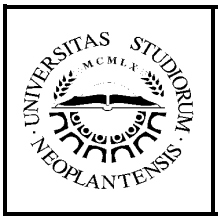


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Предраг Петковић**
Број индекса: **РА 19/2019**

Тема рада: **Једно решење синхронизације у репродукцији видео садржаја са применом у виртуелној реалности**

Ментор рада: **Проф. др Илија Башичевић**

Нови Сад, јул, 2023

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		**Предраг Петковић**
Ментор, **МН**:		**проф. др Илија Башичевић**
Наслов рада, **НР**:		**Једно решење синхронизације у репродукцији видео садржаја са применом у виртуелној реалности**
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2023**
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		**7/50/0/15/9/0/0**
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кључне речи, **ПО**:		**Синхронизација репродукције видео садржаја, виртуелна реалност, платформа за развој видео игара, игре са више играча**
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		Када је потребно синхронизовати репродукцију видео садржаја између више корисника, једно од решења је синхронизација њихових временских печата. Решење је реализовано као видео игра са више играча у виртуелној реалности. Главни клијент, тј. мастер, шаље податке о свом временском печату свим осталим клијентима. Остали клијенти синхронизују свој временски печат у односу на временски печат који су добили од главног клијента, гледајући разлику између та два временска печата. Дат је опсег у ком би требала да се налази разлика временских печата који представља задовољавајући услов синхронизације репродукције видео садржаја.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		**14.7.2023**
Чланови комисије, **КО**:	Председник:	**проф. др Мирослав Поповић**
	Члан:	**проф. др Иван Каштелан** — Потпис ментора
	Члан, ментор:	**проф. др Илија Башичевић**

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		**Predrag Petković**	
Mentor, **MN**:		**Ilija Bašičević, PhD**	
Title, **TI**:		**Solution for synchronizing video content playback for use in virtual reality**	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2023	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**7/50/0/15/9/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		**Synchronizing video playback, virtual reality, game engine, multiplayer games**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		When it is necessary to synchronize the playback of video content between multiple users, one solution is to synchronize their timestamps. The solution is implemented as a VR multiplayer video game. The main client, i.e., the master, sends data about its timestamp to all other clients. The other clients synchronize their timestamp relative to the timestamp they received from the main client, looking at the difference between the two timestamps. A range is given within which the difference in timestamps should be, which represents a satisfactory condition for synchronizing the playback of video content.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:		**7/11/2023**	
Defended Board, **DB**:	President:	**Miroslav Popović, PhD**	
	Member:	**Ivan Kaštelan, PhD**	Menthor's sign
	Member, Mentor:	**Ilija Bašičević, PhD**	

Захвалност

Захваљујем ментору проф. др Илији Башичевићу, Саши Јагодину и Михајлу Милотићу, као и осталим члановима тима *TVverse* на стручној помоћи и саветима током израде овог рада.

Захваљујем се члановима породице и пријатељима на неизмерној подршци током целокупног студирања.

САДРЖАЈ

СПИСАК СЛИКА

СПИСАК ТАБЕЛА

Скраћенице

API – *Application Programming Interface*

VR – *Virtual Reality*

AR – *Augmented Reality*

XR – *Extended Reality*

MR – *Mixed Reality*

PUN2 – *Photon Unity Networking 2*

AOSP – *Android Open Source Project*

APK – *Android Package*

AAR – *Android Archive*

JAR - *Java Archive*

1. Увод

Будућност којом доминира продужена реалност (енг. *extended reality*) се ближи, стога се може рећи да ће та будућност ускоро постати садашњост. Продужена реалност је широки термин који обухвата виртуелну реалност (енг. *virtual reality*), проширену реалност (енг. *augmented reality*), и мешовиту реалност (енг. *mixed reality*).

Виртуелна реалност је потпуно урођено (енг. *immersive*) искуство где се корисник изолује из стварне околине и поставља у потпуно виртуелну. Корисник у виртуелној реалности може да интерагује са тродимензионалним светом који није физички присутан.

Проширена реалност представља технологију која преклапа дигитални садржај на стварни свет, пружајући додатне информације или побољшавајући стварна искуства. Неки од примера проширене реалности су игра Покемон Го, где се дигитална створења појављују у стварном свету на екрану паметног телефона, као и филтери за лице у апликацијама као што су Снепчет или Инстаграм.

Мешовита реалност је мешавина виртуелне и проширене реалности где стварни и дигитални објекти интерагују. Она производи нова окружења где физички и дигитални објекти коегзистирају у реалном времену.

Виртуелна реалност има широку примену, пре свега у сврхе забаве, са мноштвом различитих врста видео игара које пружају невероватно урођено искуство. Осим забаве, виртуелна реалност је нашла примену и у другим областима, као што су образовање, где виртуелна реалност пружа урођена, практична искуства учења која нису могућа у традиционалним учионицама,

здравство, где се виртуелна реалност користи за многе ствари, од хируршке обуке до управљања болом и терапије, па чак и туризам. Еволуција VR, и уопштено XR технологије, покренута је напретком у снази рачунара и рачунарској графици, као и физичким смањивањем сензора, те су VR искуства постала јефтинија и приступачнија. Паралелно с тим, пораст паметних телефона и мобилног рачунарства је играо значајну улогу, јер су такви уређаји опремљени снажним процесорима и вишеструким сензорима, омогућавајући покретање XR апликација на уређајима које поседују милијарде људи. Осим тога, глобална пандемија коронавируса је деловала као катализатор, убрзавајући развој и усвајање XR технологија. Како су људи тражили нове начине за рад, учење и повезивање док су физички изоловани, XR се појавио као алат који пружа средства за истраживање и интеракцију у свету социјалне дистанце.

Гледање телевизије је свакодневница великог броја људи. Било то гледање омиљене серије или филма, емисије едукативног садржаја, или пак неке спортске утакмице са друштвом, телевизија чини велики удео у животима људи. У последњих неколико године су све популарнији стриминг сервиси (енг. *streaming service*), од којих су најпопуларнији Нетфликс, Амазон Прајм Видео итд. Гледање телевизијског, стриминг или видео садржаја у VR-у има за идеју приказивање тог садржаја у виртуелном тродимензионалном простору и смањивање употребе телевизора, што доводи до смањења употребе пластике и струје, а самим тим и смањења загађења животне средине. Корисници могу организовати тзв. *watch party*, где могу са својим пријатељима гледати одабрани садржај без напуштања удобности свог дома.

За реализацију програмског решења је коришћена *Unity3D* платформа за развој видео игара (енг. *game engine*), спрегнут са Андроид библиотеком, које ја увезана као додатак у *Unity3D*. Ова библиотека служи за манипулацију видео плејером. *Photon Unity Networking 2 (PUN2)* је *Unity3D* додатак (енг. *plugin*) коришћен за реализовање функционалности игре са више играча, као и за синхронизацију података између уређаја. Програмски језици коришћени за реализацију програмског решења су *C#* и *Java.* Платформа на којој је тестирано решење је *Meta Quest 2.*

2. Теоријске основе

У овом поглављу су објашњене технологије које су у употреби за реализацију програмског решења.

2.1 Платформа за развој видео игара

Платформа за развој видео игара је софтвер дизајниран да олакша развој и стварање видео игара. Инжењери га користе за стварање видео игара за конзоле, мобилне уређаје и персоналне рачунаре. Основне функционалности које обично пружа платформа за развој видео игара укључују рендеровање за дводимензионалну или тродимензионалну графику, *physics engine* или детекцију колизије (и реакцију на колизију), звук, скриптовање, анимацију, вештачку интелигенцију, мрежу, управљање меморијом, нити и сцене.

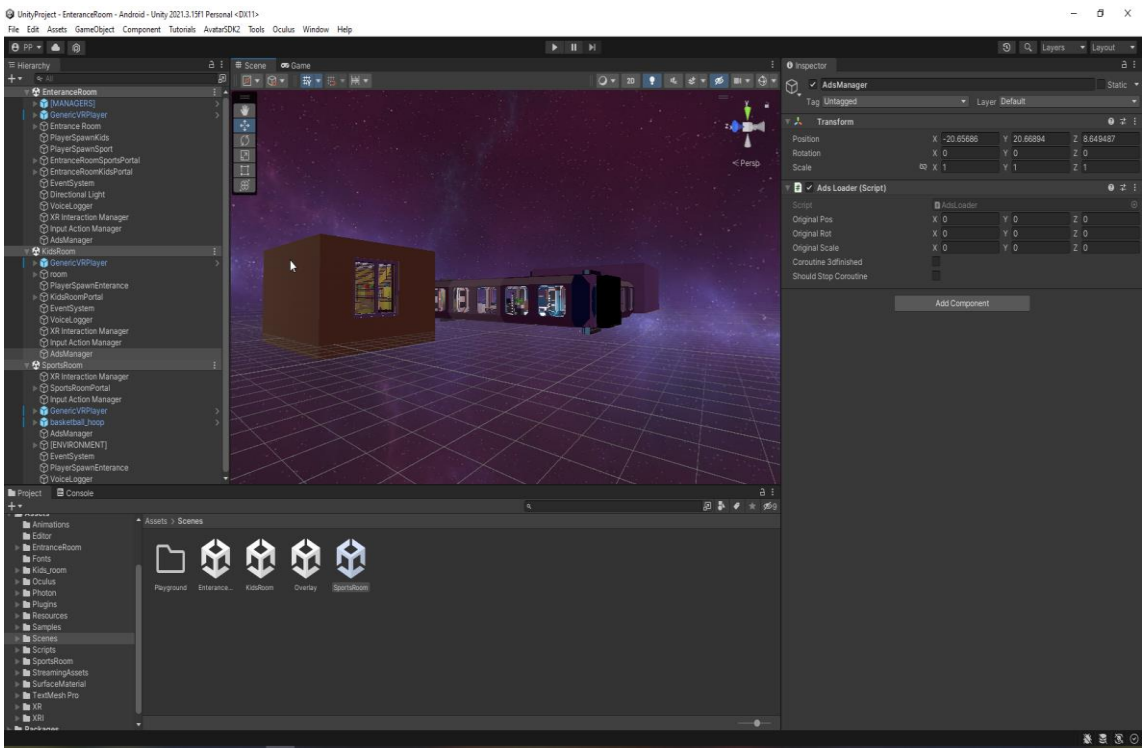
2.1.1 Unity3D платформа за развој видео игара

