Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Rédai Attila

VR JÁTÉK FEJLESZTÉSE SUGÁRKÖVETÉSSEL

KONZULENS

Dr. Magdics Milán

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Rédai Attila**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 09.

...…….
Rédai Attila

Összefoglaló

A mai világ gyors fejlődése lehetővé teszi számunkra, hogy amit 10-20 évvel ezelőtt csak science fiction filmekben láthatunk már saját magunk is megtapasztalhatjuk ezeket. Ide tartozik a virtuális valóság (VR) is, ami nap, mint nap komoly fejlődéseken megy keresztül. A VR egy virtuális környezetbe tudja helyezni a felhasználót és egy olyan élményt tud nyújtani, amit máshol nem, lehet megtapasztalni. Ahhoz, hogy a VR szemüveg használója minél jobban bele tudja élni magát ebbe a térbe a lehető legvalóság hűbb megjelenítést kell adni, amit a sugárkövetési technológiákkal kiválóan el tudunk érni.

A dolgozatban próbálom bemutatni a VR technológia és a sugárkövetés kombinációjából adódó lehetőségeket egy egyszerű VR játékon keresztül és fő célként azt próbálom elérni, hogy a lehető legkomfortosabb érzetet keltsem a VR szemüveget használó játékosban.

Részletes ismertetésre kerül a VR megjelenítés elméleti háttere és ennek megvalósítása a Unity játékmotorban. Ezen kívül a Unity-s sugárkövetés és annak gépi tanuláson alapuló szűrési technikája (DLSS) is.

A dolgozat végül ismerteti az tesztek eredményét, továbbá a fejlesztés során szerzett tapasztalatokat is összegezi.

Abstract

The rapid development of today's world allows us to experience that what we could only see in science fiction films 10-20 years ago. This includes virtual reality (VR), which is undergoing major developments day by day. VR can place the user in a virtual environment and provide an experience that cannot be experienced anywhere else. In order for the user of the VR headset to be able to immerse themself in this space as much as possible, it is necessary to give the most realistic display as possible, which we can achieve with ray tracking technologies.

In the thesis, I try to present the possibilities arising from the combination of VR technology and ray tracing through a simple VR game and the main goal is to make the player feel as comfortable as possible when he plays the videogame. The theoretical background of VR display and its implementation in the Unity game engine will be described in detail. In addition, Unity ray tracing and its Deep Learning Super Sampling (DLSS).

Finally, the thesis describes the results of the tests and summarizes the experience gained during development.

1 Bevezetés

1.1 Motiváció

A mai világban a virtuális világ már csaknem szerves részévé vált az életünknek. Egyre több elektronikai eszközzel rendelkezünk, melyek segítenek a hétköznapi életben, tanulásban, szórakoztatásunkban vagy esetleg a kikapcsolódásunkban . Idetartozik a VR eszközök is más néven virtuális valóság vagy angol nevén virtual reality. Ezek az eszközök egy olyan számítógépes környezet által létrehozott, mesterséges világot próbálnak meg létrehozni, melybe a felhasználók megpróbál minél jobban belemélyedni és beleélni magát az adott mesterséges valóságban történő eseményekbe [1].

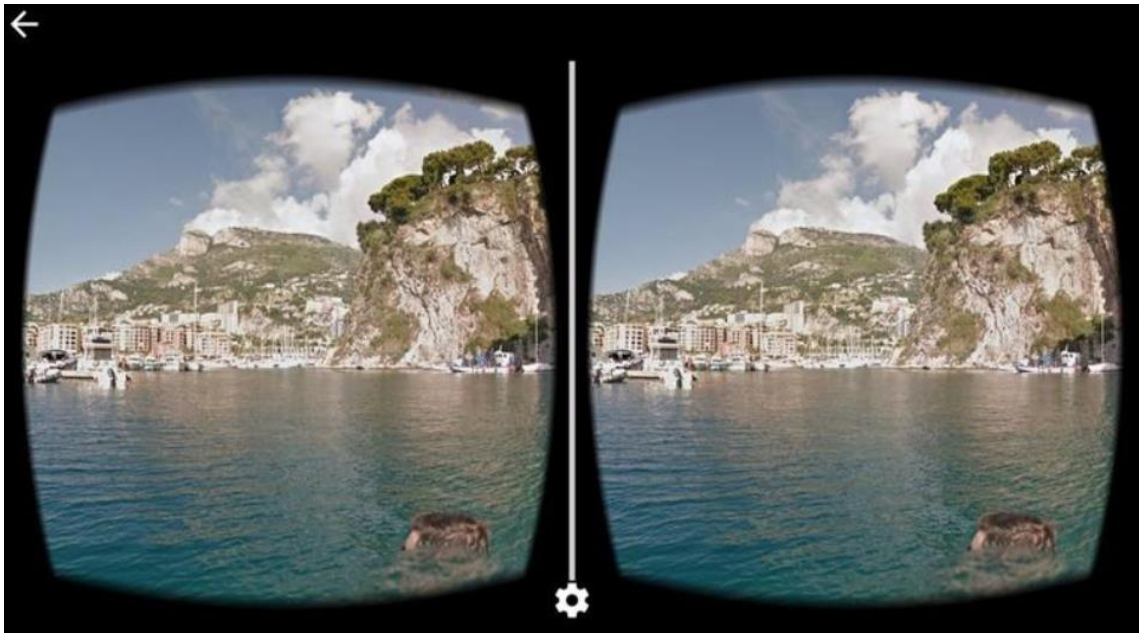
1.2 A virtuális valóság

A virtuális valóság vagy röviden VR egy olyan tér, ami a számítógépes technológiákat felhasználva létrehoz egy 360 fokban belátható szimulációs környezetet. A hagyományos interfészekkel ellentétben a VR a felhasználót az előbb említett virtuális környezetbe helyezi, hogy egy olyan magával ragadó élményben részesüljön, amit máshol nem igazán lehet elérni.

A virtuális valóság speciális, széleskörűen alkalmazható elektronikus technológiák gyűjtőneve is. Magába foglalja az oktatás, sport, ipari tervezés, építészet és tájrendezés, városrendezés, űrkutatás, orvostudomány és rehabilitáció, modellezés és a tudomány számos területét.

1.2.1 A virtuális valóság tulajdonságai

A virtuális valóság egyik legfontosabb tulajdonsága, kritériuma az, hogy a felhasználónak teljesen bele kell élnie magát, hinnie kell abban, hogy ténylegesen része a mesterséges valóságnak. Míg a másik fontos szempont, hogy a virtuális valóságban megjelenő tárgyaknak természetesnek kell tűnniük. Ahhoz hogy a felhasználóban olyan érzést keltsen, mintha benne lenne a mesterséges világban, egy úgy nevezett sztereoszkópia elvén működő eszközt kell használni.(1. ábra) Ennek lényege, hogy képalkotáskor egy olyan térlátási illúziót keltsen, mintha a valóságban lenne.



1. ábra: Szemléltetés arról, hogy mit látunk a VR szemüvegekben

1.2.2 VR headset alapvető működése

A VR szemüveg egy olyan eszköz, amit lényegében arra terveztek, hogy a környezetünket valami szoftver által létrehozott dologgal helyettesítse.

Az eszközben giroszkópos érzékelők, gyorsulásmérő és magnetométerek találhatók, amelyek meg tudják határozni a fej mozgását és ezt le tudják képezni a virtuális térben. Vannak olyan VR headset-ek amiket kamerákhoz és számítógépekhez is hozzá lehet kötni, de vannak olyan eszközök is amik csak a szemüvegből állnak és egy bennük lévő rendszerrel működnek, ilyen például az Oculus Go, Oculus Quest 2. (2. ábra)

2. ábra: Az bal oldalt az Oculus Quest 2, jobb oldalon Oculus Go látható

1.3 A feladat és cél ismertetése

A feladat egy egyszerű VR játék fejlesztése a Unity játékfejlesztő eszközével amely kihasználja a játékmotor által adott sugárkövetés lehetőségeit. A fő cél itt az lenne, hogy egy olyan megjelenítési módszert és paramétereket használjak a játékban, amely a VR eszközt használó felhasználónak komfortérzetét nem csökkenti.

A tesztelésre az egyetem által biztosított Oculus Go VR szemüveget használom, ami egy Android alapú hordozható VR eszköz. Ebből fakadóan kellett még egy szoftver (ALVR) ami lehetővé teszi, hogy a VR szemüvegen a PC erőforrását tudjam használni mivel a headset magában nem képes a valós idejű sugárkövetésre. Erről a szoftverről részletesebben a későbbiekben írok.

1.4 A dolgozat felépítése

A továbbiakban a dolgozat során részletesebben ismertetni fogom az ebben a fejezetben megemlített és hozzájuk kapcsolódó technológiák működését és pontos felhasználását a feladatban. Továbbá egyéb, a fejlesztés során használt technológiákat is bemutatok, amelyek között az ALVR is ott van és ennek a szoftvernek a beüzemeléséről és használatáról is lesz szó.

Ez követően a végzett munkára, menetére, egyes állomásaira, a munka közben felmerült nehézségekre kerül a hangsúly majd részletesen ismertetem a kész szoftver részeit és megoldásait, végül a mért eredményekről lesz szó.

