**Проект по предметот Виртуелна реалност**

1. **Вовед**

Проектот опфаќа игра кој што е изработена со Unity game engine-от. Целта во играта е да се придвижиме од дадена почетна до дадена крајна положба и притоа, тоа да го направиме за одредено време. Движејќи се низ околината, треба да се поминат сите препреки на патот за да се постигне целта.

Околината во која што се движи играчот претставува затворен простор со изглед на мал лавиринт. Покрај подот и ѕидовите, во околината постојат и неколку различни видови на препреки кои играчот треба да ги помине преземајќи различни акции.

Акциите кои што му се достапни на играчот се следните:

* Движење (нанапред) – со навалување на погледот надолу
* Скок – со подигнување на погледот нагоре
* Подигнување на објект – со доближување до објектот и насочување на погледот кон истиот.
* Отпуштање на објект – со подигнување / навалување на погледот нагоре / надолу
* Притискање на копче – со допир или со доближување до копчето и насочување на погледот кон истото (во зависност од типот на копчето).

Користејќи ги овие акции, играчот треба да ги помине препреките кои му се на патот за да ја постигне посакуваната цел.

За време на играњето, на играчот му се прикажува уште колку време му е преостанато. Кога играта ќе заврши, на екранот се прикажува за колку време е завршен обидот, како и најдоброто време на завршување. Играта содржи и позадинска музика, како и различни звуци кои се активираат при интеракција со различни објекти. Дополнително, играта има и воведен екран, екран со упатство и завршен екран.

1. **Детали за дизајнот и имплементацијата**
2. **Околина, движење и скокање**

Прв чекор во изработката на проектот ми беше поставувањето на околината во која играчот ќе се движи. Истата се состои од затворен простор кој наликува на мал лавиринт. Во неа има неколку препреки кои играчот треба да ги прескокне. Дизајнот на околината е **целосно** мој.

Наредно беше креирање на објект кој ќе претставува играч и овозможување на истиот да се придвижува и да скока во просторот. Ова беше реализирано така што кога погледот ќе се навали за одреден број на степени према надоле, играчот ќе се придвижи нанапред, а пак кога погледот ќе се подигне за одреден број на степени према нагоре, играчот ќе скокне. За сето ова да се оствари, најпрвин треба да се додаде модулot GvrEditorEmulator во сцената, кој го овозможува движењето на погледот во апликацијата. Потоа, се креира објект на кој се додаваат следниве компоненти: Transform, Rigidbody и Capsule Collider. Подесувањата може да се по лична преференца, освен кај Freeze Rotation својството на Rigidbody кое треба да се овозможи за трите оски X, Y и Z. Следно, треба да се постави главната камера како подобјект на објектот кој го претставува играчот. По ова, останува уште да се напишат скриптите кои го овозможуваат движењето и скокањето и истите да се додадат на објектот. Тоа се скриптите PlayerWalk.cs и PlayerJump.cs, соодветно. Логиката која е имплементирана и кај двете е многу слична. Најпрвин се проверува дали аголот на погледот е во потребните граници за да се изведе соодветната акција со следните услови:

*Camera.main.transform.eulerAngles.x >= toggleAngle && Camera.main.transform.eulerAngles.x <= 90F* - за движење

*Camera.main.transform.eulerAngles.x <= 360 - toggleAngle && Camera.main.transform.eulerAngles.x >= 360 - 90* - за скок

Потоа доколку условот е исполнет се повикуваат соодветните методи од Rigidbody компонентата кои прават промена на состојбата на објектот, со соодветни аргументи:

*rb.MovePosition(Vector3 position);* - за движење

*rb.AddForce(Vector3 position, ForceMode mode);* - за скок

Значајна разлика помеѓу двете скрипти е тоа што кај скриптата за скок, пред да се скокне се проверува дали објектот е на земја, за да се избегне постојано зголемување на висината на објектот. Исто така, за дополнителна сигурност е поставен и тајмер кој го мери времето поминато од последното скокање. Играчот не може да скокне се додека тоа време не надмине некое предефинирано време. Со тоа се решаваат проблемите кои можат да се јават во одредени ситуации.

