

Pannon Egyetem
Műszaki Informatikai Kar
Villamosmérnöki és Információs Rendszerek Tanszék
Programtervező Informatikus alapszak

SZAKDOLGOZAT

VR játékfejlesztés rehabilitációs célra

Kertész Domonkos

Témavezető: Dr. Guzsvinecz Tibor

2023



PANNON EGYETEM

MŰSZAKI INFORMATIKAI KAR

Programtervező informatikus BSc szak

Veszprém, 2023. március 29.

SZAKDOLGOZAT TÉMAKIÍRÁS

Kertész Domonkos

Programtervező informatikus BSc szakos hallgató részére

VR játékfejlesztés rehabilitációs célokra

Témavezető: Dr. Guzsvinecz Tibor, adjunktus

A feladat leírása:


Napjainkban a virtuális valóság típusú alkalmazások iránti érdeklődés növekedése töretlen. Nem csak a szórakoztatóiparban, hanem például orvosi területen is használják mozgás-, valamint kognitív rehabilitációs célokra, fejlesztő játék (serious game) megnevezéssel. Az említett fejlesztő játékokban gyakran használnak játékosítással (gamification) kapcsolatos elemeket, mivel az eredményesebb rehabilitáció érdekében célszerű fenntartani a páciensek motiváltságát, érdeklődését.

A feladat célja egy olyan gamification elemeket tartalmazó, multiplatform alkalmazás implementálása, amely gyógytornászok és páciensek által is használható!

Feladatkiírás:

- Dolgozza fel a témával kapcsolatos eddigi hazai és külföldi irodalmat!
- Határozza meg a szoftverrel szemben támasztott követelményeket!
- Tervezze meg és implementálja a rehabilitációs célú alkalmazást, amely figyelembe veszi a felhasználó fizikai képességeit!
- Készítsen az alkalmazáshoz felhasználói kézikönyvet!


Dr. Süle Zoltán
egyetemi docens
szakfelelős


Dr. Guzsvinecz Tibor
adjunktus
témavezető

Hallgatói nyilatkozat

Alulírott Kertész Domonkos hallgató kijelentem, hogy a dolgozatot a Pannon Egyetem Villamosmérnöki és Információs Rendszerek tanszékén készítettem a Programtervező Informatikus alap végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Dátum: Veszprém, 2023. 05. 04.



.....
Kertész Domonkos

Témavezetői nyilatkozat

Alulírott Dr. Guzsvinecz Tibor témavezető kijelentem, hogy a dolgozatot *Kertész Domonkos* a Pannon Egyetem Villamosmérnöki és Információs Rendszerek tanszékén készítette Programtervező Informatikus alap végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védelemre bocsátását engedélyezem.

Dátum: Veszprém, 2023. 05. 04.



.....

Dr. Guzsvinecz Tibor

Tartalmi összefoglaló

Az akut vagy krónikus nyaki fájdalom gyakori egészségügyi probléma, aminek a kezelése jellemzően napi fizioterápiás gyakorlatok. Olyan személyeknél, akik kevésbé motiváltak az edzés program betartására, a gyógyulás késleltetett. Szakdolgozatomban olyan virtuális valóság megoldást dolgozok ki, és fejlesztek le, ami motiválja a nyaktornára a felhasználót.

Az alkalmazás a felhasználó fizikai képességeit figyelembe véve, végigvezeti nyak tornáztató gyakorlaton, illetve lehetőséget nyújt gyógytornászoknak, hogy gyakorlatokat hozzanak létre betegeiknek, otthoni felhasználásra.

Kulcsszavak: Gamification, Virtuális valóság, Rehabilitációs szoftver, Nyak, Torna

Abstract

Acute or chronic neck pain is a common health problem that is typically treated with daily physiotherapy exercises. In individuals who are less motivated to adhere to an exercise programme, recovery is delayed. In my thesis, I design and develop a virtual reality solution that motivates the user to perform neck exercises.

The application guides the user through neck exercises based on their physical abilities, and gives physiotherapists the opportunity to create exercises for their patients to use at home.

Keywords: Gamification, Virtual reality, Rehabilitation software, Neck, Exercise

Tartalomjegyzék

Tartalmi összefoglaló	4
Abstract.....	5
Jelölésjegyzék	8
1. Motiváció	9
2. Irodalom és versenytárs elemzés.....	10
2.1. A bővített valóság (XR).....	10
2.1.1. A virtuális valóság (VR) és VR eszközök	11
2.2. XR az egészségügyben	13
2.2.1. XRHealth.....	16
3. Funkcionális követelmények	17
4. Felhasznált technológiák.....	18
4.1. Hardver	18
4.2. Unity játékmotor.....	20
4.3. Verziókezelés.....	21
4.4. Integrált fejlesztőkörnyezet	23
5. Implementáció	24
5.1. Projekt létrehozás és beállítás	24
5.2. Jelenet és játék objektumok	26
5.2.1. Objektum hierarchia	27
5.2.2. Player objektum.....	31
5.2.3. Menü, menü elemek, és térbeli objektumok.....	31
5.2.4. Gyakorlat választó menü	33
5.3. Ray casting.....	33
5.4. Gyakorlatok tárolása.....	36
5.5. A játéktér kiszámolása és objektum pozíció újraszámolás.....	40
5.6. Szkriptek	44
5.6.1. Szkript életciklusa	44
5.6.2. MainController osztály	45
5.6.3. CamControlPC és PlayerCamRotation osztályok	46

5.6.4.	GameManager osztály	46
5.6.5.	UIManager, DebugManager, BorderHelper osztályok.....	47
5.6.6.	ExerciseDictionary és FileHandler osztályok.....	47
5.6.7.	ObjectCoordinates osztály	48
5.6.8.	RayCast osztály	48
5.6.9.	ExerciseChooser osztály	49
5.6.10.	EditMode osztály	49
5.6.11.	NormalGame és Border osztályok.....	50
6.	Felhasználói instrukciók	51
7.	Lehetséges továbbfejlesztés	51
8.	Összefoglalás	52
	Irodalomjegyzék.....	53
	Mellékletek	57
	Ábrajegyzék.....	59

Jelölésjegyzék

VR: Virtual Reality (Virtuális valóság)

AR: Augmented Reality (Kiterjesztett valóság)

MR: Mixed Reality (Vegyes valóság)

XR: Extended Reality (Bővített valóság)

UI: User Interface (Felhasználói Interfész)

LINQ: Language Integrated Query (Nyelvbe ágyazott lekérdezés)

1. Motiváció

Szakedolgozatom célja egy olyan szoftver fejlesztése virtuális valóság szemüvegre, melyet fel lehet használni nyak rehabilitációs gyakorlatokra, vagy általános nyakmozgatásra. A szoftver egyszerű nyakmozgatási utasítások sorozatát mutatja a felhasználónak, ezzel érdekesebb alternatívát nyújt a mozgáshoz szokványos tornagyakorlatokhoz képest.

A virtuális környezet motiválhat olyan személyeket is a mozgásra, akik kevesebbet mozognak, szívesebben töltik az idejüket online, vagy más okokból nem akarnak vagy nem tudnak mozogni, illetve elterelheti a figyelmüket a fájdalomról olyan személyeknek, akik sérülés vagy fájdalom miatt nehezebben tudnak mozogni. [1]

Informatikusként sok időt töltök számítógép előtt, ami gyakran nyakfájdalomhoz vezet. Ez a probléma fennáll sok számítógépes munkát végző személynél, így az elsődleges célközönség a mozgáshiány miatt nyakfájdalomtól szenvedő személyek. Másodlagos célközönségként a nyaktraumát szenvedett személyeket célozom meg. A célközönséget figyelembe véve a programom tartalmaz általános használatra tornagyakorlatokat, melyeket a felhasználók a mindennapos nyakmozgatásra használhatnak, illetve lehetőséget nyújt saját tornák létrehozásához, ezzel gyógytornászok otthoni tornagyakorlatokat tudnak készíteni betegeiknek, illetve felhasználók egymás között megoszthatják gyakorlataikat.

A szakdolgozatom alatt fejlesztett szoftvert szeretném később komolyabb célokra is felhasználni, például TDK, vagy akár kutatási célra, mivel világszerte az emberek 2,6%-a szenved nyakfájdalomtól élete során [2], így fontosnak tartom ennek a témának az alapos feltárását.

2. Irodalom és versenytárs elemzés

2.1. A bővített valóság (XR)

Habár mára egyre ismertebbek a különféle bővített valóság, vagy XR technológiák, mégis léteznek olyan esetek, amelyekben nehéz pontosan meghatározni, hogy egyes alkalmazások, vagy felhasználások az XR melyik alcsoportjába tartozik. A három csoport a virtuális valóság (VR), vegyes valóság (MR), illetve kiterjesztett valóság (AR).

Az AR technológiák olyan megoldásokat tartalmaznak, amik digitális objektumokat helyeznek el a valóságban. AR applikációk általában okoseszközökkel használhatóak, okostelefon, tablet, vagy viselhető eszközök. Erre jól ismert példa a Pokémon GO játék, ami az 1. ábrán látható. A játékos az okostelefonja segítségével tud interakcióba lépni a játékkal, GPS segítségével ahogy mozog a valóságban. Úgy mozog a játékban is, illetve az okostelefon kameráján keresztül kap betekintést a játék világába. A játékhoz opcionálisan használhatóak különféle viselhető eszközök és kiegészítők. Például a Pokémon GO Plus karkötő, amivel a játékos a telefonja elővétele nélkül képes interakcióba lépni a virtuális világgal, vagy a Poké Ball Plus, a Plus karkötő továbbfejlesztett verziója, ami több lehetőséget biztosít a játékosnak a karkötőhöz képest [3] [4].



1. ábra: PokémonGO AR játék

Az MR technológiákat használó megoldások keresztezik a VR és AR technológiákat, szenzorokkal és kamerákkal érzékelik a valóságot, és virtuális objektumokat helyeznek el rajta, ezzel összemosva a valóságot és virtuális világot. MR applikációk külön erre a felhasználásra fejlesztett eszközöket igényelnek, például Microsoft HoloLens, a 2. ábrán látható, illetve speciálisan a felhasználásra kialakított teret. Az MR eltér a VR és AR technológiáktól abban, hogy szórakoztató megoldások helyett inkább ipari, oktatási és gyógyászati felhasználású [5].



2. ábra: Microsoft HoloLens használata műtéthez való felkészüléshez [6]

2.1.1. A virtuális valóság (VR) és VR eszközök

A virtuális valóság, vagy VR, egy 3 dimenziós tér számítógép által generált szimulációja, amivel felhasználók kimondottan erre a felhasználásra készített eszközökkel tudnak interakcióba lépni.

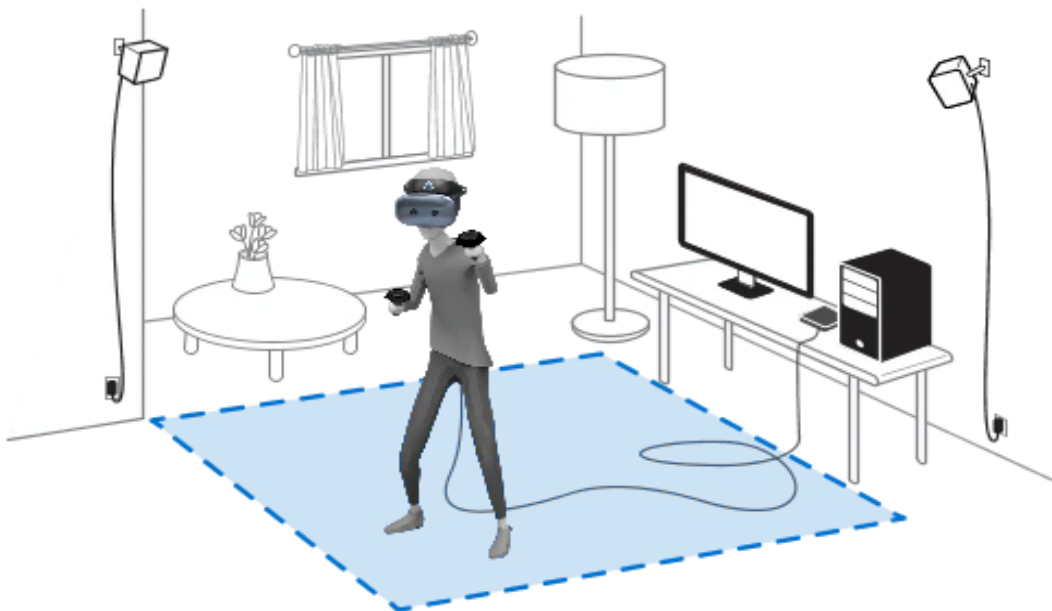
A VR koncepciója az 1950-es években jelent meg. Morton Heilig elkészítette az úgynevezett Sensorama gépét, amit az első VR gépnek tekintenek. Ez egy nagy fülke volt, amiben más-más technológiák stimulálták az emberek érzékeit: színes videók, illatok, rezgések és hangok formájában [7] [8].

A modern VR alkalmazásokhoz mindenképpen szükséges egy VR szemüveg (VR headset), amin keresztül a felhasználó belelát a virtuális térbe, ezen kívül opcionálisan kézi kontroller, és szintén opcionálisan bázis állomás. A VR szemüveg tartalmaz egy magas felbontású kijelzőt, lencsét, és szenzorokat, amikkel követni tudjuk a felhasználó fej és testmozgását. A VR technológiák alapja a sztereoszkópikus 3

dimenziós hatás, amit úgy érnek el a VR szemüvegek, hogy a kijelzőt két részre bontják, szemenként egy, és egymáshoz képest eltolt képet mutatnak, a lencsék segítenek a fókuszálásban, és csökkentik a szem terhelését. A két képet az agy összemosza, így teremtve mélység- és térérzetet.

A kontrollerekkel képes a felhasználó interakcióba lépni a virtuális környezettel. A legtöbb VR szemüveghez tartozik controller, de léteznek olyanok, amik controller nélkül is használhatóak, például Samsung GearVR, ami biztosít egy érintőpadot a szemüveg oldalán, ami érzékel több irányba húzást, illetve kattintást. A VR kontrollerek a felhasználó kezei, néhány controller esetében az ujjai, mozgását érzékelik szenzorokkal, illetve rendelkeznek gombokkal, és ravasszal, amikkel a felhasználó képes a virtuális térrel interakcióba lépni, megfogni, magához húzni, vagy magától eltolni objektumokat. Néhány controller rendelkezik haptikus jelzéssel, ami a tapintás érzetét kelti.

A VR bázis állomások feladata a VR szemüveg, és VR kontrollerek követése a térben. A bázis állomások infravörös vagy lézer jellel követik a VR eszközöket, ezzel is pontosítva a helyzetüket és elhelyezkedésüket a térben. A bázis állomásokat általában magasan kell elhelyezni, ahogy látható a 3. ábrán, hogy jól rálássanak a VR eszközökre, mivel a legkisebb vakfoltok is teljesítményvesztéshez vezetnek.



3. ábra: Bázis állomások elhelyezése [36]

Többféle VR szemüveg létezik, eltérő felszereltséggel, és eltérő technológiákkal, ezek függenek a szemüveg gyártójától, illetve a felhasználási céljától. A valóságghű videójátékokhoz készített VR szemüvegek kontrollerekkel, és akár több bázis állomással

rendelkeznek, valamint folyamatos összeköttetést igényelnek egy számítógéppel. Könnyebb alkalmazásokhoz léteznek vezeték nélküli VR szemüvegek, amikhez nincs szükség külső számítógépre, mivel rendelkeznek beépített processzorral, grafikus egységgel, tárhellyel, és akkumulátorral.

Mivel az én programomhoz nincs szükség kontrollerekre, illetve fontos a mobilitás, hogy kábelek ne akadályozzák a felhasználót a mozgásban, ezért elsődlegesen egy Android okostelefonnal működő VR szemüveget, Samsung GearVR-t választottam platformnak.

2.2. XR az egészségügyben

Az egészségügy modernizációjában fontos szerepe van a kiterjesztett valóságon alapuló megoldásoknak. XR segítségével az egészségügyi szolgáltatók jobb kezeléseket tudnak biztosítani betegek számára, illetve jobb felkészülést tudnak biztosítani az egészségügyi dolgozók számára. A technológia segít sebészeti beavatkozásokban, fájdalomkezelésben, fizikai és kognitív rehabilitációban, mentális egészségben stb.

A George Washington Egyetem agy- és mellkassebészeti beavatkozásokhoz használ egy fejlett VR eszközt, aminek használatával a sebészek a beavatkozás előtt képesek megvizsgálni a beteget [9]. Ez javított a sebészeti beavatkozások hatékonyságán, valamint így a betegek és családtagjaik jobban megértik a beavatkozást. Egy Harvard Business Review tanulmány szerint a VR környezetben végzett oktatás, mint ahogy a 4. ábrán látható, 230%-ban javította a résztvevők sebészeti teljesítményét, hagyományos módszerekhez képest [10].



