OULUN AMMATTIKORKEAKOULU

Arttu Rusanen

RUNNER BROS -UNITY PELIN KEHITYS

Runner Bros -Unity-pelin kehitys

Arttu Rusanen
Opinnäytetyö
Syksy 2022
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelman, Ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä(t): Arttu Rusanen
Opinnäytetyön nimi: Runner Bros -Unity-pelin kehitys
Työn ohjaaja(t): Eino Niemi
Työn valmistumislukukausi ja -vuosi: Syksy 2022 Sivumäärä: 24

Työn aikeeksi valitsin kehittää itsenäisesti tietokonepelin käyttäen Unity-pelimoottoria, kun sitä tuli käytettyä aikaisemmassa projektissa ja halusin muutenkin oppia käyttämään Unityä enemmän.

Kehitin peliä Unityn omilla työkaluilla, kuten sen editorilla ja ominaisuuskaupalla. Unityssa käytettävää C#-koodin kirjoittamiseen käytin Visual Studio Code-sovellusta.

Lopputuloksena sain kehitettyä yksinkertaisen tasoloikkapelin, jossa on myös toimiva hahmokauppa. Pelin julkaisin ilmaiseksi netissä pelattavaksi ja sitä voi käydä pelaamassa osoitteessa: https://arru97.itch.io/runner-bros.

Asiasanat: Unity-pelimoottori, tietokonepelit, peliohjelmointi, ohjelmointi

ABSTRACT

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

---

Author(s): Arttu Rusanen
Title of thesis: Runner Bros -Unity game development
Supervisor(s): Eino Niemi
Term and year when the thesis was submitted: Fall 2022
Number of pages: 24

For the subject I chose to independently develop a videogame using the Unity game engine. As I used it earlier in another project and I wanted to learn more about using Unity.

I developed the game using Unity's own tools like the editor and asset store and for the C# code I used Visual Studio Code.

As a result, I made a simple platforming game with a working character shop. I published the game for free one the net and it can be played at: https://arru97.itch.io/runner-bros.

Keywords: Unity-videogame engine, videogames, videogame development, programming

SISÄLLYS

1 JOHDANTO

Valitsin aiheeksi kehittää itsenäisesti tietokonepelin käyttäen Unity-pelimoottoria. Olin ollut jo vähän aikaa kiinnostunut kokeilemaan pelin kehittämistä jollakin ilmaisesti käytettävällä pelimoottorilla ja Unity vaikutta niistä helpoiten opeteltavalta. Päätin myös tehdä itsenäisen työn, niin ei tarvitsisi yrittää hakea yrityksiltä aiheita ja saisin kehittää sellaisen pelin minkä itse haluaisin.

Tutustun ja selitän tässä työssä, minkälaista on kehittää pelin itsenäisesti käyttäen Unityä. Unityä tuli kokeiltua jo aikaisemmassa kouluprojektissa, joten päätin tehdä opinnäytetyön myös sillä.

Pelin aiheeksi päätin valita kaksi ulotteisen Endless Runner -tyyppisen pelin, jossa tarkoituksena olisi juosta ja hyppiä mahdollisimman pitkälle loputtomiin jatkuvassa kentässä. Peli muuttuu koko ajan hankalammaksi siten, että kenttä liikkuu koko ajan nopeammin ja alustat missä hypitään muuttuvat vaikeammiksi. Pelissä kerätään kolikoita, joita voi käyttää kaupassa, jossa voi ostaa uusia hahmoja.

Aihe olisi myös ihan hyvin perusteltu alalle, kun yksin tai pienissä ryhmissä pelien kehitys on hyvinkin yleistä ja niin on julkaistu monia menestyneitä tietokonepelejä. Jos seuraa tietokonepelialustoja, kuten Steam, niille ilmestyy jatkuvasti myyntiin uusia tämän kaltaisia pelejä, joita on kehitetty käyttäen jotakin ilmaisesti käytettävää pelimoottoria. Menestyneitä Unityllä kehitettyjä pelejä ovat esimerkiksi Cuphead, Hollow Knight ja Among Us (1).

2 TYÖKALUT JA TEKNIIKAT

Tässä kappaleessa käyn läpi, minkälaista on pelinkehitys käyttäen Unityä.

2.1 Unity

Unity on pelimoottori, jolla voi luoda kaksi- tai kolmiulotteisia sovelluksia monille eri alustoille. Unityllä voi kehittää joko tietokone- tai mobiilipelejä, mutta sitä voi myös soveltaa muihinkin kuin pelkästään pelien kehittämiseen. Unityä voi käyttää erimerkiksi myös arkkitehtuuriin, autoteollisuuteen tai elokuvion tekoon. Unity on myös rojaltivapaa, se toimii yli 20 alustalla ja sillä on noin 1,5 miljoonaa kuukausitta käyttäjää. (2.)

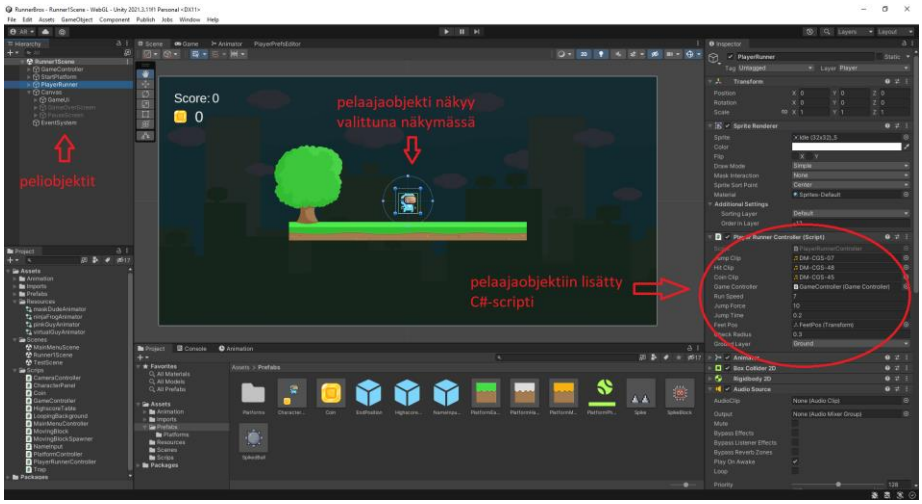
Unitylla on monia eri tilausmalleja, joka määräytyy Unityä käyttävän yrityksen tuloista. Yksilöiden tai pienten yritysten, joiden tulot eivät ylitä pienintä tulorajaa, Unitylla voi kehittää ja julkaista sovelluksia ilmaiseksi. Unity ei ole ainoa pelimoottoria, jota voi käyttää ilmaiseksi tällä tavalla. Unity kuitenkin vaikutti ainakin itselle niistä helpoiten opeteltavalta, minkä takia päätin käyttää sitä. (3.)