Unity3D је популарна платформа за развој видео игара који је развио *Unity Technologies.* Објављен је у јуну 2005. године на конференцији компаније Епл. Првобитно је направљен само за *Mac OS X* оперативни систем, док је данас *Unity3D* проширен да подржава више од двадесет пет платформи и може се користити за креирање дводимензионалних и тродимензионалних игара, као и интерактивних симулација и других искустава. *Unity3D* имплементира неке од горенаведених ствари на следећи начин:

- рендеровање: *Unity3D* подржава рендеровање дводимензионалне и тродимензионалне графике. Укључује различите функционалности за стварање богатих и импресивних визуализација, као што су ефекти честица, шејдери (енг. *shaders*), пост-процесних ефеката итд. За тродимензионалну графику, *Unity3D* подржава *Direct3D,*

OpenGL и *Vulkan*, а такође омогућава инжењерима да праве своје шејдере, и тако им дају могућност да креирају своје јединствене визуелне ефекте.

- *physics engine: Unity3D* користи Енвидијину *PhysX* платформу за симулацију физике у својим играма. Пружа подршку за динамику крутих тела (енг. *rigid body*), детекцију колизије, контролере карактера итд.
- звук: *Unity3D* долази са уграђеним аудио системом који омогућава програмерима да додају звучне ефекте, позадинску музику и позајмљивање гласа у своје игре.
- скриптовање: *Unity3D* користи језик по имену *C#* за развој игара. *Unity3D* API пружа приступ широком спектру функционалности, укључујући манипулацију објеката у сцени, обраду корисничког уноса, вештачку интелигенцију, физику итд.
- управљање меморијом: *Unity3D* пружа аутоматско управљање меморијом, што значи да инжењери не морају ручно алоцирају и деалоцирају меморију. Такође укључује подршку за вишенитни рад.
- сцене: *Unity3D* користи сценски систем за управљање светом игре. Сваки објекат у *Unity3D*-у је чвор (енг. *node*) у сцени, а односи између објеката су представљени односом родитељ-дете.



Слика 1 – Изглед *Unity3D* едитора (енг. *editor*)

2.1.2 Photon Unity Networking 2 (PUN2)

PUN2 је један од водећих мрежних решења за *Unity3D*, који је развио *Exit Games*. То је пакет који пружа *API* за прављење игара са више играча (енг. *multiplayer games*), као и једноставност коришћена одмах након инсталације. Дизајниран је да се лако интегрише са *Unity3D*-ом, јер омогућава коришћење *Unity3D* уграђених функционалности и конвенција. На пример, *PUN2* користи *Unity3D* серијализацију за комуникацију путем мреже, што олакшава слање изведених типова података. *PUN2* користи концепт „соба“ за своје сесије игара са више играча. Када се играчи повежу, придружују се једној соби. Свака соба је изолована од осталих, па догађаји у једној соби не утичу на било коју другу собу. *PUN2* користи клијент-сервер архитектуру, где један од клијената делује као главни клијент (онај који је први ушао у собу, мастер). Једино мастер може да шаље податке осталим клијентима, док остали клијенти могу само да примају податке. *PUN2* је направљен да се прилагоди и носи са великим бројем играча. Користи *Photon Cloud* сервис за хостинг, који се аутоматски скалира према потреби како би се носио са наглим порастом саобраћаја и великим бројем играча. *PUN2* подржава широк спектар платформи. Ово укључује рачунаре, конзоле, мобилне уређаје, па чак и веб прегледаче.

PUN2 пружа функционалност позивања удаљене процедуре (енг. *Remote Procedure Call - RPC*). Ова функционалност омогућава позивање функција на удаљеним објектима и синхронизацију промене стања игре између клијената, олакшавајући интеракције и синхронизацију у играма са више играча.

2.2 Виртуелна реалност

Виртуелна реалност је симулирано искуство које може бити слично или комплетно другачије од стварног света. То је рачунарски генерисано окружење које може да симулира физичко присуство у стварном или замишљеном свету. Корисник може интерактивно да се повеже са овим окружењем на начин који делује реално или физички путем посебне хардверске опреме као што су хедсет (енг. *headset*) и рукавице/контролери опремљени сензорима. Системи виртуелне реалности раде тако што прате физичке покрете корисника и претварају те покрете у виртуелно окружење. Корисник носи хедсет који има један или два екрана и стерео звук. Екрани приказују тродимензионалну слику која се може прилагодити у реалном времену док корисник помера главу. Хедсет и уређају који служе за праћење положаја руку корисника садрже сензоре који прате покрете и оријентацију корисника. Уобичајене врсте сензора укључују акцелерометре, жироскопе итд. *VR* систем користи ове податке да прилагоди поглед корисника, а некад чак и његов положај у виртуелном окружењу. Корисник може на разне начине да интерагује са виртуелним окружењем. Неки од начина су коришћење гласовних команди, ручни контролери са дугмадима или додирним плочама, као и праћење покрета руку и прстију (енг. *handtracking*). Напреднији системи могу користити рукавице или комплетна одела са сензорима за прецизнију и интуитивнију контролу.

Слика 2 - *VR* одело опремљено сензорима

2.3 OpenXR

OpenXR је отворени стандард за приступ платформама и уређајима виртуелне и проширене реалности. Развио га је *Khronos Group*. *OpenXR* има за циљ да поједностави развој *VR* и *AR* апликација, пружајући заједнички *API* који је компатибилан са широким спектром хардверских уређаја. Састоји се из два дела, слоја уређаја и слоја апликације. Слој уређаја је задужен за интеракције са хардвером, као што су улазни и излазни уређаји. То значи да се бави специфичностима уређаја који се користи, попут система за праћење, контролера и екрана. Слој апликације је место где се пише код. Пружа стандардни сет команди и функција које се могу користити за креирање *VR/AR* искустава, без обзира на основни хардвер. Најзначајнија употреба *OpenXR*-а је креирање *VR* и *AR* апликација које могу да се покрећу на више платформи са минималним променама у коду. Као отворени стандард, *OpenXR* је дизајниран да подржава будуће *VR* и *AR* технологије.

2.4 Видео плејер

Видео плејер (енг. *video player*), такође познат и као медија плејер, је врста софтвера или хардвера који репродукује мултимедијални садржај, укључујући видео записе, аудио датотеке, а понекад и слике. Видео плејери обрађују и декодирају дигиталне датотеке или токове и претварају их у аудио и видео сигнале који се могу приказати на екрану или репродуковати путем звучника. Ови фајлови су обично компримирани ради смањења њихове величине за

складиштење и пренос, а задатак видео плејера је да декомпресује ове податке и претвори их назад у секвенцу слика и звука која се може репродуковати у реалном времену. Већина видео плејера подржава широк спектар видео формата, али неки могу бити компатибилни само са одређеним форматима. Поред тога, многи видео плејери такође укључују функционалности за контролу репродукције, попут паузирања, премотавања, брзог премотавања унапред или уназад, подешавања јачине звука итд. Неки видео плејери подржавају стримовање видео садржаја, где такви плејери често укључују додатне функционалности за руковање променама у мрежним условима, попут баферовања или адаптивног стримовања са променљивом брзином преноса. Видео плејер коришћен у овом решењу је написан у програмским језицима *C++* и *Java*.

2.5 Андроид

Андроид је оперативни систем који је примарно намењен уређајима на додир, мобилним уређајима и таблетима. Једна од кључних карактеристика Андроида је отворен код. Извор Андроида је доступан јавности и омогућава развијање и прилагођавање оперативног система од стране различитих произвођача. Ово је довело до великог броја уређаја који користе Андроид као свој оперативни систем. Откад је Гугл преузео развијање Андроид оперативног система, могуће га је пронаћи и у телевизорима, аутомобилима, па чак и у ручним сатовима.

