UNIWERSYTET WROCŁAWSKI
WYDZIAŁ FIZYKI I ASTRONOMII

PIOTR WOJTASZEK

GRA EDUKACYJNA Z DZIEDZINY ASTRONOMII I FIZYKI WYKONANA W TECHNOLOGII VR

Praca inżynierska wykonana pod kierunkiem
prof. dr hab. Zbigniewa Kozy
na Wydziale Fizyki i Astronomii

WROCŁAW 2021

STRESZCZENIE PRACY

Niniejsza praca inżynierska miała na celu stworzenie aplikacji VR na urządzenie Oculus Quest za pomocą silnika Unity. Gra składa się z 6 scen, przedstawiających zagadnienia z obszaru fizyki i astronomii w prosty, a także czytelny sposób. Użytkownik ma do dyspozycji 2 kontrolery, umożliwiające wchodzenie w interakcje z obiektami w aplikacji.

W rozdziale pierwszym została zawarta specyfikacja urządzenia Oculus Quest oraz wprowadzenie odbiorcy w środowisko Unity. Wyżej wspomniane wątki zostały poruszone na samym początku, ponieważ znajomość budowy edytora Unity ułatwia zrozumienie dalszej części pracy.

Drugi rozdział zawiera opis interaktywnej zawartości aplikacji oraz kadry z aplikacji. W pierwszej scenie użytkownik może doświadczyć zjawiska Zaćmienia Słońca, nad którym sam może panować. Druga scena przedstawia animację Układu Słonecznego. Trzecia scena przedstawia symulację fizyczną Układu Słonecznego. W obydwu scenach użytkownik ma możliwość sterowania wielkością oraz prędkością modelu. Kolejny poziom daje użytkownikowi możliwość porównywania grawitacji na różnych planetach. Piąta scena przedstawia kilka gwiazdozbiorów, które użytkownik może ułożyć. Umożliwia to czytelniejsze zwizualizowanie konstelacji. Ostatnia scena jest łącznikiem pomiędzy wyżej wspomnianymi scenami.

W ostatnim rozdziale zostały opisane szczegóły dotyczące scen, ich sposobu działania oraz problemy napotkane podczas rozwijania aplikacji. Rozdział zawiera fragmenty kodu oraz kadry z edytora Unity.

ABSTRACT OF THESIS

The purpose of this thesis is to create a VR application for the Oculus Quest device using the Unity engine. The game consists of 6 scenes presenting some topics in the field of physics and astronomy in a simple and easy to understand way. The user has 2 VR controllers at his disposal, allowing him to interact with objects in the application.

The first chapter contains the specification of the Oculus Quest device and introduces the reader to the Unity environment. The second chapter describes the interactive content of the application and screenshots from the application. In the first scene, the user can experience the phenomenon of a Solar eclipse, over which he can take control himself. The second scene presents an animation of the Solar System. The third scene is a physical simulation of the Solar System. In both scenes, the user can control the size and speed of the model. The next level gives the user the ability to compare the gravity on different planets. The fifth scene shows several constellations that the user can arrange. This makes it possible to visualize the constellations more clearly. The last scene is the link between the above-mentioned scenes.

The last chapter describes the details of the scenes, how they work, and the problems encountered during the development of the application. This chapter contains code snippets and screenshots from the Unity editor.

Spis treści

WSTĘP

Informatyka jest dynamicznie rozwijającą się dziedziną współczesnej nauki. Każdego roku powstaje wiele nowych lub zmodyfikowanych technologii. Znaczna część współczesnych wynalazków miała początki w XX w. Stworzone wtedy technologie były na wyłączność wojska lub naukowców, ze względu na swoją złożoność, mały komfort użytkowania i ogromny koszt sprzętu. Ekrany dotykowe, a nawet same komputery, na początku nie były dostępne dla zwykłych konsumentów. Dopiero postępująca miniaturyzacja i spadek cen umożliwiły spopularyzowanie tych technologii. Jedną z obecnie na nowo odkrywanych technologii jest wirtualna rzeczywistość, której początki sięgają lat 60. XX w. Obecnie popularność wirtualnej rzeczywistości z roku na rok rośnie[1], co zachęca kolejne firmy do inwestowania w tę technologię. Fundamentem wirtualnej rzeczywistości są gogle VR, które po założeniu na głowę mają stworzyć iluzję 3D. Oprócz samych gogli użytkownicy mają do dyspozycji różnego rodzaju kontrolery. Gogle VR mogą się różnić w zależności od producentów, jednak zdecydowana większość dostępnych na rynku urządzeń tego typu do działania wymaga stałego podłączenia do komputera.