Jos haluaa keskittyä pelin logiikan kehittämiseen eikä välttämättä käyttää aikaa grafiikoiden tekemiseen tai niiden erikseen tilaamiseen, Unitylle on myös käytettävissä ominaisuuskauppa. Ominaisuuskaupasta voi hankkia omaan sovellukseen valmiiksi tehtyjä ominaisuuksia, kuten grafiikoita, ääniä tai työkaluja. Ominaisuudet ovat muiden kehittäjien tekemiä ja ne voivat olla joko ilmaisia tai maksullisia. Itse käytin projektissa suosituimpia 2D-peliin soveltuvia grafiikoita ja ääniä ja pari lisätyökalua.

2.2 Pelin kehitys Unityllä

Unityllä kehittäminen toimii Unityn omalla editorilla, jonka käyttö voi olla tilausmalleja seuraten ilmaista. Sovellusten tekeminen alkaa näkymistä. Pelin eri osille tehdään eri näkymiä. Eri näkymät voi olla esimerkiksi pääsivu tai pelin eri kenttiä. Projektiin määritetään sen jälkeen peli sisäisiä asetuksia ja näkymiin lisätään peliin objekteja. Nämä objektit voivat olla esimerkiksi pelihahmoja, esteitä tai käyttöliittymä. Näille objekteille lisätään osia, joilla määritetään mitä objektit teke-

vät. Osiin kuuluu esimerkiksi animaatiokontrolleja, äänilähteitä tai koodinpätkiä. Kuvasta 1 näkee valittuna pelaajaobjektin ja sen C#-skriptikomponentin.



KUVA 1. Unity Editor

Unityllä kehittäminen vaatii jonkinlaista koodin kirjoittamista, koska Unity käyttää logiikan pyörittämiseen C#-koodikieltä. Sillä pitää esimerkiksi määrittää miten peliobjektit liikkuvat tai reagoivat toistensa kanssa. Esimerkiksi pelaajaobjektin sisällä oleva C#-skripti ottaa vastaan liikkumista varten määritetyt näppäinsyötöt ja liikuttaa pelaajaobjektia sen mukaan. Koodin kirjoittamista varten tarvitsee erillisen koodinkirjoitusohjelman. Itse käytin omassa projektissa Visual Studio Code -sovellusta koodin kirjoittamiseen.

Unityssa on koodin kirjoittamista varten omia C#-komentoja objektien manipulointia varten. Näitä komentoja pitää opetella käyttämään, jotta saa pelin logiikan pyörimään luonnollisesti. Kun Unityssa luo uuden C#-skriptin, se luo kuvan 2 mukaisen esimerkkikoodin.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Test : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```

KUVA 2. Unityn luoma C#-skriptiesimerkki

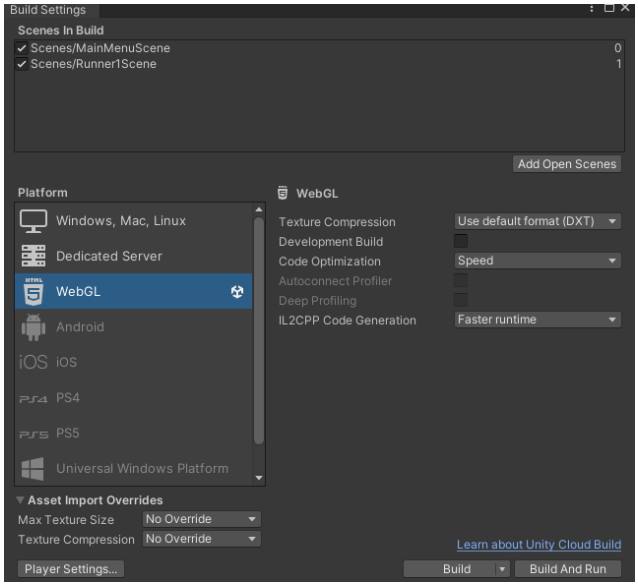
Unityn omilta nettisivuilta löytyy Unity Learn -osio, jota kannattaa hyödyntää Unity-kehitystä opetellessa. Sieltä löytyy paljon eri opetussuunnitelmia, kursseja, esimerkkiprojekteja ja tutoriaaleja Unityllä kehittämiseen liittyviin aiheisiin. Itse seurasin siellä sellaista opetussuunnitelmaa, jossa käytiin läpi 2D-pelin kehittymisen perusteet. (4)

Koska Unity on ilmaisesti käytettävissä, sille on iso yhteisö, joka tekee myös tutoriaaleja Unityn omien nettisivujen ulkopuolella. Kannattaa käyttää tämän yhteisön tekemiä hyväksi omaa sovellusta tehtäessä. Oman projektin aikana tuli monesti haettua ratkaisuja hyvinkin yksityiskohtaisille ongelmille, joille melkein aina löytyi ratkaisu joltakin foorumilta tai YouTubesta.

Unity pelin kehitysaika voi heitellä suuresti sen mukaan, kuinka suurta peliä on tekemässä ja kehittäjän taidoista. Ammatillisesti julkaistujen Unity pelien kehityksessä voi mennä useita vuosia. Oman pelin tekemisessä meni noin muutama viikko, mutta ammattilaisella siihen olisi todennäköisesti mennyt pari päivää.

2.3 Unity-pelin julkaisu

Kun pelin on saanut johonkin pelattavaan muotoon ja sen haluaa julkaista jonnekin, se pitää rakentaa pelattavaksi halutulle alustalle. Kuten kuvasta 3 huomaa, sovelluksen teossa pitää erikseen valita, mille alustalle haluaa rakentaa. Riippuen minkä alustaa valitse, pitää muistaa säätää projektin asetuksia, jotta sovellus toimii oikein. Omaa peliä varten minun piti esimerkiksi laittaa pelin soittimen kompressointi pois päältä tai se ei lähtenyt pyörimään ollenkaan.