2.5.1 Андроид библиотека

Андроид библиотека је структурно иста као и Андроид модул апликације. Она укључује све што је потребно за изградњу апликације, укључујући изворни код, датотеке ресурса, као и Андроид манифест датотеку.

Уместо да се компајлира у *APK* (енг. *Android Package*) датотеку која се извршава на уређају, Андроид библиотека се компајлира у *AAR* (енг. *Android Archive*) датотеку која се може користити као зависност за Андроид модуле апликације. *AAR* датотеке могу садржати Андроид ресурсе и манифест датотеку, што омогућава укључивање дељених ресурса, поред *Kotlin* или *Java* класа и метода. Такође се могу додавати зависности у Андроид библиотеку. Као зависности су додати *AAR* датотека која служи за коришћене

функционалности видео плејера, као и *classes.jar* датотека, која садржи компајлиран *Java* код.

Како би се користиле функционалност Андроид библиотеке користила у *Unity3D*-у, потребно је поставити *AAR* датотеку, као и све остале *AAR* и *JAR* датотеке које су укључене као зависности, на путању *Assets/Plugins/Android.* Направљена је нова прилагођена *AndroidManifest.xml* датотека, коју је такође потребно поставити на исту путању као и *AAR* датотеке.

2.6 Meta Quest 2

Meta Quest 2 је *VR* хедсет који је развио *Reality Labs,* одељак Фејзбука, сада познатијег као Мета. Представљен је 16. септембра 2020., а пуштен у продају 13. октобра те године као *Oculus Quest 2*. Преименован је 2022. у *Meta Quest 2* заједно са целом компанијом Фејзбук.

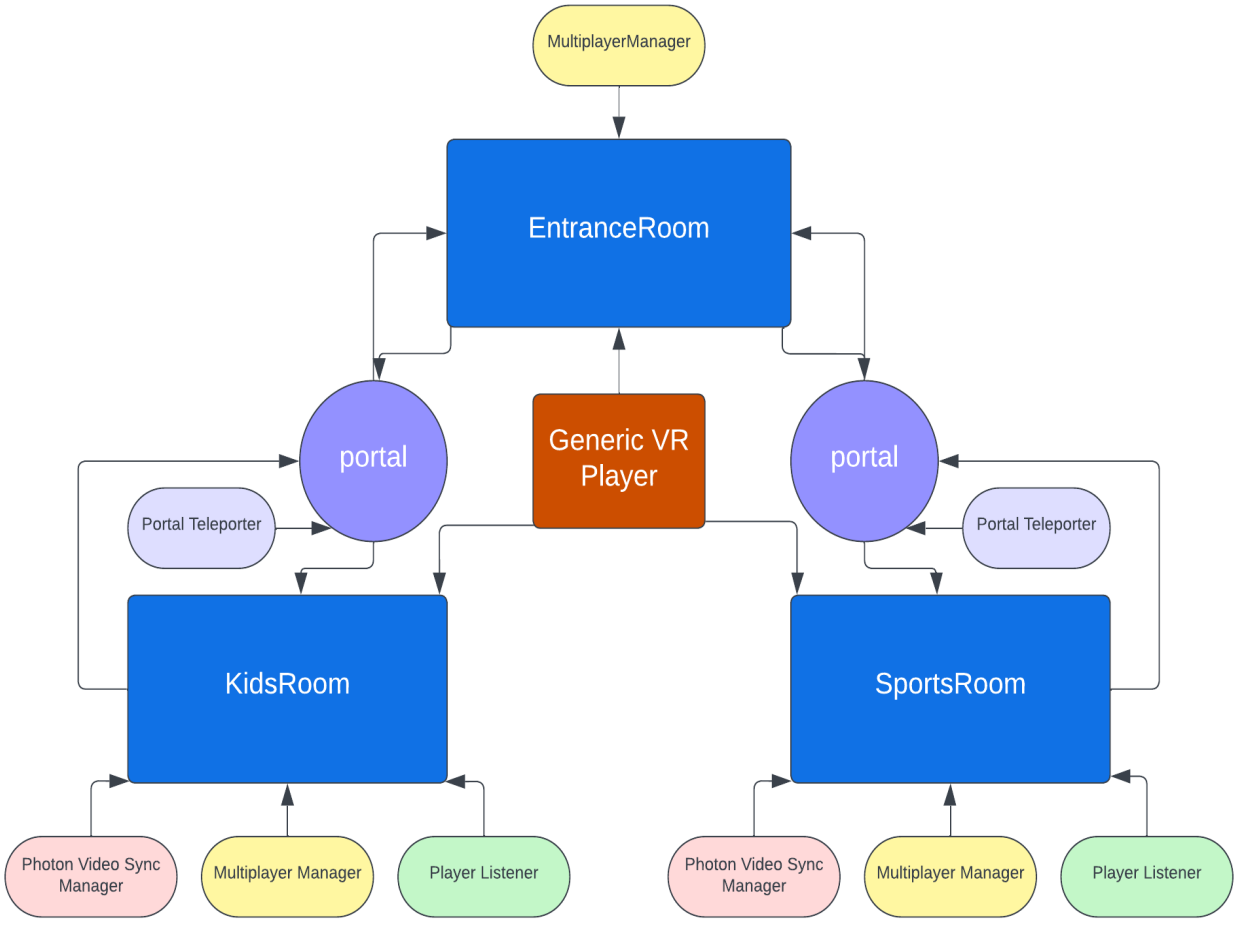
Quest 2 је дизајном слична, али побољшана верзија оригиналног *Oculus Quest*-а. *Quest 2* је лакши, са белом пластичном спољашности уместо црне, обложене тканином. Тежак је 503 грама у поређењу са 571 грамом оригиналног *Quest*-а. Такође има побољшани дисплеј са већом стопом освежавања (енг. *refresh rate*) и резолуцијом по оку, побољшане *Oculus Touch* контролере са дужим трајањем батерије и побољшане спецификације.

Quest 2 користи оперативни систем заснован на *Android Open Source Project*-у *(AOSP).*

Слика 3 - *Meta Quest 2* хедсет и контролери

3. Концепт решења

У овом поглављу је описана архитектура *Unity3D* апликације, као и опис самог програмског решења. *Unity3D* апликација је организован у три сцене, *EntranceRoom*, *KidsRoom* и *SportsRoom*.



Слика 4 – Архитектура *Unity* апликације

Апликација је организована у три сцене. Прва сцена која се учита након покретања апликације је *EntranceRoom*, која представља почетну сцену у којој корисник може да се креће и интерагује са два портала. Када корисник додирне портал, он ће се „телепортовати“, тј. учитаће се друга сцена, *KidsRoom* или *SportsRoom*, у зависности од портала који је додирнуо корисник.

KidsRoom је сцена са дечијим садржајем, у којој може да се креативно интерагује са објектима у сцени, и на такав интерактиван начин да се репродукује видео садржај. Ова сцена садржи један „телевизор“, објекат *TextureV*, који служи као површина за приказивање видео садржаја.

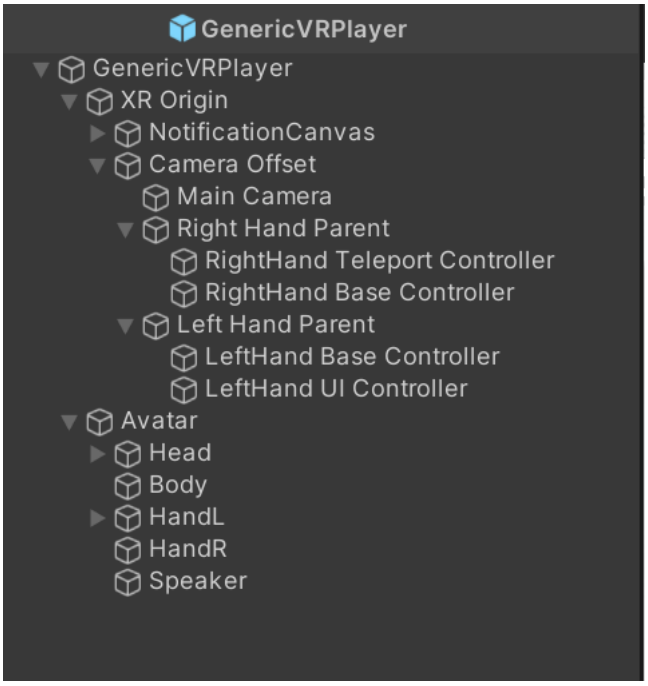
SportsRoom је сцена која је у духу спортског бара, која садржи седам „телевизора“, односно *TextureV* објеката. Поред тих телевизора се приказује статистика тренутног спортског садржаја у реалном времену.

3.1 Синхронизација преко мреже

Синхронизација података и акција преко мреже се постиже путе *PhotonView C#* скрипте коју пружа *Unity3D* пакет *PUN2.* Када се ова скрипта закачи за објекат у сцени, тај објекат постаје свестан мреже. То значи да се стање објекта може синхронизовати на различитим инстанцама апликације покренутим на различитим уређајима. *PhotonView* идентификује објекат преко мреже користећи *ViewID* и конфигурише начин на који се ажурирају удаљене инстанце тог објекта.