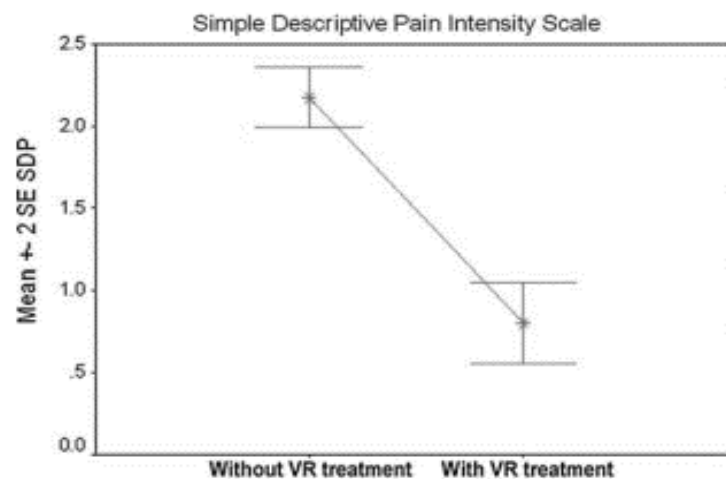
4. ábra: Orvoshallgató VR környezetben gyakorol [36]

A UConn Health PrecisionOS és Oculus VR megoldásokat használ ortopédia képzéshez. Ez a hagyományos holttesteken való gyakorláshoz képest lényeges mennyiségű időt és pénzt takarít meg, mivel így egy beavatkozást többször is el tudnak végezni [11].

Vannak kórházak, amik olyan VR megoldásokat használnak, melyekkel az orvosaik jobban megértik a betegek állapotát, ezzel növelve az empátiát az orvosokban a páciensek irányába. Olyan egészségügyi állapotokat tudnak VR környezetben szimulálni, mint az időskori demencia, Parkinson-kór, migrénes fejfájás [13]. Szociális dolgozóknál növekedett az empátia látás- és halláskárosultakkal, illetve Alzheimertől szenvedő betegekkel szemben [14].

A VR megoldások hatékony eszköznek bizonyulnak a fájdalom kezelésében és enyhítésében is. A Cedars Sinai kórház szerint a figyelem elterelése VR környezettel 24%-kal, esetekben nagyobb mértékben csökkentheti a fájdalmat [15]. Ilyen megoldások alkalmazhatóak szülő nők [16], akut és krónikus fájdalomtól szenvedő betegek kezelésére [17]. Esetekben a virtuális valóság terápia csökkentheti vagy akár meg is szüntetheti a gyógyszeres kezelés szükségét.

VR alkalmazások effektívnek bizonyultak gyermekek kezelésében is, mivel kimondottan jó figyelem elterelő, így csökkenti a fájdalom érzetet és szorongást [18], ahogy az 5. ábrán látható. Egy Washingtoni Egyetemi kutatás szerint égési sérülést szenvedett személyek kevesebb fájdalmat tapasztaltak miközben VR technológiával készült applikációval vonták el a figyelmüket, és funkcionális mágneses rezonanciavizsgálattal megállapították, hogy csökkent a fájdalomhoz köthető agyi aktivitás [19].

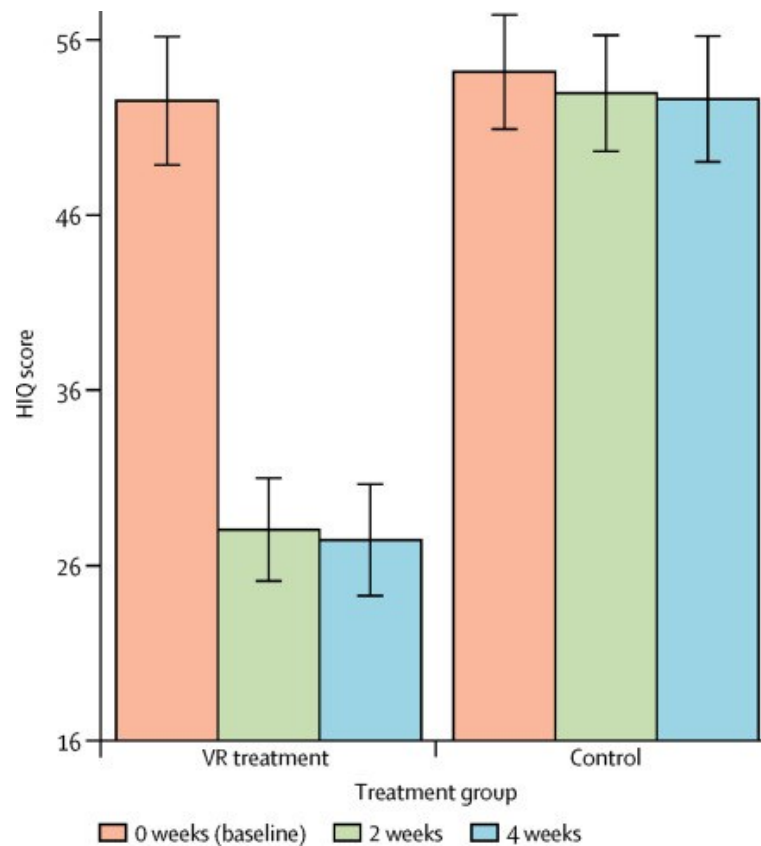


5. ábra: Fájdalomérzet VR kezeléssel és nélkül [32]

Fizikai rehabilitációs kezeléseknél is hasznos eszköznek bizonyultak a VR megoldások. Mozgásra készítő VR játékok plusz motivációt nyújtanak a gyógytornához, a terapeuták pedig személyre szabott edzésterveket képesek készíteni betegeiknek. Ezen kívül a VR rehabilitáció képes mindennapos feladatokat is gyakoroltatni a betegekkel, például bevásárlás vagy mosogatás [20]. Központi idegrendszeri sérülésben szenvedő gyerekeknél VR kezelés jelentősen javítja a mozgásfunkciókat [21].

Néhány startup, köztük a MyndVR [22] és a Rendever [23], időskori problémákra fejlesztenek megoldásokat, például kognitív és memória javításra, rehabilitációs terápiára. Tanulmányok kimutatták, hogy VR megoldásokkal javíthatóak a kognitív és motoros funkciók, kimondottan a figyelem, végtagi mozgás, egyensúly, kognitív károsodásban, vagy demenciában szenvedő időseknél [24].

Kognitív rehabilitációs problémáknál, például Sclerosis Multiplex [25] vagy agyvérzés utáni zavartság [26], tanulmányok alátámasztják, hogy VR technológia segítheti a hagyományos terápiák hatását azzal, hogy növelik az érzékszervi bemenetek mennyiségét, illetve elősegítik a különböző érzékszervek együttes használatát.



6. ábra: Téríszony kezelés VR megoldással és kontrol csoport összehasonlítása. 0, 2, és 4 hét után [28]

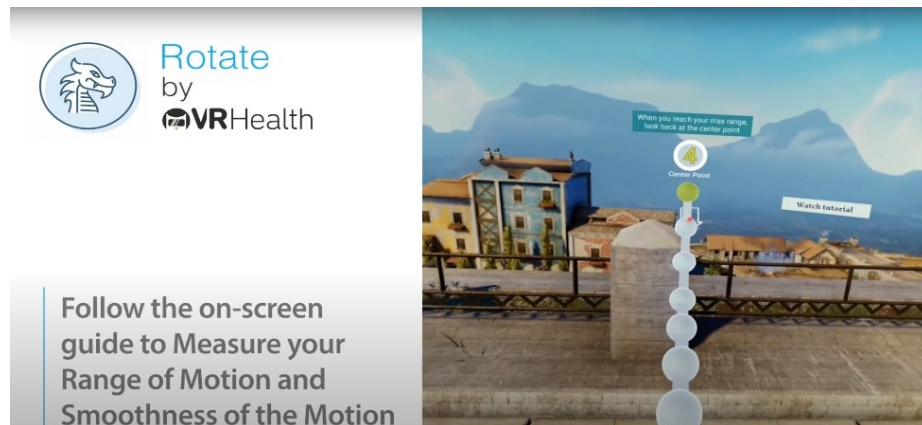
Mentális egészséget segítő VR alkalmazások használhatóak szorongás vagy trauma kezelésére, például autóbalesetet szenvedő személynek biztosíthat expozíciós terápiát pszichológus, biztonságos környezetben visszaszoktathatja az utcai környezetbe [27]. A VR általi expozíciós terápiák kimondottan hatásosak, egy kutatás szerint átlagosan 68%-kal csökkenti a tériszonyt, ahogy a 6. ábrán látható [28], de hasonló terápia alkalmazható más pszichológiai problémák kezelésére, például fóbiák, depresszió vagy PTSD.

2.2.1. XRHealth

Az Egyesült Államokbeli, Brooklynban található XRHealth IL Ltd. vállalat kiterjesztett valóság, elsősorban virtuális valóság alapú gyógy módokkal foglalkozik. Termékeik között megtalálható rehabilitációs, terápiás szoftver, illetve ezen szoftverek használatához hardver.

Három kategóriába sorolják az applikációikat. Fizikai problémákra alkalmazható gyógy módok, például nyaki sérülés vagy fájdalom, hát sérülés vagy fájdalom, légzőszervi (post Covid) rehabilitáció stb. Neurológiai problémákra alkalmazható gyógy módok, például neurológiai rendellenességekre-, kognitív és memória zavarra alkalmazható terápiás szoftver. Illetve viselkedési problémákra alkalmazható kezelések, például stressz és szorongás, depresszió, álmatlanság, nyugtalanság, függőség, figyelemhiányos hiperaktivitás-zavar [29].

Virtuális valóság alapú gyógy módot tudnak nyújtani komplexebb betegségek kezelésére is, például Parkinson kór, amit egyszerre a fizikai és neurológiai gyógy módok kategóriába is besorolnak, vagy az autizmus spektrumzavar terápiájuk, amit a neurológiai és viselkedési kategóriákba sorolnak.



7. ábra: XRHealth Rotate használat közben [36]

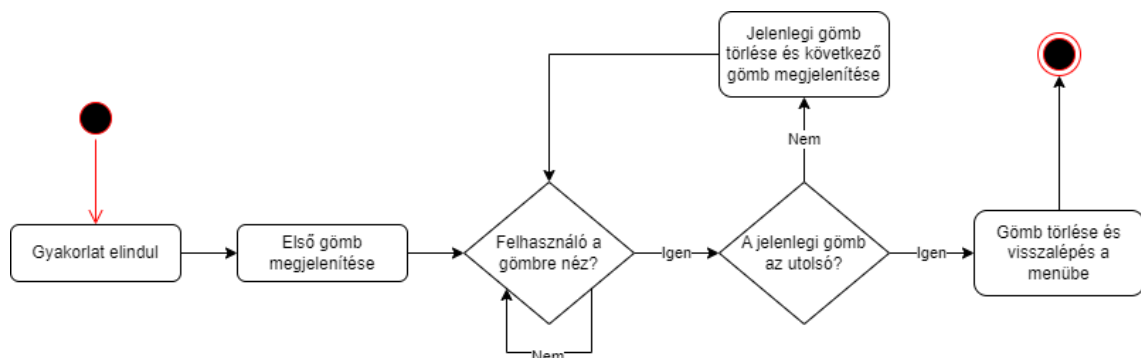
Nyak rehabilitációs szoftverük, a VRPhysio N-140 (Rotate) a gerincoszlopi régió hagyományos rehabilitációját és a gerincoszlop mozgástartományának felbecslését virtuális valóságban végző szoftver, a 7. ábrán látható. A szoftver egy egyszerű játék, amelynek gyakorlatai használhatóak hagyományos fizikoterápiái, vagy önálló otthoni mozgásra [30].

A szoftver kompatibilis több virtuális valóság szemüveggel, de csak XRHealth fiókkal és Egyesült államokbeli, ausztráliai vagy izraeli egészségbiztosítással lehetséges használni.

3. Funkcionális követelmények

A szakdolgozat feladatomban egy olyan szoftver elkészítése, ami a felhasználót végig vezeti egy nyak tornáztató gyakorlaton. A szoftver kompatibilis legyen VR szemüveggel a tornagyakorlatokhoz, és használható legyen számítógépen VR szemüveg nélkül is, tornagyakorlatok készítéséhez.

A program sárga gömbök segítségével vezesse végig a felhasználót a gyakorlaton, ennek menete a 8. ábrán látható. A gyakorlat indulásakor jelenítse meg az első gömböt a gyakorlatból, amikor a felhasználó ránéz a jelenleg aktív sárga gömbre, a gömb tűnjön el, és jelenjen meg a következő gömb, amikor már nincs megjeleníthető gömb, lépjen ki a gyakorlatból. A program képes legyen megállapítani, hogy a felhasználó mire néz a virtuális térben.



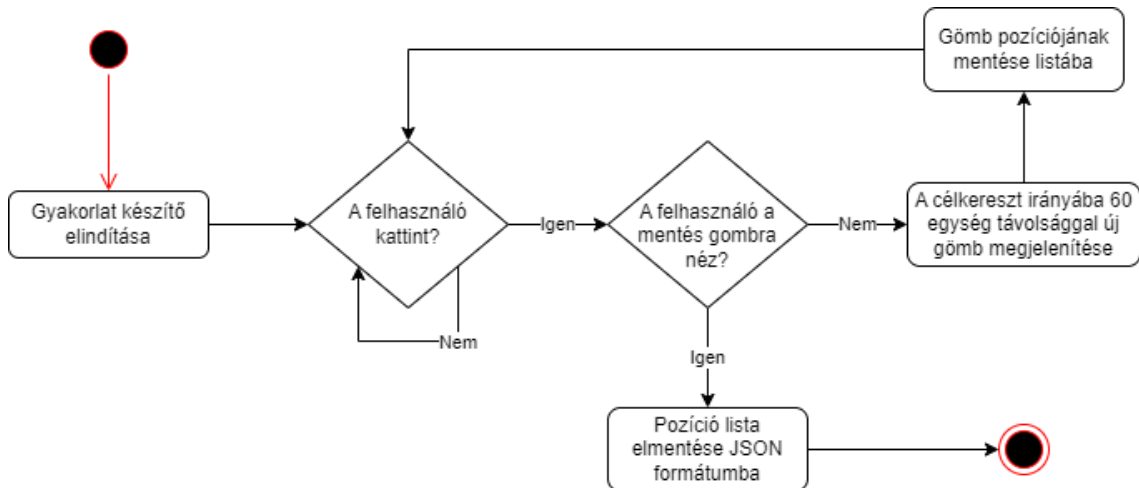
8. ábra: Gyakorlat menete

A program egy gömb pozícióját Vector3 típusként tárolja, egy gyakorlat összes gömbjének pozícióját pedig Vector3 listában tárolja.

Gyakorlat készítő módban a felhasználó kattintással tudjon lerakni sárga gömböket a virtuális térbe, amiknek a pozícióját sorrendben mentse el a program Vector3 listába, ennek a menete a 9. ábrán látható, a gyakorlaton ebben a sorrendben vezesse végig a felhasználót a program.

Az elkészített gyakorlatot JSON formátumban tárolja a program két futás között. A Vector3 listát sorrendben, a JSON formátum sztenderdnek megfelelően szerializálja, majd mentse el JSON típusú fájlba. Minden gyakorlat saját JSON fájlba legyen mentve.

A program induláskor a JSON fájlokat nyissa meg, olvassa be, és a fájlok tartalmát szerializálja Vector3 listává, a Vector3 listákat adja hozzá egy gyakorlat tárolóhoz. A program futás közben a gyakorlat tárolóban tárolja a gyakorlatokat.



9. ábra: Gyakorlat készítés menete

4. Felhasznált technológiák

4.1. Hardver

A feladat megvalósításához olyan hardvert kellett választanom, aminek a használatával mozgékony tud maradni a felhasználó, illetve fontos a VR szemüveg súlya is, egy nehezebb típus megerőltetheti a nyakat használat közben, ami a mi esetünkben nem vezetne eredményességre.

A választásom a 10. ábrán látható Oculus (ma Reality Labs) által fejlesztett, és Samsung által gyártott GearVR-ra esett, a szemüveg 2014-ben jelent meg, és egy Samsung telefont használ kijelzőként és feldolgozó egységként, ezért lényegesen olcsóbb

alternatíva a tradicionális VR szemüvegekhez képest, amik használatához szükség van egy erősebb számítógépre vagy játék konzolra. Az első modellek csak Samsung Galaxy telefonokkal működtek, de később a kompatibilis eszközök listáját lényegesen kibővítették.

A GearVR elsődleges felhasználása a videójátékok, de ezen kívül sokan használják virtuális utazásra, tanulásra és szórakozásra, GearVR platformra megjelent alkalmazásokkal a felhasználók képesek virtuális környezeteket felfedezni, filmeket vagy TV sorozatot nézni.

A GearVR erőssége a hozzáférhetőség és hordozhatóság, mivel egy kompatibilis okostelefonnal bárhol használható. A szemüveg ezen kívül lényegesen olcsóbb, mint a tradicionális VR szemüvegek, így könnyebben megfizethető a felhasználó számára főleg, ha már rendelkezik egy, a szemüveggel kompatibilis okostelefonnal.