KUVA 3. Sovelluksen rakennusikkuna.

Esimerkiksi jos pelistä haluaa julkaista tietokoneella pyörivän itsenäisesti ladattavan version, pitää valita ensimmäinen vaihtoehto eli Windows, Mac, Linux. Jos haluaa, että peli voi pelata nettiselaimella, pitää valita WebGL-vaihtoehto ja sen voi julkaista jollekin pelialustasivulle. Näihin sivuihin kuuluu Unityn oma Unity Play ja suosittu indie-videopeli julkaisusivu itch.io (5). Unityn oma alusta kuuluu pelkästään Unity-peleille, mutta Itch.io-sivulla voi olla muilla moottoreille kehitettyjä pelejä ja siellä peleistä voi tehdä maksullisia.

Pelin voi mahdollisesti myös julkaista maksullisena muille alustoille, kuten mobiili- tai konsolipelinä. En perehtynyt niillä julkaisuun, koska halusin pitää pelin pienenä ja ilmaisena. Maksulliseen julkaisuun perehtymiseen olisi vienyt liikaa aikaa aikarajoituksien takia.

3 RUNNER BROS -PELI

Tässä luvussa käyn yleisesti läpi tekemääni peliä, sen ominaisuuksia ja toimintaa. Loppujen lopuksi peli oli melko pieni. Pelistä löytyy kaksi näkymää: pääsivu ja juoksupeli. Pelin grafiikoita tai ääniä en tehnyt itse, vaan hankin ne Unity-ominaisuuskaupasta.

Peliä voi käydä pelaamassa osoitteessa: https://arru97.itch.io/runner-bros.

3.1 Juoksupeli

Pelin pääominaisuus eli juoksupeli on loputtomiin jatkuva tasoloikkapeli, jossa tarkoituksena selvitä mahdollisimman pitkälle. Pelissä juostaan ja hypitään reaaliajassa luotujen kentänpalasten päällä, joita näkee kuvassa 4. Peli muuttuu ajan myötä koko ajan hankalammaksi muuttumalla koko ajan nopeammaksi ja siten, että alustat, millä hypitään, muuttuu monimutkaisemmiksi. Pelissä ei ole loppua, mutta sen aikana voi kerätä kolikoita ja siinä on pistelaskuri, joka kasvaa ajan myötä. Ennätyspistemäärä menee talteen ja sitä voi yrittää rikkoa jokaisella juoksukerralla.

KUVA 4. Juoksupeli.

Pelaajaobjektin sisällä oleva skripti ottaa vastaan näppäinsyöttöjä ja liikuttaa hahmoa niitten mukaan. Hyppiminen on toteutettu vähän monimutkaisemmin, koska mitä enemmän hyppynappia painaa, sitä korkeammalle hahmo hyppää. Hyppyskriptin näkee kuvasta 5. Alussa myös tarkiste-

taan, minkä hahmon pelaaja oli valinnut kaupassa, ja otetaan vastaava animaattorin käyttöön. Hahmolla on osuman ottamiseen ja kolikon keräämiseen komennot, jotka näkee kuvasta 6.

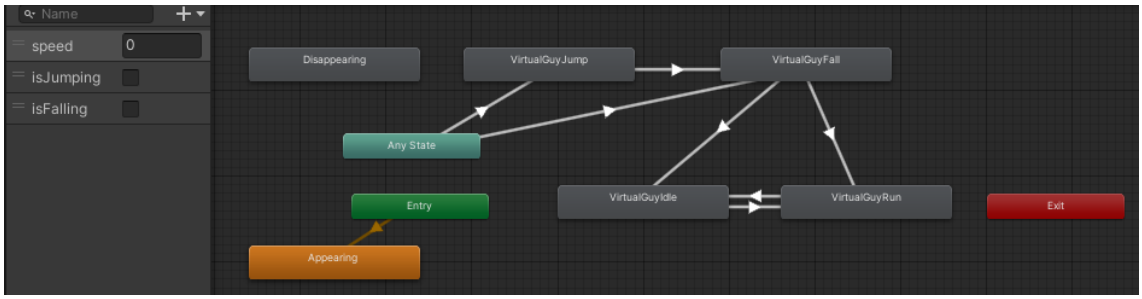
```
void Update(){
    isGrounded = Physics2D.OverlapCircle(feetPos.position, checkRadius, groundLayer);
    //tarkistetaan koskeeko pelaaja maata
    if (!gameController.gamePaused)  { //tarkistetan että peli ei ole pysäytetty
        runSpeed = runSpeed + 0.0001f; //suurennetaan juoksunpeutta ajan myötä
        if(isGrounded && Input.GetButtonDown("Jump")){ //jos kosketaan maata ja painetaan hyppynappia
            isJumping = true; //kerrotaan että hypätään
            jumpTimeCounter = jumpTime; //otetaan käyttöön ennakkoon määritetty hyppyaika
            rigidbody2d.velocity = Vector2.up * jumpForce; //suurennetaan hahmon kiihtyvyyttä ylöspäin
            animator.SetBool("isJumping", true); //kerrotaan animaattorille että hypätään
            audioSource.PlayOneShot(jumpClip); //soitetaan hypppyääni
        }

        if(Input.GetButton("Jump") && isJumping){ //jos hyppynappi on vieläkin painettuna
            if(jumpTimeCounter > 0){ //tarkistetaan että hyppyaikaa on jäljellä
                rigidbody2d.velocity = Vector2.up * jumpForce; //lisätään kiihtyvyyttä
                jumpTimeCounter -= Time.deltaTime; //pienennetään hyppyaikaa ajan myötä
            } else {
                isJumping = false; //määritetään että hyppääminen loppui
            }
        }

        if(Input.GetButtonUp("Jump")){ //jos hyppynapista päästetään irti
            isJumping = false; //lopetetaan hyppääminen
            animator.SetBool("isJumping", false); //otetaan hyppyanimaatio pois käytöstä
        }

        animator.SetFloat("speed", Mathf.Abs(rigidbody2d.velocity.x));
        //kerrotaan animaattorille kiihtyvyys jouksuanimaatioita varten
        if(rigidbody2d.velocity.y < 0){
            //tarkistetaan putoamiskiihtyvyys ja kerrotaan animaattorille että pudotaanko
            animator.SetBool("isFalling", true);
        } else{
            animator.SetBool("isFalling", false);
        }

        if(moveInput > 0){ //käännetään hahmo ympäri riippuen mihin liikutaan
            sprite.flipX = false;
        } else if(moveInput < 0){
            sprite.flipX = true;
        }

        if(rigidbody2d.velocity.y < -100){ //lopetaan peli jos pudotaan pois kentästä
            animator.SetBool("isFalling", false);
            Hit();
        }
```

KUVA 5. Pelaajaskriptin hyppäämiseen liittyvä osuus.