2 Irodalomkutatás és technológiák

2.1 VR Unity játékmotorban

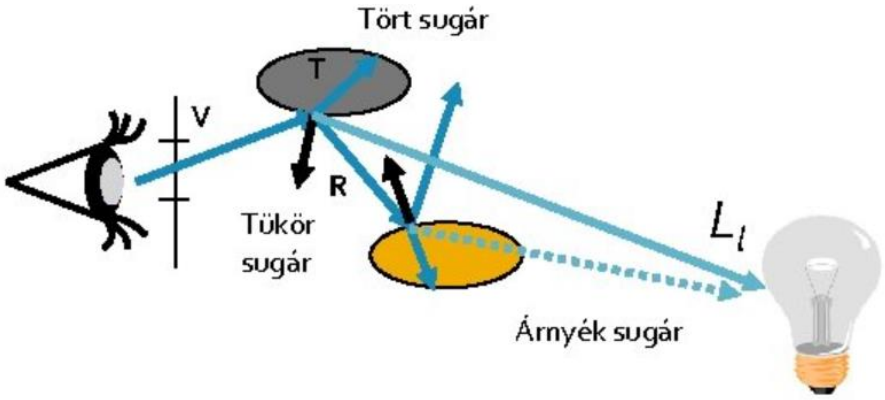
A VR játékfejlesztést támogatja az ingyenes Unity játékmotor mellyel könnyen készíthetők saját fejlesztésű VR játékok. A Unity VR szemüveg nézetét és a vetítési mátrixait automatikusan tudja igazítani a fej helyzetéhez és a látómezőhöz. Sok egyéb eszközt is biztosít, mellyel tesztelni lehet a teljesítményt és játékélményt. Lehetőséget ad ülő és szobaléptékű VR fejlesztésre is. A Unityban a VR rendereléséhez egy úgy nevezett Single Pass Stereo rendering-et használ aminek elve az, hogy egyetlen nagy szélességű képet készít és azt szétosztja a bal és jobb szemre illeszkedő képre.

2.1.1 SteamVR plugin

A Unity Asset Store-ban elérhető SteamVR plugin mely előre elkészített scripteket és prefabokat tartalmaz, mellyel lehetséges a SteamVR rendszerén belül tesztelni a Unity-ben fejlesztett játékokat.

2.2 Sugárkövetés

A sugárkövetés feladata egy háromdimenziós virtuális világról egy kétdimenziós képet alkotni, amelyet aztán a képernyőn meg lehet jeleníteni. A valóságot azért látjuk, mert a fényforrásokból induló fény egy része a tárgyak felületéről visszaverődve a szemünkbe jut. A sugárkövetés során a képszintézist úgy végezzük el, hogy valamilyen pontossággal szimuláljuk a természet e folyamatát. (3. ábra)



3. ábra: Sugárkövetés

2.2.1 Unity sugárkövetés

A Unity motor lehetővé teszi nekünk, hogy valós idejű sugárkövetést alkalmazhassunk játékainkban. Valósághűen szimulálni tudja a fény viselkedését és azt hogyan lép kölcsönhatásba a különböző fizikai tárgyakkal és anyagokkal. A High Definition Render Pipeline (HDRP) sugárkövetés komponense támogatja a hardveres gyorsítást is, amely lehetővé teszi azt is, hogy az összes objektum visszaverődését figyelembe tudja venni még akkor is, ha a képernyőn kívülre esnek [2].

Ez a funkció a Unity 2019.3 verzióban került be a motorba. Annak érdekében, hogy a sugárkövetés el tudjuk érni a Unity-ben ezzel a verzió vagy újabbal kell a projektet indítani. [3]

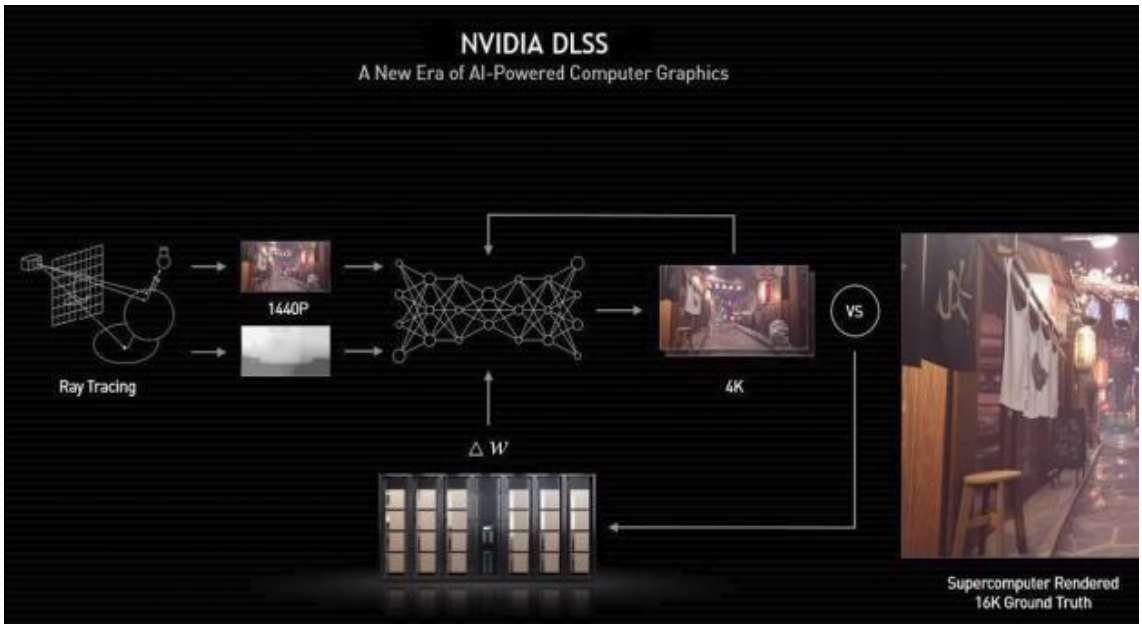
2.2.2 Unity NVIDIA DLSS szűrési technika

A sugárkövetés sokkal valósághűbb képeket tud visszaadni a hagyományos, raszterizáción alapuló inkrementális képszintézishez képest, viszont ez nagy számítási igénnyel is jár. Ezért jelent meg az NVIDIA által fejlesztett DLSS (Deep Learning Super Sampling) ami natív módon támogatott a Unity fejlesztők számára a 2021.2 verziók után [4].

Az NVIDIA megoldása az lett, hogy kevesebb pixel számot és MI használatával nagy felbontású képet készítenek. Az MI neurális hálózat alapú [5] és három bemeneti paramétere van:

1. alacsony felbontású, alias kép
2. az aktuális képkocka mozgásvektorjai
3. nagy felbontású kép

Ezekből a bemenetekből a DLSS gyönyörűen éles, nagy felbontású képet állít össze, amelyre az utófeldolgozás és az UI/HUD kerül (4. ábra). Ezáltal megkapjuk a kellő teljesítény mellett a sugárkövetés valósághű képét. [6]

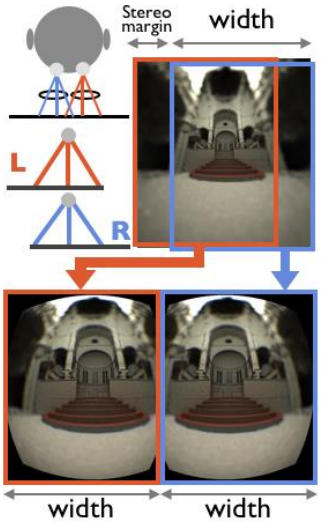


4. ábra: NVIDIA DLSS működési ábrája

2.2.3 Valós idejű sugárkövetés virtuális valóság eszközön

A VR headset egy feltörekvő árucikk, amely nagy frame rate-et biztosító, széles látószögű, fejre helyezhető kijelző, különösképpen az Oculus Rift-ek. Ez a termék megnyitja előttünk lehetőséget, hogy magával ragadó virtuális valóság élményre szert tehessünk. A gyors, alacsony késleltetésű, sima és valósághű megjelenítési módszerek egyre létfontosságúbbak lettek a VR eszközök számára.

Az egyik ilyen megjelenítési módszer a Stereo rendering. Az 5. ábrán, a bal és jobb oldali képen is lehet látni a vetítést, amit az Oculus Rift is használ párhuzamosan. Mivel sok azonos pixel van a bal és jobb oldali képeken, és ha a jelenet léptéke is elég nagy, akkor ezt a két tulajdonságokat kihasználva egy nagyobb széleségű képet készítünk, egyetlen nézőpontból majd pedig az utó feldolgozásban a képet szétosztjuk bal és jobb részre. Ennek a módszernek köszönhetően csökkenthetjük a sugárkövetés adta nagy számítás igényt valamint a véletlenszerű sugarakból adódó zaj is ugyan az lesz mindkét szemre adott képen [7].



5. ábra: Stereo rendering pipeline

2.3 ALVR ismertetése és használata

Az ALVR egy nyílt forráskódú, távoli VR eszközökhöz készült szoftver, ami segítségével SteamVR játékokat lehet játszani. A program támogatja a Gear VR / Oculus Go / Oculus Quest Wi-Fin keresztüli kommunikációját a számítógéppel [8].

Az ALVR használatához kelleni fog a szerver és kliens oldali alkalmazása is. A szerver használatához egyszerűen le kell tölteni és a SteamVR telepítése után a számítógépre el is lehet indítani a szervert. A kliens telepítése esetén viszont egy kicsit bonyolultabb a helyzet. Ebben a dolgozatban én Oculus Go VR headset-et használtam, így az erre való telepítési módszerét fogom részletesebben ismertetni.

Ahhoz, hogy az Oculus Go-ra lehessen külső applikációt telepíteni, rendelkezni kell egy olyan Oculus-os fiókkal, aminek van egy szervezete (organization). Ezt meg tudjuk tenni ezen a linken. Ha létrehoztunk egy szervezetet és bejelentkeztünk az Oculus-os alkalmazásba a telefonodon akkor a Device menüpont alatt a headset beállításainál meg fog jelenni a Developer Mode és ezt engedélyezni kell. Miután a headset developer módban van, akkor csatlakoztassuk, az eszközt USB kábelen keresztül a számítógépre. Ahhoz, hogy telepíteni tudjunk külső alkalmazásokat le kell tölteni a SideQuest programot, ami ha látja VR headset akkor képes telepíteni bármilyen appot a szemüvegre. A kliens applikációt az újabb verziójú release-ben meg lehet találni [9].