Tematem poniższej pracy jest aplikacja VR przybliżająca użytkownikom funkcjonowanie fizyki oraz pojęć związanych z astronomią, przeznaczona do uruchomienia na goglach Oculus Quest. Głównym moim celem było poznanie nowych narzędzi pomagających w tworzeniu tego typu aplikacji oraz stworzenie kompletnej aplikacji. Podstawowym założeniem aplikacji była jej prostota, tak aby każdy użytkownik, bez względu na doświadczenia z technologią, był w stanie bezproblemowo jej używać. Chciałem wykorzystać zalety technologii wirtualnej rzeczywistości, wysoką immersję oraz możliwość interakcji, do ukazania rzeczy, których zwykły ekran monitora i komputera nie jest w stanie oddać, takich jak np. skala Układu Słonecznego i możliwość interakcji z przedmiotami. Kolejnym ważnym celem aplikacji było ukazanie zjawisk fizycznych i astronomicznych w jak najbardziej czytelny i zrozumiały sposób, choć niekoniecznie wiernie oddający rzeczywistość.

[1] https://www.mordorintelligence.com/industry-reports/virtual-reality-market, [dostęp: 08.06.2021r.].

ROZDZIAŁ 1

CEL I GŁÓWNE ZAŁOŻENIA PROJEKTU

Motywacją do podjęcia tematyki VR była aktualnie wykonywana przeze mnie praca, w której tworzę i rozwijam aplikacje na gogle VR, oraz chęć poznania nowych narzędzi opracowanych przez firmę Unity Technologies.

Głównym założeniem projektu było stworzenie aplikacji, która jest interesująca dla użytkownika nie tylko ze względu na wykorzystaną technologię wirtualnej rzeczywistości, ale również ze względu na jej zawartość. Uznałem, że dobrym wyborem będzie podjęcie tematu fizyki i astronomii, które są w moim kręgu zainteresowań. Przy projektowaniu aplikacji starałem się zawrzeć w niej zagadnienia, które zaangażują oraz zaciekawią użytkownika. Zdecydowałem, że gra będzie wykorzystywać moc obliczeniową gogli Oculus Quest, dzięki czemu do jej uruchomienia nie jest wymagany komputer. Użytkownik będzie mieć większą swobodę ruchu, bez niekomfortowych odczuć związanych z otaczającymi go przewodami.