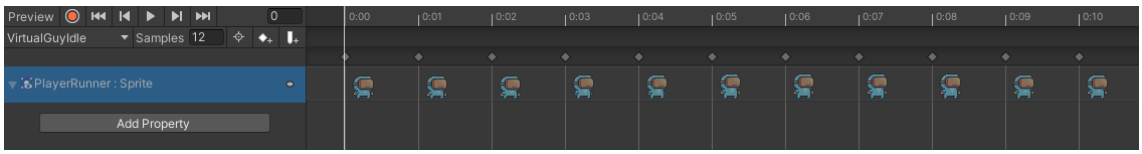
```
public void Hit() { //pelaajan osumanotto komento
    animator.SetBool("isJumping", false);
    animator.SetBool("isFalling", false);
    //kerrotaan animaattori lopettamaan muut animaatiot
    audioSource.PlayOneShot(hitClip); //soitetaan osumaääni
    animator.Play("Disappearing"); //kerrotaan animttoori pelaamaan katoamis animaatio
    gameController.EndGame(); //kerrotaan pelinohjaajalle lopettaa peli
}

public void AddCoin(){ //kolikon lisäämis skripti
    audioSource.PlayOneShot(coinClip); //soitetaan kolikkoääni
    gameController.AddCoin(); //kerrotaan pelinohjaajalle lisäämään kolikko
}
```

KUVA 6. Pelaajaskriptin osuma- ja kolikkokomento.

Hahmon animointia varten objektilla on animaatiokontrolleri, jonka näkee kuvasta 7 ja sen sisällä on tarvittavat animaatiot. Nämä animaatiot tein valmiista kuvista Unityn omalla animaatiotyökalulla. Eri animaatiot laitetaan pyörimään sen mukaan, mitä muuttujia animaatiokontrollerille lähetetään. Esimerkkianimaation näkee kuvasta 8.



KUVA 7. Unity-animaattori



KUVA 8. Esimerkki juoksuanimaatiosta

Pelissä on kontrolleri, joka suorittaa kentän luonnin reaaliajassa. Kontrolleri liikkuu koko ajan oikealle ja sen vasemmalla puolella on seinä, joka pakottaa pelaajaa liikkumaan oikealle. Pelin kamera on määritetty osaksi kontrolleria, minkä takia kameraa liikkuu kontrollerin mukana, mutta seuraa pelaajaa y-akselissa, jotta pelaaja ei katoa ruudulta. Kameran skriptin näkee kuvasta 9.

```
public class CameraController : MonoBehaviour
{
    public Transform player;
    Vector3 offset;

    // Update is called once per frame
    void Update()
    {
        Vector3 position = transform.position ;
        position.y = (player.position + offset).y;
        transform.position = position;
        //muutetaan kameran sijaintia y-akselissa pelaajaobjektin mukaan.
    }
}
```

KUVA 9. Kameraskripti.

Pelin kontrolleri hoitaa pelin muita logiikoita, kuten pelin pysäyttämisen, jatkamisen ja käyttöliittymän ylläpitämiseen. Kontrollerin skriptiä näkee kuvista 10, 11 ja 12.

```
void Start(){
    gamePaused = false;
    Time.timeScale = 1;

    previousEndPosition = startPlatform.Find("EndPosition").position;
    //otetaan aloitus kentän palasen loppukohta

    int startingSpawnPlatforms = 3;
    for (int i = 0; i < startingSpawnPlatforms; i++){
        SpawnPlatform();
        //luodaan aloksu kolme kentän palasta
    }

    if (testingPlatform != null) {
        Debug.Log("Using Testing Plarform");
    }
}

void Update(){
    scoreCounter.text = score.ToString("f0");
    //muutetaan käyttöliittymän pistetekstiä

    if(!gamePaused){ //jos peli ei ole pysäytetty
        score += Time.deltaTime * scoreMultiplier;
        //suurennetaan pisteitä riippuen ajasta ja moninkertojasta
        gameSpeed = gameSpeed + 0.00005f;
        //suurennetaan pelin nopeutta kokoajan

        if(Input.GetButtonDown("Cancel")) { Pause(); }
        //suoritetaan pelin pysäytys komento jas painetaan menunappia
    }
    else {
        if(Input.GetButtonDown("Cancel")) { Resume(); }
        //suoritetaans en sijaan pelin jatkamis komento jos peli on pyäystetty
    }

    transform.position += new Vector3(gameSpeed * difficulty * Time.deltaTime, 0, 0);
    //muutetaan kontrollerin sijaintia riippuen pelin vaikeudesta ja ajasta

    if (Vector3.Distance(player.GetPosition(), previousEndPosition) < PLAYER_DISTANCE_SPAWN_PLATFORM_PART){
        SpawnPlatform();
        //aloitetaan kenttien luominen riippuen kuinka kaukana pelaaja on edellisestä palasesta
    }
}
```