Sajnos a GearVR rendelkezik néhány elég gyenge ponttal is. A VR élmény minősége függ a használt okostelefontól. és sok eszköz nem kompatibilis a szemüveggel. A grafikus és számítási kapacitást szintén limitálja az okostelefon hardvere, ami kevésbé magával ragadó és realisztikus élményhez vezethet, tradicionális VR szemüvegekhez képest. Ezen kívül a GearVR fejlesztését és támogatását megszakították 2019-ben, mivel Samsung más VR és AR termékekre fordította a hangsúlyt. Oculus továbbra is támogatja a GearVR eszközöket.



10. ábra: Samsung GearVR szemüveg és okostelefon

4.2. Unity játékmotor

A Unity egy népszerű motor, amit 2005-ben a Unity Technologies nevű cég adott ki, és azóta is aktívan fejleszt. Eredetileg csak az Apple által készített Mac OS X operációs rendszerre lehetett játékokat készíteni, később a támogatott operációs rendszerek számát kibővítették Windows, iOS, Android és konzolok támogatásával, a jelenleg támogatott platformok a 11. ábrán láthatóak.

A Unity motor erőssége a könnyű kezelhetőség és elérhetőség. Gyakori választás mind kezdő, mind tapasztalt fejlesztők körében, köszönhetően a felhasználóbarát felhasználói interfésznek, illetve a Unity téma köré épült online közösségnek, akik megosztják egymással a tudást és tapasztalatokat. A Unity motor ezen kívül rendelkezik előre elkészített eszközökkel, és elemekkel, amiket könnyen és gyorsan lehet integrálni a fejlesztett játékba.

A motor eleinte 3 nyelvet támogatott, C#, egy általános felhasználású magas szintű nyelv, amihez a motor a Mono nevű scripting API-t biztosítja, Boo, egy Python szintaktika által inspirált általános felhasználású nyelv, illetve UnityScript, egy Boo alapú JavaScript szintaktikájú nyelv, amit kimondottan a Unity motorhoz fejlesztettek. Utóbbi kettő támogatása megszűnt 2017-ben a kis számú felhasználók miatt, így a Unity mára csak a C# nyelvet támogatja.

A motor másik erőssége, hogy több platformot is támogat, így könnyedén lehet több platformra is fejleszteni minimális munkával. Emiatt Unity segítségével fejleszteni olcsóbb és effektívebb azokhoz a motorokhoz képest, amik csak egy platformot támogatnak.

A Unity legnagyobb hátránya a teljesítmény és optimalizációs problémák, főleg nagyobb projekteknél. Ezek kevesebb másodpercenkénti képkocka számhoz, és hosszabb töltési időhöz vezethet, amik ronthatják a játékos élményt. Ezen kívül a magas licenszdíjak belépési korlátot állíthatnak egyes fejlesztőknek, különösen a kezdőknek.



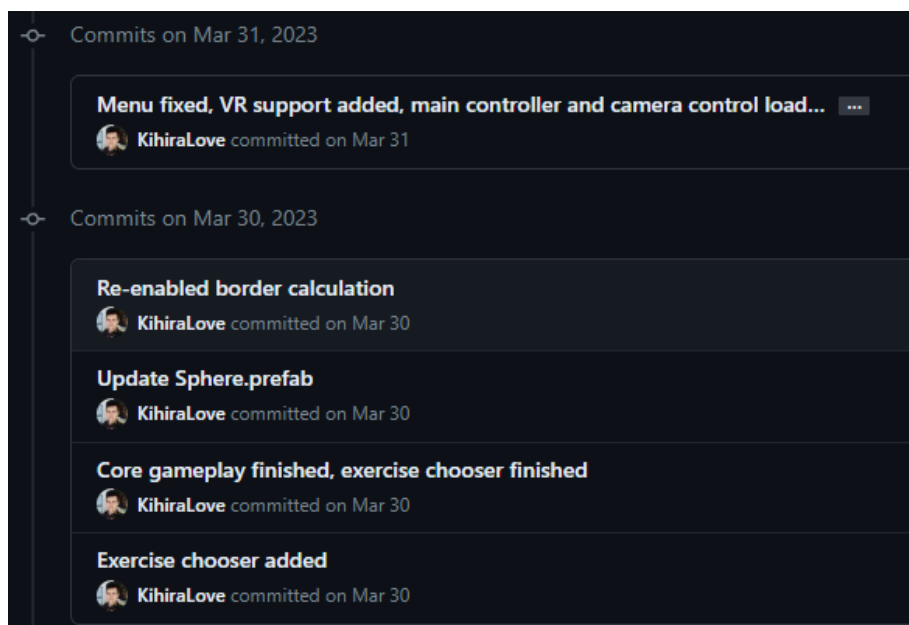
11. ábra: Unity által támogatott platformok [36]

4.3. Verziókezelés

A Git egy népszerű verziókezelő rendszer, amelyet fejlesztők a kódbázisuk módosításainak kezelésére használnak. 2005-ben jelent meg, a Linux Kernel verziókezeléséhez készítette Linus Torvalds, és azóta a szoftverfejlesztés szabványos eszközévé vált.

A Git használata több okból is fontos. Először is, lehetővé teszi a fejlesztők számára, hogy nyomon kövessék a kódjukban idővel bekövetkező változásokat, a 12. ábrán látható néhány változtatásom, amit a feladat megoldása alatt hoztam létre. Így szükség esetén könnyebben visszaállíthatják a korábbi verziókat. Ez különösen hasznos lehet olyan projekteknél, amiken több fejlesztő dolgozik párhuzamosan, ami miatt könnyen előfordulhatnak hibák vagy konfliktusok. Ezen kívül a Git lehetővé teszi a fejlesztők számára, hogy leágazzanak a fő kódbázisról, így új funkciókat próbálhatnak ki anélkül, hogy befolyásolnák azt. Ezeket az ágakat később igény szerint vissza lehet vezetni akár a fő ágba, akár másik ágakba.

A Git az együttműködést is megkönnyíti, mivel több fejlesztő egyszerre dolgozhat ugyanazon a kódbázison, és változtatásaikat zökkenőmentesen egyesíthetik. Ez megkönnyíti a kódváltozások kezelését, és biztosítja, hogy mindenki a kód legfrissebb verzióján dolgozik. Továbbá a Git elágazási és összevonási funkciói megkönnyítik a kódbeli konfliktusok kezelését és a kódban felmerülő problémák megoldását.

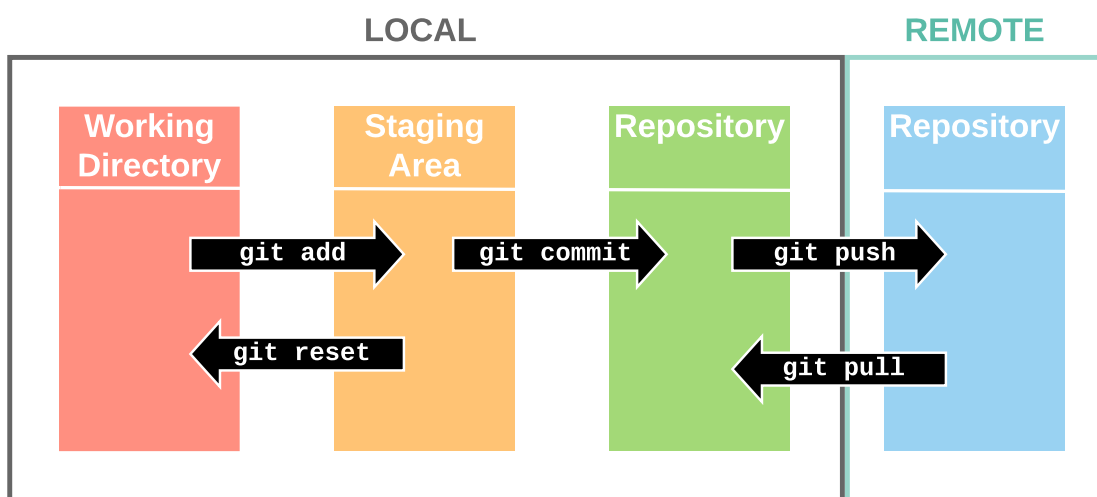


12. ábra: A szakdolgozat feladat megoldása közben létrehozott változtatások (commit) GitHub online felületen

A Git úgy működik, hogy létrehoz egy olyan tárolót, repository-t, amely a fejlesztők által végrehajtott összes kódmódosítást tárolja. Minden egyes változtatást a Git nyomon követ, amik később szükség esetén megtekinthetők vagy visszaállíthatók.

A Git további más funkciókat is biztosít, amelyek megkönnyítik a kódváltozások kezelését, például címkék létrehozását és egyesítését, problémák és hibák nyomon követését, valamint a kódváltozások részletes előzményeinek megtekintését. Ezek a funkciók megkönnyítik a fejlesztők számára a közös munkát és a kódbázisuk módosításainak kezelését, javítva a termelékenységét és csökkentve a hibák vagy konfliktusok valószínűségét.

A szakdolgozat feladatom megoldásához GitHub-ot választottam, mivel biztonságot ad, hogy a kódot egy távoli szerveren tudom tárolni, így bármilyen lokális problémából adódó adatvesztéskor nem veszítem el a már elkészült kódot. A GitHub honlapján le tudom követni a haladást, és probléma esetén bármikor vissza tudok állítani egy korábbi verziót. Valamint a GitHub Desktop alkalmazás felhasználói interfészével kimondottan egyszerű a Git használata, a változtatásaimat könnyen hozzá tudom adni a kódhoz (commit) és könnyen fel tudom tölteni a szerverre (push), ez a folyamat látható a 13. ábrán.



13. ábra: Módosítások kezelése GIT tárolóval [36]

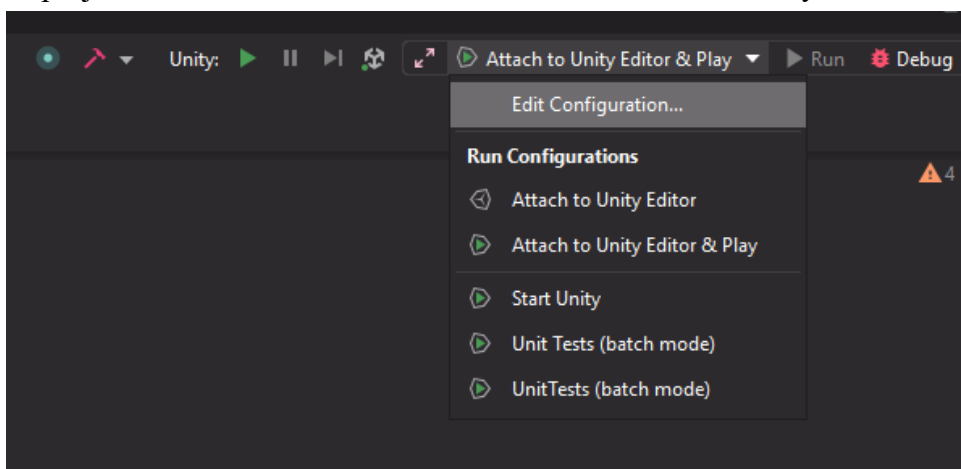
4.4. Integrált fejlesztőkörnyezet

A JetBrains Rider egy nagy teljesítményű integrált fejlesztőkörnyezet (IDE), amelyet .NET, ASP.NET, .NET Core, Xamarin, és Unity applikációk fejlesztésére terveztek. Számos nyelvet támogat, köztük a .NET nyelveket, mint C#, VB.NET, F#, ASP.NET Razor, web fejlesztésre használt nyelveket például JavaScript, TypeScript, jelölőnyelveket mint például XAML, XML, HTML, stílus leíró nyelveket, mint a CSS és SCSS, valamint JSON formátumot, ami szerializált objektumokat képes tárolni, és az SQL nyelvet, adatbázisokhoz

Az egyik fő erőssége a fejlett kódelemzési és refaktorálási képesség, amelyek lehetővé teszik a fejlesztők számára, hogy gyorsan és hatékonyan azonosítsák és javítsák a kódjukban lévő problémákat. Az IDE számos hibakeresési és elemzési eszközt is tartalmaz, amelyek megkönnyítik a Unity projektek teljesítmény problémáinak azonosítását és megoldását.

A JetBrains Rider Unity-vel együtt használva számos előnyt kínál más integrált fejlesztői környezetekkel szemben. Az egyik legfontosabb előny a Unity specifikus funkciók támogatása, beleértve a Unity Editor-t, és a Unity projektek hibakeresésének lehetőségét közvetlenül az IDE-ben, a Rider beágyazott Unity irányítópult látható a 14. ábrán. Ez megkönnyíti a fejlesztők számára a Unity projektekkel való munkát és a nagyobb projekteken való együttműködést más fejlesztőkkel.

A Rider számos együttműködési és verziókezelő eszközt is tartalmaz, beleértve a Git, Subversion, Mercurial, Perforce, és TFS verzió kezelőket, amiket közvetlenül a fejlesztő környezetből tudunk használni. Ez megkönnyíti a fejlesztőcsapatok számára a nagyobb projekteken való közös munkát és a kódváltozások hatékonyabb kezelését.



14. ábra: JetBrains Rider beágyazott Unity irányítópult

Más integrált fejlesztői környezetekkel, például a Visual Studio-val összehasonlítva a JetBrains Rider számos egyedi funkciót és előnyt kínál Unity applikációk fejlesztéséhez. Az egyik legfontosabb előnye a keresztplatformos támogatás, amely lehetővé teszi a fejlesztők számára, hogy Unity projekteken számos operációs rendszeren dolgozzanak. Valamint, a Rider egyszerűbb és testreszabhatóbb felületet kínál, így a fejlesztők a számukra leginkább szükséges funkciókra és eszközökre összpontosíthatnak.

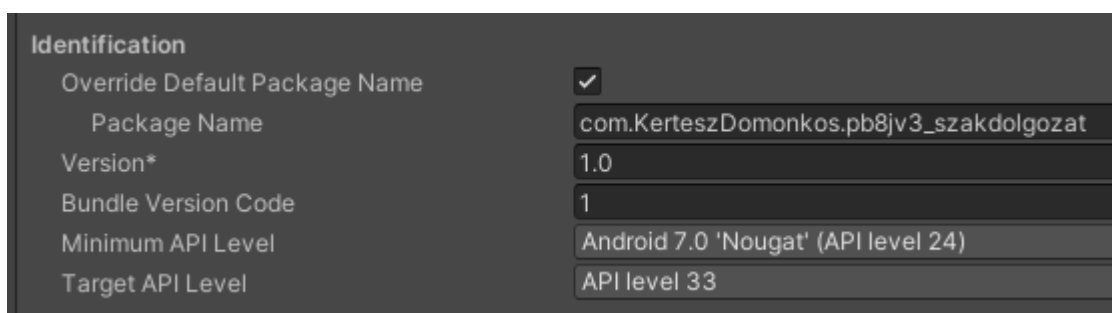
A Rider hátránya, hogy a használatához szükségünk van licenszre, míg a Visual Studio ingyenes, de szerencsére a JetBrains minden termékéhez ingyenes licenszt biztosít minden egyetemi hallgató számára.

5. Implementáció

5.1. Projekt létrehozás és beállítás

Első lépésként létrehoztam egy Git repository-t, ami a kódot és kód változtatásokat tárolja. Ezt a GitHub honlapján tudtam megtenni regisztrálás és bejelentkezés után, a New Repository gombra kattintva, majd a GitHub Desktop alkalmazásban megnyitottam a repository-t, egy lokális mappába a clone repository opcióval.

A Unity használatához telepítenünk kell a Unity Hub alkalmazást, és a segítségével le kell töltenünk a számunkra megfelelő Unity Editor verziót azokkal a modulokkal, amikre szükségünk lesz. Az én esetemben ez a Unity Editor 2021.3.22f1 verziója, valamint a szükséges modulok, Universal Windows Platform Build Support, Android Build Support, OpenJDK, Android SDK & NDK Tools. Ezután létre kell hoznunk egy új 3D projektet a telepített Editor verziót kiválasztva, abba a mappába, amibe létrehoztuk a Git tárolónkat.

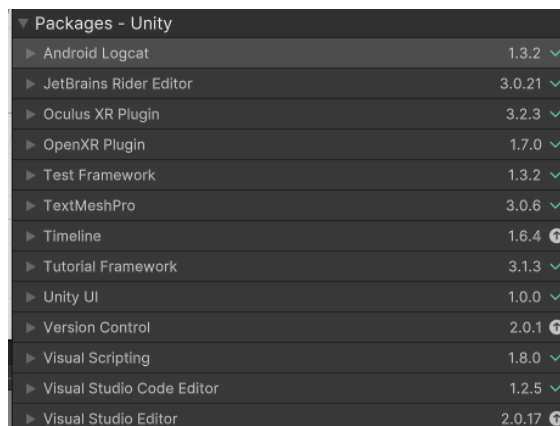


15. ábra: Android azonosító és API szint beállítások

Miután a projekt létrejött be tudjuk állítani a minimum és a target Android API szintet, amit használni szeretnénk, ezt az Edit > Project Settings > Player > Android Player > Other Settings menüpontban tehetjük meg, ez látható a 15. ábrán. A Google Play elvárja, hogy a lehető legmagasabb API szintet válasszuk, mint target API, ezért a projektem API szintjét 33-ra állítottam, a minimum API szintet pedig 24-es szintre, ami az Android „Nougat” szintje, mivel a rendelkezésre álló teszt eszköz ezt az operációs rendszert használja.