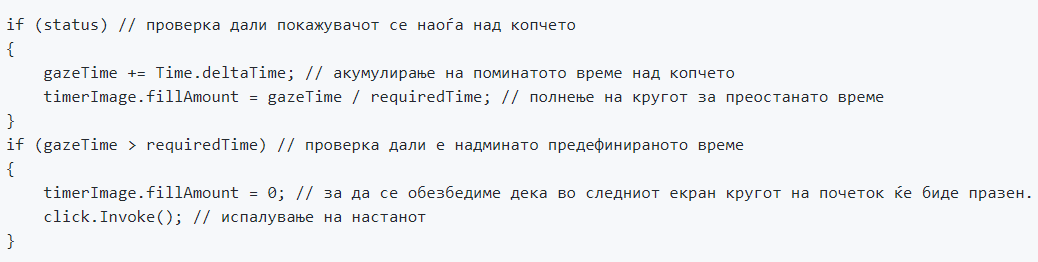
За креирањето на објектот кој го претставува играчот и пишувањето на скриптите се послужив со неколку различни туторијали, написи на форуми и официјалната документација. Еден видео-туторијал кој би можел да го издвојам е следниов:

[[Tutorial] Movement In Mobile VR: Look Walk](https://www.youtube.com/watch?v=kBTn2pGwZUk).

1. **Екрани**

Следно продолжив со дизајнирање на изгледот на воведниот екран, екранот со упатството и завршниот екран. Трите екрани се разделени во три различни сцени, посебни од сцената од првата домашна задача. Овие екрани се состојат од копчиња со кои може да се врши интеракција преку задржување на погледот над истите. Копчињата се поставени на Panel објект, кој обезбедува поставување на позадинска боја и истиот е поставен на Canvas објект.

По стартувањето на апликацијата се прикажува воведниот екран. На него има три копчиња. Со "притискање" на првото копче се влегува во сцената од првата домашна задача. По завршувањето на играта ни се прикажува завршниот екран. На завршниот екран има две копчиња, преку кои можеме или да се вратиме во сцената од претходната домашна задача, или да ја прекинеме апликацијата. Второто копче на воведниот екран служи за влез во сцената во која е прикажано упатството за движење и скокање и дополнително има копче преку кое можеме да се вратиме назад кон воведниот екран. Последното копче од воведниот екран е копчето за излез од апликацијата.

За остварување на овие функционалности најпрвин е потребен GvrEditorEmulator, кој овозможува движење на погледот, и GvrEventSystem на кој треба да се додаде компонентата GvrPointerInputModule, преку која се овозможува посакуваната интеракција. Потоа, потребно ни е да имаме главна камера на која ќе и се придружи GvrRecticlePointer објект. Овој објект ни го дава крукчето на екранот, т.е. покажувачот преку кој се врши интеракцијата со копчињата и дополнителни функционалности за остварување на истата. Дополнително, на камерата и е придружен и Canvas објект, на кој пак е придружен Image објект кој има форма на круг и е иницијално празен. Преку овој објект се прикажува уште колку време е преостанато пред да се "притисне" копчето. Скриптата која што го овозможува кликнувањето на копчињата преку задржување на погледот над истите е GazeClick.cs. Нејзината имплементација е тривијална и се состои од проверка дали покажувачот се наоѓа над некое од копчињата и доколку е тоа случај, тогаш се мери колку време истиот ќе се задржи над соодветното копче. Ако тоа време надмине некое предефинирано време, тогаш ќе се изврши соодветната акција. Дополнително, се полни и кругот кој што ни прикажува уште колку време ни е преостанато до кликнување. Кодот кој го овозможува ова е следниот:

Оваа скрипта се додава на секое од копчињата. Исто така, на секое од копчињата мора да се додаде и Event Trigger скриптата, која ни ја нуди Unity и да се постават соодветни настани за влез на покажувачот над некое копче и излез надвор од него. Преку нив ќе се менува вредноста на променливата status и ќе се ресетираат gazeTime и timerImage.fillAmount променливите од GazeClick.cs скриптата.

По испалувањето на настанот, потребно е да се промени сцената. За менување на сцените имам напишано посебна скрипта која е именувана како ChangeScene.cs. Менувањето на сцените се изведува преку повик на методот:

UnityEngine.SceneManagement.SceneManager.LoadScene(sceneName)

За изработка на екраните се послужив со неколку различни туторијали, написи на форуми и официјалната документација. Меѓу нив, би издвоил два видео-туторијали:

[START MENU in Unity](https://www.youtube.com/watch?v=zc8ac_qUXQY)

[Unity VR Tutorial - Button Gaze Timer Interaction](https://www.youtube.com/watch?v=zdNBZsJdg9c)

1. **Интеракција со објекти**

Наредно, имплементирав 3 различни типови на интеракција помеѓу играчот и дадени објекти од сцената, и тоа:

1. **Подигнување и носење на коцка, која ќе послужи за да се прескокне една од повисоките препреките**