KUVA 10. Kontrollerin aloitus- ja päivityskomennot.

```
public void EndGame(){
    gamePaused = true;
    Time.timeScale = 0;
    //pysäytetään peli ja aika

    int totalCoins = PlayerPrefs.GetInt("playerCoins");
    totalCoins = totalCoins + coinCount;
    PlayerPrefs.SetInt("playerCoins", totalCoins);
    TotalCoinCounter.text = totalCoins.ToString();
    //laitetaan kerätyt kolikot talteen ja näytetän kaikki kolikot näkymiin

    finalScoreCounter.text = score.ToString("f0");
    //laitetaan lopulliset pisteet näkymiin
    if(score > PlayerPrefs.GetFloat("recordScore")){
        recordScoreCounter.text = score.ToString("f0");
        PlayerPrefs.SetFloat("recordScore", score);
        //tarkistetaan onko uusi pistemäärä suurempi kuin edellinen ennätys ja päivitetään se tarvittaessa
    }
    else{
        recordScoreCounter.text = PlayerPrefs.GetFloat("recordScore").ToString("f0");
        //muulloin laitetaan vanha ennätys näkymiin
    }

    GameUI.SetActive(false); //otetaan pelin käyttöliittymä pois käytöstä
    GameOverSceen.SetActive(true); //laitetaan pelin lopetusruutu aktiiviseksi
}
```

KUVA 11. Pelin lopetuskomento.

```
public void AddCoin(){
    coinCount++;
    coinCounter.text = coinCount.ToString();
}

public void Restart(){
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(scene.name);
    gamePaused = false;
    Time.timeScale = 1;
}

public void Reset(){
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(scene.name);
    gamePaused = false;
    Time.timeScale = 1;
}

public void Pause(){
    gamePaused = true;
    Time.timeScale = 0;
    PauseScreen.SetActive(true);
}

public void Resume(){
    gamePaused = false;
    Time.timeScale = 1;
    PauseScreen.SetActive(false);
}

public void Exit(){
    SceneManager.LoadScene(sceneName:"MainMenuScene");
}
```

KUVA 12. Loput kontrollerissa olevat komennot.

Kentän luonti toimii siten, että kameran ulkopuolelle oikealle syntyy jatkuvasti sekalaisesti ryhmästä valittuja kentän palasia, jotka liikkuvat vasemmalle, kunnes ne poistuvat ruudulta ja katoavat ajan myötä. Kun kentänosien määrä ylittää tietyn rajan, niitten ryhmä vaihtuu vaikeampaan. Ryhmiä pelissä on kolme ja niissä on viisi erilaista palasta. Kentän luontia varten olevat koodinpätkän näkee kuvista 13 ja 14.

```
private Difficulty GetDifficulty(){
    if (platformCount >= 8){ //tarkistetaan kentän palasten määrä
        scoreMultiplier = 1.5f; //suurennetaan pistekertojaa
        return Difficulty.Medium; //vaihdetaan vaikeustaso
    }
    else if(platformCount >= 16){
        scoreMultiplier = 2.0f;
        return Difficulty.Hard;
    }
    else return Difficulty.Easy;
}
```

KUVA 13. Koodinpätkä vaikeustason vaihtamisesta.

```
private Transform SpawnPlatform(Transform platform,Vector3 spawnPosition){
    Transform platformTransform = Instantiate(platform, spawnPosition, Quaternion.identity);
    return platformTransform;
    //luodaan ja palautetaan kentänpalanen
}

private void SpawnPlatform(){
    List<Transform> chosenPlatformList;

    switch (GetDifficulty()) { //valitaan kenttälista vaikeustason mukaan
        default:
        case Difficulty.Easy: chosenPlatformList = platformListEasy; break;
        case Difficulty.Medium: chosenPlatformList = platformListMedium; break;
        case Difficulty.Hard: chosenPlatformList = platformListHard; break;
    }

    Transform chosenPlatform = chosenPlatformList[Random.Range(0, platformListMedium.Count)];
    //valitaan sekalaisesti listalta kentän palanen

    if (testingPlatform != null) { //tarkistetaan onko testi kentänpala käytössä
        chosenPlatform = testingPlatform; //muutetaan kokolista testipalaksi
    }

    Transform previousPlatform = SpawnPlatform(chosenPlatform, previousEndPosition);
    //lähetään kentän luonti komennolle valittu palanen ja edellisen palasen loppukohta
    previousEndPosition = previousPlatform.Find("EndPosition").position;
    //etsitään kentänpalasen loppukohta, joka oli määritetty erikseen joka palaselle
    platformCount++; //suurennetaan palasten määrää
}
```

KUVA 14. Koodinpätkä kentän palasten luonnista.

Kentän palasissa on ansoja ja pelissä on erikseen syntyviä liikkuvia piikkipalikoita, joita pitää väistää. Ansoissa on skripti, joka kertoo pelaajaobjektille, että pelaaja otti osuman. Kolikoilla sen sijaan skripti, joka pyytää pelaajaobjektia lisäämään kolikkomäärää ja tuhoaa itsensä. Peli loppuu, kun pelaaja osuu ansaan tai vasempaan sivurajaan tai tippuu kentästä. Ansojen ja kolikoiden koodin näkee kuvista 15 ja 16.

```
public class Trap : MonoBehaviour
{
    void OnCollisionEnter2D(Collision2D other)
    {
        PlayerRunnerController player = other.gameObject.GetComponent<PlayerRunnerController>();

        if(player != null){ //tarkistetaan oliko se objekti mitä kosketettiin pelaaja
            player.Hit(); //kerrotaan pelaajalle että se osui ansaan
        }
    }
}
```

KUVA 15. Ansojen skripti.

```
public class Coin : MonoBehaviour
{
    void OnTriggerEnter2D(Collider2D other)
    {
        PlayerRunnerController controller = other.gameObject.GetComponent<PlayerRunnerController>();

        if (controller != null)
        {
            controller.AddCoin();
            Destroy(gameObject);
        }
    }
}
```

KUVA 16. Kolikkoskripti.