Egyetlen egy dolog kell még, hogy tudjuk használni Unity-ben is, az Unity Asset Store-ban még le kell tölteni a SteamVR plugint és importálni kell a projektbe. Ha ez megvan, akkor a Windows menüpont alatt megjelenik egy SteamVR Input opció és ott generálni kell egy input fájlt a SteamVR-hoz. Ezt követően pedig már könnyen használható az Oculus Go Unity-ban való tesztelésre.

3 Tervezés

3.1 A játékkoncepció

A játék alapjául egy híres régi mobil játékot vettem, ami nem más, mint a Fruit Ninja (6. ábra). Azért hogy ki tudjam használni a sugárkövetés adta lehetőségeket ezért a játékot egy kicsit megcsavartam és nem csak annyiból áll, hogy az előttünk lévő gyümölcsöket kell szétvagdosni egy szamuráj karddal, hanem a hátunk mögött is jelenhetnek meg gyümölcsök. Azért hogy a játékos észleni tudja a háta mögött lévő gyümölcsöket is, tükröt helyezek el a játékban és ezzel már képes a háta mögötti dolgokat is látni.

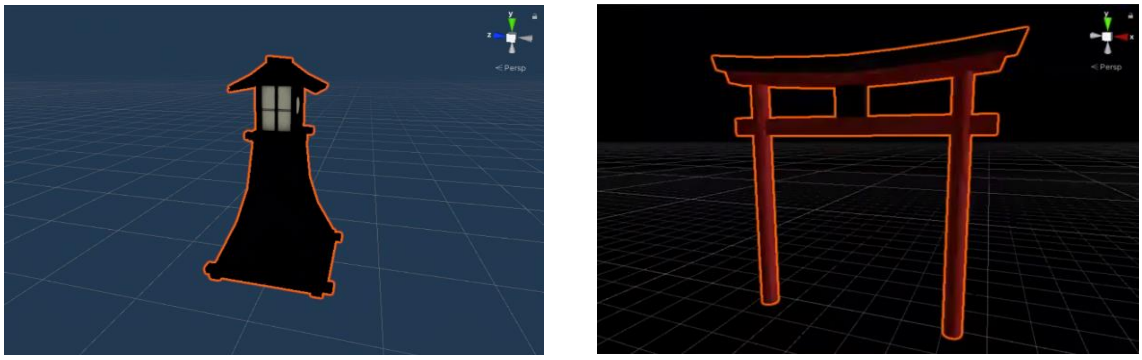
6. ábra: Fruit Ninja játék

3.2 Játékmenet

A játék során a játékos előtt és mögött véletlenszerűen fognak megjeleni gyümölcsök. A gyümölcsök száma, feldobási magasága és az , hogy elöl vagy hátul jelenik meg is teljesen véletlenszerű. A játék célja hogy a játékos minél több pontot tudjon összeszedni. A játék nehézségét lehet azzal állítani, hogy milyen gyorsan jelennek meg a gyümölcsok és a mozgási sebességük változtatásával lehetséges könnyíteni vagy épp bonyolítani a játékon.

3.3 A pálya stílusa és kialakítása

A játék stílusa mivel ninjákról van, szó ezért csak japán lehet. A játékhoz szükséges 3D modelleket a Unity Asset Store-ból szereztem be, mert én nem értek a 3D modellezéshez. Különböző japán stílusú modelleket kerestem mit például torii kapu, lámpások és egyéb különböző dolgok, amelyek Japánra emlékeztetnek és ezzel is próbálva egy kicsit visszaadni a játék hangulatát. Mivel nem túl sok ingyenesen elérhető modellt lehet találni az asset store-ban ezért eléggé nehézkes egy igazán látványos pályát kialakítani. (7. ábra)



7. ábra: A játékban felhasznált modellek

A kialakítása a pályának egy kocka alakú szoba, mivel az Oculus Go nem lehet mozogni ezért a játékos a szoba közepén van elhelyezve. A játékosnak egy kontrollere van ezért csak egy karddal rendelkezik. A gyümölcsök a földről indulnak, úgy ahogy a játékmenetben említettem.

3.4 Gyümölcsök szétvágása

Mint az előző 3D modelleket is a gyümölcsöket is az asset store-ból szereztem be. A játékban 5 darab különböző típusú gyümölcs jelenhet meg, alma, avokádó, banán, citrom és körte. Az hogy milyen gyümölcs fog megjeleni a játékos előtt (vagy mögött) is teljesen véletlenszerűsítve van. Ahhoz, hogy a gyümölcsöket ketté lehessen vágni a karddal egy nyílt forráskódú C# Unity scriptet használok [10]. Ennek köszönhetően már könnyedén szétvághatók lesznek a gyümölcsök.

3.5 A gyümölcsök megjelenése

A játékmenet során már sablonosan elmeséltem, hogy a gyümölcsök véletlenszerűen jelennek meg. Ebben a részben részletesebben ismertetem, hogy hogyan tervezem egy adag gyümölcs létrehozását a játékban. (8. ábra)

8. ábra: 3D gyümölcs modellek

Az elgondolás az, hogy a játékos köré egy nem látható téglatestet helyezünk és csak ennek a területén jelenhetnek meg a gyümölcsök. Mivel nem akarjuk, hogy a játékosba vagy esetleg a játékos jobb vagy bal oldalán jelenjen meg gyümölcs ezért ezeket az eseteket kizárjuk. Amikor a gyümölcs megjelenik, egy véletlenszerű erőnagysággal feldobjuk, és hogy egy kicsit izgalmasabb is legyen meg is csavarjuk. Az, hogy elöl vagy hátul jelenjen meg is véletlenszerű, viszont annak, hogy a játékos mögött jelenik, meg kisebb eséllyel kell megtörténnie mint, hogy előtte, hogy ezzel is elkerüljük a sok forgolódással járó kellemetlenséget.

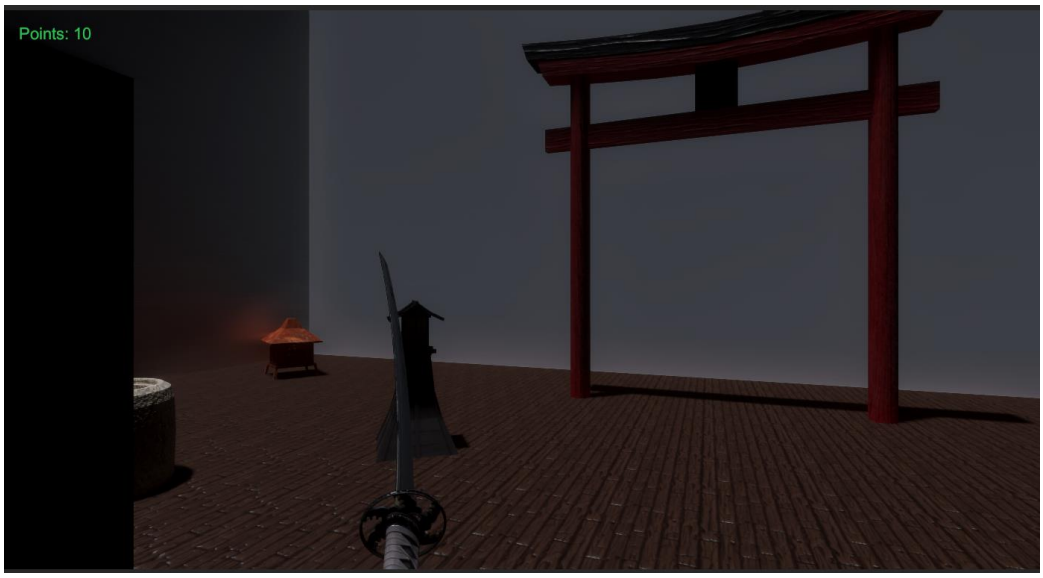
4 Fejlesztés

4.1 Játékos

A játékos a SteamVR által nyújtott prefab, ami tartalmazza a VR szemüveget támogató scriptet és ezen kívül még a kontrollerekhez is ad egy vezérlő scriptet. Mivel az Oculus Go csak egy vezérlővel rendelkezik ezért a játékos egyik kontroller vezérlőjét deaktiválni kellett. Ezeken kívül már csak a magasságot kellet nagyából belőni olyan magasra, hogy a felhasználónak ne okozzon diszkomfortérzést. Ezt követően már lehetett is használni a Unity tesztrendszerén.

4.2 Pontozás

A pontozást úgy oldottam meg, hogy létrehoztam egy SliceListener C# scriptet amiben megírtam a Unity OnTriggerEnter függvényét amelyben figyelem, hogy az eltalált GameObject tag jelzése „Point”. Ha igen akkor egy PointsSystem objektumon meghívjuk a GetPoint() nyilvános függvényét és ez a függvény a játékos által is látható „Points” felirat számát növeli 10 ponttal. Mivel a „Point” taget nem állítom be, a kettévágott gyümölcsökre ezért egy gyümölcs szétvágásáért csak egyszer kaphat pontot a játékos. (9. ábra)



9. ábra: A pontozó rendszer látható jobb felső sarokban

4.3 Ezy-Slice alkalmazása

Ahogy már említettem a 3.4 Gyümölcsök szétvágása résznél a gyümölcsök szétvágásához egy nyílt forráskódú C# scriptet használtam. Ennek használatához készítenem kellett még két scriptet, ami a SliceListener amit a Pontozás említettem és a másik pedig a Slicer lett. A SliceListenerben még annyi kiegészítés történt, hogy az osztály kapott egy Slicer adat mezőt, aminek az is Touched adat mezőjét a fizikai kizáróba történő belépés esetén aktiválódó OnTriggerEnter állítja true-ra ha valamihez hozzáért a Slicer GameObject.

