pelaaja saa lisävoimia tai parantuvat. Ilman näitä ei peliä kovin kauaa jaksa pelata, joten päätettiin lisätä ne peliin mukaan. Pelissä täytyy myös olla useammanlaisia vihollisia, sillä peli olisi hyvin tylsä, kun kokoajan tuhotaan samoja vihollisia. Tästä syystä päätettiin lisätä peliin erilaisia vihollisia. Vihollisia on kaksi erilaista, toinen on hieman nopeampi, mutta heikompi ja toinen on hitaampi, mutta kestävämpi. Molemmat viholliset tekevät saman verran vahinkoa pelaajaan.

4.3 Toteutus

Peli toteutettiin C#-ohjelmointikielellä käyttäen Unity-pelinkehitystyökalua. Ensimmäisenä työstettiin sitä, että pelihahmo liikkuu hyvin ja pystyy hyppimään vain, kun hahmon jalat koskettavat maata. Hahmon liikuttaminen oikealle ja vasemmalle tapahtuu painamalla joko A- tai D-näppäimiä. Riippuen siitä kumpaa näppäintä painaa annetaan hahmolle vauhtia, että hahmo liikkuu ja liike loppuu, kun näppäimestä päästää irti. Hahmoa voi myös liikuttaa nuolinäppäimillä. Hahmon hyppy on toteutettu siten, että hahmon alle aseteltiin pieni näkymätön alue, joka tarkistaa onko hahmo seisomassa maan päällä, ettei hahmo pysty hyppimään loputtomasti ilmassa. Hahmon liikuttamiseen käytetty koodi näkyy kuviossa 9. FixedUpdate-funktiota kutsutaan aina tasaisin väliajoin pyörittämään sisällään oleva koodi. Ensin koodissa otetaan talteen horisontaalinen syöttö, jonka avulla tarkistetaan onko pelaajan hahmo liikkeellä. Mikäli pelaajan hahmo ei ole saavuttanut maksiminopeuttaan tai kääntää suuntaa, lisätään pelaajan hahmolle liikettä. Tämän jälkeen tarkistetaan liikkuuko pelaajan hahmo nopeammin kuin sille annettu maksiminopeus. Mikäli hahmo liikkuu liian nopeasti, asetetaan hahmon nopeus maksiminopeudeksi. Sitten tarkistetaan onko pelaaja liikkumassa eri suuntaan kuin hahmo katsoo. Jos pelaajan hahmo katsoo eri suuntaan kuin liikkuu, käännetään hahmo. Tämän jälkeen tarkistetaan, onko pelaaja painanut hyppynappia ja jos on, soitetaan AudioSourcella hyppyääni sekä lisätään pelaajahahmolle vertikaalista vauhtia imitoimaan hyppäämistä. Tämän jälkeen asetetaan hyppäämiseen käytetty muuttuja epätodeksi.

```
39      void FixedUpdate ()
40      {
41          // Cache the horizontal input.
42          float h = Input.GetAxis("Horizontal");
43
44
45          // If the player is changing direction (h has a different sign to velocity.x)...
46          // ... or hasn't reached maxSpeed yet...
47          if(h * GetComponent<Rigidbody2D>().velocity.x < maxSpeed)
48              // ... add a force to the player.
49              GetComponent<Rigidbody2D>().AddForce(Vector2.right * h * moveForce);
50
51          // If the player's horizontal velocity is greater than the maxSpeed
52          if(Mathf.Abs(GetComponent<Rigidbody2D>().velocity.x) > maxSpeed)
53              //  set the player's velocity to the maxSpeed in the x axis.
54              GetComponent<Rigidbody2D>().velocity = new Vector2(Mathf.Sign(
55                  GetComponent<Rigidbody2D>().velocity.x) * maxSpeed ,
56                     GetComponent<Rigidbody2D>().velocity.y );
57
58          // If the input is moving the player right and the player is facing left...
59          if (h < 0 && facingRight) {
60              // flip the player.
61
62              Flip ();
63          }
64          // Otherwise if the input is moving the player left and the player is facing right...
65          else if (h > 0 && !facingRight) {
66              // flip the player.
67              Flip ();
68          }
69          // If the player should jump...
70          if(jump)
71          {
72
73              AudioSource.PlayClipAtPoint(jumping,transform.position,0.5f);
74              // Add a vertical force to the player.
75              GetComponent<Rigidbody2D>().AddForce(new Vector2(0f, jumpForce));
76
77
78              jump = false;
79          }
```