3.2 Корисник у апликацији

Корисник је представљен *GenericVRPlayer* шаблоном (енг. *prefab*). Шаблон у *Unity3D*-у представља елемент поставке (енг. *asset*) који може да конфигурише објекат са свим својим компонентама и објектима потомцима, као шаблон који може поново да се искористи. Скрипте накачене на овај шаблон су *PhotonView, PlayerNetworkSetup* и *MultiplayerVRSynchronization*. Потомци овог објекта су *XR Origin,* на који је накачена *AvatarInputConverter* скрипта, као и скрипте које служе за кретање и окретање у апликацији, и *Avatar* објекат, који служи за визуелизацију корисника. *Avatar* објекат се састоји од главе, тела, леве и десне руке.



Слика 5 - *GenericVRPlayer* шаблон

Слика 6 – Изглед *Avatar* објекта

3.2.1 XR Origin

XR Origin је родитељски објекат који служи као референтна тачка за сво праћење везано за *XR*. *XR Origin* садржи објекат који представља камеру, као и објекте који представљају руке. Камера представља корисников поглед или хедсет у виртуелном свету. Позиционира се и ротира на основу покрета корисникове главе у стварном простору. Објекти који представљају руке или ручне контролере у виртуелном свету, позиционирају се и ротирају на основу

покрета корисникових руку у стварном простору. У нашем случају, лева рука служи за кретање корисника по простору, док десна служи за окретање, тј. ротирање, као и телепортовање на одређену локацију. Ови објекти садрже скрипту по имену *TrackedPoseDriver,* која управља позицијом и ротацијом објекта на који је закачена, и то на основу података о праћењу са *XR* уређаја.

3.2.1.1 Синхронизација Avatar објекта

Синхронизација *Avatar* објекта се постиже *AvatarInputConverter C#* скриптом, која синхронизује позицију и ротацију главе аватара са објектом камере, тело аватара се синхронизује са главом аватара, док се објекти руку аватара синхронизују са објектима руку *XR Origin*-а. Ово је одрађено како би се сви делови аватара кретали као један.

3.2.2 Проблем појављивања више XR Origin-а у истој сцени

Овај проблем се манифестује кад више корисника уђе у исту *Photon* собу. Овај проблем може узроковати још неколико проблема, где би један од њих био постојање више камера у сцени, и тада је нејасно која камера би била задужена за рендеровање сцене, што може довести до неочекиваног понашања.

За решавање овог проблема је задужена *PlayerNetworkSetup C#* скрипта, тако што код локалног корисника за удаљене *XR Origin* објекат постави да буде неактиван, где онда остаје само удаљени *Avatar* објекат који рендерује локална камера.

3.2.3 Синхронизација корисника

Синхронизација корисника се постиже *MultiplayerVRSynchronization C#* скриптом, која синхронизује позицију и ротацију *GenericVRPlayer* шаблона, као и његових компоненти између корисника, укључујући и *Avatar* објекат. Ова функционалност служи како би корисници видели аватаре других корисника у реалном времену, пружајући урођено искуство као да су стварно једни поред других.

3.3 Комуникација са Photon серверима

За комуникацију са *Photon* серверима је задужена *MultiplayerManager C#* скрипта, као и једним делом *PortalTeleporter C#* скрипта.

Када корисник први пут укључи апликацију, појавиће се у *EntranceRoom*, где ће се повезати на сервер и ући у *Photon* лоби. *EntranceRoom* је једина сцена

где ће се корисник налазити у лобију, док ће у осталим сценама корисник бити у *Photon* соби. Лоби представља „област за чекање“, где корисници могу да пронађу постојеће собе или креирају нове собе. Корисници углавном не могу да комуницирају у лобију, чак и кад се налазе у истој инстанци. Комуникација се обавља у собама, које суштински представљају сесију игре са више играча. Кад корисник креира собу, остали корисници могу да се придруже тој соби.

Када корисник додирне портал, део *C#* скрипте *PortalTeleporter* је задужен за учитавање нове сцене, у зависности који портал је дотакнут. Ово означава „телепортовање“ у другу собу. У зависности да ли је корисник у лобију или соби, након што пређе у нову сцену, он ће изаћи из собе или лобија. Ово ће узроковати раскидање везе са сервером, где ће корисник, прелазећи у нову сцену био поново повезан на сервер. Ако корисник не улази први пут у апликацију, чекаће се док он не раскине везу са сервером како би се поново повезао. Ово се ради како би се инстанцирао *GenericVRPlayer* шаблон, јер ако би шаблон већ постојао у осталим сценама као у *EntranceRoom*, сваки нови корисник који уђе у собу би покушао приступити истом *XR Origin*-у.

Након што се поново повезао, ако сцена није *EntranceRoom*, корисник ће направити нову собу уколико она не постоји, а уколико постоји придружиће јој се. Онда се инстанцира *GenericVRPlayer* на прослеђеној позицији са прослеђеном ротацијом.

3.4 Манипулација плејером

За манипулацију плејером је задужена *PlayerListener C#* скрипта, која је закачена за *TextureV* објекат, који представља „телевизор“. Поред многих функционалности које пружа *PlayerListener*, за ово решење су најбитније: комуникација са Андроид делом, иницијализација плејера, стартовање плејера (са и без помераја), паузирање плејера, настављање репродукције плејера, стопирање плејера, добављање тренутног временског печата (енг. *timestamp*) видеа и премотавање плејера (енг. *seek*). Иницијализација, стартовање, паузирање, настављање репродукције и стопирање плејера се позивају синхронизовано код свих корисника који се тренутно налазе у *Photon* соби.

Када се стартује плејер, он се додаје у листу активних плејера (може их бити више у сцену) ако није већ додат.

3.5 Синхронизација репродукције видео садржаја

За синхронизацију репродукције видео садржаја је задужена *PhotonVideoSyncManager C#* скрипта. Главни клијент, онај који је први шаље своје податке, међу којима су временски печати његових плејера, свим осталим клијентима. Клијенти пореде своје временске печате са добијеним, и ако разлика није задовољавајућа, премотаће своје плејере на одговарајући временски печат.

Ова скрипта, заједно са *PlayerListener* и *MultiplayerManager* скриптама ће детаљно бити објашњене у наредном поглављу.

4. Програмско решење

У овом поглављу је објашњено програмско решење скрипти и делова наведених у претходном поглављу.

4.1 PortalTeleporter

Скрипта садржи енумерациони тип *SCENES* са вредностима *ENTRANCE_ROOM = 0, KIDS_ROOM = 1, SPORTS_ROOM = 2.* Жељена вредност се поставља у *Unity3D* едитору у зависности од тога у коју сцену корисник треба да се телепортује.

У методи *Awake,* која се позива на почетку *Unity3D* апликације, пита се да ли је корисник у лобију или соби, користећи својства (енг. *properties*) скрипте *PhotonNetwork* (део пакета *PUN2*) *InLobby* и *InRoom.* Ако је први услов испуњен, позива се метода *PhotonNetwork* скрипте *LeaveLobby,* а ако је други услов испуњен, онда се позива *LeaveRoom.*

Метода *ScenesToString* враћа одговарајући стринг у зависности која је вредност постављеног типа *SCENES.* Вредност враћеног стринга може бити “EntranceRoom”, “KidsRoom” или “SportsRoom”.

OnTriggerEnter је функција повратног позива (енг. *callback*) метода која се позива кад корисник изврши колизију са порталом (додирне га). У овој методи се позива метода *Teleport,* у којој се позива метода скрипте *PhotonNetwork* по називу *LoadLevel*, која служи за учитавање нове сцене. Њој се као параметар прослеђује име сцене, што је у овом случају повратна вредност методе *ScenesToString.*

4.2 MultiplayerManager

Поље	Опис
public static MultiplayerManager instance	Служи за прављење Синглтон (енг. *Singleton*) шаблона, како би се осигурало постојање само једне инстанце класе у сцени
public Transform spawnTransform	Позиција и ротација GenericVRPlayerPrefab поља кад се инстанцира
private bool sceneEntrance	Провера да ли је тренутна сцена *EntranceRoom*
private GameObject GenericVRPlayerPrefab	Шаблон који се инстанцира након улажења у собу

Табела 1 – Поља класе *MultiplayerManager*

Дефиниција	Опис
public static bool ConnectUsingSettings()	Повезује корисника на *Photon* сервере
public static bool CreateRoom(string roomName, RoomOptions roomOptions = null, TypedLobby typedLobby = null, string[] expectedUsers = null)	Прави нову *Photon* собу са прослеђеним параметрима
public static bool JoinLobby()	Придружује корисника подразумеваном *Photon* лобију
public static bool JoinRandomRoom()	Придружује корисника насумичној *Photon* соби
public static GameObject Instantiate(string prefabName, Vector3 position, Quaternion rotation, byte group = 0, object[] data = null)	Инстанцира шаблон на прослеђену позицију са прослеђеном ротацијом
public static void Disconnect()	Раскида везу корисника и *Photon* сервера

Табела 2 – Методе класе *PhotonNetwork* коришћене у решењу

Дефиниција	Опис
private IEnumerator ConnectToPhotonServers()	Корутина која чека да стање корисника пређе у ClientState.Disconnected, па онда позива ConnectUsingSettings
private void CreateAndJoinRoom()	Поставља име *Photon* собе и максималан број корисника у њој на 20, па позива CreateRoom