1.1 Środowisko programistyczne

Unity3D

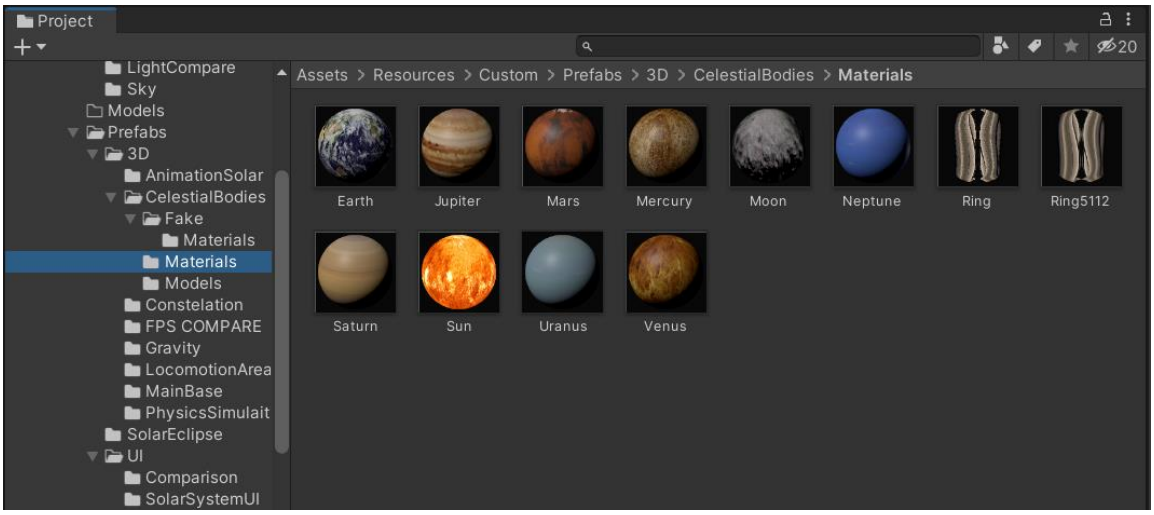
Unity jest międzyplatformowym silnikiem gry rozwijanym od 2005 roku przez Unity Technologies. Umożliwia tworzenie aplikacji na komputery, konsole, telefony oraz na gogle VR. Platforma Unity jest chętnie wykorzystywana nie tylko do tworzenia gier, ale również w wielu innych branżach, takich jak motoryzacja, transport, produkcja oraz architektura. Silnik jest chętnie wykorzystywany przez wielkie korporacje takie jak Microsoft i CD Project Red oraz przez małe studia gier i niezależnych twórców. Przy użyciu Unity zostały stworzone takie gry jak *Pokemon Go, Gwint, Cuphead, Ori and the blind forest* oraz *Beat Saber*.

Działanie edytora Unity można podzielić na dwa fundamentalne stany, *edit mode* oraz *play mode*. Pierwszy tryb zapisuje na bieżąco wszystkie zmiany poczynione w aplikacji np. przesunięcie obiektu. Drugi tryb służy do testowania aplikacji, po wyjściu z tego trybu wszelkie zmiany np. zmiana wielkości obiektu, przepadają i wracają do stanu sprzed wejścia w *play*

mode. Edytor Unity jest podzielony na okna, najczęściej używanymi są: *Inspector, Game, Scene, Project* oraz *Hierarchy.*

Project

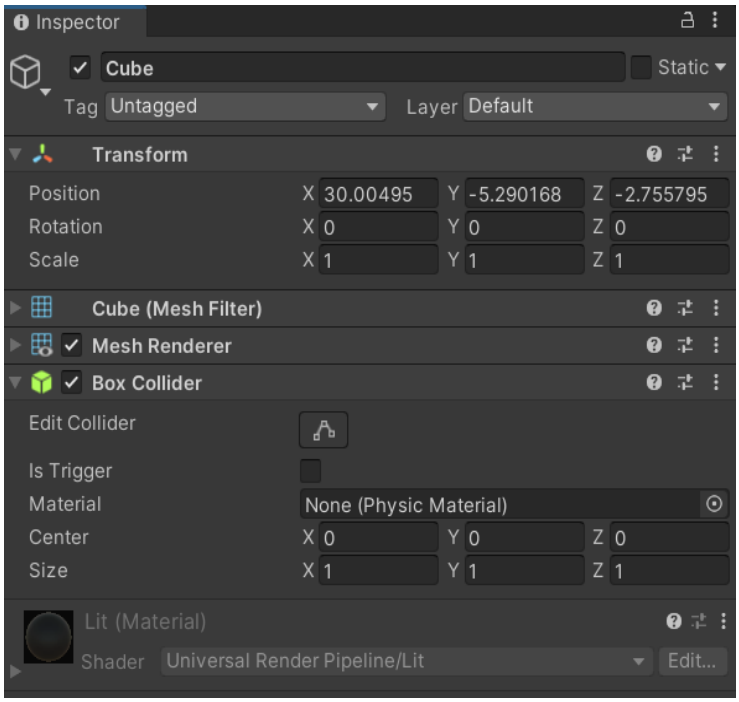
Project jest miejscem, w którym można znaleźć wszystkie *assety,* czyli elementy składające się na aplikacje, np.: modele 3D, skrypty C#, muzyka oraz tekstury. Umożliwia tworzenie nowych plików i folderów, a także przenoszenie oraz podgląd już istniejących.