KUVIO 9. Hahmon liikuttamisen koodi

Tämän jälkeen peliin työstettiin ampuminen. Ampuminen tapahtuu hiiren vasemmalla painikkeella tai näppäimistön vasemmalla ctrl-näppäimellä. Kun ampumisnappia on painettu, peli tarkistaa missä päin peliruutua hiirtä painettiin ja lähettää siihen suuntaan ammuksen, joka pysähtyy seinään tai viholliseen. Ammutut panokset katoavat kuitenkin yhden sekunnin kuluttua. Tällä tavoin estetään panosten lentäminen loputtomiin pelikentän ulkopuolella. Ampumiseen käytin kahta eri koodipätkää jotka näkyvät kuvioissa 10 ja 11. Kuviossa 10 käytin Update-funktiota, joka kutsutaan jokaisella piirretyllä kuvalla. Update-funktiossa tarkistetaan onko pelaaja painanut Fire1- tai Fire2-näppäimiä. Riippuen kumpaa on painettu, kutsutaan joko funktiota ampumaan normaaliammuksen tai kananmunan. Normaalin ammuksen funktiossa haetaan ensin ammuksen lähtökohta, joka on pelaaja hahmon silmä. Sen jälkeen luodaan ammus Instantiate-funktiolla, joka hakee pelin tiedoista Bulletprefab-nimisen esiluodun objektin. Tämän jälkeen ammuksen koodi, joka näkyy kuviossa 11, jatkaa. Ensin ammukselle asetetaan ikäraja käyttämällä destroy(GameObject, 1) –funktiota, jolla tuhotaan ammus yhden sekunnin jälkeen. Update-funktiossa liikutetaan ammusta transform.Translate-funktiolla. Ammuksen osuessa johonkin kutsutaan OnTriggerEnter2D-funktiota, jossa tarkistetaan, että onko osuttu peliobjekti merkattu viholliseksi, jos on, soitetaan ääni joka kertoo, että viholliseen on osuttu. Seuraavaksi kutsutaan vihollisen koodista Hurt-

funktiota, joka aiheuttaa viholliseen vahinkoa. Viimeisenä tuhotaan ammus. Ammuksen osuessa seinään tapahtuu samalla tavalla, ilman vihollisen vahingoittumista ja toisella äänellä.

```
21      // Update is called once per frame
22      void Update () {
23          //if fire1 button is pressed
24          if (Input.GetButtonDown ("Fire1")) {
25              //shoot a bullet
26              Shoot();
27
28          }
29          //if fire2 button is pressed
30          else if (Input.GetButtonDown ("Fire2")) {
31              //shoot an egg
32              ShootEgg();
33
34          }
35
36      }
37
38      //function for shooting bullets
39      void Shoot () {
40
41          //get the location of firepoint
42          Vector2 firePointPosition = new Vector2 (firePoint.position.x, firePoint.position.y);
43
44          //create a bullet
45          Instantiate (BulletPrefab, firePoint.position, firePoint.rotation);
46
47      }
```

KUVIO 10. Ammuksen luonti koodi

```
12      //Use this for initialization
13      void Start () {
14
15          //Destroy the bullet after one second
16          Destroy(gameObject, 1);
17      }
18
19      //Update is called once per frame
20      void Update () {
21
22          //Moving the bullet
23          transform.Translate (Vector3.right * Time.deltaTime * speed);
24
25      }
26
27      //If something collides with the bullet
28      void OnTriggerEnter2D (Collider2D col)
29      {
30          //If bullet collides with the enemy
31          if (col.tag == "Enemy") {
32
33              //Play the enemy_hit audioclip
34              AudioSource.PlayClipAtPoint(enemy_hit,transform.position);
35
36              //Get the enemys script and call Hurt function
37              col.gameObject.GetComponent<Enemy>().Hurt();
38
39              //Destroy the bullet
40              Destroy (gameObject);
41          }
42
43          //If the bullet collides with an object tagged as "Ground" or "Wall"
44          else if(col.gameObject.tag == "Ground" || col.gameObject.tag == "Wall")
45          {
```