Табела 3 – Методе класе *MultiplayerManager*

Назив	Опис
void OnConnected()	Веза је успостављена, али корисник не може да извршава никакве операције на *Photon* серверима
void OnConnectedToMaster()	Корисник је повезан на мастер сервер
void OnJoinedLobby()	Корисник се придружио *Photon* лобију
void OnJoinRandomFailed()	Корисник није успео да се придружи насумичној *Photon* соби. Соба или је пуна или не постоји
void OnCreatedRoom()	Корисник је креирао *Photon* собу и ушао у исту
void OnJoinedRoom()	Корисник је ушао у *Photon* собу, небитно да ли је он направио или се придружио
void OnPlayerEnteredRoom(Player newPlayer)	Удаљени корисник је ушао у *Photon* собу
void OnLeftLobby()	Корисник је напустио *Photon* лоби
void OnLeftRoom()	Корисник је напустио *Photon* собу
void OnDisconnected(DisconnectCause cause)	Корисник је изгубио везу са *Photon* серверима

Табела 4 – Функције повратног позива *MonoBehaviourPunCallbacks* класе коришћене у решењу

У *Start* методи (позива се након *Awake* методе) се прво проверава да ли је први улаз у апликацију, преко поља *firstEntrance* скрипте *MainManager.* Ако је ово случај, позваће се *ConnectUsingSettings,* иначе се покреће корутина *ConnectToPhotonServers*.

Након тога се позива функција повратног позива *OnConnected*, где се само потврђује веза са сервером исписујући лог, потом се позива *OnConnectedToMaster*, где се проверава да ли је тренутна сцена *EntranceRoom.*

Ако јесте, позива се метода *JoinLobby*, након тога се позива функција повратног позива *OnJoinedLobby.* Ако није, позива се метода *JoinRandomRoom*, која ће позвати функцију повратног позива *OnJoinedRoom* ако соба већ постоји, а ако не постоји позваће функцију повратног позива *OnJoinRandomFailed,* који ће позвати методу *CreateAndJoinRoom*. Ова метода ће позвати функцију повратног позива *OnCreatedRoom*, где се потврђује креирање собе исписивањем лога, а потом ће се позвати *OnJoinedRoom*. У овој функцији повратног позива ће се позвати метода *Instantiate*, која ће инстанцирати *GenericVRPlayerPrefab* поље.

Кад се у скрипти *PortalTeleporter* позове *LeaveLobby* или *LeaveRoom*, позваће се или *OnLeftLobby* или *OnLeftRoom*, у њима ће се позвати *Disconnect,* што узрокује позивање функције повратног позива *OnDisconnected*, где се потврђује да је корисник изгубио везу са серверима исписивањем лога.

Сваки пут када нови корисник уђе у собу позива се *OnPlayerEnteredRoom.*

4.3 Java класе

У Андроид библиотеци се налазе три јава класе коришћене за спрегу са *C#*-ом. Те класе су *PlayerWrapper*, *PlayerListener* и *UnityMessenger.*

PlayerWrapper садржи методу за *initSurfaceTexture*, која иницијализује површину на којој се исцртава, тј. рендерује видео садржај. Ова површина се назива *SurfaceTexture.* Такође садржи методе за управљање видео плејером, које се позивају из *C#* скрипте *PlayerListener.* Те методе су: *initialize*, која иницијализује видео плејер, *CreatePlayer, startPlayer, startPlayerOffset, pausePlayer, resumePlayer, stopPlayer, destroy player,* које су методе за управљање плејером, *unitySeekBarUpdateCallback unityPlaybackUpdateProgress,* које служе за управљање репродукције у виду управљањем временским печатом.

UnityMessenger је класа која садржи методе *sendSingleMessage* и *sendCompiledMessage*, које служе за слање порука методи *C#* скрипте *PlayerListener* по називу *ParseAndroidMessage.*

PlayerListener је класа која на одређене догађаје позива функције повратног позива. Функција повратног позива која је од значаја за решење се зове *on_state_changed*, која се позива кад се промени стање видео плејера. Шаље поруку методи *ParseAndroidMessage* у облику StateTextBox:ново стање.

4.4 PlayerListener

PlayerListener је *C#* скрипта која је, поред многих функционалности, у овом решењу задужена за манипулацију видео плејером.

Поље	Опис
public PhotonView photonView	Инстанца *PhotonView* закачене на *TextureV* објекат
public AndroidJavaObject iwp	Видео плејер који рендерује на *quad*-у објекта на који је закачен
public enum IWPState	Стања видео плејера, вредност може бити Idle, Started, Paused, Resumed, Stopped
public IDictionary<int, IWPState> IWPStates	Чува стања плејера
public IWPState CurrentIWPState	Тренутно стање плејера, узима вредност IWPStates[PhotonVideoSyncManager.playerNumber]
private bool analytics	Ако је *true,* омогућава слушање функција повратног позива из *PlayerListener Java* класе
public string currentTimestamp	Тренутни временски печат видео плејера
public string playerState	Тренутно „стање“ плејера, тј. порука добијена из Андроида
public float videoOffset	Померај са којим ће кренути репродукција видео садржаја
private bool videoPlayerAdded	Проверава да ли је *PlayerListener* додат у листу активних *PlayerListener*-а

Табела 5 – Поља класе *PlayerListener* коришћена у решењу

Дефиниција	Опис
public void InitializePlayer()	Врши иницијализацију плејера
private void AddPlayerToVideoPlayers()	*PUNRPC* који, ако је вредност *videoPlayerAdded false,* додаје *PlayerListener* у листу активних *PlayerListener*-а, и поставља *videoPlayerAdded* на *true*
private void StartPlayerRPC(string uri)	*PUNRPC* који стартује плејер репродукујући видео садржај са прослеђеног *uri*-ја
public void StartPlayer(string uri)	Позива *AddPlayerToVideoPlayers* и *StartPlayerRPC*, позивајући методу *RPC* над *photonView* објектом за све кориснике, чак и оне који се касније придруже соби
private void StartPlayerOffsetRPC(string uri, float offset)	*PUNRPC* који стартује плејер репродукујући видео садржај са прослеђеног *uri*-ја, са прослеђеним померајом за све кориснике
public void StartPlayerOffset(string uri, float offset)	Позива *StartPlayerOffsetRPC*, позивајући методу *RPC* над *photonView* објектом, чак и оне који се касније придруже соби
private void PausePlayerRPC()	*PUNRPC* који паузира плејер ако није већ паузиран, у супротном позива *ResumePlayer*
public void PausePlayer()	Позива *PausePlayerRPC*, позивајући методу *RPC* над *photonView* објектом за све кориснике
public void ResumePlayer()	Наставља репродукцију плејера
private void StopPlayerRPC()	*PUNRPC* који стопира плејер
public void StopPlayer()	Позива *StopPlayerRPC*, позивајући методу *RPC* над *photonView* објектом за све кориснике
private void SeekBarUpdate()	Извршава добављање тренутног временског печата плејера
public void SeekPlayerToTimestamp(string timestamp)	Премотава плејер на прослеђени временски печат
public void ParseAndroidMessage(string message)	Парсира и обрађује поруку добијену из Андроида

Табела 6 – Методе класе *PlayerListener* коришћене у решењу

Метода *InitializePlayer* прво прави инстанцу *Java* класе *PlayerWrapper*, која садржи функционалности манипулације плејером, те иницијализује *iwp* објекат, позивајући *Call* (служи за позивање *Java* метода) методу над њим

CreatePlayer методу *PlayerWrapper*-а. *И*ницијализација плејера се завршава позивањем *initialize* методе *PlayerWrapper*-а.

StartPlayerRPC метода проверава да ли је *CurrentIWPState* плејера *Idle* или *Stopped* (иницијална вредност је *Idle*), ако јесте *iwp* позива *startPlayer* методу *PlayerWrapper*-а са прослеђеним *uri*-јем. Након тога се *CurrentIWPState* поставља на *Started.*

StartPlayerOffsetRPC метода ради исто као и *StartPlayerRPC*, једина разлика је репродукција видео садржаја почиње од прослеђеног помераја (енг. *offset*).

PausePlayerRPC метода прво проверава да ли је *CurrentIWPState* постављен на *Resumed*, и ако јесте, позваће се *ResumePlayer.* Ако није, *iwp* позива *pausePlayer* методу *PlayerWrapper*-а. Након тога се *CurrentIWPState* поставља на *Paused*, па се зауставља извршавање *SeekBarUpdate* методе позивањем *CancelInvoke* методе.

У методи *ResumePlayer iwp* позива *resumePlayer* методу *PlayerWrapper*-а, наставља извршавање *SeekBarUpdate* методе позивањем методе *InvokeRepeating*, која се позива на одређени временски период, у зависности колика је вредност прослеђеног параметра *repeatRate.* Након тога се *CurrentIWPState* поставља на *Resumed.*

У методи *StopPlayerRPC iwp* позива *stopPlayer* методу *PlayerWrapper*-а, потом позива *CancelInvoke* методу, а онда *CurrentIWPState* поставља на *Stopped.*

SeekBarUpdate метода поставља вредност поља *currentTimestamp* на повратну вредност *unitySeekBarUpdateCallback*, методе класе *PlayerWrapper*, коју је позвао *iwp* објекат користећи методу *Call*. Вредност поља *currentTimestamp* представља тренутни временски печат видео плејера.