Rys. 1. Wygląd okna *Project*

Inspector

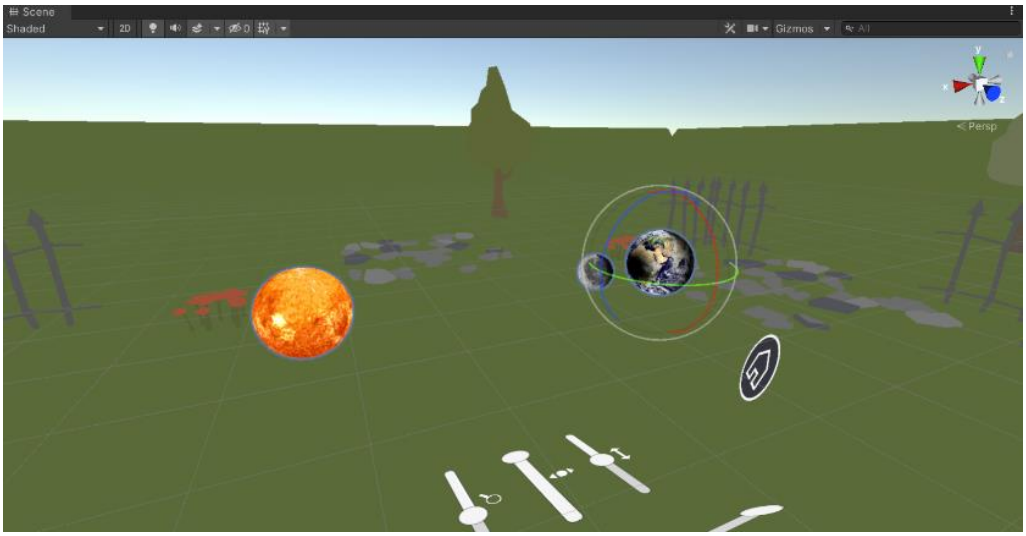
Inspector jest oknem umożliwiającym podgląd oraz modyfikowanie właściwości aktualnie wybranego obiektu, takich jak pozycja lub wielkość. W tym miejscu możemy dodawać kolejne komponenty, czyli klasy, które nadają obiektowi nowe właściwości, np.: wykrywanie kolizji, odtwarzanie muzyki lub dodanie skryptu stworzonego przez programistę. Domyślnie wszystkie pola w klasie z modyfikatorem *public* są widoczne i mogą być zmodyfikowane w tym miejscu, bez konieczności zmiany ich wartości w kodzie.



Rys. 2. Wygląd okna *Inspector*

Scene

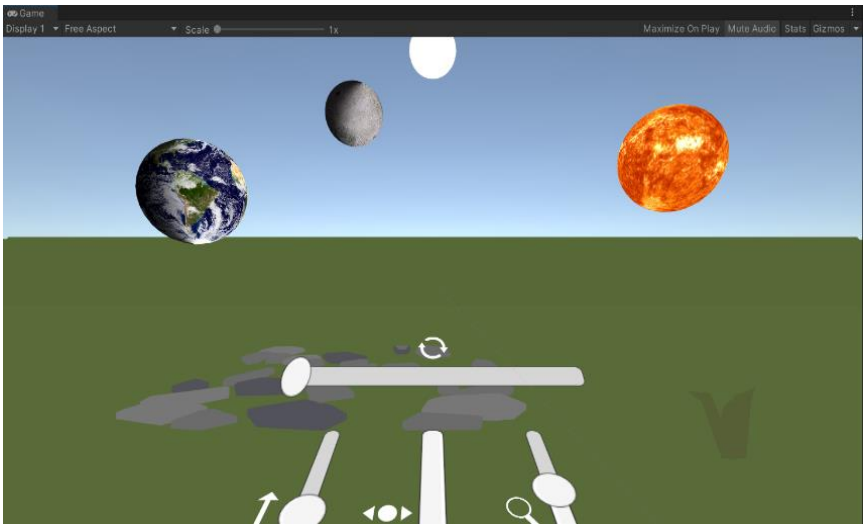
Okno *Scene* umożliwia poruszanie obiektami znajdującymi się na scenie. Obiekty można dodawać poprzez przenoszenie *assetów* z okna *Project*. Programista ma do dyspozycji kilka podstawowych narzędzi do dostosowywania obiektów we wszystkich trzech osiach, umożliwiają one przesuwanie, obracanie oraz zmienianie wielkości obiektu.