Liikkuvien ansojen skripti on hieman monimutkaisempi, kun ne liikuttavat itseään ja tuhoavat itsensä ajan myötä tai kun ne koskettavat jotain muuta kuin pelaajaa. Niitten luomiseen liittyvä skripti on lisättä pelin kameraan. Niiden oman skriptin ja niiden luontia varten pelikamerassa olevan koodin näkee kuvista 17 ja 18.

```
public class MovingBlock : MonoBehaviour
{
    public float speed = 3.0f;
    private float secondsToDestroy = 30f;

    void Start(){
        StartCoroutine(DestroySelf());
        //aloiteaan itsensä tuhoamis komento
    }

    void FixedUpdate(){
        Vector2 position = GetComponent<Rigidbody2D>().position;
        //tarkistetaan objektin nykyiden sijainti

        position.x = position.x - Time.deltaTime * speed;
        //muutetan sijaintia vasemmalle riippuen ajasta ja nopeudesta

        GetComponent<Rigidbody2D>().MovePosition(position);
        //kerrotaan uusi sijainti objektille
    }

    IEnumerator DestroySelf()
    {
        yield return new WaitForSeconds(secondsToDestroy);
        Destroy(gameObject);
        //tukotaan objekti kun määritetty aika on kulunut
    }

    void OnCollisionEnter2D(Collision2D other)
    {
        PlayerRunnerController player = other.gameObject.GetComponent<PlayerRunnerController>();

        if(player != null){ //tarkistetaan oliko objekti mitä kosketetiin pelaaja
            player.Hit(); //jos oli lähetään pelaajalle osuma komento
        }
        else{ //jos se ei ollut pelaaja tuhotaan objekti
            Destroy(gameObject);
        }
    }
}
```

KUVA 17. Liikkuvien ansojen skripti

```
public class MovingBlockSpawner : MonoBehaviour
{
    public GameObject movingBlock;
    public float maxY;
    public float minY;
    public float x;
    public float timeBetweenSpawn;
    private float spawnTime;
    //unityn sisällä määritetyn muuttujat

    void Update(){
        if(Time.time > spawnTime){ //jos aikaa on kulunut tiettä määrä
            Spawn(); //suoritetaan luomis komento
            spawnTime = Time.time + timeBetweenSpawn; //resetoidaan luomisaika
        }
    }

    void Spawn(){
        float randomY = Random.Range(minY, maxY);

        Instantiate(movingBlock, transform.position + new Vector3(x, randomY, 10), transform.rotation);
        //luodaan uusi liikkuva ansa sekalaisesti y-akselin mukaan
    }
}
```

KUVA 18. Kamerassa oleva liikkuvien ansojen luontiskripti.

3.2 Pääsivu ja pelikauppa

Pelin pääsivu ja hahmokauppa ovat osa samaa näkymään. Pääsivun pelikuvake vaihtaa näkymän juoksupeliin, mutta kauppapainike vain vaihtaa käyttöliittymässä hahmokaupan aktiiviseksi. Tälle näkymälle on myös oma kontrolleri, joka muuttaa käyttöliittymää ja sen koodia näkee kuvista 19 ja 20.

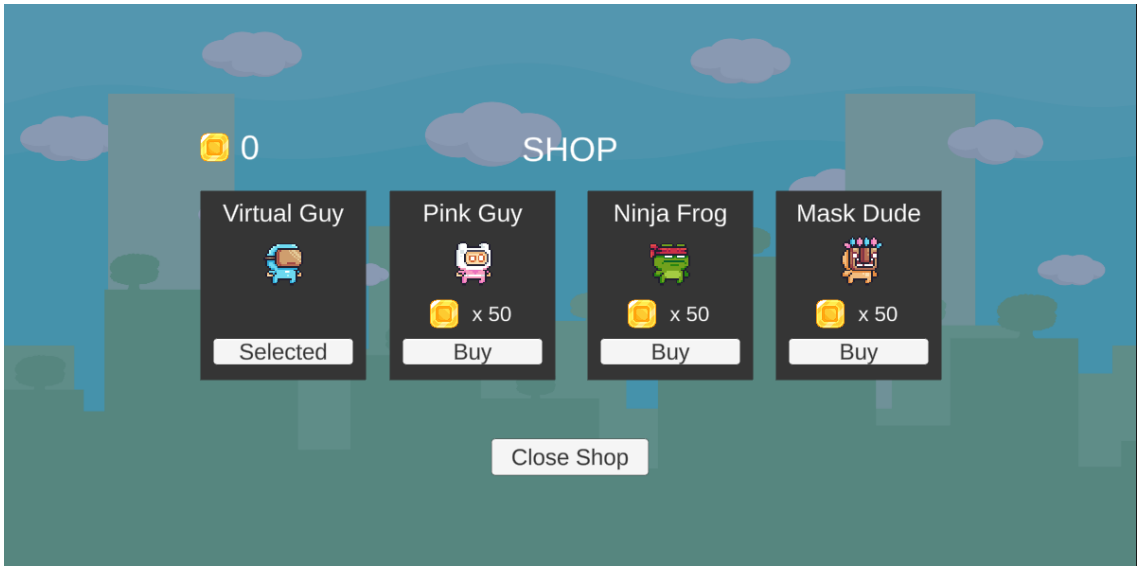
```
void Start(){
    audioSource = GetComponent<AudioSource>();

    if(PlayerPrefs.GetInt("firstTime") != 1){
        PlayerPrefs.SetInt("virtualGuy", 1);
        PlayerPrefs.SetString("playerCharacter", "virtualGuy");
        PlayerPrefs.SetInt("firstTime", 1);
        //tarkistetaan pelataanko peliä ekaa kertaa
        //ja laitetaan yksi hahmoista valituksi
    }

    coinCount.text = PlayerPrefs.GetInt("playerCoins").ToString();
    //otetaan muististi kolikoiden määrä näkymiin
}
```

KUVA 19. Kontrollerin aloituskomento

```
public void LoadRunner1(){
    PlayButtonSound();
    SceneManager.LoadScene(sceneName:"Runner1Scene");
    //vaihdetaan juoksupelinäkymään
}

public void OpenShop(){
    PlayButtonSound();
    mainMenu.SetActive(false);
    shopMenu.SetActive(true);
}

public void CloseShop(){
    PlayButtonSound();
    shopMenu.SetActive(false);
    mainMenu.SetActive(true);
}

public void CharacterBought(){
    alertText.text = "Character Bought!";
    audioSource.PlayOneShot(buyClip);
    coinCount.text = PlayerPrefs.GetInt("playerCoins").ToString();
    //vähennetään hahmon ostamiseen käytetyt kolikot muistista
}

public void NotEnoughCoins(){
    audioSource.PlayOneShot(failClip);
    alertText.text = "Not enough coins!";
}

public void PlayButtonSound(){
    audioSource.PlayOneShot(buttonClip);
}
```