За нејзина имплементација, најпрвин е потребно да се додаде GvrReticlePointer компонента на камерата на играчот, но овој пат, Max Reticle Distance својството треба да е поставено на 2. Со тоа, играчот нема да може да влегува во интеракција со објекти кои се оддалечени од него на растојание поголемо од 2. Исто така, на камерата на играчот се додава и еден празен објект, кој е именуван како Guide, и кој има Box Collider компонента која е иницијално оневозможена. Потоа, се креира нов 3D објект од тип Cube, кој ќе ја претставува коцката. На таквиот објект, покрај стандардните компоненти, му се додаваат и компонентите Rigidbody и EventTrigger. Следно, се креира нова скрипта PickObject.cs, која се додава на коцката, а во која се имплементирани сите методи кои ја овозможуваат интеракцијата. Кога играчот ќе се доближи до коцката на растојание помало од 2 и Reticle Pointer-от ќе покажува кон истата, тогаш се испалува настанот PointerEnter, преку кој се повикува методата OnPickUp() од PickObject.cs скриптата. Преку оваа метода, Transform објектот на коцката се поставува како дете на Transform објектот на Guide објектот. Во оваа метода се извршуваат уште неколку подесувања, меѓу кои едно е овозможувањето на Box Collider компонентата на Guide објектот. Тоа служи за да не се дозволи коцката да "поминува низ ѕидови" додека е носена од играчот. Последно, за да коцката се спушти на земја, откако претходно била подигната, потребно е погледот на играчот да се подигне/навали за одреден агол према нагоре/надолу. Спуштањето на објектот е имплементирано преку OnPutDown(bool look) методата, чиј аргумент look укажува на тоа во која насока е придвижен погледот(нагоре/надолу). За различна насока на придбижување на погледот, коцката се отпушта на различен начин. Замислено е објектот да се отпушта со подигнување на погледот према нагоре. Другиот начин е поставен само за да се избегне колизија на објектот со подот.

1. **Отворање на врата при стоење над даден објект и нејзино повторно затворање откако играчот повеќе не е врз објектот**

Потребни се два 3D објекта, од кои едниот ја претставува вратата, а другиот има улога на копче. Копчето е иницијално со црвена боја и е поставено на подот. Доколку играчот застане врз копчето, тоа се обојува зелено и во исто време вратата почнува постепено да се отвора. Во оној момент кога играчот веќе нема да е врз копчето, тоа ќе се обои црвено и вратата ќе почне да се затвора. За да успееме да поминеме низ неа, треба да почекаме таа целосно да се отвори. Скриптата која ја овозможува оваа интеракција е OpenDoor.cs и истата е додадена на објектот кој го претставува копчето. Во неа, потребни ни се објектот кој ќе ја претставува вратата, како и брзините на отворање и затворање. Методите од OpenDoor.cs кои се повикуваат при испалување на настаните кои го регистрираат застанувањето на играчот врз копчето и негово излегување од истото се OnTriggerEnter() и OnTriggerExit(), соодветно. Преку овие методи се контролира вредноста на clicked променливата од истата скрипта. Во зависност од таа вредност, и во зависност од тоа дали вратата е целосно отворена или затворена, во Update() методата се менува векторот на позицијата на Transform објектот на вратата.

1. **Поместување на препрека преку "притискање" на копче.**

Овој тип на интеракција е многу сличен на интеракцијата со копчињата од претходната домашна задача. Единствените разлики се во тоа што копчето се "притиска" веднаш штом ќе го придвижиме покажувачот над него, и по "притискање" на копчето, наместо да се смени сцената, се поместува препрека со што се отвара простор низ кој играчот може да помине. Поместувањето се прави преку доделување на нов вектор на позиција на Transform објектот на препреката.

Како помош при изработката на овие функционалности ми послужија следниве два видео-туторијали:

[Pick up and Move Objects | Unity 2017 Tutorial](https://www.youtube.com/watch?v=JtflOvhOO1Y)

[Unity Beginner Tutorial : Triggers](https://www.youtube.com/watch?v=JHZN_vMqqr4)

1. **Аудио**

Додадов и позадинска музика во главната сцена. Тоа го направив на начин што на главната камера, онаа која што е придружена како дете на играчот, и додадов компонента Audio Source. На таа компонента го поставив соодветниот аудио фајл со музиката која треба да се пушти, и ги поставив својствата Play On Awake и Loop на true. Со тоа, аудиото ќе се пушти уште на почетокот на сцената и нема да прекине се до нејзиниот крај.

Следно, додадов неколку звуци кои се активираат при интеракција со различни објекти или приближување кон даден објект. Такви звуци може да се слушнат при подигнување на коцката, притискање (и отпуштање) на копчињата и при доближување кон вратата додека таа се отвора или затвора. За даден објект да емитира некаков звук, најпрвин е потребно да се додаде Audio Source компонента на тој објект. Потоа, на таквата компонента треба да се постави соодветниот аудио фајл со звукот. Останатите подесувања зависат од самата ситуација.