Rys. 3. Wygląd okna *Scene*

Game

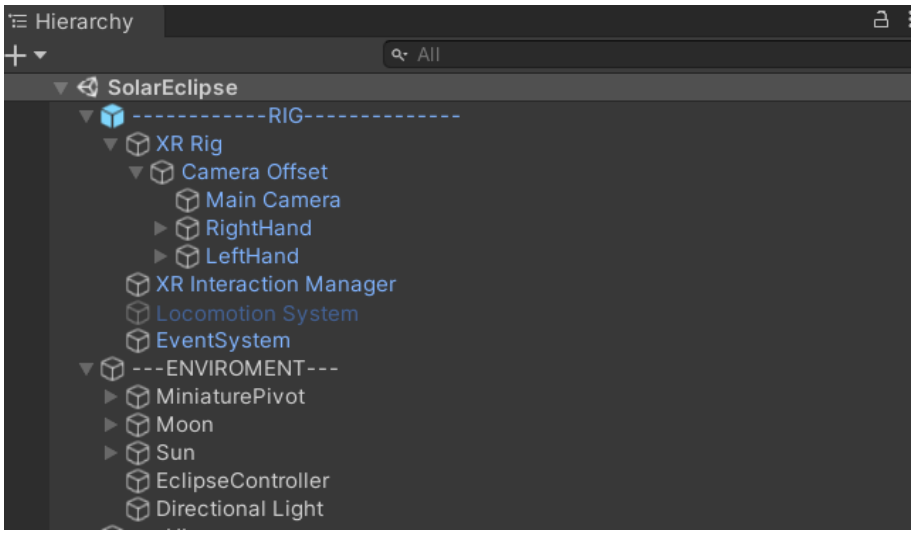
Okno *Game* jest podglądem tego, co użytkownik zobaczy w finalnej wersji aplikacji. Umożliwia ono szybkie testowanie aplikacji bez konieczności wychodzenia z edytora Unity.



Rys. 4. Wygląd okna *Game*

Hierarchy

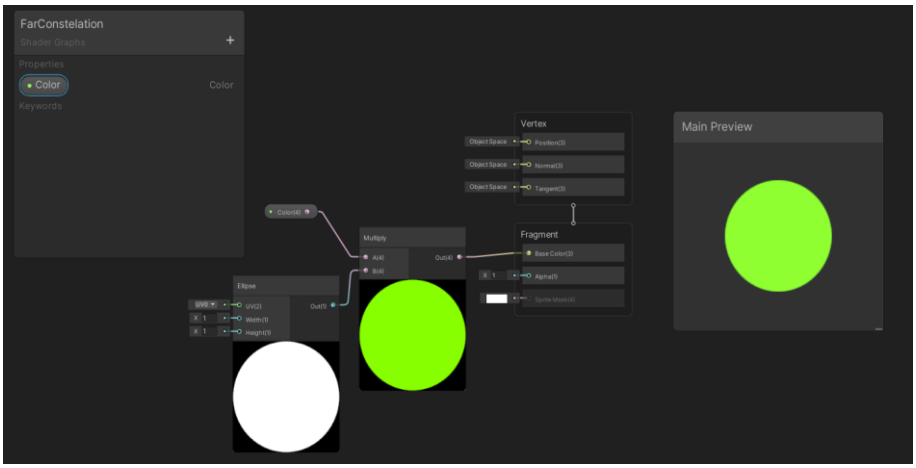
Okno *Hierarchy* zawiera listę wszystkich obiektów znajdujących się na scenie. Poprzez przeciągnięcie elementu z okna *Project*, można za pomocą niego dodać obiekt na scenę.