Meg kell adnunk az Android SDK és NDK, OpenJDK, és Gradle elérési útvonalakat, a Edit > Preferences > External Tools menüpontban, ha a Unity nem találta meg automatikusan. Ezután meg kell nyitnunk a File > Build Settings ablakot és itt ki kell választanunk az Androidot platformként. Ezen kívül kikapcsoltam a Build App Bundle lehetőséget, hogy apk formátumú fájlt készítsen a Build opció, amit közvetlenül tudok telepíteni az okostelefonra, és kikapcsoltam az Export Project lehetőséget, mivel nem szeretném Android Studio-ban megnyitni a projektet, JetBrains Riderrel viszont igen, ezért a Edit > Preferences > External Tools menüben, az External Script Editor lehetőségnél kiválasztottam a JetBrains Rider integrált fejlesztői környezetet.

A fejlesztés során szükségem volt különböző eszközökre, amik a 16. ábrán láthatóak, ezeket a Window > Package Manager menüben tudtam hozzáadni a projektemhez. Ezek az eszközök a JetBrains Rider Editor, ami a Rider és Unity közti integrációhoz nyújt eszközöket, az Oculus XR Plugin, ami a GearVR és Oculus alkalmazásokhoz szükséges, az Android Logcat, ami a hardveren történő debugolást könnyíti meg, az Input System, ami Unity egyik könyvtára a bemenetek könnyű feldolgozásához, a TextMeshPro, ami a térben lévő szövegek megjelenítéséhez szükséges, és a Unity UI, ami a programom felhasználó interfészének felépítését segíti.



16. ábra: A szakdolgozat feladat megvalósításához használt Unity csomagok

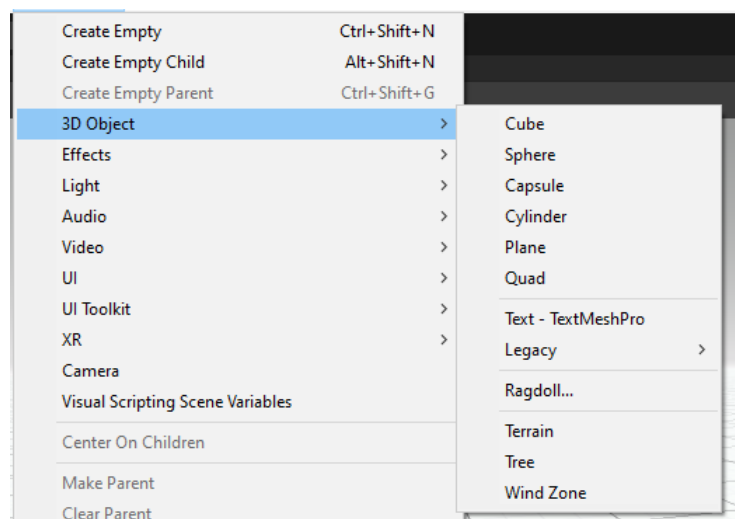
5.2. Jelenet és játék objektumok

A Unity Editor első megnyitásakor egy üres jelenet fogad minket, Level.unity néven. A Unity jelenet egy olyan szint vagy környezet, amelyben a játék objektumok, karakterek és egyéb eszközök elhelyezésre és elrendezésre kerülnek. A jelenet tartalmazza az összes szükséges elemet és beállítást, amely a játék egy adott részének, például egy szintnek vagy menüképernyőnek a létrehozásához szükséges.

Jelenet létrehozásakor a felhasználó különböző játék objektumokat, például karaktereket, akadályokat, terepet és világítást, valamint hang- és vizuális effekteket adhat hozzá és konfigurálhat, a játékobjektumok hozzáadása a 17. ábrán látható. Ezeket a játék objektumokat a Unity beépített eszközeivel, például a Transform eszközzel a Scene nézetben lehet manipulálni és elrendezni a jeleneten belül.

A jelenetek a Unity játékfejlesztési folyamat alapvető részét képezik, lehetővé téve a fejlesztők számára, hogy komplex játékvilágokat hozzanak létre a játék objektumok és eszközök meghatározott területekbe való rendezésével. Egy teljes játék létrehozásához több jelenet is használható, amelyek mindegyike egy-egy különböző szintet vagy környezetet képvisel.

A Unity azt is lehetővé teszi a fejlesztők számára, hogy a játék során zökkenőmentesen váltsanak a jelenetek között, lehetővé téve a játék különböző részei közötti zökkenőmentes átmenetet. A jelenetek használatával a fejlesztők olyan magával ragadó élményeket hozhatnak létre a játékosok számára, amelyek egy összefüggő világot érzékeltetnek.

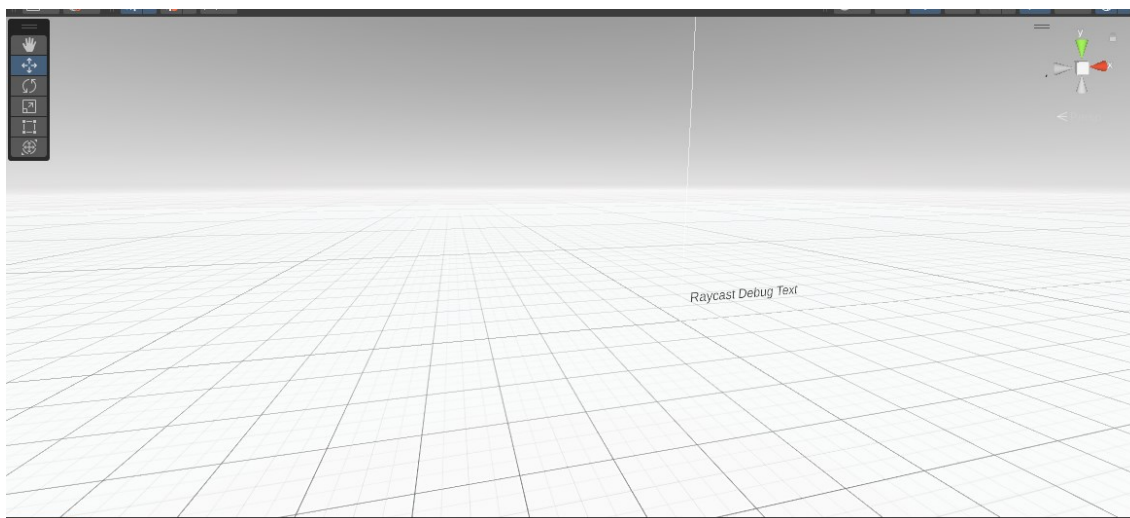


17. ábra: Unity virtuális térben elhelyezhető játék objektumok

Az én applikációmban kimondottan fontos a teljesítmény, mivel a Unity kimondottan erőforrás igényes, és Android telefonon szeretnénk futtatni a programot, amik nem rendelkeznek kimondottan erős processzorral, és mivel VR applikációról beszélünk, a legkisebb problémák, például akadozás, késés, képkockák kiesése rosszul érezhető a felhasználónál, mivel a képernyő csupán néhány centiméterre van a szemétől, így a kis problémák is könnyen észrevehetőek.

Valamint Android rendszeren a jelenetek közti váltás nem történik meg azonnal. Gyorsabb okostelefonoknál egy megakadásként érzékelhetjük a jelenet váltást, ami szintén rosszul érezhető, lassabb okostelefonoknál pedig várakozást igényel.

Ezen okok miatt úgy döntöttem, hogy a jelenetben csak minimális játék objektumot helyezek el, mint ahogyan az a 18. ábrán látható. Ahelyett, hogy statikus, előre felépített jelenetek között váltok, egy jelenetet használok csak, és a játék objektumokat kódból, dinamikusan hozom létre, amikor szükséges, és törlöm őket amikor már nincs rájuk szükség. Így a lehető legkevesebb erőforrást használja a program, és nincs szükség jelenetek közti váltásra sem.



18. ábra: A virtuális tér indítás előtt üres, mivel majdnem minden játék objektumot kódból hozunk létre

5.2.1. Objektum hierarchia

Unityben a hierarchia, ahogyan a játék objektumok szerveződnek és elrendeződnek a jeleneten belül. A hierarchia a játékobjektumok közötti szülő-gyermek kapcsolatra utal, ahol egy szülő objektumhoz egy vagy több gyermek objektum kapcsolódhat. Egy objektum lehet egyszerre szülő és gyerek is.

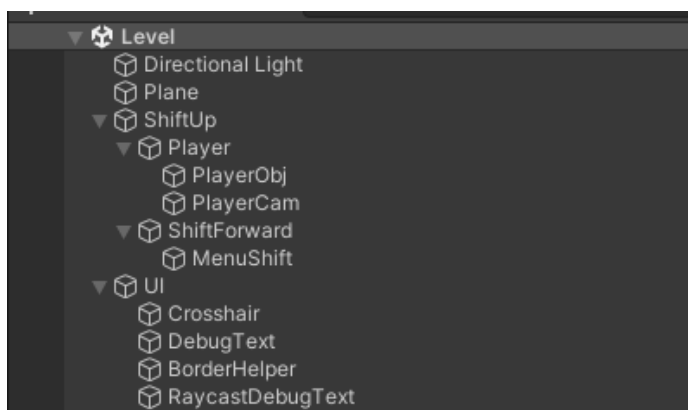
A hierarchia több okból is fontos a Unityben. Először is, lehetővé teszi a fejlesztők számára, hogy komplex játékobjektumokat hozzanak létre több kisebb objektum egyetlen entitássá történő egyesítésével. Például egy autó egy versenyjátékban több kisebb objektumból, például kerekekből, ajtókból és alvázból állhat, amelyek mindegyike egy szülőobjektum alá csoportosítható. Ez megkönnyíti az autó egészének mozgását és manipulálását, ahelyett, hogy minden egyes részt külön-külön kellene kezelni.

Másodszor, a hierarchia lehetővé teszi a játékobjektumok jobb szervezését és kezelését. A fejlesztők a hierarchiák segítségével csoportosíthatják a kapcsolódó objektumokat, és logikus struktúrát hozhatnak létre a játékukban, ami megkönnyíti a navigálást és a megértést. Ez különösen hasznos lehet, ha nagy és összetett projekteken dolgozunk.

Harmadszor, a hierarchia fontos a játékobjektumok viselkedésének szabályozásához. A szkriptek vagy komponensek szülő objektumokhoz való csatolásával a fejlesztők egyszerre irányíthatják az összes gyermekobjektum viselkedését, ahelyett, hogy minden egyes objektumon külön-külön kellene elvégezniük ugyanazokat a változtatásokat. Ez időt és energiát takaríthat meg, és megkönnyíti a konzisztens és koherens játékmenet létrehozását.

Az én applikációmban azok a játék objektumok, amik konstansan léteznek, és nem törlődnek soha a program futása alatt, statikus hierarchiába vannak rendezve, ahogy az a 19. ábrán látható.

Az első ilyen objektum egy Plane nevet viselő sík. Az objektum rendelkezik egy Transform komponenssel, hogy a virtuális térben rendelkezzen pozícióval, tengelyforgással és mérettel, valamint egy Mesh Renderer komponenssel, aminek megadok egy egyszerű négyzetrács textúrájú anyagot, és így a sík négyzetrácsos lesz. Ez kelti a virtuális tér látszatot, így a felhasználó nem csak szürkeséget lát maga körül.



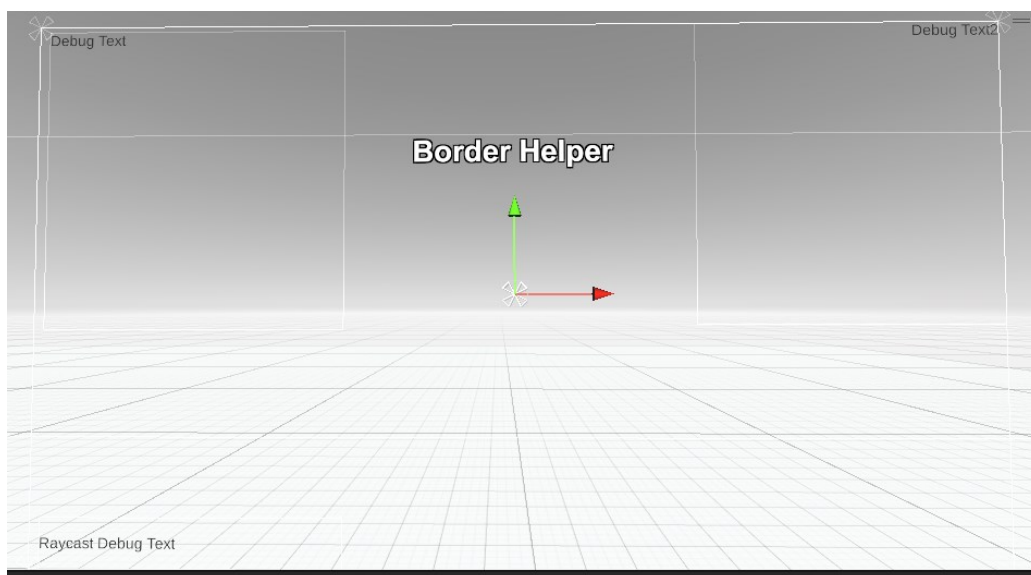
19. ábra: A statikus játékobjektumok hierarchiája

A következő objektum a virtuális teret megvilágító irányított fény, Directional Light néven. Az irányított fények hasznosak az olyan hatások létrehozásához, mint a napfény egy kültéri jelenetben. Az irányított fények sok szempontból úgy viselkednek, mint a nap, és úgy gondolhatunk rájuk, mint távoli fényforrásokra, amelyek végtelenül messze léteznek. Az irányított fénynek nincs azonosítható forráshelye, így a fényobjektum bárhol elhelyezhető a jelenetben. A jelenet összes objektumát úgy világítja meg, mintha a fény mindig ugyanabból az irányból érkezne. A fény távolsága az objektumoktól nincs meghatározva, így a fény nem csökken.

A hierarchia része a felhasználó interfész is, ez egy UI nevű objektum, ami rendelkezik egy Canvas komponenssel, ami az interfész elemeket mozgatja dinamikusan, hogy igazodjanak különböző méretű képernyőkhöz, valamint rendelkezik egy Canvas Scaler komponenssel, ami magát a Canvas-t méretezi át a képernyőmérethez igazodva.

A UI több gyermek objektummal is rendelkezik, egy célzó kereszttel, ami mindig a képernyő közepén jelenik meg, és segít a felhasználónak, a célzásban, valamint több debugolóhoz szükséges TextMeshPro elemet, amikre különféle üzeneteket és értékeket írtam ki a fejlesztés során, ezzel segítve a hibakeresést, ezek láthatóak a 20. ábrán.

Az utolsó objektum a hierarchiában egy olyan objektum, ami csak pozícióval és gyerek objektumokkal rendelkezik. Ez az objektum, a ShiftUp, a pozíció komponensében tárolja a Plane síktól való távolságot. A gyerek objektuma, ShiftForward, szintén csak pozícióval rendelkezik, és a felhasználótól 10 egységre előre helyezkedik el. Valamint a



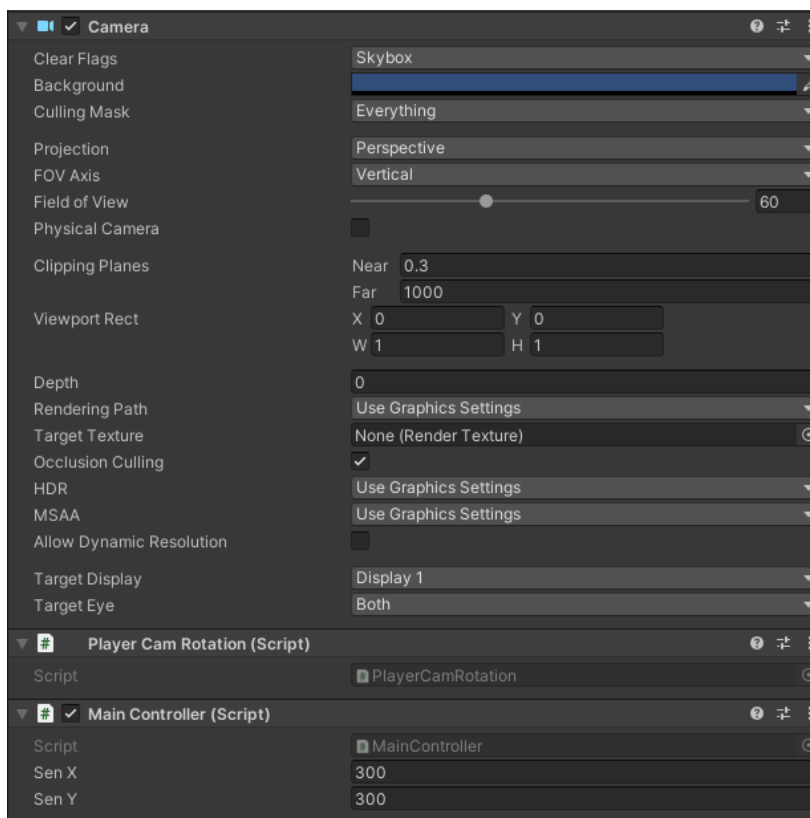
20. ábra: A felhasználói interfészt képző Canvas objektum és elemei, középen a célkereszttel

ShiftForward is rendelkezik egy csak pozícióval rendelkező gyerek objektummal, MenuShift, ami további 40 egységre helyezkedik el, a játékos előtt.