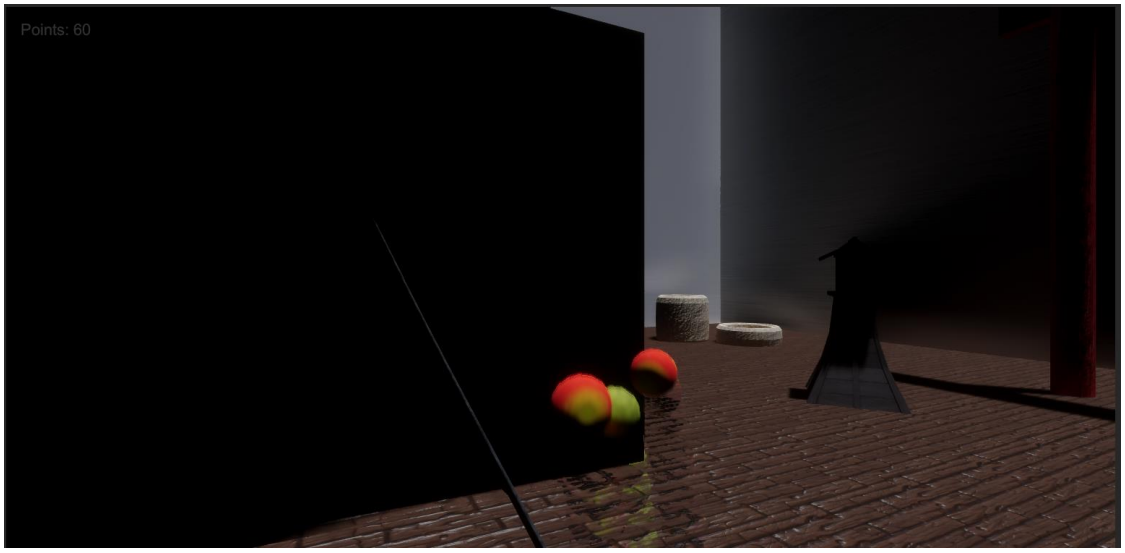
A Slicer osztály az Update függvényében figyeli azt, hogy az isTouched logikai mező változik-e. Ha true lesz, akkor az itt látható kód fut le:

```
if (isTouched == true)
    {
        isTouched = false;

        Collider[] objectsToBeSliced = Physics.OverlapBox(
        transform.position,
        new Vector3(1, 0.1f, 0.1f),
        transform.rotation, sliceMask);

        foreach (Collider objectToBeSliced in objectsToBeSliced)
        {
            SlicedHull slicedObject = SliceObject(
            objectToBeSliced.gameObject, materialAfterSlice);

            GameObject upperHullGameobject = slicedObject
            .CreateUpperHull(objectToBeSliced.gameObject,
            materialAfterSlice);
            GameObject lowerHullGameobject = slicedObject
            .CreateLowerHull(objectToBeSliced.gameObject,
            materialAfterSlice);

            upperHullGameobject.transform.position = objectToBeSliced
            .transform.position;
            lowerHullGameobject.transform.position = objectToBeSliced
            .transform.position;

            upperHullGameobject.transform.localScale = objectToBeSliced
            .transform.localScale;
            lowerHullGameobject.transform.localScale = objectToBeSliced
            .transform.localScale;

            upperHullGameobject.layer = (int) Layer.Sliceable;
            lowerHullGameobject.layer = (int) Layer.Sliceable;

            MakeItPhysical(upperHullGameobject);
            MakeItPhysical(lowerHullGameobject);

            Destroy(objectToBeSliced.gameObject);
        }
    }
```

Ebben a kódrészletben történik meg a gyümölcsok kettévágása. Az elején összeszedjük az összes Collider, amit a Slicer GameObject megérintett és ezen a collider tömbön végig menve az összes objektumot két részre szedi. A felső és alsó részeknek is be kell állítani az eredeti objektum pozícióját és be kell állítani a layerét is, hogy akár többször is szét lehessen vágni egy gyümölcsöt. De ahogy a pontozás résznél említve volt a taget itt nem állítom be mivel nem akarom, hogy többször is pontot kapjon a játékos egy gyümölcsért. A MakeItPhysical() függvénynek pedig csak annyi dolga van, hogy az újonnan létrehozott GameObjectek-re ráhelyezzen egy MeshCollidert és egy Rigidbodyt.

4.4 Gyümölcsdobáló

A gyümölcsdobálásért (10. ábra) a játékban a CreateFruits osztály felel. Ennek az osztálynak a paramétereiben lehet beállítani, hogy milyen nagy területen, milyen magasra, milyen gyorsan dobja fel a gyümölcsöket. Ezeken a tulajdonságokon kívül még van lehetőség itt beállítani, hogy milyen gyors legyen maga a játék.



10. ábra: A gyümölcsdobáló játékban

Megvalósításánál egy négyzet alapú területet vettem ahonnan véletlenszerűen jelenhetnek meg a gyümölcsok. De ahogy írtam már korábba a négyzet egyes területein nem jelenhetnek meg gyümölcsök, ez úgy lett megoldva, hogy amikor a random szám legenerálodik akkor vizsgálom az, hogy hova esne az a pont a négyzeten belül, és ha egy tiltott mezőbe, akkor ehhez a számhoz hozzáadok vagy elveszek attól függően, hogy hol van elöl vagy hátul. Itt látható ez a kód részlet:

```
private IEnumerator SpawnFruit(int frontOrBack)
    {
        isRunning = false;

        float randomX = frontOrBack > 3 ?
                SpawnFrontCoord() : SpawnBackCoord();
        float randomZ = frontOrBack > 3 ?
                SpawnFrontCoord() : SpawnBackCoord();

        if (randomZ < size.x / 4 && randomZ > 0)
            randomZ += size.x / 4;
        if (randomZ > -size.x / 4 && randomZ < 0)
            randomZ -= size.x / 4;

        Vector3 pos = center + new Vector3(randomX, 0, randomZ);

        GameObject newFruit = Instantiate(Fruitsprefab[
          Random.Range(0, Fruitsprefab.Count)], pos, Quaternion.identity);
        ThrowUp(newFruit, randomX);

        yield return new WaitForSeconds(spawnTime);
        isRunning = true;
    }
```

Miután megvan a pont, választunk egy gyümölcsöt a listából, amiben benne vannak a gyümölcsök prefab-jai. Ezt követően meghívjuk a ThrowUp() függvényt, amely majd feldobja a gyümölcsöt a levegőben. Ebben a metódusban két dolgot állítunk be, a GameObjekt Rigidbody-ának a gyorsaságát (velocity) és forgását (rotation).

4.5 HDRP alkalmazása Unityben

Ahhoz hogy a Unityben tudjam használni a sugárkövetést először a projektben a Package Manager-ben hozzá kell adnom a HDRP csomagot mivel nem egy HDRP projekt sablont használtam hanem egy VR-t. Ezt követően létre kellett hozni egy render pipeline asset-et, amit be kellett állítani a projekt beállításain belül a minőség (Quality) menüpontot alatt. Létre kellett hozni egy új szintet, ami majd alkalmazni fogja az újonnan létrehozott pipeline asset-et. Amikor ezzel megvoltam az új render pipeline asset-ben engedélyeznem kellett a valós idejű sugárkövetést. Miután engedélyeztem, a Unity figyelmeztetett, hogy a sugárkövetés nem támogatott az én eszközömön. Ezt a problémát könnyen orvosoltam a HDRP által biztosított varázslóval. Ez a varácsló az összes hibát fixálni tudja. Ilyen volt például hogy a sugárkövetésnek szüksége van a Direct3D 12-re. Amint megvoltam ezzel, a projekt beállításaihoz visszatérve még engedélyeznem kellett néhány dolgot a HDRP asset-ben. Ezek voltak a screen space ambient occlusion, screen space global illumination, screen space reflection és a screen space shadows. Aktiválásukkal már teljes mértékben ki tudtam használni a Unity által adott sugárkövetés lehetőségeit.

4.6 Ray Trace beállítása

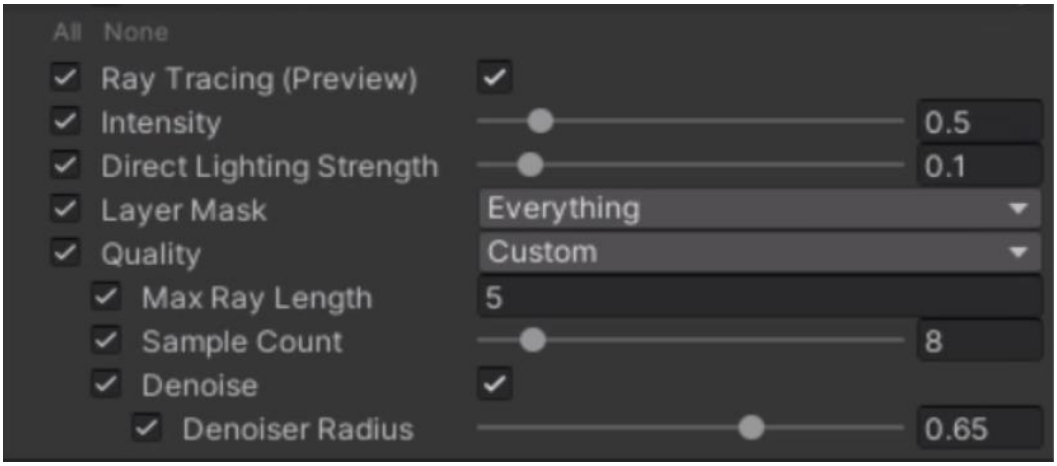
Miután beállítottam a HDRP-t hogy lehessen a sugárkövetést használni a Unity-ben, elkezdtem utána nézni annak, hogy milyen beállításai vannak magának a sugárkövetésnek.

Ahhoz, hogy a jeleneteben tudjuk állítani a sugárkövetés paramétereit ahhoz kellenek Volume Framework-ök. Kellett egy olyan volume ami az egész térre hatással van ez pedig nem más volt, mint a Global Volume. A Global Volume annyit jelent, hogy végtelen visszaverődés és, hogy lefedi az egész világot függetlenül attól, hogy hol van a kamera éppen. Itt tároljuk azokat a beállításokat, amik konstansok. Ilyen például az ég és tone mapping.