KUVIO 11. Ammuksen koodi

Kun hahmo liikkui ja pystyi ampumaan työstettiin vihollisia. Vihollisten käyttäytyminen on hyvin yksinkertaisesti luotu. Vihollinen liikkuu eteenpäin niin pitkälle kunnes se törmää johonkin, jolloin se kääntyy ja jatkaa matkaansa, paitsi jos vihollinen törmää pelaajaan. Tämä reaktio on luotu käyttämällä kuviossa 5 näkyvää OnCollisionEnter2D-funktiota. Kun vihollinen törmää pelaajaan, jatkaa vihollinen samaan suuntaan ja alkaa työntää pelaajaa sekä tekemään vahinkoa. Vihollista liikutetaan FixedUpdate-funktion sisällä. Funktiossa tarkistetaan myös, ovatko vihollisen elämäpisteet vähemmän kuin nolla ja onko se merkattu kuolleeksi, jos on, kutsutaan vihollisen kuolinfunktiota.

```
34    void OnCollisionEnter2D(Collision2D coll){
35
36        if (coll.gameObject.tag == "Wall" || coll.gameObject.tag =="Enemy") {
37            Flip ();
38        }
39
40    }
41
42    void FixedUpdate ()
43    {
44
45        // Set the enemy's velocity to moveSpeed in the x direction.
46        GetComponent<Rigidbody2D>().velocity = new Vector2(transform.localScale.x * moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
47
48
49        // If the enemy has zero or fewer hit points and isn't dead yet
50        if(HP <= 0 && !dead)
51            // call the death function.
52            Death ();
53    }
```

KUVIO 12. Vihollisen liikkumisen koodi

Vihollisia luodaan peliin Spawner-nimisten peliobjektien avulla, jotka on aseteltu pelikentän yläpuolelle. Kuviossa 13 näkyvää InvokeRepeating-funktiota käytetään luomaan jatkuvasti uusia vihollisia. Funktiolle annetaan tiedoksi mitä funktiota halutaan kutsua sekä kuinka usein sitä kutsutaan. Kutsussa Spawn-funktiossa käytetään hyväksi koodin alussa olevaa enemies-taulukkomuuttujaa, johon on asetettu sisälle esiluodut vihollisten peliobjektit. Seuraavaksi luodaan vihollinen Instantiate-funktiolla. Funktiolle annetaan enemyIndex-muuttuja, joka kertoo, minkä vihollisen funktio luo.

```
6       public float spawnTime = 5f;            // The amount of time between each spawn.
7       public float spawnDelay = 3f;           // The amount of time before spawning starts.
8       public GameObject[] enemies;            // Array of enemy prefabs.
9
10
11      void Start ()
12      {
13          // Start calling the Spawn function repeatedly after a delay .
14          InvokeRepeating("Spawn", spawnDelay, spawnTime);
15      }
16
17
18      void Spawn ()
19      {
20          // Create a random enemy from the enemies Array
21          int enemyIndex = Random.Range(0, enemies.Length);
22          Instantiate(enemies[enemyIndex], transform.position, transform.rotation);
23
24      }
25  }
```