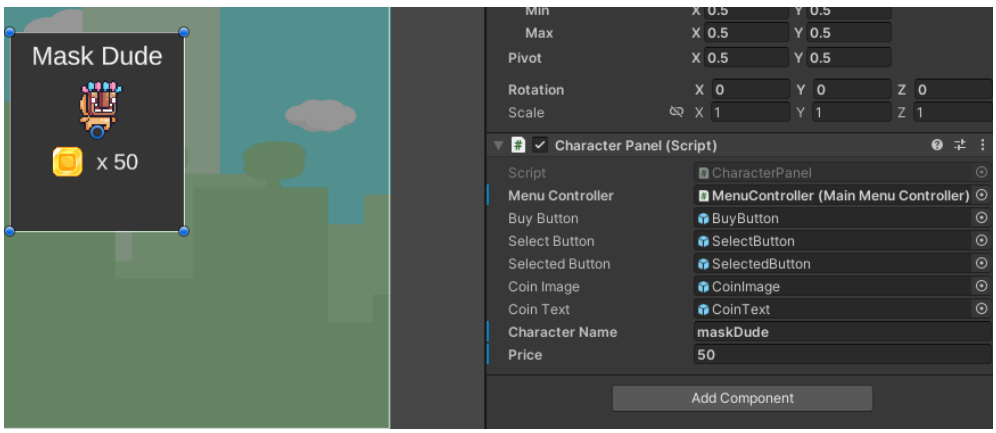
KUVA 20. Loput kontrollerin komennot.

Hahmokaupassa voi ostaa erinäköisiä hahmoja ja sen näkee kuvassa 21. Hahmoilla ei ole pelissä muuta eroa kuin näkö. Jos peliä haluaisi laajentaa, hahmoilla voisi olla sellaisia eroja, että toinen hyppää korkeammalle, juoksee nopeampaa tai kestää useamman osuman.



KUVA 21. Hahmokauppa

Hahmopaneelit on tehty itse tehdystä esimerkkipaneelista, johon erikseen määritetään hahmolle nimi ja hinta. Jokaisessa hahmopaneelissa on skripti, jonka näkee kuvista 22, 23 ja 24. Skripti tarkistaa, onko hahmoa ostettu, ja suorittaa sen ostamisen.



KUVA 22. Valittu hahmopaneeli ja siihen lisätty skripti.

```
void Start(){
    if(PlayerPrefs.GetInt(characterName) == 1){ //tarkistetaan onko hahmo ostettu
        characterBought = true;
        coinImage.SetActive(false);
        coinText.SetActive(false);
        //määriteen hahmo ostetuksi ja laitetaan kolikon kuva ja teksi pois käytöstä
    }
}

void Update(){
    if(characterName == PlayerPrefs.GetString("playerCharacter")){
        characterSelected = true;
    }
    else{
        characterSelected = false;
    }
    //tarkistetaan muistiita onko hahmo valittu

    if(characterBought & !characterSelected){
        BuyButton.SetActive(false);
        SelectButton.SetActive(true);
        SelectedButton.SetActive(false);
    }
    else if(characterSelected){
        BuyButton.SetActive(false);
        SelectButton.SetActive(false);
        SelectedButton.SetActive(true);
    }
    else{
        BuyButton.SetActive(true);
        SelectButton.SetActive(false);
        SelectedButton.SetActive(false);
    }
    //laitetaan tietty nappi aktiiviseksi riipuen onko hahmo ostettu tai valittu
}
```

KUVA 23. Hahmopaneeliskriptin aloitus- ja päivityskomento.

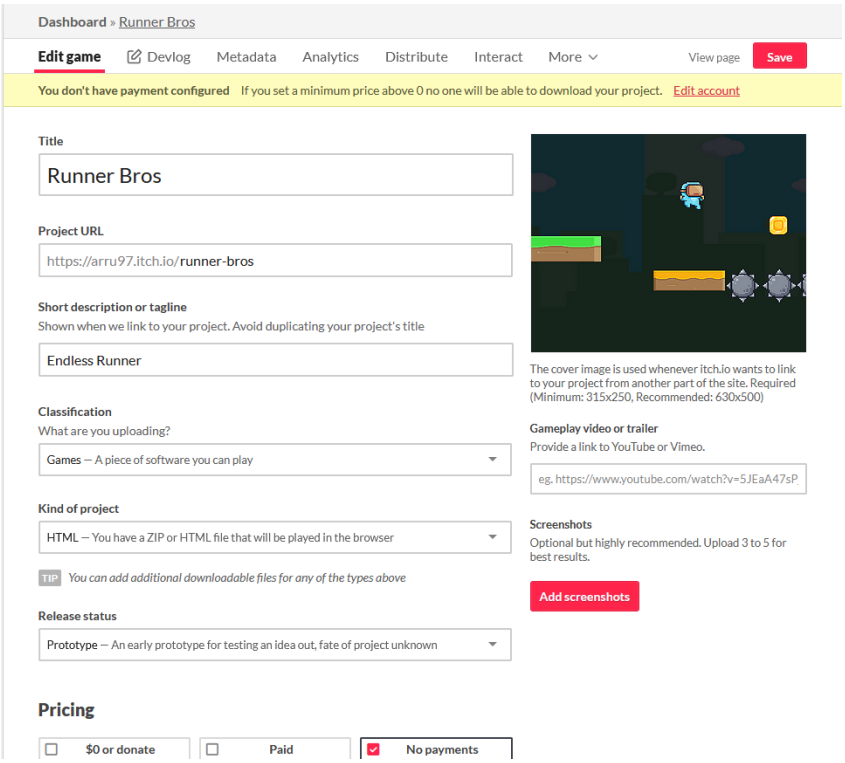
```
public void Buy(){ //hahmon ostokomento
    int playerCoins = PlayerPrefs.GetInt("playerCoins");
    //otetaan muistista kolikoiden määrä

    if(playerCoins >= price){ //tarkistetaan onko koikoit tarpeeksi
        PlayerPrefs.SetInt(characterName, 1); //määritetään muistiin hahmo ostetuksi
        playerCoins = playerCoins - price;
        PlayerPrefs.SetInt("playerCoins", playerCoins); //vähennetään muistista kolikoita
        characterBought = true;
        coinImage.SetActive(false);
        coinText.SetActive(false);
        menuController.CharacterBought();
    }
    else{
        menuController.NotEnoughCoins();
        //jos kolikoita ei ollut tarpeeksi kolikoita lähetään siitä viesti kontrollerille
    }
}

public void Select(){ //hahmon valintakomento
    menuController.PlayButtonSound();
    PlayerPrefs.SetString("playerCharacter", characterName);
}
```

KUVA 24. Hahmopaneeliskriptin osto- ja valintakomennot.