Fontos megemlíteni, hogy a gyerek objektumok megörökölik a szülő objektum pozícióját, és a saját lokális pozíciójukat hozzáadva helyezkednek el a virtuális világban. Ebben az esetben ez azt jelenti, hogy a ShiftUp (0, 60, 0) pozícióját megörököli a ShiftForward, aminek a lokális pozíciója (0, 0, 10), és így a virtuális világban a (0, 60, 10) pozícióban fog elhelyezkedni. Ugyan így a MenuShift megörököli a ShiftForward pozícióját, így a (0, 0, 40) lokális pozícióval, (0, 60, 50) pozícióban fog elhelyezkedni.

Ezek az objektumok azért fontosak mert a pozíciójuk használatával tudjuk megállapítani kódból, hogy hova szeretnénk a program használata közben dinamikusan játék objektumokat létrehozni. Ezzel a megoldással, ha kódból elmozgatjuk ezeket az objektumokat, referencián keresztül máshol le tudjuk kérni az új pozíciót.

Az utolsó gyerek objektuma a ShiftUp objektumnak a Player objektum, ami szintén egy üres objektum, és csak arra szolgál, hogy szülője legyen, és összefogjon két objektumot, a PlayerCapsule és PlayerCam objektumokat.

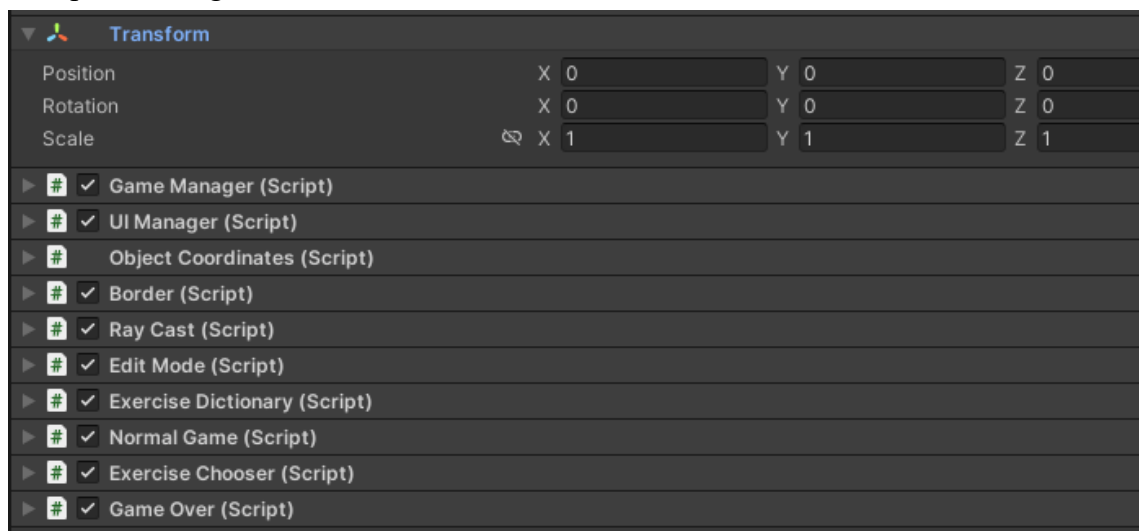


21. ábra: A virtuális térben lévő Camera objektum beállításai és komponensei

5.2.2. Player objektum

A PlayerCam objektum a jelenet fő kamerája, ezen keresztül lát a felhasználó. Rendelkezik egy Camera komponenssel, aminek a beállításában megadtam látószöveget, ami 60 fok, az égbolt (SkyBox) színét, valamint, hogy VR szemüvegen futtatva mindkét szem képe ebből a kamerából eredjen. A PlayerCam objektumhoz ezen kívül két szkript van komponensként hozzáadva. Ezek a beállítások láthatóak a 21. ábrán.

A PlayerObj objektum egy láthatatlan objektum, és komponensei azok a szkriptek melyek a MonoBehaviour osztályból vannak származtatva, ahogy az a 22. ábrán látható. A MonoBehaviour a Unity Mono API része- Egy MonoBehaviour osztályból származtatott osztály életciklusa megegyezik annak a játékobjektumnak az életciklusával, melynek komponense. Mivel a PlayerObj statikusan a virtuális térbe van elhelyezve, ezért a program indulásakor létrejön, és a program bezárásakor törlődik. Így a komponens szkriptek mindig futnak.



22. ábra: A PlayerObj objektum komponensei

5.2.3. Menü, menü elemek, és térbeli objektumok

A program használatához szükséges egy menü, ez általában a felhasználó interfész részét képezi, viszont a VR akadályokat állít elé, nincs kurzor, amit a képernyőn mozgathatnánk, ezért a kurzor célját átveszi a célzókereszt, ami a UI része és rögzített helyen van a képernyőn, a többi UI elemmel együtt, vagyis mindig a felhasználó fejével együtt forognak, és mindig ugyan akkora távolságra vannak egymástól. Emiatt lehetetlen a célzókeresztrel a UI más elemeire célozni.

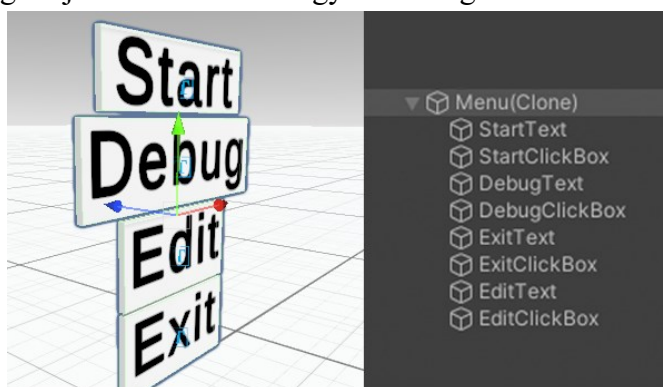
Erre a problémára az az egyszerű, és VR applikációknál alkalmazott megoldás, hogy a menü nem a UI része, hanem a virtuális térben lévő objektumok, amikre rá tudunk kattintani, Ray Casting segítségével. A menü elemeit és minden más objektumot, dinamikusan hozom létre kód segítségével. Ehhez segítenek a prefabok.

A prefab, a prefabricated object (előre gyártott objektum) rövidítése, egy előre elkészített játék objektum, amely többször is felhasználható a projekt során. A prefabok lényegében sablonok, amelyeket a jelenettől külön lehet létrehozni és szerkeszteni, majd szükség szerint létrehozni és a jelenetbe helyezni, erre egy példa a 23. ábrán látható.

A prefabok egy vagy több játékobjektumból állhatnak, amelyek mindegyike saját tulajdonságokkal, komponensekkel és szkriptekkel rendelkezik. Ebben az esetben a MenuPrefab tartalmazza a menü gombjait, amik egyszerű téglatest alakú objektumok, illetve a gombok TextMeshPro típusú szövegét, amik a téglatestek előtt helyezkednek el. Ha a menühöz egy új gombot akarok hozzáadni, csak annyi a dolgom, hogy a prefabot szerkesztem, a Scene view ablakban.

A prefabok használata Unityben számos előnnyel jár. Időt és energiát takaríthat meg, mivel lehetővé teszi a fejlesztők számára, hogy a projekt során újra felhasználhassanak játék objektumokat. Valamint segít a konzisztencia és a karbantarthatóság biztosításában, mivel a prefabon végrehajtott változtatások a projekt során a prefab minden példányában megjelennek. Ezen kívül javítja a teljesítményt, mivel a prefabok használata csökkenti az egyedi játékobjektumok számát a jelenetben, és ezáltal csökkenti a játékmotor feldolgozási terhelését.

A menüt képző téglatest objektumok rendelkeznek Box Collider komponenssel, ami miatt képes más játék objektumokkal ütközni, ami nélkül nem tudnánk Ray Castot alkalmazni rajta, illetve mindegyik téglatest elem rendelkezik egy egyedi névvel, aminek a segítségével meg tudjuk különböztetni egymástól a gombokat a kódban.



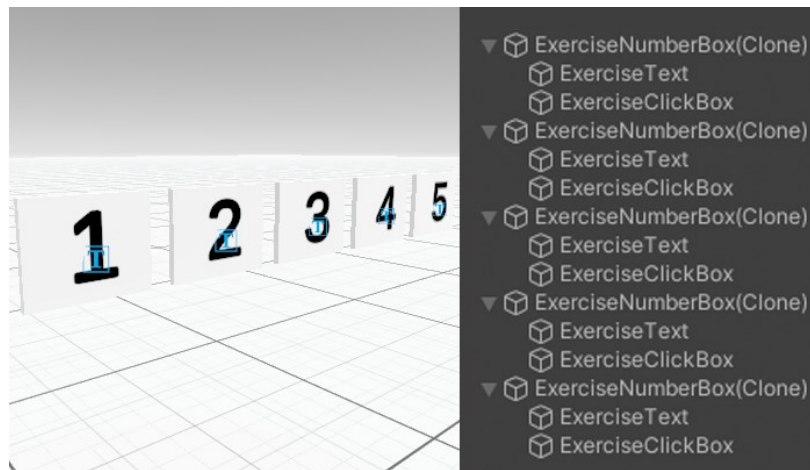
23. ábra: Menü prefabból létrehozott játék objektumok és hierarchiájuk

5.2.4. Gyakorlat választó menü

A főmenüből a Start gombbal a gyakorlat választó menübe lépünk, ahol ki tudjuk választani, hogy melyik gyakorlatot szeretnénk elindítani. A gyakorlat választó menü, a főmenühöz hasonlóan téglatest alakú objektumokból épül fel, a főmenüvel ellentétben ezeknek az objektumoknak a pozícióját és feliratát dinamikusan számolom ki, mivel csak annyi gombot szeretnék a gyakorlat választó menüben megjeleníteni ahány gyakorlat van.

Ezt úgy érem el, hogy a gombhoz létezik egy prefab, és azt klónozzom, valahányszor új gombot szeretnék létrehozni. Az első gomb pozíciója (-30, 80, 50), egy sorban 10 gomb helyezkedik el, és összesen 5 sornyi gomb elhelyezése lehetséges. A sorok és oszlopok között 6 egység hely van, hogy a gombok ne csússzanak össze. A gombok megjelenítése a 24. ábrán látható

Minden gomb rendelkezik egy TextMeshPro gyerek objektummal, ami a gomb felirata, a felirat a gomb számozása 1-től 50-ig, ez alapján tudom azonosítani a gombokat kódból.



24. ábra: Gyakorlat választó menü játék objektumai és hierarchiája

5.3. Ray casting

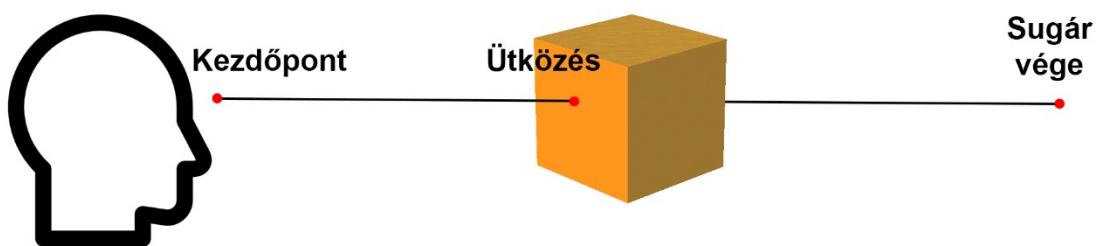
A ray casting egy sokoldalúan felhasználható technika, amely a játék fejlesztés, számítógépes grafika, számítógépes geometria, és renderelés fontos része. A ray casting alapja, hogy egy virtuális vonalat vagy sugarat, ray-t, lövünk egy objektumtól, hogy megállapítsuk, metszi-e vagy ütközik-e más objektummal, ahogy az a 25. ábrán látható. Ezzel a technikával meg tudjuk állapítani többek között, hogy két objektum ütközik-e, milyen objektumok láthatóak a játékos számára és melyek vannak takarásban, és számomra a legfontosabb, mire néz rá a játékos.

A Unity számos beépített lehetőséget biztosít a Ray casting megvalósítására, köztük a Raycast, a SphereCast, a CapsuleCast és a BoxCast. Mindegyik módszer másképp működik, és különböző típusú játék mechanizmusokra optimalizált.

A Raycast legegyszerűbb formája, egy egyenes vonal és a jelenet más objektumai közötti ütközések észlelésére szolgál. A SphereCast egyenes vonal helyett gömb alakú sugarat vet a kezdőpont köré, a CapsuleCast hasonló a SphereCasthoz, de kapszula alakú sugarat bocsát ki, végül a BoxCast egy téglatest alakú sugarat vet ki, ezek hasznosak lehetnek nagyobb vagy összetettebb objektumokkal vagy nagy mennyiségű objektummal való ütközések észleléséhez.

Ezekon a beépített lehetőségeken kívül a Unity támogatja az egyéni Ray casting implementációkat is, így a fejlesztők saját algoritmusokat és szkripteket hozhatnak létre. Ez hasznos lehet összetettebb vagy speciálisabb játékmechanizmusok megvalósításához, vagy a Ray casting teljesítményének optimalizálásához nagy vagy összetett játékkörnyezetekben.

A Ray Casting implementálására saját megoldást használtam. A felhasználó látótere közepéből lövök sugarat. A sugarat úgy számítom ki, hogy a Camera játék objektum ViewPortPointToRay függvénynek paraméterként megadom a (0.5, 0.5, 0) Vector3 értéket, ezzel megkapok egy a képernyő közepéből kiinduló irányvektort a Camera fordulásának irányába.



25. ábra: Ray casting segítségével állapítjuk meg, mire néz a felhasználó

Az irányvektor direction értékét megszorozom 100-zal, és az így kapott vektort a Physics Unity beépített osztály Raycast függvényével kilövöm. A függvény paraméterei a sugár kiindulópontja, a sugár hosszúsága és iránya vektorként, és out paraméter módosító kulcsszóval ellátott RaycastHit típusú változó, a függvény true eredménnyel tér vissza, ha a sugár ütközik egy objektummal, és false értékkel, hogyha nem. Ütközés esetén a RaycastHit változóban kapom meg a sugárral ütköző objektumokat, amit az out referencián keresztül érek el. A RaycastHit változóból lekérem az objektum nevét, amivel

a sugár először ütközött és a különböző lehetséges objektum neveket egy switch-case kezeli le, például az ExitClickBox nevű objektumra, ha kattint a felhasználó, ami a menü része, a program bezáródik, az implementáció a 26. ábrán látható.

A Ray casting menüknél és gyakorlat készítő módban minden képkockán lefut, de sugarat csak akkor lő ki, amikor a felhasználó kattint, mivel csak azt szeretnénk megtudni, hogy milyen objektumra vagy milyen irányba kattint, és így optimalizáltabb a futás. Gyakorlat alatt minden képkockán kilőjük a sugarat, nincs szükség, hogy a felhasználó kattintson, mivel azt szeretnénk megtudni, hogy a felhasználó ránéz-e a gömbökre.

```
private void CheckForRayHit()
{
    Ray ray = cam.ViewportPointToRay(pos: new Vector3(x: 0.5f, y: 0.5f, z: 0f));
    Vector3 rayDirection = ray.direction * 100f;
    if (_ui.debug.Enabled)
    {
        Debug.DrawRay(start: ray.origin, dir: rayDirection, Color.red);
    }
    if (!MainController.Instance.IsMainInput()) return;
    if (!Physics.Raycast(ray.origin, rayDirection, out RaycastHit hit)) return;
    GameObject o = hit.collider.gameObject;
    _ui.debug.RaycastDebugText = "Ray collided with " + o.name;
    switch (o.name)
    {
        case "DebugClickBox":
            _game.SwitchDebugMode();
            return;
        case "StartClickBox":
            _game.State = GameState.ExerciseMenu;
            return;
        case "EditClickBox":
            _game.State = GameState.EditMode;
            return;
        case "ExitClickBox":
            Application.Quit();
            return;
    }
}
```

26. ábra: Ray casting implementálása a Menü navigálásához

A gyakorlatok tárolását a program futása közben úgy oldottam meg, hogy egy gyakorlat közben használt gömbök pozícióját egy Vector3 listában tárolok, és az összes gyakorlatot egy Vector3 lista listában, amit az ExerciseDictionary Singleton osztály tárol. Az ExerciseDictionary osztály a program indulásakor beolvassa fájlból a gyakorlatokat, valamint segéd függvényeket biztosít, amikkel új gyakorlatot tudunk hozzáadni a gyakorlatok listájához, és le tudunk kérni index alapján gyakorlatokat.