KUVIO 13. Spawner-koodi

Hahmon elämäpisteet ovat esillä hahmon yläpuolella yksinkertaisella vihreällä palkilla. Hahmolla voi olla enintään 100 elämäpistettä. Aina, kun vihollinen törmää hahmoon vähennetään 10 pistettä. Kuviossa 14 näkyy kuinka tämä on tehty koodissa. TakeDamage-funktiossa käsitellään tilannetta, kun hahmo vahingoittuu. Ensin asetetaan, että hahmo ei pysty hyppäämään samanaikaisesti kun vahingoittuu. Seuraavaksi annetaan hahmolle pieni hyppy taaksepäin ja hieman ylös. Sen jälkeen vähennetään hahmon elämäpisteistä määritetty vahinko, jonka jälkeen kutsutaan UpdateHealthBar-funktiota. Funktiolla päivitetään hahmon päällä olevaa vihreää palkkia, joka näyttää paljonko pelaajalla on elämäpisteitä jäljellä.

```
80      void TakeDamage (Transform enemy)
81      {
82          // Make sure the player can't jump.
83          playerControl.jump = false;
84
85          // Create a vector that's from the enemy to the player with an upwards boost.
86          Vector3 hurtVector = transform.position - enemy.position + Vector3.up * 5f;
87
88          // Add a force to the player in the direction of the vector and multiply by the hurtForce.
89          GetComponent<Rigidbody2D>().AddForce(hurtVector * hurtForce);
90
91          // Reduce the player's health by 10.
92          health -= damageAmount;
93
94          // Update what the health bar looks like.
95          UpdateHealthBar();
96
97      }
98
99
100     public void UpdateHealthBar ()
101     {
102         // Set the health bar's colour to proportion of the way between green and red based on the player's health.
103         healthBar.material.color = Color.Lerp(Color.green, Color.red, 1 - health * 0.01f);
104
105         // Set the scale of the health bar to be proportional to the player's health.
106         healthBar.transform.localScale = new Vector3(healthScale.x * health * 0.01f, 1, 1);
107     }
```

KUVIO 14. Hahmon elämäpisteiden koodi

Jokaisesta vihollisesta, jonka pelaaja saa ammuttua, saa viisi pistettä. Pelissä viholliset pudottavat ensiapupakkauksia sekä munia hahmolle nostettavaksi. Ensiapupakkaukset parantavat hahmoa 10 elämäpisteen verran, eli yhden osuman. Munia, jotka pelaaja nostaa, käytetään erillisinä ammuksina, jotka tappavat vihollisen yhdellä osumalla. Jokaisessa munanipussa on kolme heitettävää munaa. Vihollisten pudottamat tavarat voi myös ampua rikki, jolloin hahmo ei saa kyseistä tavaraa. Pistemäärä ja hahmon munien määrä näytetään pelialueen vasemmassa alanurkassa. Kuviossa 15 näkyvä koodi on vihollisen kuolemisen funktiosta. Kun vihollinen kuolee, ensin asetetaan randomDrop-muuttujaan numero satunnaisesti 0 ja 100 väliltä, tämä määrittää pudottaako vihollinen tavaran. Jos randomDrop-muuttujassa oleva numero on 90 tai suurempi vihollinen pudottaa ensiapupakkauksen. Jos numero on pienempi kuin 90 ja suurempi tai yhtä suuri kuin 70, pudottaa vihollinen hahmolle kananmunia ammuttavaksi. Seuraavaksi asetetaan vihollisen dead-muuttuja todeksi ja lisätään score scriptin score-muuttujaan 5. Kuviossa 16 näkyy kuinka pisteet lasketaan. Yksinkertaisesti scriptissä on vain yksi muuttuja, johon asetetaan saadut pisteet ja ne näytetään päivittämällä kuvaruudulla olevaa GUIText osiota, johon on upotettu score scripti.

```
67      void Death()
68      {
69          //Determine if the enemy drops something
70          randomDrop = Random.Range(0, 100);
71
72          if (randomDrop >= 90) {
73              //Create healthdrop
74              Instantiate (HealthDrop, transform.position, transform.rotation);
75          }
76          else if(randomDrop < 90 && randomDrop >= 70){
77              //Create eggdrop
78              Instantiate (EggDrop, transform.position, transform.rotation);
79          }
80
81
82          // Set dead to true.
83          dead = true;
84
85          //Add 5 score
86          score.score += 5;
```

KUVIO 15. Vihollisten kuolemisen koodi

```
using UnityEngine;
using System.Collections;

public class Score : MonoBehaviour
{
    public int score = 0;                    // The player's score.


    void Update ()
    {
        // Set the score text.
        GetComponent<GUIText>().text = "Score: " + score;

    }

}
```

KUVIO 16. Pisteiden koodi.