A szobának van egy külön dedikált Volume-ja, ami kezeli az exposure-t, white balance-t és a ködöt. Ennek a Volume-nak a módját local-ra állítottam mivel nem szeretem volna, hogy az egész világra hatással legyen. Ezt követően egy új profilt kellett készítenem hozzá, hogy a HDRP beállításait tudjam felülírni.

4.6.1 Ambient Occlusion

Az első felülírás az Ambient Occlusion volt, röviden AO. Ennek lényege, hogy a való életben a lyukak, gyűrődések, metszéspontok vagy egymáshoz közel eső felületek elzárják a szórt fény útját és ezért sötétebnek tűnnek [11].

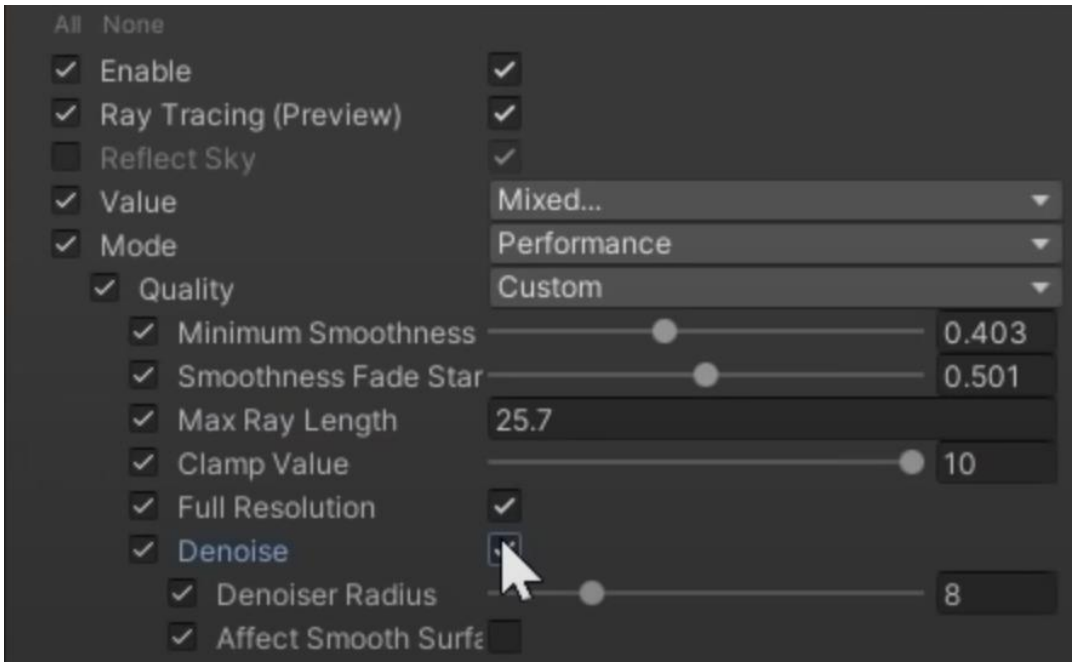


11. ábra: Az AO beállítási paraméterei a Global Volume-ban

A fenti képen láthatók az AO beállításai (11. ábra). Itt az Intensity a sötétség erőségének állítására szolgál. Itt érdemesebb volt minél kisebb értéken tartani, hogy a mesterséges sötétség minél hamarabb elő tudjon állni ezzel is csökkentve a sugárkövetés nagy számítási igényét. A következő érdekesebb paraméter a Layer Mask, amivel azt lehet beállítani, hogy melyik layer tagel ellátott GameObjekt-eken érvényesüljön az AO. Ezt az elemet lehet arra használni, hogy a teljesítményt növeljük, például valamilyen művészi vagy design okok miatt. Ezt követően a Max Ray Length opció volt, amit még megnéztem. Itt minél nagyobbra állítjuk a sugár hosszát annál nagyobb eséllyel fog eltalálni egy felületet. Ezt a beállítást az eredeti értékén hagytam mivel ez tűnt a legoptimálisabbnak. Az utolsó paraméter, amit az AO-ban megnéztem a Sample Count ugyebár ez az a minták száma, amik véletlen keletkeznek képkockánként. Ha megnöveljük a minták számát, akkor a kép tisztasága is javulni fog viszont ezzel nagy terhelést is kap a számítógép. Ezért itt a cél az, hogy minél kisebb értéken tartsuk a minták számát, azért hogy ne legyen, olyan zajos a kép a véletlen minták miatt ezért alkalmazunk zajszűröket is, amik a lehetővé teszi az alacsonyszámú minták mellett a tiszta képet.

4.6.2 Screen Space Reflection

A következő Volume felülírás a Screen Space Reflection volt. Ennek a lényege, hogy a jelentben a képernyő területén lévő adatokat újra felhasználva tükröződést tudjon képezni a játékban. A Screen Space Reflectiont általában a finom tükröződéseknél használják, mint például padlófelületek vagy tócsákban lévő visszatükröződésekhez. Ez a technika igen költséges, de megfelelő beállítások mellett igazán nagyszerű eredményeket érhetünk el.

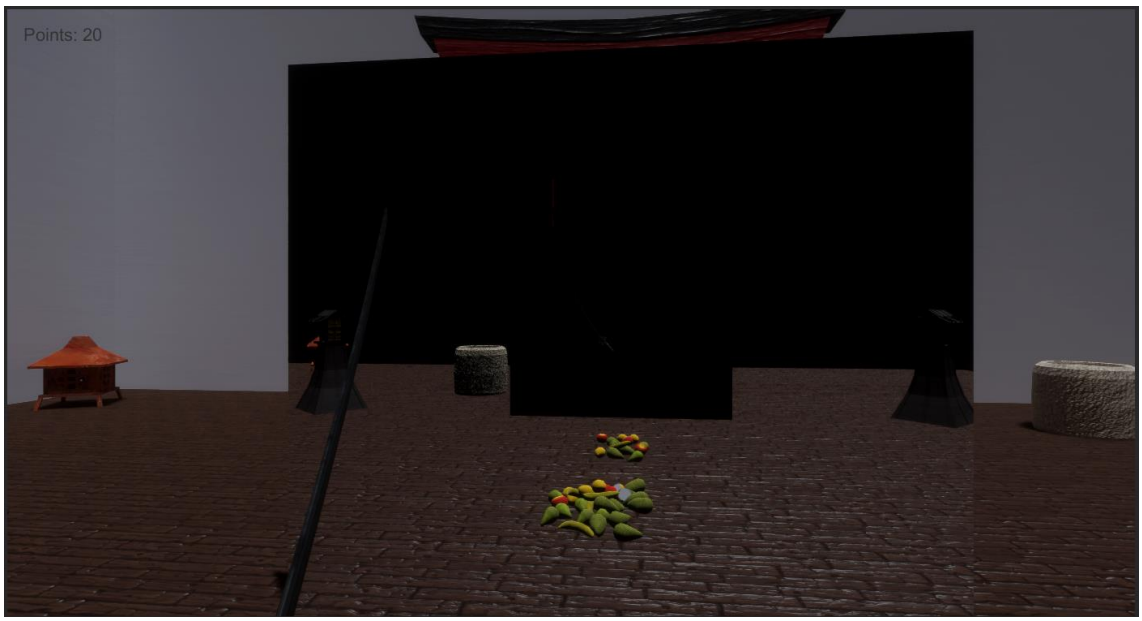


12. ábra: Screen Space Reflection beállításai

Mielőtt elkezdtem volna a beállításokat (12. ábra) a Screen Space Reflection-höz előtte ezt a felülírás a globális és lokális volume-on is be kellett állítani, mivel ha csak a globálison állítottam volna be, akkor a játékban elfordulhatott volna ún. fényszivárgás is, ami igen érdekes eredményeket tud létrehozni. Amikor engedélyeztem és a ray tracinget is bekapcsoltam az első dolog, amit állítottam az a Clamp Value. Ezzel lehet növelni a maximum erősségét sugárnak, ami majd visszatér. Ennek növelésével lehetett a visszaverődés fényeségét növelni. Viszont ennek egy hátránya is van, mégpedig az, hogy a zaj is nőni fog a képen. Ezt követően két fontos paramétert vizsgáltam meg, ami az anyagok simaságához is közük van a Minimum Smoothness és a Smoothness Fade Star. Ezek a tulajdonságok határozzák meg, hogy az anyag milyen sima legyen mielőtt alkalmazni lehet rajta a Screen Space Reflection. Itt mivel valós idejű a játék ezért itt azt a két beállítást annyira magasan kellett tartanom amennyire csak lehetséges volt, hogy a

jelenet minél jobb legyen. Ahogy az AO-nál is itt is megtalálható a Max Ray Length opció ami hasónló képen működik itt is azaz, hogy milyen hosszúak legyenek a sugarak, amik majd ütköznek egy felülettel. Itt, ha túl rövidre állítjuk a sugarakat, akkor fényszivárgás történt.

Amihez igazán használtam ezt a volume felülírást az a tükör volt (13. ábra), ami a játék mechanika egyik fő része. Mivel a performance mód csak egy visszapattantást engedélyez ezért át kellett állnom quality módra. Itt már lehet állítani, hogy hány pattanás legyen a visszaverődésnél. Itt vigyáznom kellett, mert amikor túl magasra állítottam a visszapattantás értéket a Unity használhatatlana vált a nagy számításigény miatt. Mivel az én a játékomban csak két tükröt használok a visszapattantás paraméter értékét kettőre állítottam és ezzel az értékel már olyan eredményeket értem el, ami kielégítő volt a játék számára.