SeekPlayerToTimestamp премотава видео плејер на прослеђени временски печат тако што *iwp* позива *unityPlaybackUpdateProgress* методу *PlayerWrapper*-а. Након тога се *CurrentIWPState* поставља на *Started.*

ParseAndroidMessage се позива сваки пут кад се у *Java* класи *PlayerListener* позове функција повратног позива која садржи слање поруке ка *Unity3D*-у, користећи *sendSingleMessage* методу класе *UnityMessenger.* Ова порука је у формату string:string, и смешта се у променљиву **string[] dest**, где се порука дели по двотачки. Уколико је вредност dest[0] “SeekbarInitialization”,

позваће се *InvokeRepeating* над *SeekBarUpdate.* Уколико је вредност dest[0] “StateTextBox”, вредност поља *playerState* ће се ажурирати на dest[1].

4.4.1 NativeTexturePlugin

NativeTexturePlugin је *C#* скрипта која је накачена на исти објекат као и *PlayerListener*, а то је *TextureV* објекат. Служи за рендеровање видео садржаја на *TextureV* објекту.

Садржи поље *playerListener* које је типа *PlayerListener,* и оно представља *PlayerListener* који је закачен на *TextureV* објекат исти као и *NativeTexturePlugin.* Такође садржи корутину *CallPluginAtEndOfFrames*, у којој се врти бесконачна петља, где се у свакој итерацији петље чека на крај оквира (енг. *frame*). Након тога се над пољем *iwp* које садржи *playerListener* позива *Java* метода *jRender*.

4.5 PhotonVideoSyncManager

Скрипта садржи променљиву *PlayerState* која је набројивог типа и означава стања плејера која су потребна за реализацију логике решења.

Вредност	Опис
Initialized = 1	Почетно стање
Seeking	Стање које се поставља кад се позове метода за премотавање плејера
PrepareAfterSeeking	Стање које се поставља када је вредност временског печата плејера -1 током трајања премотавања плејера
Playing	Стање које се поставља када се репродукује видео садржај
Waiting	Стање које се поставља када су задовољени сви услови синхронизације репродукције видео садржаја

Табела 7 - Вредности променљиве *PlayerState*

Поље или метода	Опис
public string timestamp	Временски печат плејера
public int viewID	Идентификациони број плејера
public void Update(string timestamp, int id)	Ажурира горенаведена поља

Табела 8 - Поља и методе структуре *VideoSyncParams*

Поље	Опис
public PlayerState state	Тренутно стање плејера
public float lastRecordedTimestamp	Последњи забележен временски печат
public List<float> playbackIntervals	Листа свих разлика временских печата

Табела 9 - Поља класе *PlayerListenerSyncParams*

Поље	Опис
private PlayerListener playerListener	Користи се за чување тренутне инстанце *PlayerListener*-а
private float timestampDiff	Користи се за чување разлике временских печата мастера и тренутног клијента
private VideoSyncParams syncParams	Параметри које шаље мастер
public static List<PlayerListener> videoPlayers	Листа видео плејера која се попуњава у методи *PlayerListener*-а *AddPlayerToVideoPlayers*
public static Dictionary<int, PlayerListener> playerListenerPhotonIDs	Речник чији је кључ идентификациони број, а вредност *PlayerListener* са тим бројем
public static Dictionary<string, PlayerListenerSyncParams> playerParams	Речник чији је кључ име објекта на ком се налази *PlayerListener,* а вредност параметри синхронизације

Табела 10 - Поља класе *PhotonVideoSyncManager*

Кључ поља *playerListenerPhotonIDs* је *ViewID* својство скрипте *PhotonView*, које представља идентификациони број објекта у мрежи, и исти је за све кориснике. С обзиром да се *PhotonView* и *PlayerListener* налазе на истом објекту (*TextureV),* узима се *ViewID* како би се представила иста инстанца *PlayerListener*-а преко мреже.

Иницијална вредност поља *playerParams* се поставља тако да иницијална вредност поља *state* буде *initialized,* а вредност поља *lastRecordedTimestamp* буде -1.

Дефиниција	**Опис**
private IEnumerator SetStateToWaiting(PlayerListenerSyncParams parameters)	Корутина која поставља стање параметра на *Waiting,* па га након одређеног времена (у решењу након тридесет секунди), враћа на *Playing*
private bool IsCurrentState(PlayerState parametersState, PlayerState state)	Пореди параметре и враћа резултат тог поређења
private void SetCurrentState(PlayerListenerSyncParams parameters, PlayerState state)	Поставља стање првог параметра на вредност другог параметра
private (float, float) CalculateUpperAndLowerBound(PlayerListenerSyncParams parameters)	Рачуна горњу и доњу границу међу којима би требао да се налази временски печат
private void SeekToAccordingTimestamp(PlayerListenerSyncParams parameters, PlayerListener playerListener, float receivedTimestamp)	Рачуна одговарајући временски печат и премотава видео плејер на тај временски печат

Табела 11 - Методе класе *PhotonVideoSyncManager*

Метода *CalculateUpperAndLowerBound* рачуна горњу и доњу границу тако што на поље параметра *lastRecordedTimestamp* додаје реципрочну вредност *SerializationRate* на горњу границу, а одузима је како би се направила доња граница. Ово се ради јер је периода позивања *OnPhotonSerializeView* реципрочна вредност *SerializationRate,* па се толики опсег даје временском печату.

Метода *SeekToAccordingTimestamp* рачуна одговарајући временски печат тако што све запамћене разлике временских печата запамћених у пољу параметра *parameters*, *playbackIntervals,* и сабира их са добијеним временским печатом од мастера, као и са тренутном разликом временских печата. На овај израчунати временски печат се не премотава само први пут, кад се премотава само на добијени временски печат. Разлике се сабирају како би се достигла разлика која се направи док се извршава премотавање.

4.5.1 OnPhotonSerializeView

OnPhotonSerializeView је метода пакета *PUN2* која служи за слање жељених података између корисника. Да би се користила ова метода, објекат за који је накачена скрипта која ће користити методу мора да има *PhotonView* скрипту накачену на себи. Такође класа која користи ову методу мора да имплементира интерфејс *IPunObservable.* Метода се позива онолико пута у секунди колика је вредност својства *SerializationRate* класе *PhotonNetwork. OnPhotonSerializeView* се позива све док постоје бар два корисника у *Photon* соби.

OnPhotonSerializeView функционише тако што се испита вредност својства *IsWriting* једног од параметара ове методе, *photonStream.* Ако је вредност овог својства *true,* то значи да корисник шаље податке, у супротном прима податке. Само корисник који је први ушао у *Photon* собу може да шаље податке, и њега ћемо називати мастером.

У овом решењу, мастер пролази кроз листу активних видео плејера и за сваки видео плејер прво ажурира поље *syncParams,* позивајући методу *Update* над њим, прослеђујући *currentTimestamp* и *ViewID* тренутног видео плејера. Након тога се прави променљива типа стринг која садржи два горенаведена прослеђена параметра, који су одвојени знаком „|”. Ову променљиву мастер шаље осталим клијентима.

На другој страни, када се прими податак, прво се примљена стринг променљива растави на део пре и после „|”. Први део се смешта у променљиву *receivedTimestamp,* а други у променљиву *viewID.* Поље *playerListener* добија вредност поља *playerListenerPhotonIDs* за кључ *viewID,* што представља тренутну инстанцу *PlayerListener*-а.

Након тога се прави објекат *parameters* која је претходно изведеног типа *PlayerListenerSyncParams*, и добија вредност поља *playerParams* за кључ који представља тренутно име објекта на ком се налази *PlayerListener.* После тога се прави променљива типа *float* по називу *currentTimestamp*, и она узима временски печат тренутног видео плејера и конвертује га у *float.*

Прва ствар која се проверава је да ли је тренутна вредност поља *state* објекта *parameters* постављена на *Waiting*, користећи методу *IsCurrentState*. Ако је то случај, прелази се у наредну итерацију петље.

Следећа ствар која се проверава је да ли је тренутна вредност поља *state* објекта *parameters* постављена на *Seeking,* користећи методу *IsCurrentState*.

Ако је то случај, рачунају се горња и доња граница користећи методу *CalculateUpperAndLowerBound*, где се њена повратна вредност смешта у променљиве *upperBound* и *lowerBound.* Након тога се проверава да ли је вредност *currentTimestamp* негативна, и ако јесте, вредност поља *state* објекта *parameters* се поставља на *PrepareAfterSeeking,* користећи методу *SetCurrentState.* Ако ово није случај, проверава се да ли је вредност *currentTimestamp* већа или једнака *lowerBound,* и мања или једнака *upperBound.* Ако ово није случај, вредност поља *state* објекта *parameters* се поставља на *Playing,* користећи методу *SetCurrentState.* Ова последња провера служи како би се покрио случај када се премотавање одради превише брзо, и вредност временског печата никад не постане -1, што доводи до проблема да се стање никад неће поставити на *PrepareAfterSeeking,* а самим тим ни на *Playing.*

Наредна ствар која се проверава је да ли је вредност *currentTimestamp* негативна или је вредност *receivedTimestamp* негативна. Ако је ово случај, прелази се у наредну итерацију петље.