KUVIO 17. Pelaajan elämäpisteet, pistemäärä sekä munien määrä.

Kun pelin perusmekaniikat toimivat, alkoi pelin hienosäätäminen, vihollisia sekä pelaajan hahmoa. Ensimmäisenä vihollisille lisättiin silmienräpäyttelyn, sekä pientä liikettä antamaan hieman elävyyttä. Hahmolle lisättiin silmän liikkeen hiiren mukaisesti, tämä näyttää kuin hahmo tähtäisi sinne minne ampuu. Lisäämällä pelihahmoille pientä liikettä, saadaan pelistä paljon mukavampi pelata verrattuna vain paikallaan oleviin hahmoihin. Pelikentästä tehtiin laajempi, jotta olisi enemmän liikkumistilaa sekä poistettiin ns. turvalliset alueet, ettei hahmo voi olla samassa paikassa kokoaikaa. Lisäksi kentälle lisättiin tausta.

4.4 Testaus

Puhekielessä testaus termillä tarkoitetaan lähes mitä vain kokeilemista. Ohjelmistojen testauksessa testaustermi määritellään suunnitelmalliseksi virheiden etsimiseksi suorittamalla ohjelmaa tai sen jotain osaa. Tässä määritelmässä keskeisimmät sanat ovat suunnitelmallinen ja etsiminen: usein testaus tapahtuu kokeilemalla ohjelmaa umpimähkäisesti jollain syöttöaineistolla, varsinkin, jos testaajana on ohjelman tekijä. Testauksen tarkoituksena on enemmänkin osoittaa ohjelman toimivuus, kuin löytää virheitä. Testauksen määrä ei aina tarkoita, että testaus on ollut tehokasta. Muutaman tunnin testauksella, joka on suunniteltu huolella voi johtaa parempaan tulokseen kuin usealle päivälle pitkitetty umpimähkäinen testaaminen. (Haikala & Märijärvi 2000, 226)

Tätä työtä kehitettiin iteratiivisella kehitystavalla, joka on ketterää ohjelmistokehitystä. Eli sovellusta kehitetään pienissä paloissa, jotka testataan valmistuttuaan ja joissa nähdään lopputuloksen kehittyvän pikkuhiljaa. Iteratiivinen kehitys asetti testaukselle lisää vaatimuksia. Kun sovellusta kehitetään pienissä osissa, on todennäköistä, että syntyy regressiota. Tämän takia syntyi tarve jatkuvalle integraatiolle. Sovelluskehittäjän kannalta ketterä ohjelmistokehitys aiheutti sosiaalisille taidoille uusia vaatimuksia sekä sovellusten iteratiiviseen rakentamiseen. (Gofore 2013. Hakupäivä 27.5.2014)

Peliä testattiin aina, kun koodiin tehtiin jonkinlaisia muutoksia. Mitään erillistä testaustyökalua työssä ei käytetty vaan pelkästään kokeilemalla mikä toimii ja mikä ei. Mikäli jokin asia ei toiminut, sitä työstettiin, että peli saatiin toimimaan halutulla tavalla. Ammuksen luomisen kanssa oli aluksi paljon ongelmia, jostain syystä peli halusi pelkästään ampua sivuille. Jos ongelmaan ei löydetty ratkaisua, Internetistä etsittiin apua, kuinka ongelma ratkaistaan ja yleensä löytyi edes suuntaa antavaa apua. Esimerkkinä testaus prosessista käytetään vihollisten törmäyksien jälkeisen kääntymisen testaamista. Kun vihollisille lisättiin koodiin, että törmätessään seinään tai toiseen viholliseen vihollinen kääntyy. Huomattiin, että seinään törmätessään viholliset eivät kääntyneetkään, vaan jäivät liikkumaan seinää vasten. Tämä johtui siitä, että seinien tunnisteessa (Tag) oli pieni kirjoitusvirhe koodinpuolella. Ongelma ratkaistiin tarkistamalla if- ehto jossa tunnistetta käytetään (Kuvio 12, rivi 36) ja korjaamalla siitä kirjoitusvirhe, jonka jälkeen viholliset kääntyivät seiniin törmätessään.