A JSON két struktúrára épül. Név és érték párok gyűjteménye például objektum, rekord, struktúra, szótár, hash-tábla, kulcsos lista, asszociatív tömb. És értékek rendezett listája például tömb, vektor, lista. Ezek univerzális adatszerkezetek, szinte minden modern programozási nyelv támogatja őket valamilyen formában, így a JSON tökéletes ilyen nyelvek közti adatcserére. A JSON formai követelménye a 27. és 28. ábrán látható.

```

graph LR
    start(( )) -- object --> L1(( { ))
    L1 --> W1[whitespace]
    W1 --> L2(( , ))
    L2 --> W2[whitespace]
    W2 --> S[string]
    S --> L3(( ))
    L3 --> W3[whitespace]
    W3 --> C(( : ))
    C --> V[value]
    V --> L4(( ) --> end(( ))
    L1 --> L4
    L2 --> L4
    L3 --> L4
  
```

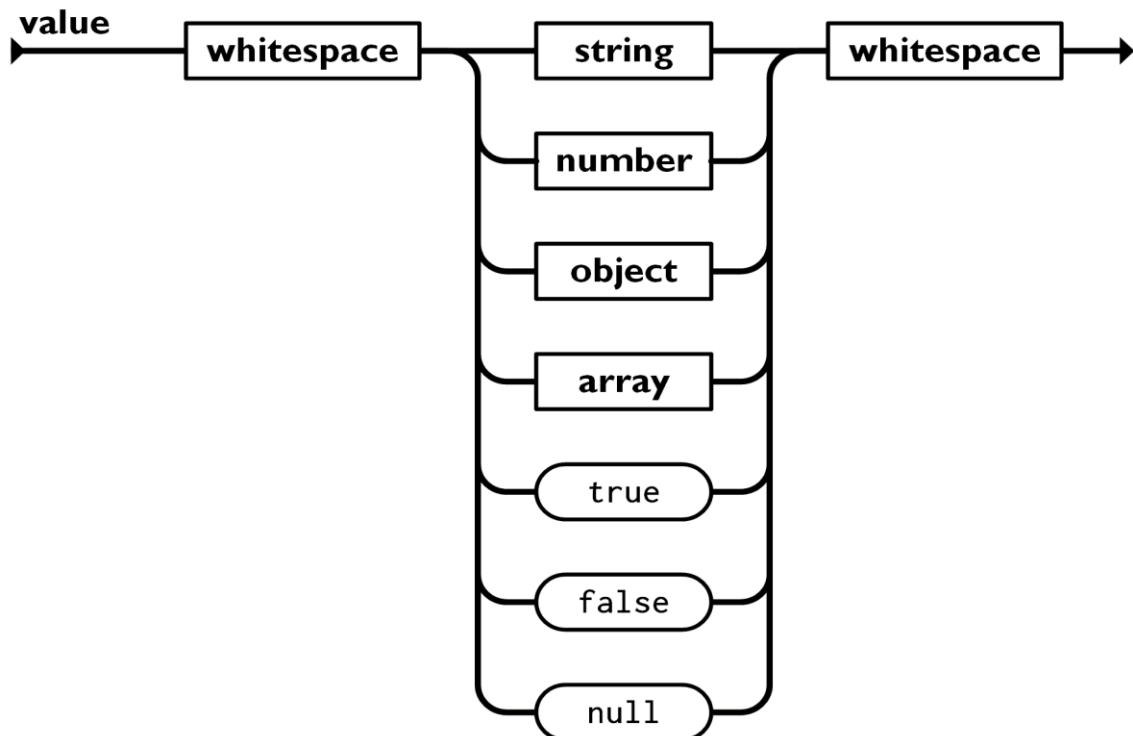
The diagram illustrates the grammar for a JSON object. It starts with an arrow labeled "object" pointing to a circle containing an opening curly brace "{". From this circle, the path splits into three main branches that eventually merge back into a single path leading to a circle containing a closing curly brace "}". The top branch consists of a rectangle labeled "whitespace". The middle branch starts with a circle containing a comma ",", followed by a rectangle labeled "whitespace", then a rectangle labeled "string", and finally a circle containing an empty space " ". The bottom branch starts with a rectangle labeled "whitespace", followed by a circle containing a colon ":", then a rectangle labeled "value", and finally a circle containing an empty space " ". All three branches merge into a single path that leads to the closing curly brace "}".

függvényeket írtam, mivel csak Vector3 listákat szeretnék JSON formátumba írni és onnan beolvasni, ezért nem kell sok típust lefednem a függvényekkel.

A `SerializeVectorListToJson` függvény, aminek az implementációja a 29. ábrán látható egy vektor listát kap paraméterként, és egy sztring értéket térít vissza, ami a vektor listát tárolja JSON formátumban. A függvényben `StringBuilder`-t használok a szerializáláshoz. A JSON szabványnak megfelelően a függvény elején egy „{ Vector3 :[„ tartalmú sztringet fűzök hozzá, az üres `StringBuilder`-hez, ez a JSON kezdetét jelző nyitó kapcsos zárójel, a típus megnevezése, a típus és érték elválasztó kettőspont, és a tömb kezdetét jelző nyitó szögletes zárójel.

Ezután végig iterálok a vektor listán, és vektoronként hozzáfűzöm a `StringBuilder`hez a következő formában, ({\"x\":) sztring, ami a vektor nyitó kapcsos zárójele, az érték típusa, és a típus és érték elválasztó kettőspont, hozzáfűzöm a vektor x értékét, 3 tizedes pontossággal. Majd egy (,\"y\":) tartalmú sztringet fűzök a `StringBuilder`hez, ami az típus érték párokat elválasztó vessző, a következő típus, majd a típus és érték közötti kettőspont, és hozzá fűzöm a vektor y értékét.

A vektor z értékét az y értékkel megegyező módszerrel fűzöm hozzá, és végül egy záró kapcsos zárójellel lezárom a JSON formátumú vektort, a listában utolsó vektor kivételével minden vektor után egy típus érték párok közti elválasztó vesszőt fűzök, és az utolsó vektor után a tömb lezáró záró szögletes zárójelet és a JSON lezáró záró kapcsos



28. ábra: JSON érték felépítése [36]

zárójel (`}]`). Ezután a `StringBuilder`-t sztringé alakítom és visszatérítem. A visszatérített JSON formátumú sztringet egy másik függvény kiírja egy JSON kiterjesztésű fájlba, a fájl nevének pedig a gyakorlat számát adja meg a függvény, például `3.JSON`.

A `DeserializeJsonToVectorList` egy JSON fájlból kiolvasott tartalmat kap meg sztringként, és egy lista `Vector3` értékkel tér vissza. Először levágjuk a vektorokat tartalmazó tömb nyitó és záró szögletes zárójelet, az első vektor nyitó kapcsos zárójelét, és az utolsó vektor záró kapcsos zárójelét.

Az így kapott rész sztringet feldaraboljuk a `Split` függvénnyel a vektorok közti záró kapcsos zárójel, vektorokat elválasztó vessző, és nyitó kapcsos zárójel karakter hármasokra, és egy sztring tömbbe elmentjük az így kapott sztringeket. A tömböt LINQ segítségével dolgozom fel.

A LINQ, Language Integrated Query vagy nyelvbe ágyazott lekérdezés, a .NET keretrendszer egyik összetevője, amely hatékony, deklaratív szintaxist biztosít különböző forrásokból, például gyűjteményekből, adatbázisokból és XML dokumentumokból származó adatok lekérdezéséhez. Lehetővé teszi a fejlesztők számára, hogy összetett lekérdezéseket írjanak tömör és olvasható módon, anélkül, hogy alacsony szintű részletekkel kellene foglalkozniuk, például azzal, hogy hogyan iteráljanak az adatstruktúrákon.

```
public string SerializeVectorListToJson(List<Vector3> vectorList)
{
    StringBuilder sb = new StringBuilder();
    sb.Append("{ \"Vector3\": [");

    for (int i = 0; i < vectorList.Count; i++)
    {
        Vector3 v = vectorList[i];
        sb.Append("{ \"x\": ");
        sb.Append(v.x.ToString( format: "F3" ));
        sb.Append(", \"y\": ");
        sb.Append(v.y.ToString( format: "F3" ));
        sb.Append(", \"z\": ");
        sb.Append(v.z.ToString( format: "F3" ));
        sb.Append("}");

        if (i < vectorList.Count - 1)
        {
            sb.Append(",");
        }
    }

    sb.Append("]");
}
```

29. ábra: Vektor lista JSON formátumba szerializáló függvény implementációja

A LINQ a C# és a VB.NET programozási nyelvekre épül, és több szabványos lekérdezési operátort biztosít, amelyekkel szűrni, rendezni, csoportosítani és átalakítani lehet az adatokat.

Egyik fő előnye, hogy egységes szintaxist biztosít a különböző forrásokból származó adatok lekérdezéséhez. Akár egy adatbázis, akár egy XML dokumentum vagy egy objektumgyűjtemény lekérdezése történik, a szintaxis ugyanaz marad, ami megkönnyíti a kód írását és karbantartását.

A LINQ másik előnye, hogy elősegíti a funkcionális programozási stílust, ami tömörebb és kifejezőbb kódot eredményezhet. A LINQ lekérdezések olyan műveletek sorozatából állnak, amelyek az eredeti forrás módosítása nélkül alakítják át és szűrik az adatokat. Ez megkönnyíti a hibakeresést és elemzést.

Az én LINQ kódom a sztring tömb elemein végig iterál, minden egyes vektort felbont a három komponensére, x, y, z, ezeket az értékeket float típusú változóba Parse-ol, és egy új Vector3 értéket készít belőle. Az iteráció után a Vector3 értékeket egy listaként visszatéríti, ennek az implementációja a 30. ábrán látható.

```
public List<Vector3> DeserializeJsonToVectorList(string json)
{
    int startIndex = json.IndexOf(value: "{", StringComparison.Ordinal) + 1;
    int endIndex = json.LastIndexOf(value: "}", StringComparison.Ordinal);

    string[] vectorJsonStrings = json.Substring(startIndex, length: endIndex - startIndex) // string
        .Split(separator: new[] { "{", " " }, StringSplitOptions.RemoveEmptyEntries);

    return (from vectorJson:string in vectorJsonStrings
        select vectorJson.Replace(oldValue: "{", newValue: "")
            .Replace(oldValue: "}", newValue: "") // string
            .Split(separator: new[] { ' ', ' ' }, StringSplitOptions.RemoveEmptyEntries) // string[]
        into components:string[]
        let x:float = float.Parse(components[0].Split(separator: ':')[1])
        let y:float = float.Parse(components[1].Split(separator: ':')[1])
        let z:float = float.Parse(components[2].Split(separator: ':')[1])
        select new Vector3(x, y, z)).ToList();
}
```

30. ábra: JSON deszerializálása vektor listává LINQ segítségével

5.5. A játéktér kiszámolása és objektum pozíció újraszámolás

Mivel az applikációm rehabilitációs célra készítettem, fontosnak tartottam, hogy a program figyelembe vegye a felhasználó fizikális képességeit. Erre azért volt szükség, hogy ne vezessen esetleges sérüléshez, ha a program olyan mozdulatot kér a felhasználótól amire az nem képes.

Ezért a gyakorlat indulásakor a program megkéri a felhasználót, egy felhasználó interfészre írt szöveggel, hogy fordítsa a fejét balra amennyire csak tudja. Ezt úgy ellenőrzöm, hogy ha az utolsó két másodpercben a kamera szöge 2 fokok határon belül van, akkor a felhasználó elérte a balra fordulás határát, ha nem akkor újra indítom a számlálót. Majd ugyanerre megkéri a program jobbra, felfelé, és lefelé is, a bal oldalra fordulást ellenőrző függvény a 31. ábrán látható.

A határokat térbeli koordinátaként mentem el, méghozzá úgy, hogy amikor lejár a 2 másodperces időzítő, a kamera pozíciójához hozzáadom a kamera irányvektorának és a játékos-objektum közötti távolság szorzatát. Ez megadja azt a pozíciót, ami 50 egységre van a játékostól abban az irányban amerre a játékos néz, a bal oldali pozíció kiszámolásának függvénye a 32. ábrán látható.

```
private void SetLeftBorder()
{
    if (!_helperTextCalled)
    {
        _ui.borderHelper.SetBorderTextLeft();
        _helperTextCalled = true;
    }

    float yRotation = _playerCamRotation.EulerAngles.y;
    if (yRotation is > 360 or < 180) return;
    if (Math.Abs(_lastDegree - yRotation) < 2.0)
        _pop--;
    else
    {
        _pop = pop;
        _lastDegree = yRotation;
        return;
    }

    if (_pop != 0) return;
    LeftDegree = yRotation;
    _borderFlag = BorderFlag.Left;
    _pop = pop;
    _helperTextCalled = false;
    _ui.debug.SpawnDebugObject(LeftBorder);
}
```

31. ábra: A játéktér határának megállapítása a felhasználótól balra

```

_borderRotationDegrees[0] = value;
Vector3 tempBorder = _playerCamRotation.CameraPosition + _playerCamRotation.ForwardVector * _objectCoordinates.SpawnDistanceFromPlayer;
tempBorder.y = 60;
LeftBorder = tempBorder;

```

32. ábra: Bal határ térbeli pozíciójának kiszámolása a kamera forgásszögéből

Ezeket a pozíciókat normalizálom, ami későbbi számítások miatt szükséges. A jobb és bal oldali határ y koordinátáját 60-ra állítom, ami a kamera távolsága a (0, 0, 0) ponttól. A fenti és lenti határ x koordinátáját pedig 0-ra állítom.

Ezek segítségével a játék a gyakorlat betöltésekor újra számolja a pozíciókat, hogy mindenképpen bent legyenek a határok között. A számoláshoz a program négy részre osztja a virtuális teret. Először újra számol minden pozíciót, ami a felhasználótól balra helyezkedik el, ezeken végig iterál, készít egy új pozíciót, aminek x és z koordinátái megegyeznek az újra számolandó pozíció x és z koordinátaival, az y koordinátája pedig 60. Ekkor alkalmazhatunk polárkoordináta számolásokat, mivel a kamera, az eredeti határ, a felhasználó által megadott határ, és az újra számolandó pozíció egy síkban helyezkednek el. Létrehozok még egy (0, 60, 50) koordinátájú pozíciót, ami a számolásban fog segíteni, és az összes pontba számolok vektort a kamera pozícióból.

Kiszámolom a vektorok közötti szögeket úgy, hogy α a kamera-segédpont és kamera-eredeti határ vektorok közti szög, γ a kamera-segédpont és kamera-felhasználó által megadott határ vektorok közti szög, β a kamera-segédpont és kamera-újra számolandó pozíció vektorok közötti szög. Ekkor a kamera-segédpont és kamera-új pozíció vektorok közti szög,

$$\delta = \gamma * \beta / \alpha$$

átszámolva radiánba,

$$\tau = \frac{\pi}{180} * \delta$$

az új pozíció z értéke

$$z = r * \cos(\tau)$$

x értéke

$$x = r * \sin(\tau) * -1$$

ahol r a kamera és eredeti pozíció távolsága. Azért szükséges negatívvá változtatni az értéket, mert a kamera $x = 0$ koordinátán helyezkedik el, így a tőle balra eső részében a virtuális környezetnek minden x koordináta negatív, és az egyenlet csak pozitív értékeket tud kiszámolni. Ezzel az x és z, valamint az eredeti pozíció y koordinátájával létrehozom az új pozíciót, ennek az implementációja a 33. ábrán látható.

Ezután az összes kamerától jobbra eső pozíciót számolom újra. Ez hasonlóan történik, mint a bal oldali pozíciók újra számolása. Ugyan úgy kiszámolom a vektorokat, és a vektorok közti szögeket, ezek segítségével megkapom a τ szöget. Átszámolom radiánba

$$\tau = \pi / 180 * 90 - \delta$$

itt azért szükséges 90 fokból kivonni, mert másik tengelytől szeretnénk számolni a koordinátát. Ezután megkapjuk az x és y értékeket

$$x = r * \cos(\tau)$$

$$z = r * \sin(\tau)$$

az új pozíció y értéke pedig az eredeti pozíció y értéke.

```
private List<Vector3> ScalePositionsLeft(List<Vector3> positions)
{
    ObjectCoordinates coords = ObjectCoordinates.Instance;
    Border border = Border.Instance;

    Vector3 player = transform.position;
    Vector3 origin = coords.MenuShiftPosition;

    Vector3 playerOriginVector = origin - player;
    Vector3 playerBaseBorderVector = coords.BaseLeftBorder - player;
    Vector3 playerNewBorderVector = border.LeftBorder - player;

    float alpha = Vector3.Angle(playerOriginVector, playerBaseBorderVector);
    float gamma = Vector3.Angle(playerOriginVector, playerNewBorderVector);

    for(int i = 0; i < positions.Count; i++)
    {
        if (!(positions[i].x < 0)) continue;

        Vector3 positionNormalizedToPlane = new Vector3(positions[i].x, y.60, positions[i].z);
        Vector3 playerBaseObjectVector = positionNormalizedToPlane - player;

        float r = coords.SpawnDistanceFromPlayer;
        float beta = Vector3.Angle(playerOriginVector, playerBaseObjectVector);
        float ro = (gamma * (beta / alpha));
        double roRads = (Math.PI / 180) * ro;

        double z = r * Math.Cos(roRads);
        double x = r * Math.Sin(roRads) * -1;

        Vector3 newVector = new Vector3((float)x, positions[i].y, (float)z);
        positions[i] = newVector;
    }

    return positions;
}
```

33. ábra: A felhasználótól balra lévő pozíciók igazítása a játéktérhez

Ezután újból két részre osztom a virtuális teret, először újra számolok minden pozíciót, ami a kamera felett helyezkedik el, utána pedig az összeset, ami alatta. Így minden pozíciót kétszer számolok újra, egyszer horizontálisan, egyszer pedig vertikálisan.