13. ábra: A működő tükrökről a játékban

4.6.3 Screen Space Global Illumination

A Screen Space Global Illumination feladata az lenne, hogy természetes hatású megvilágítás hozzon létre a jelenetben. Ezt úgy próbálja elérni, hogy dinamikus indirekt fényt ad az objektumokhoz a látható képernyőn. Ezen kívül még van egy olyan feladata is, ami az emisszív felületekből, területi fényforrásokból számolt dinamikus megvilágítás. Itt például lehet szó a neonlámpáról vagy valami más világító felületről.

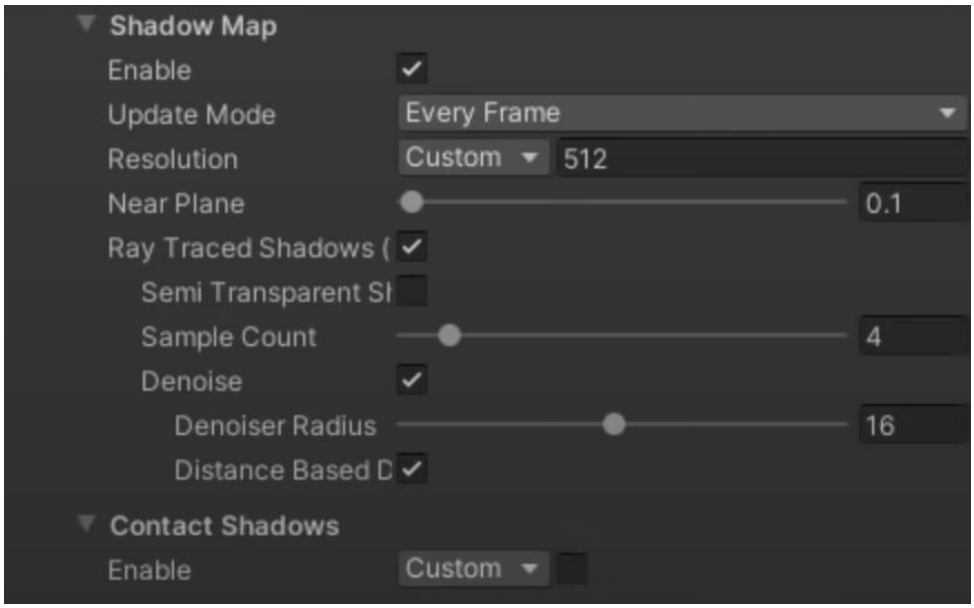
14. ábra: Screen Space Global Illumination beállítások

A sugárkövetés egyik leglátványosabb felülírása szerintem a Screen Space Global Illumination (14. ábra). Itt is elég kicsinek találtam az intenzitást az effektben ezért a Clamp Value opcióval ezt növeltem. Mivel itt is, ha növeljük ezt a paramétert, akkor a zajszint is nőni fog ezért itt alacsonyabban tartottam, mint visszaverődésnél, mert itt sokkal szembe tűnőbb és zavaróbb volt az eredmény, amikor nagyon magas volt. Azért hogy az előző beállítással keletkezet zajt csökkenthessem, a minták számát kellet megemelnem (Sample Count). Ezt követően észleltem, hogy egy kis fényszivárgás is van a jelenetben. Ennek az orvosolásához egy másik felülírást is hozzá kellett adni a volume-hoz. Ennek a neve a Ray Trace Settings és ennek a Direction Shadow Fallback paraméterét nullára kellett állítani és ez teljesen megoldotta a fényszivárgási gondot.

Ahhoz, hogy a minőségen tovább tudjak javítani a Sample Counton kívül volt még egy elérhető paraméter ami a Bounce Count volt. Ha a visszaverődést nagyobbra állítottam sokkal szebb képet kaptam viszont a játék közben sokkal több akadozás is volt, ezért ezt az értéket egyre állítottam, hogy a megfelelő teljesítményt el tudjam érni, ami az élvezetes játék élményhez kell.

4.6.4 Ray Trace Shadows

A Ray Trace Shadows-t az árnyékok számításához használjuk. A hagyományos és a sugárkövetés között csak annyi különbség van, hogy a sugárkövetéses árnyékok sokkal valósághűbbek mit a hagyományos, shadow mapping algoritmussal számított árnyékok és kevesebb erőforrást is igényelnek.



15. ábra: Ray Trace Shadows beállítások

Ez a beállítás (15. ábra) egy kicsit eltér az előzőektől mivel ez nem a volume egyik felülírása, hanem a fény objektumokon a Shadow Map alatt található jelölőnégyzetet bepipálva aktiválódnak. Ez azt is jelenti, hogy minden fényhez ezt a beállítás külön be kell állítani.

A Ray Trace Shadows egyik előnye még, hogy nem számít, milyen távolságban van a fényforrás és a tárgy, a homályossága az árnyéknak nem változik. Itt is lehetséges növelni, hogy az adott fény mekkora területen fejtse ki a hatását. Ezt a beállítást a fényen belül a Shape alatti Radius-szal lehet állítani. Ennél a beállításnál minél nagyobbra állítjuk annál nagyobb lesz a szétszórtsága az árnyéknak hasonlóképpen, mint a való életben.

4.7 DLSS alkalmazása

Unity-be bekerült DLSS technológia az Nvidia által lett kifejlesztve ezért a projektben külön engedélyezni kell a Nvidia-t. Ez úgy csináltam meg, hogy a Package Maneger-ben a Built-in menü fülre kellett állni és ezt követően rá kellett keresni az Nvidia-ra. Miután engedélyeztem és a projekt is betöltötte a szükséges fájlokat minden kamerán engedélyezni lehetet a DLSS funkciót.

Mivel ez én projektemben csak is egy kamera volt ezért nem tartott sokáig beállítani azt. A kamera beállításain belül a Rendering menü pont alatt engedélyezni kellett a Dynamic Resolution paramétert és utána a DLSS is. Ezzel a két engedélyezéssel pedig már a kamerám alkalmazta is a DLSS (. ábra).

4.8 Egyéb, a fejlesztés során keletkezet problémák és annak megoldásai

Ebben a részben szeretnék egy kicsit részletesében írni azokról a hibákról, amik a fejlesztés során keletkeztek. Ezeknek megoldásáról vagy arról, hogy próbáltam meg kijavítani ezeket a hibákat és mi okozhatta a hibát.

4.8.1 Oculus Go a számítógépes erőforrást használja

A legelső akadály, amivel a szakdolgozat projektének készítése során találkoztam az, hogy az Oculus Go egy androidos rendszerrel működik és semmi féle hivatalos módszer nincs arra, hogy a számítógépes VR játékokat játszhassuk. Ezért az interneten kellet keresnem valami megoldásra azért, hogy majd tesztelni tudjam a játékot.

Hamar találtam is egy megoldást, ami az ALVR volt melynek telepítéséről és használatáról már részletesen írtam az 2.3 ALVR ismertetése és használat című fejezetben. Amint megtaláltam ez a nyílt forráskódú programot azt reméltem, hogy ezt követően minden gond nélkül ki tudom próbálni a VR szemüveget egy számítógépes játékon, de sajnos az ALVR kliens oldali applikációját már nem lehetet letölteni az Oculus store-ból. Ezért kellett keresnem még egy programot, amivel lehetséges nem hivatalos forrásokból származó alkalmazást telepíteni az Oculusra. Ez végül a SideQuest volt, amit találtam és ezzel már ezt a problémát is ki tudtam küszöbölni.

4.8.2 A Headset fejmozgás követése

Miután sikerült a VR headset-et csatlakoztatni a számítógéphez elkezdtem dolgozni a Unity-s projektemen. Beállítottam az alap dolgokat a scene-ben mint pedálul a SteamVR által biztosított prefab ami a VR eszközt tudja kezelni. Ezt követően tesztelni szerettem volna a headset, hogy jól működik-e a Unity-n belül és a játékot szépen be is töltötte de amikor elkezdtem mozgatni a fejemet a kamera azt nem követte le.

Itt elég sokáig kellet vizsgálódnom, hogy mi is okozhatja a gondot. Először azt hittem, hogy a SteamVR prefab-ja hibás és ezért megnéztem, hogy a másik prefab amit az újabb VR eszközökhöz készítettek megoldja-e a gondot, de sajnos ez nem volt eredményes. A második gond amire gondoltam az volt, hogy lehet a Unity-ben van valami hiba. Ennél próbálkoztam új projekt létrehozással és újabb verziójú Unity-vel is, de ezek sem oldották meg a problémát.

Az utolsó dolog, ami eszembe jutott még az, hogy az ALVR nem jó. Ez azért volt az utolsó dolog, mert amikor teszteltem a VR-t játékokkal a számítógépemen ilyen fajta gond nem volt. Ezért utána kellet néznem, hogy van-e valami hasonló program mint az ALVR és találtam egy másik ALVR is de ennek az volt a problémája, hogy nem támogatta az Oculus Go kontrollerét. Amikor kipróbáltam a Unity-ben megoldódott a gond és már tudtam forgatni a fejemet a játékon belül. Itt már csak az volt a probléma, hogy nem tudom használni a kontrollert, ami meg a játék egyik elengedhetetlen eleme. Ezért megpróbálkoztam a legelőször megtalált ALVR újabb verziójával és szerencsémre ez is működött és ezzel sikerült elhárítani a tesztelést hátráltató akadályt.

4.8.3 Csak a Directional Light működik sugárkövetésnél

Ezt a hibát akkor észleltem, amikor a sugárkövetést állítottam be a projektbe. Azt vettem észre, hogy amikor létrehoztam a Volume objekteket a pont fényforrásaim nem működtek. Semmilyen fényt nem bocsátottak ki és bármenyire nagy intenzitásúra állítottam a lámpát nem segített.

Itt elkezdtem az interneten keresgélni, hogy mi okozhatja a problémát, de semmi érdemleges információt nem találtam rá. Ezt követően jobban megvizsgáltam a pont fény paramétereit és egy olyan figyelmeztetést találtam, hogy a pont és spot fények nem tudnak ray trace árnyékot vetni. Így, hogy a hibára valami megoldást találhassak ezen a szálon kezdtem tovább nyomozni.