После тога се проверава да ли је тренутна вредност поља *state* објекта *parameters* постављена на *PrepareAfterSeeking*, користећи методу *IsCurrentState.* Ако је ово случај, онда се вредност поља *state* објекта *parameters* се поставља на *Playing,* користећи методу *SetCurrentState.*

Након тога се проверава да је поље *playerState*, које је део *playerListener*-а, има вредност „playing“. Ако ово није случај, прелази се на наредну итерацију петље.

Следећа ствар која се проверава је да ли је тренутна вредност поља *state* објекта *parameters* постављена на *Seeking,* користећи методу *IsCurrentState.* Ако је ово случај, прелази се на наредну итерацију петље.

Потом се рачуна разлика *receivedTimestamp* и *currentTimestamp,* те се та разлика смешта у поље *timestampDiff.*

Након тога се проверава да ли је вредност поља *timestampDiff* већа од 0,15 или мања од -0,35. Ове вредности су изабране као услов који би задовољавао синхронизацију репродукције видео садржаја. Ако је услов задовољен, позива се метода *SeekToAccordingTimestamp*, која ће премотати *playerListener* на одговарајући временски печат. Трећи параметар који се прослеђује овој методи је *receivedTimestamp.* Након тога се у поље *playbackIntervals* објекта *parameters* додаје *timestampDiff.* Ако услов није задовољен, проверава се да ли је тренутна вредност поља *state* објекта *parameters* постављена на *Waiting,* користећи

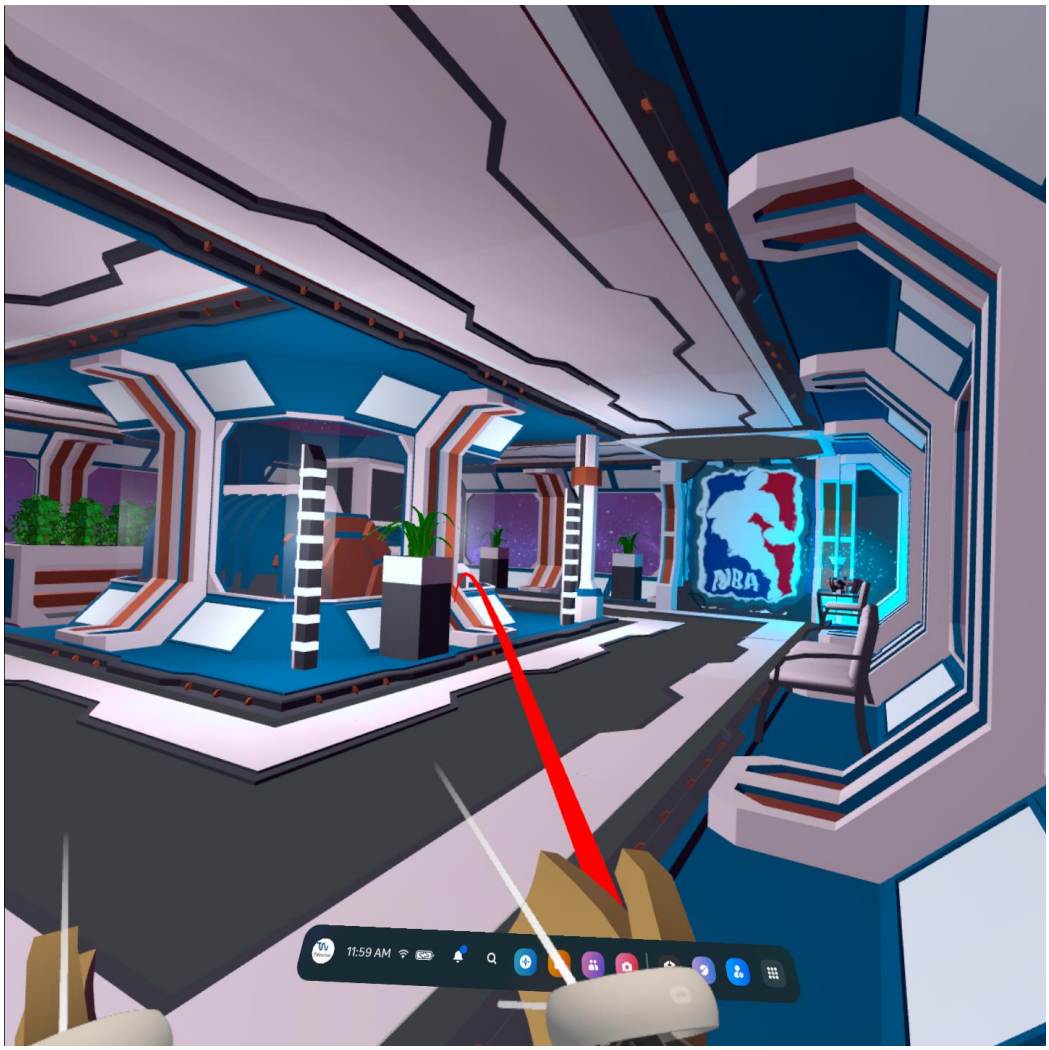
методу *IsCurrentState.* Ако ово није случај, покреће се корутина *SetStateToWaiting.*

Након тога се вредност поља *lastRecordedTimestamp* објекта *parameters* поставља на *currentTimestamp.*

5. Резултати

Решење је тестирано са највише три корисника у *Photon* соби. Један корисник је мастер и он шаље податке, остали примају. Сви корисници могу да стартују, паузирају, наставе репродукцију и стопирају видео плејер, позивајући *PUNRPC* методе описане у поглављу програмско решење. Ове функционалности су тестиране исписивањем логова. Стартовање плејера је потврђено логом „Start player called.“, паузирање логом „Pause button pressed.“, настављање репродукције логом „Player resumed.“, и стопирање логом „Stop button pressed.“.

Синхронизација репродукције видео садржаја је тестирана исписивањем вредности променљивих *receivedTimestamp* и *currentTimestamp* на пријемној страни. Решење је тестирано са највише четири покренута видео плејера истовремено. Даљи резултати тестирања репродукције видео садржаја ће бити приказани и описани у наставку поглавља. Тестирање је рађено по три пута за сваки број покренутих видео плејера истовремено, од један до четири плејера.



Слика 7 – Изглед *EntranceRoom*

5.1 Један видео плејер

Решење са једним покренутим видео плејером је тестирано у сцени *KidsRoom*, док су сви остали тестови обављени у сцени *SportsRoom.*

При првом тестирању, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 19,051. Након четири премотавања, вредност *receivedTimestamp* је била 23,371, а вредност *currentTimestamp* је била 23,416, што задовољава услове синхронизације.

При другом тестирању, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 26,771. Након два премотавања, вредност *receivedTimestamp* је била 28,451, а вредност *currentTimestamp* је била 28,313, што задовољава услове синхронизације.

При трећем тестирању, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 25,291. Након два премотавања, вредност *receivedTimestamp* је била 26,611, а вредност *currentTimestamp* је била 26,517, што задовољава услове синхронизације. Након тридесет секунди, разлика временских печата није задовољавала услове синхронизације, тако да је корисник који прима податке морао још једном да премотава плејер. Након тог премотавања, вредност *receivedTimestamp* је била 57,691, а вредност *currentTimestamp* је била 57,640, што задовољава услове синхронизације.

Редни број тестирања	**Примљен временски печат**	**Тренутни временски печат**	**Број потребних премотавања како би се постигла синхронизација**	**Потребно време како би се постигла синхронизација**
1.	23,371s	23,416s	4	4,32s
2.	28,451s	28,313s	2	1,68s
3.	26,611s	26,517s	2	1,32s

Табела 12 - Статистика након успешне синхронизације једног видео плејера

Слика 8 – Изглед *KidsRoom* са покренутим једним видео плејером

5.2 Два видео плејера

При првом тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 29,771. Након три премотавања, вредност *receivedTimestamp* је била 33,091, а вредност *currentTimestamp* је била 32,960, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 26,571. Након дванаест премотавања, вредност *receivedTimestamp* је била 40,611, а вредност *currentTimestamp* је била 40,597, што задовољава услове синхронизације.

При другом тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 33,811. Након два премотавања, вредност *receivedTimestamp* је била 34,811, а вредност *currentTimestamp* је била 35,144, што задовољава услове синхронизације. За видео плејер који је

други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 29,491. Након шест премотавања, вредност *receivedTimestamp* је била 37,811, а вредност *currentTimestamp* је била 37,915, што задовољава услове синхронизације.

При трећем тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 28,291. Након дванаест премотавања, вредност *receivedTimestamp* је била 40,331, а вредност *currentTimestamp* је била 40,618, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 25,811. Након тридесет премотавања, вредност *receivedTimestamp* је била 50,851, а вредност *currentTimestamp* је била 50,843, што задовољава услове синхронизације.