Az alsó és felső pozíciók újra számolása ugyan úgy történik, mint a jobb és bal oldali pozícióké, annyi különbséggel, hogy a számoláshoz használt ideiglenes pozícióknak az x értékét állítja 0-ra, ezután az összes objektumunk egy síkban helyezkedik el, és kiszámíthatjuk a vektorokat, majd szögeket ugyan azon módszerrel.

A felhasználótól felfele lévő pozíciókhoz a z és y értékeket megkapom

$$z = r * \cos(\tau)$$

$$y = r * \sin(\tau) + 60$$

az x érték pedig egyértelműen az eredeti pozíció x értéke. A felhasználótól lefele lévő pozíciók a z és y értékek pedig

$$z = r * \cos(\tau)$$

$$y = r * \sin(\tau) * -1 + 60$$

az x érték pedig az eredeti pozíció x értéke. Ezeknél a számításoknál azért kell 60 egységet hozzáadnunk mivel a kamera 60 egységre helyezkedik el az origótól az y tengelyen. Valamint a lenti számolásnál szükséges negatívvá átváltani az értéket, mivel meg szeretném fordítani a tengelyt, és ez után adom hozzá a 60 értéket, hogy elmozgassam a kamera irányába.

Az így kapott pozíciók a kapott határon belül lesznek, viszont a számolás nem teljesen pontos, és ezért egy esztétikai javító függvénnyel, ami a 34. ábrán látható, átszámolom a kamerától való távolságukat, és ugyan arra az értékre állítom, a konzisztencia eléréséhez. Ezt úgy teszem meg, hogy minden pozícióhoz számolok egy vektort a kamera pozícióból, ezt normalizálok, hogy irányvektort kapjak, és megszorozom a kívánt távolsággal, ami 60 egység, végül hozzá adom a kamera pozíciójához a kapott vektort, és az így kapott pozíció lesz a gyakorlatban használt új pozíció.

```
private List<Vector3> NormalizePositions(List<Vector3> positions)
{
    return positions.Select(
        position :Vector3 =>
            transform.position + Vector3.Normalize(position - transform.position) * ObjectCoordinates.Instance.SpawnDistanceFromPlayer
    ).ToList(); // List<Vector3>
}
```

34. ábra: A játéktérre átszámolt pozíciók esztétikai módosítása, így minden pozíció 60 egységes helyezkedik el a felhasználótól

5.6. Szkriptek

5.6.1. Szkript életciklusa

A Unity szkriptek készítéséhez az osztályt, amit szkriptként szeretnék futtatni, a MonoBehaviour osztályból kell származtatnom, és egy játék objektumhoz kell csatolnom komponensként. Ilyenkor a szkript életciklusa megegyezik a játék objektumével, aminek a gyereke.

Az MonoBehaviour osztályból több függvényt megörököl ilyen esetben az osztályunk, amik a szkript életciklusának különböző pontjain automatikusan lefutnak. Ilyen például az Awake függvény, ami a szkript objektum inicializálásánál egyszer hívódik meg, a Start függvény, ami a szkript objektum aktiválódásakor egyszer hívódik meg, az Update függvény, ami a program futása alatt minden képkocka kirajzolásakor meghívódik, a FixedUpdate, ami a képkockák sebességétől függetlenül, a fizikai szimuláció frekvenciájával megegyezően hívódik meg, másodpercenként 50-szer.

Ezek a függvények biztosítják a program futásának alapját, minden szkript Update vagy Awake függvénnyel indul, ami inicializálja a szkriptben használt változókat, lekér vagy beállít adatokat. Az Update és FixedUpdate függvények pedig a szkriptek viselkedését tartalmazzák.

```
private static GameManager _instance;

public static GameManager Instance
{
    get
    {
        if (_instance == null)
            Debug.LogError( message: "Game Manager singleton not instantiated!");
        return _instance;
    }
}

private void Awake()
{
    _instance = this;
}
```

35. ábra: Singleton implementáció

Az osztályok készítésénél gyakran alkalmaztam Singleton programtervezési mintát, amit a MonoBehaviour függvények és a C# beépített Property mező leegyszerűsítenek, erre példa a 35. ábrán látható.

Minden Singleton osztály tárolja a saját példányát, private static változóban, rendelkezik egy Instance property-vel, ami visszatéríti a Singleton példányát, valamint az Awake függvényben a példányt beállítjuk a this kulcsszóval. Az Awake függvény a szkript objektum inicializálásánál meghívódik, és a Singleton példány változóba elmenti a szkript objektumot. Ezután bármelyik másik osztályból elértem a példányt és tudom használni az Instance property segítségével.

5.6.2. MainController osztály

A MainController a Unity MonoBehaviour osztályból származtatott osztály, a Singleton programtervezési minta alapján készült, és a PlayerCam játék objektum komponense. Az osztály feladata, hogy a program indulásakor ellenőrizi a platform típusát, Windows vagy Unity Player estén a PlayerCam játék objektumhoz hozzá adja komponensként a CamControlPC szkriptet, Android esetén pedig TrackedPoseDriver komponenst ad a játék objektumhoz. Ezek felelnek a kamera mozgatásáért számítógépes, ennek az implementációja a 26. ábrán látható, illetve VR környezetben.

Az osztály ezen kívül rendelkezik egy IsMainInput nevű függvénnyel, ami igazgal tér vissza, ha a bal egér gomb vagy a VR szemüveg érintő padja le van nyomva a függvény meghívásakor.

```
private void RotateCam()
{
    float mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime * senX;
    float mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime * senY;

    _yRotation += mouseX;
    _xRotation -= mouseY;

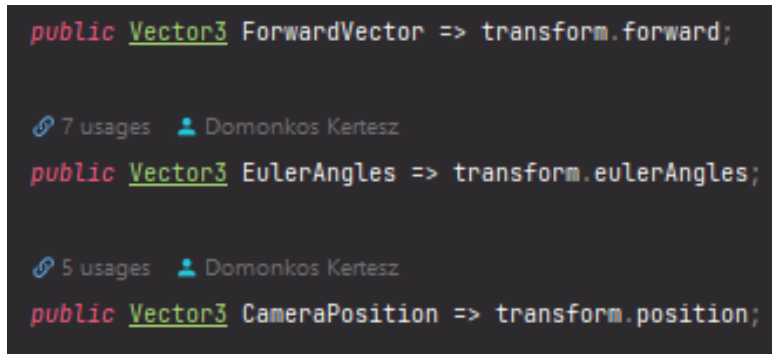
    transform.rotation = Quaternion.Euler(_xRotation, _yRotation, z0);
}
```

36. ábra: Kamera forgását számoló függvény

5.6.3. CamControlPC és PlayerCamRotation osztályok

A CamControlPC a Unity MonoBehaviour osztályból származtatott osztály, és a kamera mozgatásáért felel számítógépes környezetben. Az osztály minden képkockára kiszámolja az előző képkocka óta történt egérmozgást a Unity Input könyvtár segítségével, majd módosítja a jelenlegi kamera forgást a kiszámított értékekkel.

A PlayerCamRotation a Unity MonoBehaviour osztályból származtatott osztály, a PlayerCam játék objektum komponense, a Singleton programtervezési minta alapján készült, és ezen az osztályon keresztül a programból bárholnan lekérhető a kamera pozíciója, irányvektora, forgás szögei, Property segítségével, ez a 37. ábrán látható.



```
public Vector3 ForwardVector => transform.forward;

7 usages Domonkos Kertesz
public Vector3 EulerAngles => transform.eulerAngles;

5 usages Domonkos Kertesz
public Vector3 CameraPosition => transform.position;
```

37. ábra: A Camera objektumhoz tartozó irányvektor, forgásszög, és pozíció lekérését segítő Property-k

5.6.4. GameManager osztály

A GameManager a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, a Singleton programtervezési minta alapján készült, és a program menetéért felel. Tárolja a jelenlegi státuszt, ami a GameState enum típusként, aminek a lehetséges értékei, Menu, ExerciseMenu, EditMode, BorderCalculation, InGame, GameOver, és Status property-n keresztül lehet elérni más osztályokból.

A GameManager osztály hozza létre a játék menü objektumait, amikor a játék státusza Menu, és törli a játék objektumokat, amikor kilépünk a menüből. Ezen kívül a SwitchDebugMode függvénnyel létrehozza vagy kitörli azokat a játék objektumokat, amik vizuális segítséget nyújtottak a debugoláshoz, például a Ray casting sugarát kirajzolja, a láthatatlan objektumok pozíciójára elhelyez látható objektumokat.

Ezen kívül rendelkezik két függvénnyel, a SpawnObject paraméterként egy prefab-ot kap GameObject típusként, és egy Vector3 pozíciót, és visszatér egy a virtuális térben létrejött új játékobjektumra mutató referenciával, amit a prefab-ból klónozott és a kapott

pozícióra helyezett el. A DestroyObject egy játék objektumot kap paraméterként, törli a játék objektumot az aktív objektumok közül, és nem tér vissza semmivel.

5.6.5. UIManager, DebugManager, BorderHelper osztályok

A UIManager a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, a Singleton programtervezési minta alapján készült, és a felhasználói interfész elemek tartalmának frissítéséért felel. Minden interfész elemhez tárol referenciát, amin keresztül új értéket lehet megadni. Ezen kívül az osztály tárolja a DebugManager és BorderHelper osztályokat, és minden képkockán kiírja a felhasználói interfészre a kamera jelenlegi x és y szögét, amennyiben a debug funkció be van kapcsolva.

A BorderHelper osztály olyan funkciókat tárol, amikkel a felhasználói interfészre utasításokat tudunk a felhasználónak kiírni, például, hogy fordítsa a fejét egy adott irányba, amiket a felhasználó fizikai képességeinek felmérésekor használunk.

A DebugManager olyan függvényeket tárol, amikkel a program futása közbeni értékeket tudjuk a felhasználói interfészre, illetve a konzolra kiírni. Ezen kívül rendelkezik egy SpawnDebugObject függvénnyel, ami egy pozíciót kap paraméterként és megjelenít egy debugoláshoz használható élén piros gömböt a térben. A függvény a gömb játék objektumához egy referenciával tér vissza.

5.6.6. ExerciseDictionary és FileHandler osztályok

Az ExerciseDictionary osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, a Singleton programtervezési minta alapján készült és az a gyakorlat tároló, ami a JSON formából beolvasott és szerializált vektor lista gyakorlatokat tárolja egy List<List<Vector3>> változóban.

Az osztály a program indulásakor megnyitja a gyakorlatokat tartalmazó mappát és beolvassa az összes .JSON kiterjesztésű fájl nevét egy sztring tömbbe, majd egyenként megnyitja a fájlokat, beolvassa és a FileHandler osztály segítségével vektor listává alakítja, ennek a függvénye a 38. ábrán látható.

Az osztályból függvénnyel, gyakorlat szám alapján lehet gyakorlatot lekérni, és függvénnyel lehet új gyakorlatot a gyakorlatok listájához hozzáadni.

A FileHandler osztály tartalmazza a JSON és vektor lista közti szerializáló és deszerializáló függvényeket, illetve a SaveVectors függvényt, ami egy vektor listát kap paraméterül, és elmenti egy új JSON kiterjesztésű fájlba a szerializáló függvény segítségével.

```
private void InitializeDictionary()
{
    FileHandler fileHandler = new FileHandler();
    string[] fileNames = Directory.GetFiles( fileHandler.ExercisesPath, searchPattern: "*.json");
    if (fileNames.Length == 0) return;
    foreach (string fileName in fileNames)
    {
        string json = File.ReadAllText(fileName);
        List<Vector3> exercise = fileHandler.DeserializeJsonToVectorList(json);
        AddExercise(exercise);
    }
    Debug.Log( message: "There are " + _exercises.Count + " exercises loaded from JSON");
}
```

38. ábra: Függvény a gyakorlatok betöltéséhez a program indulásakor

5.6.7. ObjectCoordinates osztály

Az ObjectCoordinates osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, a Singleton programtervezési minta alapján készült és azért felel, hogy a virtuális térben lévő láthatatlan játékobjektumok pozícióját a programból bárhonnán le lehessen kérni.

Az osztályból le lehet kérni a pozíciót, ahova a menü objektum megjelenik, a játéktér határainak pozícióit, a felhasználótól való távolságot, ahol a játék alatt használt gömbök megjelennek.

5.6.8. RayCast osztály

A RayCast osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, és a menüben történő Ray casting-ért felel.

A kilőtt sugárnak ellenőrzi az ütközéseit, és lekezeli egy switch-case használatával. A DebugClickBox ki- és bekapcsolja a debug funkciókat, a StartClickBox a játék státuszát ExerciseMenu értékre állítja be, az EditClickBox a játék státuszát EditMode értékre állítja be, az ExitClickBox pedig bezárja a programot.

5.6.9. ExerciseChooser osztály

Az ExerciseChooser osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, a Singleton programtervezési minta alapján készült és a gyakorlat választó menü létrehozásáért és törléséért felel.

Az osztály tárolja a választott gyakorlat számát, aminek segítségével tölti be a program a választott gyakorlatot a gyakorlat választó menü törlése után. Az osztály rendelkezik függvénnyel a gyakorlat választó menü elemek pozíciójának kiszámolásához, a menü elemek létrehozásához, aminek a függvénye a 39. ábrán látható, és a menü elemek törléséhez a virtuális térből.

Ezen kívül az osztály kezeli a gyakorlat választó menühöz a Ray casting megoldását is, amivel ellenőrizni tudjuk, hogy a felhasználó melyik gyakorlatot választotta.

```
private void SpawnBoxes()
{
    int exerciseNum = _exercises.NumberOfExercises;
    if (exerciseNum == 0)
    {
        _game.State = GameState.Menu;
        return;
    }

    for (int i = 1; i <= exerciseNum; i++)
    {
        GameObject obj = _game.SpawnObject(exerciseChooserBoxPrefab, position: NextVector());
        TextMeshPro tmp = (TextMeshPro)obj.transform.GetChild(0).GetComponentInChildren(typeof(TextMeshPro));
        tmp.text = i.ToString();
        _boxArray.Add(obj);
    }
}
```

39. ábra: Gyakorlatválasztó menü gombokat megjelenítő függvény

5.6.10. EditMode osztály

Az EditMode osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, és a gyakorlat készítő menetéért felel. Amikor a játék státusza EditMode, elindul a gyakorlat készítő, ilyenkor, ha a felhasználó kattint, kiszámolunk egy pozíciót, a kamera pozíciójához hozzáadjuk a kamera irány vektorának és a kamera és objektum kívánt távolságának szorzatát. Ez a pozíció lesz a játékban a gömb pozíciója, amit meg is jelenítünk, és a pozíciót hozzáadjuk egy listához.

Amikor a felhasználó végzett a pozíciók felvételével, a mentés és kilépés gombbal ki tud lépni a gyakorlat készítőből. Ilyenkor a program törli az összes gömböt és a mentés

és kilépés gombot a virtuális térből, és a pozíciók listáját a FileHandler osztály segítségével szerializálja és kiírja egy JSON fájlba.

A szkript prefabból készíti a mentés és kilépés gombot, és tartalmaz Ray casting függvényt a gomb megnyomásának ellenőrzésére.

5.6.11. NormalGame és Border osztályok

A Border osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, a Singleton programtervezési minta alapján készült és a játék kezdésekor a felhasználó nyakának fordulási határait méri fel, ehhez számolásokat és függvényeket tartalmaz.

A határok megállapításánál az osztály kiírja a felhasználói interfészre az utasításokat, számontartja a fordulási szöget, számlálóval követi, hogy a fordulás lelassult-e és igény esetén újra indítja azt, kiszámolja a határ pozíciókat, a kamera pozíciójából, irány vektorából, és a kívánt távolságból.