Hahmot ostetaan juoksupelistä kerätyillä kolikoilla. Pelin sisäisiin kauppoihin voi lisätä toiminnon, että pelikolikoita voisi ostaa oikealla rahalla. Tähän peliin en sitä soveltanut, koska halusin pitää pelin ilmaisena ja tähän soveltuminen olisi todennäköisesti vienyt liikaa aikaa.

3.3 Pelin julkaisu

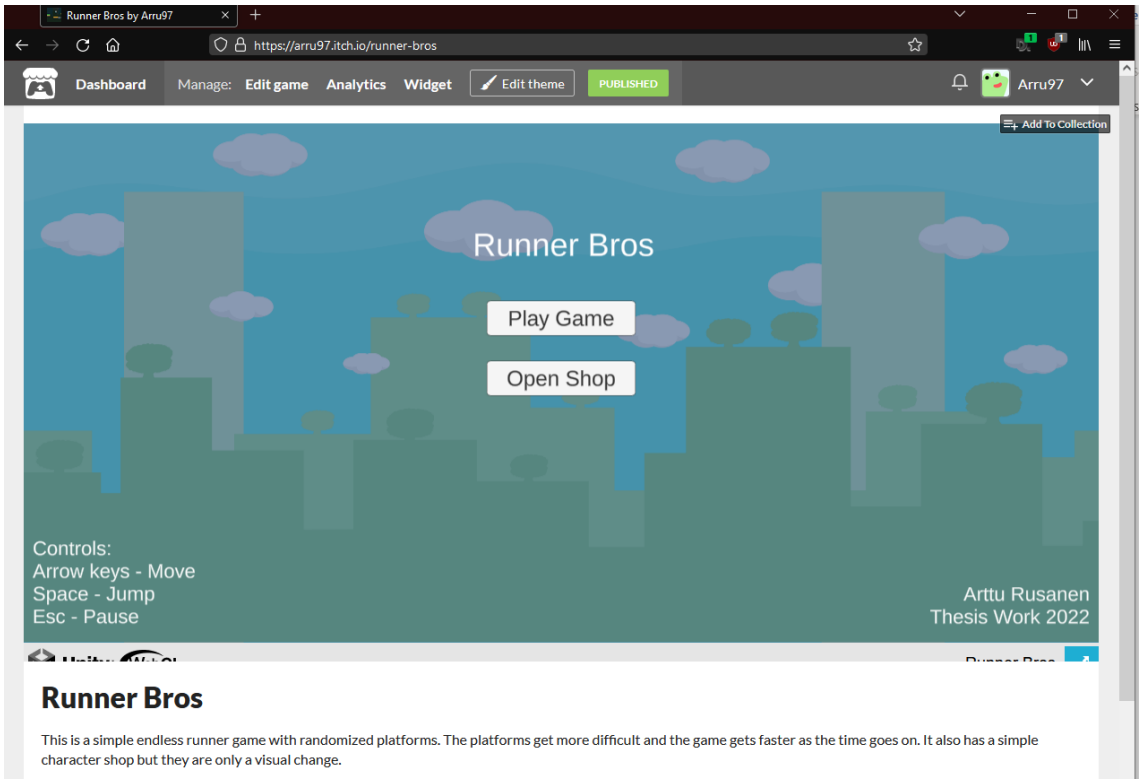
Itse halusin, että peli on pelattavissa nettiselaimen kautta. Aluksi ideana oli julkaista peli Unityn omalle pelialustasivulle, mutta peliä sinne upottaessa minulle oli ongelmia, joihin en löytänyt tai keksinyt ratkaisua. Tämän takia päätin julkaista pelin itch.io-sivulla (5). Julkaisua varten peli piti ensin rakentaa WebGL-muodossa. Siitä muodostuneen kansion muutin zip-tiedostomuotoon ja latasin sen itch.io-sivulle. Itch.io-sivulle piti tehdä käyttäjä ja pelille piti antaa tietoja, kuten mikä on pelin nimi ja mihin genreen se kuuluu. Pelin muokkausta varten olevan sivun näkee kuvasta 25.



KUVA 25. Pelein muokkaussivu itch.io-sivulla.

4 JOHTOPÄÄTÖKSET JA POHDINTA

Lopputuloksena opin itsenäisesti pelin kehitystä Unityllä ja sain tehtyä vaatimusten mukaan toimivan Unity-pelin. Sain toteutettua peliin toimivan juoksupelin ja sen kanssa toimivan hahmokaupan. Julkaisin pelin pelattavan version nettiin, jota voi kuka tahansa voi käydä pelaamassa ilmaiseksi. Pelin näkee pelattavana itch.io-sivulla kuvassa 26.



KUVA 26. Pelin pelausikkuna itch.io-sivulla.

Verrattuna muihin ammatillisesti julkaistuihin peleihin lopputulos on melko pieni. Ammatillisia pelejä saatetaan tosin kehittää vuosia ja niitä yleensä tekee kokeneemmat ammattilaiset. Koska ehdin kehittää peliä vain muutaman viikon, sanoisin sen olevan ensimmäiseksi itsenäisesti kehitetyn Indie-peliksi melko hyvä. Pelillä olisi paljonkin laajentamisen varaa, jos siihen varaisi enemmän aikaa. Voin mahdollisesti laajentaa peliä opinnäytetyön ulkopuolella tulevaisuudessa.

LÄHTEET

1. Create & Learn Team 2022. Top Games Made with Unity: Unity Game Programming. Hakupäivä: 7.11.2022. https://www.create-learn.us/blog/top-games-made-with-unity/

2. Unity 2022. Hakupäivä 7.11.2022. https://unity.com/.

3. Unity 2022. FAQ. Hakupäivä 7.11.2022. https://unity.com/faq.

4. Unity Learn 2022. Hakupäivä 16.11.2022 https://learn.unity.com/

5. Itch.io 2022. Hakupäivä 16.11.2022 https://itch.io/