Редни број тестирања	Видео плејер по редоследу стартовања	Примљен временски печат	Тренутни временски печат	Број потребних премотавања како би се постигла синхронизација	Потребно време како би се постигла синхронизација
1.	Први	33,091s	32,960s	3	3,32s
	Други	40,611s	40,597s	12	14,04s
2.	Први	34,811s	35,144s	2	1s
	Други	37,811s	37,915s	6	8,32s
3.	Први	40,331s	40,618s	12	12,04s
	Други	50,851s	50,843s	30	25,04s

Табела 13 - Статистика након успешне синхронизације два видео плејера

5.3 Три видео плејера

При првом тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 37,451. Након четири премотавања, вредност *receivedTimestamp* је била 41,811, а вредност *currentTimestamp* је била 42,090, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у

собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 35,251. Након два премотавања, вредност *receivedTimestamp* је била 37,891, а вредност *currentTimestamp* је била 37,760, што задовољава услове синхронизације. За видео плејер који је трећи покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 31,531. Након осам премотавања, вредност *receivedTimestamp* је била 53,891, а вредност *currentTimestamp* је била 53,917, што задовољава услове синхронизације.

При другом тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 32,171. Након четири премотавања, вредност *receivedTimestamp* је била 41,891, а вредност *currentTimestamp* је била 41,813, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 28,131. Након четири премотавања, вредност *receivedTimestamp* је била 37,131, а вредност *currentTimestamp* је била 37,451, што задовољава услове синхронизације. За видео плејер који је трећи покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 28,691. Након осам премотавања, вредност *receivedTimestamp* је била 46,011, а вредност *currentTimestamp* је била 46,064, што задовољава услове синхронизације.

При трећем тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 38,411. Након пет премотавања, вредност *receivedTimestamp* је била 42,051, а вредност *currentTimestamp* је била 42,136, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 35,131. Након четири премотавања, вредност *receivedTimestamp* је била 40,131, а вредност *currentTimestamp* је била 40,320, што задовољава услове синхронизације. За видео плејер који је трећи покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 27,891. Након шеснаест премотавања, вредност

receivedTimestamp је била 41,571, а вредност *currentTimestamp* је била 41,696, што задовољава услове синхронизације.

Редни број тестирања	Видео плејер по редоследу стартовања	Примљен временски печат	Тренутни временски печат	Број потребних премотавања како би се постигла синхронизација	Потребно време како би се постигла синхронизација
1.	Први	41,811s	42,090s	4	4,36s
	Други	37,891s	37,760s	2	2,64s
	Трећи	53,891s	53,917s	8	22,36s
2.	Први	41,891s	41,813s	4	9,72s
	Други	37,131s	37,451s	4	9s
	Трећи	46,011s	46,064s	8	17,32s
3.	Први	42,051s	42,136s	5	3,64s
	Други	40,131s	40,320s	4	5s
	Трећи	41,571s	41,696s	16	13,68s

Табела 14 - Статистика након успешне синхронизације три видео плејера

5.4 Четири видео плејера

При првом тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 30,171. Након осам премотавања, вредност *receivedTimestamp* је била 46,571, а вредност *currentTimestamp* је била 46,765, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 25,691. Након четири премотавања, вредност *receivedTimestamp* је била 35,411, а вредност *currentTimestamp* је била 35,544, што задовољава услове синхронизације. За видео плејер који је трећи покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 25,291. Након тридесет шест премотавања, вредност *receivedTimestamp* је била 74,451, а вредност *currentTimestamp* је била 74,410, што задовољава услове синхронизације. За видео плејер који је четврти

покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 18,611. Након три премотавања, вредност *receivedTimestamp* је била 25,611, а вредност *currentTimestamp* је била 25,691, што задовољава услове синхронизације.

При другом тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 29,091. Након три премотавања, вредност *receivedTimestamp* је била 31,451, а вредност *currentTimestamp* је била 31,533, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 26,171. Након четири премотавања, вредност *receivedTimestamp* је била 30,811, а вредност *currentTimestamp* је била 30,762, што задовољава услове синхронизације. За видео плејер који је трећи покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 22,571. Након два премотавања, вредност *receivedTimestamp* је била 25,571, а вредност *currentTimestamp* је била 25,882, што задовољава услове синхронизације. За видео плејер који је четврти покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 18,811. Након осам премотавања, вредност *receivedTimestamp* је била 27,811, а вредност *currentTimestamp* је била 27,752, што задовољава услове синхронизације.

При трећем тестирању, за видео плејер који је први покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 26,531. Након четири премотавања, вредност *receivedTimestamp* је била 31,531, а вредност *currentTimestamp* је била 31,530, што задовољава услове синхронизације. За видео плејер који је други покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 23,651. Након тридесет премотавања, вредност *receivedTimestamp* је била 57,771, а вредност *currentTimestamp* је била 58,059, што задовољава услове синхронизације. За видео плејер који је трећи покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 20,731. Након шест премотавања, вредност *receivedTimestamp* је била 31,131, а вредност *currentTimestamp* је била 30,997,

што задовољава услове синхронизације. За видео плејер који је четврти покренут, када је корисник који прима податке ушао у собу и први пут покренуо премотавање плејера, вредност *receivedTimestamp* је била 19,091. Након десет премотавања, вредност *receivedTimestamp* је била 35,811, а вредност *currentTimestamp* је била 36,074, што задовољава услове синхронизације.

Редни број тестирања	**Видео плејер по редоследу стартовања**	**Примљен временски печат**	**Тренутни временски печат**	**Број потребних премотавања како би се постигла синхронизација**	**Потребно време како би се постигла синхронизација**
1.	Први	46,571s	46,765s	8	16,40s
	Други	35,411s	35,544s	4	9,72s
	Трећи	74,451s	74,410s	36	49,16s
	Четврти	25,611s	25,691s	3	7s
2.	Први	31,451s	31,533s	3	2,36s
	Други	30,811s	30,762s	4	4,64s
	Трећи	25,571s	25,882s	2	3s
	Четврти	27,811s	27,752s	8	9s
3.	Први	31,531s	31,530s	4	5s
	Други	57,771s	58,059s	30	34,12s
	Трећи	31,131s	30,997s	6	10,40s
	Четврти	35,811s	36,074s	10	16,72s

Табела 15 - Статистика након успешне синхронизације четири видео плејера

Слика 9 – Изглед *SportsRoom* са четири покренута видео плејера

6. Закључак

У овом раду је приказана реализација решења које омогућава синхронизацију репродукције видео садржаја у виртуелној реалности. Ово омогућава људима да се повежу и проводе квалитетно време заједно, при чему не морају физички бити на истом месту.

Приказ резултата тестирања решења показује је разлика временских печата између удаљених корисника прихватљиво мала, а у великој већини ситуација и неприметна. У неким случајевима та разлика износи чак 0,01 секунду, или десет милисекунди, што је за људско око и ухо неприметно. Ова разлика временских печата се у већини случајева достигне веома брзо и не толико великим бројем потребних премотавања видео плејера.

Даље унапређење решења је могуће смањивањем услова за потребну разлику временских печата како би се синхронизација репродукције видео садржаја сматрала успешном. Тренутно тај опсег износи пола секунде, и са таквим опсегом је решење тестирано, али се може рећи да је смањивање тог опсега реална опција. Потенцијални проблем који би се могао појавити услед смањивања опсега јесте повећан број потребних премотавања видео плејера. Овај проблем може довести до лошијег искуства корисника, где би корисник имао утисак „сецкања“ репродукције видео садржаја.

7. Литература

[1] Jingming Xie, Research on Key Technologies Base Unity3D Game Engine, Guangzhou Panyu Polytechnic College, The 7th International Conference on Computer Science & Education, Melbourne, Australia, 2012.

[2] “Unity – Manual: Unity User Manual 2022.3(LTS)”, *Unity Documentation*, 2023. [Online]. Available: https://docs.unity3d.com/Manual/UnityManual.html. [Accessed: 15-June-2023]

[3] “Photon Unity Networking 2: Main Page”, *PUN2 Documentation*, 2023. [Online]. Available: https://doc-api.photonengine.com/en/PUN/v2/index.html. [Accessed: 15-June-2023]

[4] “Virtual Reality (VR) | Definition, Development, Technology, …”, 2023. [Online]. Available: https://www.britannica.com/technology/virtual-reality. [Accessed: 16-June-2023]

[5] “The OpenXR Specification”, 2023. [Online]. Available: https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html. [Accessed: 16-June-2023]

[6] “Android - Overview”, 2023. [Online]. Available: https://www.tutorialspoint.com/android/android_overview.htm. [Accessed: 17-June-2023]

[7] “Create an Android library | Android Studio | Android Developers”, 2023. [Online]. Available: https://developer.android.com/studio/projects/android-library. [Accessed: 17-June-2023]

[8] “Meta Quest 2 review | Tom’s Guide”, 2023. [Online]. Available: https://www.tomsguide.com/reviews/oculus-quest-2-review. [Accessed: 17-June-2023]