A NormalGame osztály a Unity MonoBehaviour osztályból származtatott osztály, a Player játék objektum komponense, és a gyakorlat menetért felel, ez a 40. ábrán látható. Az osztály elindítja a gyakorlatot amikor a státusz InGame, lekéri a választott gyakorlat számát a gyakorlat választó menüből, majd a szám alapján lekéri a gyakorlat vektor listáját a gyakorlat tárolóból. Ezután újra számolja a pozíciókat, hogy a felhasználó nyakfordulás szög határain belül legyenek, végül egymás után megjeleníti őket.

```

if (_game.State != GameState.InGame) return;
if (!_isExerciseLoaded)
{
    LoadExercise();
    _currentObject = _game.SpawnObject(spherePrefab, GetNextVector().Value);
}

Ray ray = Camera.main.ViewportPointToRay(pos: new Vector3(x: 0.5f, y: 0.5f, z: 0f));

if (!Physics.Raycast(ray.origin, direction: ray.direction * 100f, out RaycastHit hit)) return;
GameObject o = hit.collider.gameObject;
_ui.debug.RaycastDebugText = "Ray collided with " + o.name;
if(o.name == "Sphere(Clone)")
{
    NextObject();
}

```

40. ábra: Gyakorlat betöltéséért és menetéért felelő függvény

6. Felhasználói instrukciók

A használathoz a programot Android eszközre apk formátumú fájlal fel kell telepíteni, majd a telepített applikációt elindítani. Az applikáció induláskor megjelenít egy feliratot, amiben megkéri a felhasználót, hogy csatlakoztassa az eszközt a GearVR szemüveghez.

Csatlakozás után a felhasználó a menüben találja magát, ahonnan elindíthat gyakorlatot vagy gyakorlat készítőt úgy, hogy a célkereszttel a kívánt menüpontra céloz, és megérinti a szemüveg oldalán lévő érintőpadot.

Gyakorlat készítő módban szintén az érintőpaddal tudja a felhasználó lerakni a gömböket, ezzel kijelölve a gyakorlat útját. A gyakorlat elkészítése után a kilépés és mentés gombra célozva, az érintőpad megérintésével tudja elmenteni a gyakorlatot és visszalépni a menübe.

Gyakorlat indításakor először a gyakorlat választó menüben kell kiválasztani a kívánt gyakorlatot. Ezután a program megkéri a felhasználót, hogy a fejét fordítsa balra, jobbra, fel és le amennyire csak tudja, ezzel kijelölve a játékteret. Ezután a program végig vezeti a felhasználót a gyakorlaton.

7. Lehetséges továbbfejlesztés

Mint minden programban, az én megoldásomban is számos helyen adódik lehetőség továbbfejlesztésre. Jelenlegi formájában a gyakorlatok csak a fej forgásszögét nézik, és tornáztatják, itt lehetőség a fej jobb és bal oldalra való döntésének támogatása.

A támogatott VR szemüvegek listája bármikor bővíthető, hiszen a Unity erre lehetőséget biztosít, erre nekem eszköz hiányában nem volt lehetőségem, de a jövőben könnyen megvalósítható.

A gyakorlatok megosztása eszközök között jelenlegi formájában nem a legegyszerűbb, hiszen egy JSON fájlt kell mozgatni az eszközök között. Ennek könnyítésére fejleszthető a jövőben olyan gyakorlat exportáló funkció, ami egy gyakorlatot valamilyen tömörített formában a vágólapra másol, illetve vágólapról importál.

8. Összefoglalás

Dolgozatomban bemutattam a virtuális valóságot és típusait, a virtuális valóság eszközöket és működésüket, feltártam a virtuális valóság alapú gyógymódok felhasználhatóságát, kitérve a fizioterápiára, és a VR alapú gyógymódok mentális hatására. Elemeztem egy hasonló terméket, ami elérhető a piacon.

A kutatómunka alatt tanultak alapján, a feladat megvalósításához megfelelő hardver és fejlesztő szoftverek kiválasztása után, elkészítettem egy VR alapú nyak tornáztató alkalmazást. A fejlesztést Unity motoron végeztem el, és Android és Windows platformon futtathatóra terveztem az applikációt.

A fejlesztés során komplex térbeli számításokat készítettem, vektorokkal és koordinátákkal, alkalmaztam tanulmányaim alatt, és azon kívül tanult módszereket. Készítettem megoldást összetett objektumok fájlba mentéséhez, és fájlból beolvasásához. Számomra új algoritmusokkal és módszerekkel ismerkedtem meg, amiket alkalmaztam a feladat megoldásában, például Ray casting, vagy Unity szkriptelés. Optimalizáltam a programomat, hogy a Unity erőforrásigényével is könnyen tudja egy Androidos telefon futtatni.

Az applikációm használható nyak tornáztató gyakorlatok létrehozására és nyak tornáztatására VR környezetben, egy könnyen hordozható VR szemüveggel, elősegítve a személyi egészségét a felhasználóknak.

Irodalomjegyzék

- [1] Zeljka Mihajlovic, Sinisa Popovic, Karla Brkic és Kresimir Cosic, „A system for head-neck rehabilitation exercises based,” *Multimedia Tools and Applications*, 2018.
- [2] Dong Woo Shin, Jae Il Shin, Ai Koyanagi, Louis Jacob, Lee Smith, Heajung Lee, Yoonkyung Chang és Tae-Jin Song, „Global, regional, and national neck pain burden in the general population, 1990–2019: An analysis of the global burden of disease study,” 2019.
- [3] „Nintendo Pokémon Go Plus,” Nintendo, [Online]. Available: <https://www.nintendo.hu/pokemon-go-plus/>. [Hozzáférés dátuma: 4 May 2023].
- [4] „Nintendo Switch Pokéball Plus,” Nintendo, [Online]. Available: <https://www.nintendo.hu/switch-poke-ball-plus/>. [Hozzáférés dátuma: 4 May 2023].
- [5] „Mixed Reality Use Cases and Challenges in 2022,” Rinf Tech, 2022. [Online]. Available: <https://www.rinf.tech/mixed-reality-use-cases-and-challenges-in-2022/>. [Hozzáférés dátuma: 4 May 2023].
- [6] Drummond K.H, Houston T és Irvine T, „The rise and fall and rise of virtual reality,” *Vox Media*, 2014.
- [7] Virtual reality systems, San Diego: Academic Press Limited, 1993.
- [8] „George Washington University Hospital,” George Washington University, [Online]. Available: <https://www.gwhospital.com/conditions-services/surgery/precision-virtual-reality>. [Hozzáférés dátuma: 4 May 2023].
- [9] G. B. MD, „How Virtual Reality Can Help Train Surgeons,” Harvard Business Review, 2019. [Online]. Available: <https://hbr.org/2019/10/research-how-virtual-reality-can-help-train-surgeons>.
- [10] „UConn Health is training orthopaedic surgery residents using VR solutions from PrecisionOS™ and Oculus,” UConn Health, 18 March 2020. [Online]. Available:

-
- <https://business.oculus.com/case-studies/uconn-health/>. [Hozzáférés dátuma: 4 May 2023].
- [11] „VR and the Future of Healthcare,” Cedars Sinai, 1 Sep 2020. [Online]. Available: <https://www.cedars-sinai.org/blog/virtual-reality-future-healthcare.html>. [Hozzáférés dátuma: 4 May 2023].
- [12] Elizabeth Dyer, Barbara J Swartzlander és Marilyn R Gugliucci, „Using virtual reality in medical education to teach empathy,” *National Library of Medicine*, 2018.
- [13] L. Lagnado, „Enlisting Virtual Reality to Ease Real Pain,” *The Wall Street Journal*, 24 Jul 2017.
- [14] Melissa S Wong, Brennan M R Spiegel és Kimberly D Gregory, „Virtual Reality Reduces Pain in Laboring Women: A Randomized Controlled Trial,” 2020.
- [15] Naseem Ahmadpour, Hayden Randall, Harsham Choksi, Antony Gao, Christopher Vaughan és Philip Poronnik, „Virtual Reality interventions for acute and chronic pain management,” 2019.
- [16] L. Minor, „For Children in the Hospital, VR May Be the Cure for Anxiety,” *The Wall Street Journal*, 2018.
- [17] H. Hoffman, „Virtual Reality Pain Reduction,” University of Washington Seattle and U.W. Harborview Burn Center, 2008. [Online]. Available: <http://www.hitl.washington.edu/projects/vrpain/>.
- [18] A. Tugend, „Meet Virtual Reality, Your New Physical Therapist,” *The New York Times*, 21 Apr 2021. [Online]. Available: <https://www.nytimes.com/2021/04/21/health/virtual-reality-therapy.html>.
- [19] Burcu Metin Ökmen, Meryem Doğan Aslan, Güldal Funda Nakipoğlu Yüzer és Neşe Özgirgin, „Effect of virtual reality therapy on functional development in children with cerebral palsy: A single-blind, prospective, randomized-controlled study,” *PubMed Central*, 2019.
- [20] „MyndVR,” [Online]. Available: <https://www.myndvr.com>. [Hozzáférés dátuma: 4 May 2023].
- [21] „Rendever,” [Online]. Available: <https://www.rendever.com>. [Hozzáférés dátuma: 4 May 2023].

-
- [22] Shizhe Zhu, Youxin Sui, Ying Shen, Yi Zhu, Nawab Ali, Chuan Guo és Tong Wang, „Effects of Virtual Reality Intervention on Cognition and Motor Function in Older Adults With Mild Cognitive Impairment or Dementia: A Systematic Review and Meta-Analysis,” *PubMed Central*, 2021.
- [23] „Virtual reality technology could strengthen effects of traditional rehabilitation for multiple sclerosis,” *Science Daily*, 6 July 2022. [Online]. Available: <https://www.sciencedaily.com/releases/2022/06/220630142201.htm>. [Hozzáférés dátuma: 4 May 2023].
- [24] Irene Cortés-Pérez, Francisco Antonio Nieto-Escamez és Esteban Obrero-Gaitán, „Immersive Virtual Reality in Stroke Patients as a New Approach for Reducing Postural Disabilities and Falls Risk: A Case Series,” *Brain Sci*, 2020.
- [25] C. Metz, „A New Way for Therapists to Get Inside Heads: Virtual Reality,” *The New Your Times*, 2017.
- [26] Prof Daniel Freeman, Polly Haselton, Jason Freeman, Bernhard Spanlang és Sameer Kishore, „Automated psychological therapy using immersive virtual reality for treatment of fear of heights: a single-blind, parallel-group, randomised controlled trial,” *The Lancet*, %1. kötet5, %1. szám8, pp. 625-632, 2018.
- [27] „What we treat,” XRHealth, [Online]. Available: <https://www.xr.health/what-we-treat/>. [Hozzáférés dátuma: 4 May 2023].
- [28] „VRPhysio Home N-140 Rotate ID Information,” XR HEALTH IL LTD, [Online]. Available: <https://accessgudid.nlm.nih.gov/devices/07290016986099>. [Hozzáférés dátuma: 4 May 2023].
- [29] L. Kelion, „Microsoft HoloLens 2 augmented reality headset unveiled,” *BBC*.
- [30] Brenda Kay Wiederhold, Kenneth Gao, Camelia Sulea és Mark D. Wiederhold, „Virtual Reality as a Distraction Technique in Chronic Pain Patients,” *ResearchGate*, 2014.
- [31] „Five Ways VR Is Being Used In Modern Healthcare,” [Online]. Available: <https://vrscout.com/news/five-ways-vr-is-being-used-in-modern-healthcare/>. [Hozzáférés dátuma: 4 May 2023].

-
- [32] „Git: Reference Sheet,” New Zealand eScience Infrastructure, [Online]. Available: <https://support.nesi.org.nz/hc/en-gb/articles/360001508515-Git-Reference-Sheet>. [Hozzáfézés dátuma: 5 May 2023].
- [33] „JSON.org,” [Online]. Available: <https://www.json.org/json-en.html>. [Hozzáfézés dátuma: 4 May 2023].
- [34] „Verifying your setup,” Vive, [Online]. Available: https://www.vive.com/eu/support/cosmos-external-tracking-faceplate/category_howto/verifying-your-setup.html. [Hozzáfézés dátuma: 4 May 2023].
- [35] „What platforms are supported by Unity?,” Unity Technologies, [Online]. Available: <https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity->. [Hozzáfézés dátuma: 5 May 2023].
- [36] „XRHealth youtube csatorna,” XRHealth, [Online]. Available: <https://www.youtube.com/@xrhealth2836>. [Hozzáfézés dátuma: 4 May 2023].

Mellékletek

Mappaszerkezet

```
+pb8jv3_szakdolgozat
|   .gitattributes
|   .gitignore
|
+---Assets
|   |
|   +---Exercises
|   |
|   +---Materials
|   |     Controller.mat
|   |     Grid.mat
|   |     Debug_sphere_material.mat
|   |     sphgere_material.mat
|   |     Skybox_Light.mat
|   |
|   +---Scenes
|   |     Level.unity
|   |
|   +---Plugins
|   |
|   +---Prefabs
|   |     Debug.prefab
|   |     DebugSphere.prefab
|   |     ExerciseNumberBox.prefab
|   |     ExitDebugMode.prefab
|   |     Menu.prefab
|   |     Sphere.prefab
|   |
|   +---Scripts
|   |   |
|   |   +---Controllers
|   |   |     CamControlPc.cs
|   |   |     MainController.cs
|   |   |
|   |   +---Data
|   |   |     Border.cs
|   |   |     ExerciseDictionary.cs
|   |   |     ObjectCoordinates.cs
|   |   |     PlayerCamRotation.cs
|   |   |     Raycast.cs
|   |   |
|   |   +---Enums
|   |   |     BorderFlag.cs
|   |   |     GameState.cs
|   |   |
|   |   +---GameModes
|   |   |     EditMode.cs
|   |   |     ExerciseChooser.cs
|   |   |     GameOver.cs
|   |   |     NormalGame.cs
|   |   |
|   |   \---Managers
```

```
| | |
| | |GameManager.cs
| | |UIManager.cs
| | | \---SubManagers
| | |     BorderHelper.cs
| | |     DebugManager.cs
| | |     FileHandler.cs
| | |
| | +---TextMesh Pro
| | |
| | +---Textures
| | |     Grid_Light_512x512.png
| | |
| | +---XR
| | |
| | \---XRI
|
+---Documents
|     ClassHierarchy.drawio
|
+---Library
|
+---Logs
|
+---obj
|
+---Packages
|
+---ProjectSettings
|
\---UserSettings
```

Ábrajegyzék

1. ábra: PokémonGO AR játék	10
2. ábra: Microsoft HoloLens használata műtéthez való felkészüléshez [6].....	11
3. ábra: Bázis állomások elhelyezése [36]	12
4. ábra: Orvoshallgató VR környezetben gyakorol [36].....	13
5. ábra: Fájdalomérzet VR kezeléssel és nélkül [32].....	14
6. ábra: Tériszony kezelés VR megoldással és kontrol csoport összehasonlítása. 0, 2, és 4 hét után [28]	15
7. ábra: XRHealth Rotate használat közben [36].....	16
8. ábra: Gyakorlat menete	17
9. ábra: Gyakorlat készítés menete	18
10. ábra: Samsung GearVR szemüveg és okostelefon	19
11. ábra: Unity által támogatott platformok [36].....	20
12. ábra: A szakdolgozat feladat megoldása közben létrehozott változtatások (commit) GitHub online felületen	21
13. ábra: Módosítások kezelése GIT tárolóval [36].....	22
14. ábra: JetBrains Rider beágyazott Unity irányítópult.....	23
15. ábra: Android azonosító és API szint beállítások	24
16. ábra: A szakdolgozat feladat megvalósításához használt Unity csomagok.....	25
17. ábra: Unity virtuális térben elhelyezhető játék objektumok	26
18. ábra: A virtuális tér indítás előtt üres, mivel majdnem minden játék objektumot kódból hozunk létre.....	27
19. ábra: A statikus játékobjektumok hierarchiája.....	28
20. ábra: A felhasználói interfészt képző Canvas objektum és elemei, középen a célkereszttel	29
21. ábra: A virtuális térben lévő Camera objektum beállításai és komponensei	30
22. ábra: A PlayerObj objektum komponensei	31
23. ábra: Menü prefabból létrehozott játék objektumok és hierarchiájuk	32
24. ábra: Gyakorlat választó menü játék objektumai és hierarchiája.....	33
25. ábra: Ray casting segítségével állapítjuk meg, mire néz a felhasználó	34
26. ábra: Ray casting implementálása a Menü navigálásához.....	35
27. ábra: JSON felépítése [36]	36

28. ábra: JSON érték felépítése [36].....	37
29. ábra: Vektor lista JSON formátumba szerializáló függvény implementációja.....	38
30. ábra: JSON deszerializálása vektor listává LINQ segítségével	39
31. ábra: A játéktér határának megállapítása a felhasználótól balra	40
32. ábra: Bal határ térbeli pozíciójának kiszámolása a kamera forgásszögéből	41
33. ábra: A felhasználótól balra lévő pozíciók igazítása a játéktérhez	42
34. ábra: A játéktérre átszámolt pozíciók esztétikai módosítása, így minden pozíció 60 egységes helyezkedik el a felhasználótól.....	43
35. ábra: Singleton implementáció	44
36. ábra: Kamera forgását számoló függvény	45
37. ábra: A Camera objektumhoz tartozó irányvektor, forgásszög, és pozíció lekérését segítő Property-k.....	46
38. ábra: Függvény a gyakorlatok betöltéséhez a program indulásakor.....	48
39. ábra: Gyakorlatválasztó menü gombokat megjelenítő függvény	49
40. ábra: Gyakorlat betöltéséért és menetéért felelő függvény.....	50