5 POHDINTA

Opinnäytetyöni alkuvaiheessa tein hyvin suppean ja hätäisesti suunnitelman minkälainen pelistä pitäisi tulla. Tämä myöhemmässä vaiheessa aiheutti pieniä hankaluuksia, kun en ollut päättänyt etukäteen, mitä meinaan tehdä ja täytyi suunnitella mitä tekee samalla kuin ohjelmoi aikaisempia osia. Olisi ollut hyvä tehdä edes karkea hahmotelma vaikka paperille, miltä pelin käyttöliittymä näyttäisi sekä miltä kenttä näyttäisi. Testauksessa olisi ollut hyvä käyttää samaa menetelmää jokaisessa testaustilanteessa.

Pelin graafisessa ulkoasussa olisi paljon paranneltavaa, koska en itse ole kovin taiteellinen ihminen ja jouduin hakemaan Internetistä lähes kaiken grafiikan. Pyrin pitämään hahmoissani jotain samankaltaisuutta, ettei mikään näytä liikaa poikkeavalta. Pelaajahahmolle olisi voinut lisätä jotain muutakin animaatiota kuin vain silmän pieni liikkuminen hiiren perässä.

Projektin koodien jäsentely omasta mielestäni onnistui hyvin, kaikki koodi ei ole samassa paikassa, vaan on ositeltu moneen osaan sekä kommentoitu mitä koodin eri pätkät tekevät. Tämä helpottaa jatkokehitystä huomattavasti, ettei uusien ohjelmoijien tarvitse käyttää aikaa koodin läpikäymiseen.

Jatkoprojektina pelille voisi hyvin olla pelin laajentaminen useammille eri kentille, lisäämällä vihollisia sekä erilaisia aseita. Pelille myös pystyisi kehittämään tällä tavoin jonkinlaisen tarinan, joka pitäisi pelaajan pidempään kiinnostuneena. Lopulta sain pelin tehtyä loppuun. Olen työhöni tyytyväinen, vaikka se ei ole graafisesti kovin kaunis.

LÄHTEET

Design a Game 2014. Why Is Unity so popular for videogame development? Hakupäivä 4.5.2014 http://designagame.eu/2013/12/unity-popular-videogame-development/

Gofore 2013. Menetelmien Aikakaudet. Hakupäivä 27.5.2014 http://gofore.com/avainsana/kettera-ohjelmistokehitys/

Hernando Cadet Technology 2013. A Brief History of C Sharp. Hakupäivä 25.3.2015 http://www.hernandocadett.com/content/brief-history-c-sharp#

Haikala, I. & Märijärvi, J. 2000. Ohjelmistotuotanto. Helsinki: Satku.

Juuso Hietalahti 2015. 2D Versus 3D. Hakupäivä 8.4.2015 http://www.gameproducer.net/2009/04/05/2d-versus-3d/

Rocket 5 Studios 2015. Make A 2D Game in Unity3D Using Only Free Tools Part 1. Hakupäivä 8.4.2015 http://www.rocket5studios.com/tutorials/make-a-2d-game-in-unity3d-using-only-free-tools-part-1/

TechTerms.com 2015. Sprite. Hakupäivä 27.4.2015 http://techterms.com/definition/sprite

Unity Technologies 2014a. Unity Scripting Reference Hakupäivä 4.5.2014 http://docs.unity3d.com/Documentation/ScriptReference/index.html

Unity Technologies 2014b. Gameplay in 2D. Hakupäivä 31.3.2015 http://docs.unity3d.com/Manual/Overview2D.html