Egy jó pár óra keresést követően, sok oldal átböngészése és különböző videók vizsgálódása után arra jöttem rá, hogy ray trace árnyéknak teljesen jól kellene működnie ezért már csak egy megoldás jutott eszembe az pedig, hogy a HDRP verziójával van valami gond.

Ehhez, hogy a HDRP verziót állítani lehessen újból keresnem kellett mert a Unity által biztosított Package Manager nem biztosit olyan lehetőséget, amivel lehetséges lenne régebbi vagy alfa verziójú package-et használni. A régebbi verzióra való állításra csak abban az esetben van lehetőség ha régebbi Unity verziót használunk, de mivel a Real Time Ray Tracing még csak most került be a Unity játékmotorba ezért ez a lehetőség nem volt opció. Az újabb verzióra való frissítésre csak egy lehetséges megoldást találtam ami pedig, hogy a Unity GitHub oldalán lévő HDRP használom ami viszont csak az alpha kiadású Unity verzióval működik.

Hogy használni tudjam a legújabb verziójú HDRP-t le kellett töltenem az alpha Unity-t és a projektemet is frissítenem kellett erre a verzióra. Ahhoz, hogy a projekt ezt az új HDRP használja a projekt mappájában lévő manifest fájlban át kellett írnom a HDRP elérési útvonalat, hogy a gépen lévő package-et használja. Ezekkel elkészülve megpróbáltam elindítani a projektet de a Unity azonnal összeomlott. Megpróbáltam többször is elindítani, de nem sikerült. Ezt követően megpróbáltam a régi HDRP-vel is, de azzal sem volt eredményes az indítás.

Így hát erre a problémára végül nem sikerült semmilyen jó megoldást találni. Szóval ennek a hibbannak a javítását csak úgy lehet megoldani ha a Unity ad ki egy hivatalos frissítést a HDRP-hez.

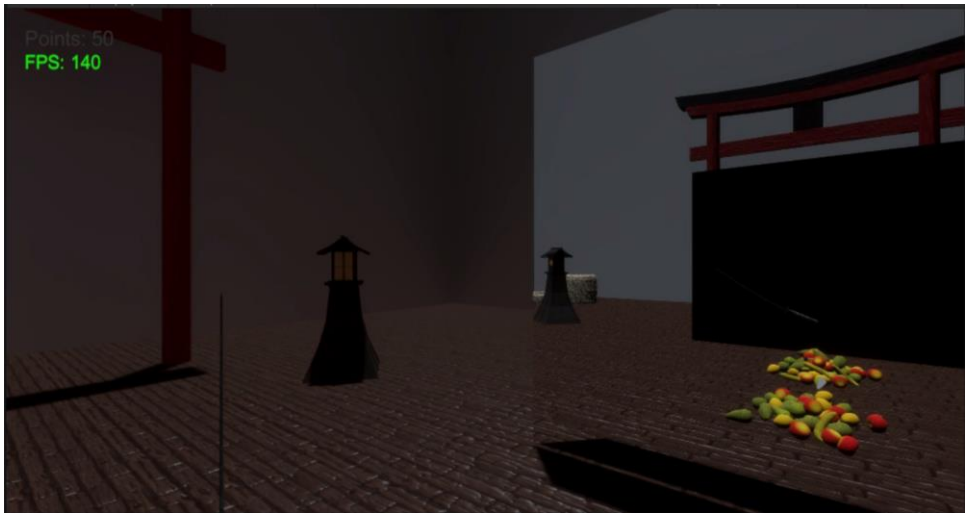
4.8.4 DLSS nem látható a Unityben

Ez a legújabb technológiák közé tartozik, ami a Unity-be bekerült így még valószínűleg nem a legstabilabb eszköz. Mikor be szerettem volna állítani, hogy a kamera ezt a képalkotási technikát alkalmazza nagy meglepetésemre nem találtam azt az opciót a Render menüpontja alatt. Elkezdtem próbálkozni azzal, hogy új kamerát hozok létre és abban állítom majd be, de ezekben az új kamerákban sem lehetett megtalálni a Nvidia által készített DLSS.

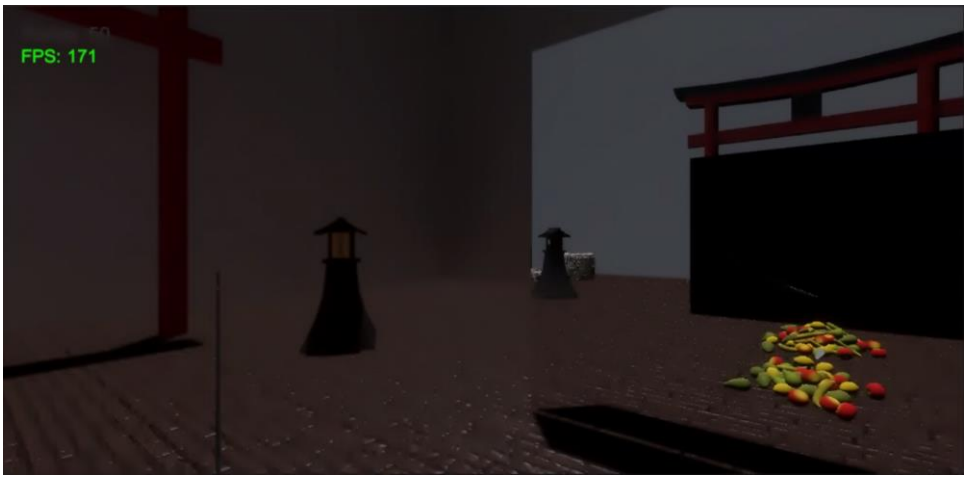
Ezután az interneten kezdem el keresgélni, hogy mi okozhatja ezt a gondot és a hivatalos Unity fórumon találtam is egy ilyen kérdésről szóló postot. Ahogy olvasgattam a visszajelzéseket egy olyan megoldás találtam, ami a HDRP verzióját lecseréli az alpha kiadásra. Ez sajnos nem igazán jó megoldás számomra mivel az előző fejezetben is ezzel próbálkoztam és az alpha nem volt egyáltalán olyan állapotban, amiben én ezt tudtam volna használni.

Mivel a fórumokon nem találtam más megoldást ennek a hibának az orvosolásához ezért utána néztem, hogy melyik Unity-s verzióban érkezett be a DLSS és láttam, hogy a 2021.2 verzióban érkezett be és pont volt egy elérhető frissítés ehhez a verzióhoz. Miután letöltöttem és a projektemet is felfrissítettem elindítva a projektet sajnos arra kellett rádöbbennem, hogy még mindig nem látszik a DLSS eszköz a kamerákon.

Utolsó reményemben a DLSS hivatalos dokumentációjához nyúltam hátha találok ott valami érdemleges információt. Miközben olvasgattam láttam egy olyan bekezdést, hogy engedélyezni kell a Nvidia-t a projekten belül. A dokumentáció instrukcióját követve a Package Managerben a Build-in opciót kellett kiválasztani és ott megjelent az Nvidia engedélyezése. Miután engedélyeztem és a projekt betöltött láthatóvá vált a DLSS opció a kamerán.



16. ábra: A játék FPS száma DLSS nélkül



17. ábra: A játék FPS száma DLSS bekapcsolása után

Ahogy a 16. és 17. ábra is demonstrálja a DLSS használatával 30 kép per másodperces (FPS) teljesítményjavulást is elérhetünk, ami igen nagy szamnak minősül a játékoknál.

4.9 VR és Ray Tracing kompatibilitási hiba Unity-ben

A legnagyobb hiba, amivel a projekt készítésekor szembesültem az volt, amikor tesztelni akartam a sugárkövetés beállításai után a VR headset-tel a játékot és valami oknál fogva nem tudott elindulni a VR eszközön a játék viszont a számítógépen futott hibamentesen. Ennek a nyomozására szerencsére túl sok időt nem kellet szánni mert a Unity egy értesítést adott arról, hogy az XR csak a Direct3D 11-gyel működik ezért a projekt beállításaiban átállítottam, hogy ezt a verziót használja. Miután újra indult a Unity és már a Direct3D 11 használta el is indult a játék viszont a Unity Ray Trace-hez Direct3D 12 kell ami így lehetetlené tette a két technológia összehozását Unity játékfejlesztő rendszeren belül.

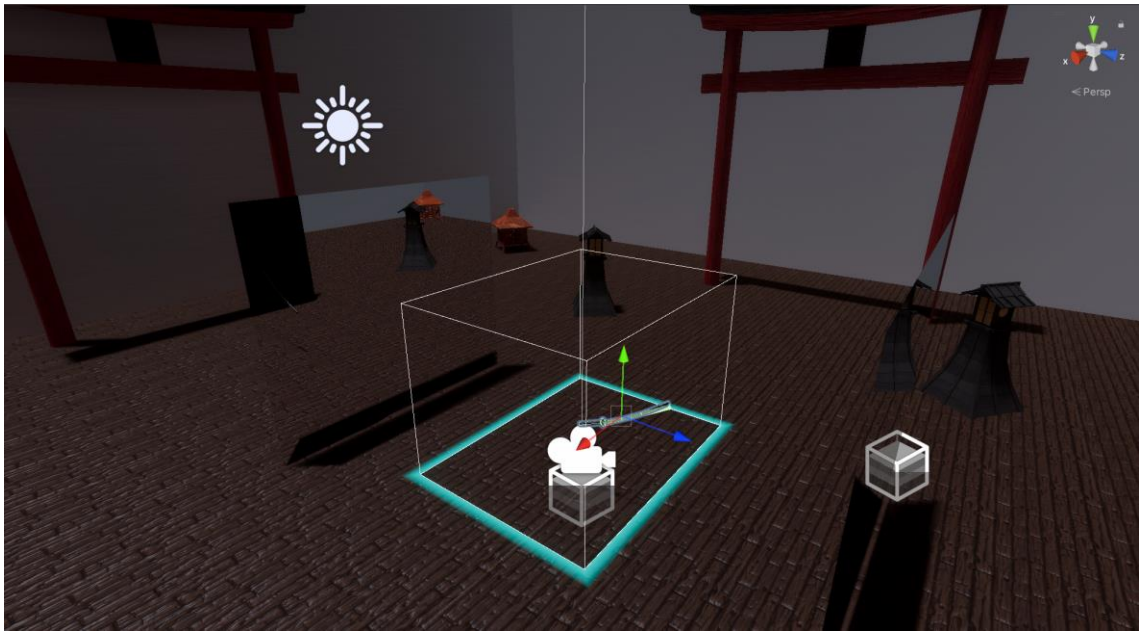
Ennek a hiba megoldására sajnos semmilyen behatással nem tudtam javítani és valószínű, hogy ez a probléma csak akkor fogjuk tudni megoldani amikor a Unity-ben XR már támogatja a Direct3D 12 is. Ezt viszont már csak a közeljövőben kijövő újabb verziójú Unity-k fogják tudni.