Кај коцката, својствата Play On Awake и Loop на Audio Source компонентата се иницијално поставени на false. При подигнување на коцката, преку соодветниот објект се повикува методата која го активира звукот:

audioSourceObject.Play();

Кај копчињата, проблемот е решен на истиот начин. Во моментот на испалување на соодветниот настан се активира соодветниот звук. Повиците на методите се напишани во веќе постоечките скрипти.

Разлика има кај звукот кој се слуша при отворањето и затворањето на вратата. Доколку сме поблиску до вратата, звукот ќе биде погласен, а во спротивно, тој ќе биде потивок. За да се овозможи ваквиот ефект, потребно е Spatial Blend својството на соодветната Audio Source компонента да се постави на 1, со што се добива т.н. 3D звук. Дополнително, се поставува Volume Rolloff својството на 3D звукот на Linear Rolloff, со што релацијата помеѓу јачината на звукот и оддалеченоста од објектот се моделира преку линеарна крива. Во соодветната скрипта, регулирањето кога звукот да почне да се емитува и кога да прекине се прави со две знаменца, moving и doorSoundOn. Доколку вратата се движи (т.е. moving е true), а звукот не е уклучен, тогаш тој се уклучува и doorSoundOn променливата се поставува на true. Звукот ќе биде исклучен во оној момент кога вратата целосно ќе се затвори или целосно ќе се отвори, по што doorSoundOn, како и moving повторно се враќаат на false.

Аудио фајловите кои ги содржат звуците кои ги користев се превземени од:

[Позадинската музика](https://www.dl-sounds.com/royalty-free/blazer-rail/)

[Подигнување на коцката](https://freesfx.co.uk/Category/Grunts-Groans-and-Moans/259)

[Притискање/отпуштање на копчињата](https://www.freesoundslibrary.com/button-click-sound-effect/)

[Движење на вратата](https://bigsoundbank.com/detail-0513-portail-automatique-coulissant.html)

Како помош ги користев следниве два видео-туторијали:

[How to play background music in Unity](https://www.youtube.com/watch?v=eB3I4l0AED0)

[Play sound upon Collision in Unity](https://www.youtube.com/watch?v=yE0JdtVTnVk)

1. **Тајмер**

Следно, имплементирав тајмер кој мери колку време е поминато од почетокот на главната сцена. Доколку ова време надмине некое предефинирано време, тогаш играта завршува. Времето кое му е преостанато на играчот е постојано во неговото видно подрачје, во горниот десен агол.

Ова е реализирано преку додавање на Canvas објект на главната камера, кој што е именуван како Timer Canvas. На тој објект му се придружува Text Mesh компонента, на која всушност ќе се испишува времето кое е преостанато. Дополнително, на објектот му се додава и скриптата GameTimer.cs. Во оваа скрипта се вршат пресметките за тоа колку време е преостанато, по што резултатот се запишува на Text Mesh компонентата во соодветен формат. Доколку преостанатото време е над (или еднакво на) една минута, форматот во кој се испишува истото е MM:SS. Доколку тоа е под една минута, тогаш се прикажува во формат SS. Дополнително, доколку преостанатото време е под 10 секунди, тогаш бојата со која се испишува истото се менува од зелена во црвена.

Оваа функционалност ја изработив без користење на дополнителни ресурси за помош.

1. **Време на завршување**

Последно, имплементирав прикажување на времето за кое ќе биде завршен тековниот обид, како и прикажување на најдоброто време на завршување. Во сцената во која се прикажува завршниот екран, покрај двете копчиња за нов обид и крај на играта, додадов и текст компонента на која би се испишале тековното време и најдоброто време на завршување. Дополнително, доколку тековното време на завршување е всушност новото најдобро време на завршување се испишува и соодветна порака. Ваквото динамичко подесување на текстот се врши во DisplayTime.cs скриптата. Најголем предизвик при имплементацијата на барањето беше "префрлувањето" на вредноста за тековното време на завршување од една сцена во друга. Овој проблем е решен со користење на статичка променлива. Најдоброто време на завршување се зачувува и по исклучување на играта. Ова е постигнато со користење на PlayerPrefs API-то.

За имплементација на функционалноста ми помогна следниот одговор на Stack Overflow:

<https://stackoverflow.com/a/44542726>

1. **Видео**

Линк до видео на YouTube од играта:

<https://www.youtube.com/watch?v=9hwpocu1DyI>