5 Eredmények, tesztek, és értékelésük

A fejlesztést két nagyobb szakaszra bontottam. Az első részben a VR headset-en egy játszható játékot készíteni, a második nagy rész pedig a ray trace alkalmazás a játékon belül.

Az első rész készítésénél nem okozott gondot, hogy az itthoni gépen fejlesszem a játékot mivel az egyetem kölcsönadta nekem a VR eszközt. A tesztelés során nem csak én, de megkértem ismerőseimet is, hogy teszteljék a játékot és mondják el véleményüket, hogy milyen javítási lehetőségeket vagy problémákat érzékeltek, amikor a játékkal játszottak. Ilyen javaslatok voltak pedálul, hogy a gyümölcsök feldobásánál, a gyümölcsök száma legyen véletlenszerű vagy az, hogy egy kicsit megcsavarva dobjuk fel a gyümölcsöket és itt volt még egy olyan észrevétel is, ami szerint a játék túl nehéz mivel túl gyorsan mentek a gyümölcsök. Ezeket a javaslatokat beleépítve a játékomba újabb teszteket futtattam és ezt addig ismételgetem amig egy elfogadható hiba értékig el nem értem.

A második nagy szakasz a sugárkövetés bekötése volt a játékba. Itt már kicsit problémás volt a munka mivel itthon nem rendelkezem olyan videokártyával, ami támogatja az RTX-et. Ezért, hogy ezt tesztelni tudjam az egyetem egyik számítógépét kellett használatba vennem. Mivel én nem Budapesten élek, hanem Debrecenben azért, hogy tudjam használni a gépet több alkalommal mint, hogy Pesten fent vagyok távoli asztali kapcsolattal dolgoztam amikor itthon voltam. Ennek egyetlen egy hátránya volt még pedig az, hogy nem tudtam tesztelni a VR szemüveget akkor, amikor a sugárkövetes bekerült és ezért végeredményül az, hogy a VR nem tudja használni a sugárkövetést Unity környezetben csak a legvégén derült ki. Ettől eltekintve sok mindent ki tudtam próbálni a Unity sugárkövetésében és sok új tapasztalttal gazdagodtam. Ennek a résznek a tesztelése csak a Unity teszt felületén volt lehetőségem viszont ez is bőven jó volt, hogy a hibáka észlelni tudjam és javítani tudjam őket. (18. ábra)



18. ábra: Kép a játék végső állapotáról

5.1 Továbbfejlesztési lehetőségek

A játék a jelenlegi állapotában egy végtelen játék, azaz addig lehet játszani míg a játékos, megunja. Viszont egy játék nem annyira izgalmas, ha nincsenek célok vagy valami jutalom azért, hogy játszol. Ezek elengedhetetlen elemei egy játék számára mivel itt az a cél, hogy a játékos ne csak egyszer játszon vele, hanem többször is vegye elő és szórakozzon vele. Ezért maga a játék még rengeteg lehetőséget tár elénk a fejlesztés részén, de ezen kívül még a megjelenítésen is dolgozni lehet. Itt ebben a fejezetben pár gondolatot osztok, meg mik lehetnének ezek a fejlesztések.

- Ahogy az előbb is említettem a játékba lehetséges lenne különböző kihívásokat tenni, amiért a játékos virtuális trófeákat szerezhetne. Ilyen kihívás lehet például, hogy a három vagy több gyümölcsöt is szét tud vágni egyetlen suhintással vagy esetleg, hogy egy adott időn belül nem esik le egyetlen gyümölcs sem érintetlenül.
- A játék mechanikát lehetne azzal bővíteni, hogy más tárgyakat is feldobunk, ami nem gyümölcs és ennek szétvágása büntetéssel járna. Itt például olyanra gondolok, mint, hogy az eredeti telefonos játékban, amit alapul vettem voltak bombák és azok szétvágása életvesztéssel járt. Ehhez hasonló elemek kerülhetnének még be a játékba azzal a céllal, hogy még izgalmasabb legyen.

- A megjelenítés terén olyan továbbfejlesztési lehetőséget látok, hogy sokkal szebb és jobb modelleket helyezünk el a játékban annak érdekében, hogy a sugárkövetés adta lehetőségeket még jobban ki tudjuk használni. Ezzel elérhetjük azt, hogy a játékos még jobban bele tudja élni magát a játékba és még mélyebb nyomot hagyjon benne azért, hogy még egyszer akarjon játszani a játékkal.

A fentebbi lehetőségek csak egy pár példát adtak arra, hogy esetleg milyen fejlesztési lehetőségek vannak még ebben a projektben.

5.2 Értékelés

Habár a fő célt nem tudtam elérni, ami pedig az volt, hogy egy olyan VR játékot készíteni ami ray trace is alkalmaz de a szakdolgozat projekt készítése során volt lehetőségem még jobban elmélyedni a játékfejlesztés és a Unity fejlesztő környezetében elmélyülni. Rengeteg újdonságot és tapasztalatot szereztem, amit akár a munka világában is jól lehet hasznosítani.

Az, hogy most ez a projekt nem volt teljesen sikeres az nem azt jelenti, hogy a jövőben nem lehet majd használni a VR-t és a sugárkövetést egyszerre a Unity-n belül. Nagy valószínűséggel már zajlanak ezek a fejlesztések a Unity-nél hiszen a jövőben egyre több játék fog készülni a VR szemüvegekre és ezeknek a következő generációjára is és elengedhetetlen eszköz lesz a sugárkövetés ahhoz, hogy egy valósághű és ámulatba ejtő virtuális világot tudjunk készíteni.

6 Összefoglaló

A munkám során rengeteg akadállyal és problémákkal kellett szembe néznem. Ezen hibák többségét sikerült megoldanom hosszas keresgetés és próbálkozások segítségével. Nagyon sajnálom, hogy a végén a projekt lényegi része nem jött össze a Unity korlátai miatt, de ettől függetlenül rengeteg új technologiát és eszközt ki tudtam próbálni a projekt keretein belül. Ez a projekt rengeteg potenciált tartalmaz a továbbfejlesztés szempontjából, amiről egy kicsit részletesebben is fogok írni a 5.1 Továbbfejlesztési lehetőségek fejezetben. Egyik fő célom ezzel a szakdolgozattal az volt, hogy kipróbálhassam a Unity által újonnan adott Ray Trace eszközét. A VR headset-en nem tudtam meg nézni végül az eredményt de a Unity teszt felületén is igazán élvezetes volt gyönyörködni a sugárkövetéssel létrehozott játékot.

Irodalomjegyzék

[1] Virtuális valóság: Wikipedia, https://hu.wikipedia.org/wiki/Virtuális_valóság (letöltve: 2021. okt. 12.)

[2] Unity Ray Trace: https://unity.com/ray-tracing (letöltve: 2021. okt. 13.)

[3] László, S. K., György, A., & Ferenc, C. (2006). Háromdimenziós grafika, animáció és játékfejlesztés (Vol. 68). ComputerBooks: https://cg.iit.bme.hu/~szirmay/3Dgraf.pdf (letöltve: 2021. okt. 13.)

[4] NVIDIA DLSS hivatalos oldala: https://developer.nvidia.com/blog/nvidia-dlss-natively-supported-in-unity-2021-2/ (letöltve: 2021. nov. 02.)

[5] Xiao, L., Nouri, S., Chapman, M., Fix, A., Lanman, D., & Kaplanyan, A. (2020). Neural supersampling for real-time rendering. *ACM Transactions on Graphics (TOG)*, *39*(4), 142-1: https:/dl.acm.org/doi/abs/10.1145/3386569.3392376 (letöltve: 2021. nov. 02.)

[6] Edelsten, A., Jukarainen, P., & Patney, A. (2019). Truly next-gen: Adding deep learning to games and graphics. In In NVIDIA Sponsored Sessions (Game Developers Conference): https://www.gdcvault.com/play/1026184/Truly-Next-Gen-Adding-Deep (letöltve: 2021. nov. 02.)

[7] Foveated Real-Time Ray Tracing for Virtual Reality Headset: http://research.lighttransport.com/foveated-real-time-ray-tracing-for-virtual-reality-headset/asset/abstract.pdf (letöltve: 2021. nov. 05.)

[8] ALVR GitHub repo: https://github.com/polygraphene/ALVR (letöltve: 2021. nov. 09.)

[9] ALVR GitHub releases oldala: https://github.com/polygraphene/ALVR/releases (letöltve: 2021. nov. 09.)

[10] EzySlice GitHub repo: https://github.com/DavidArayan/ezy-slice (letöltve: 2021. nov. 09.)

[11] Unity HDRP hivatalos dokumentációja: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-

definition@10.6/manual/Override-Ambient-Occlusion.html (letöltve: 2021. nov. 21.)