Rys. 5. Wygląd okna *Hierarchy*

Shader Graph

Shader Graph jest oficjalną paczką opracowaną przez Unity Technologies, będącą wizualnym kreatorem shaderów. Umożliwia ona tworzenie shaderów bez znajomości języka HLSL. Tworzenie odbywa się za pomocą węzłów i połączeń oraz pozwala na podgląd zmian w czasie rzeczywistym.



Rys. 6. Okno *ShaderGraph* oraz przykładowy shader

XR Interaction Toolkit[2]

Oficjalna paczka do Unity będąca *frameworkiem* umożliwiającym szybką konfiguracją podstawowych elementów potrzebnych do stworzenia aplikacji VR i AR. W jej skład wchodzą komponenty odpowiedzialne za interakcję z obiektami 3D i interfejsem użytkownika oraz obsługę kontrolerów.

Asset Store[3]

To oficjalny sklep Unity, w którym można kupować i pobierać elementy, z ang. *assets*, pomagające w procesie tworzenia aplikacji, takie jak modele 3D, muzyka lub narzędzia wzbogacające edytor o nowe funkcje. Podobną funkcję pełni *Package Manager,* w którym można szukać zakupionych *assetów* oraz paczki rozszerzające możliwości edytora.

[2] https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@1.0/manual/index.html, [dostęp: 2.06.2021r.].
[3] https://assetstore.unity.com/, [dostęp: 2.06.2021r.].

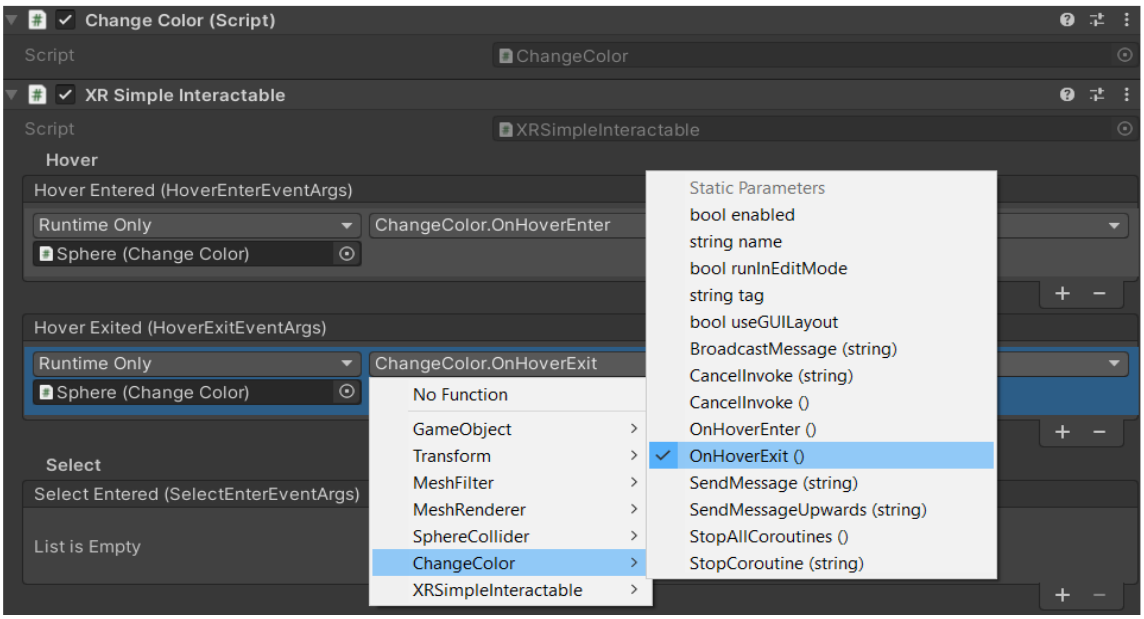
Przykładowe tworzenie prostej interakcji z obiektem

W oknie *Project* należy utworzyć klasę *ChangeColor*, w której zostaną zaimplementowane interakcje. W klasie należy utworzyć dwie metody, *OnHoverEnter()* zmieniającą kolor obiektu na żółty oraz *OnHoverExit()* zmieniającą kolor przedmiotu na biały.

Tabela 1. Fragment klasy *ChangeColor*

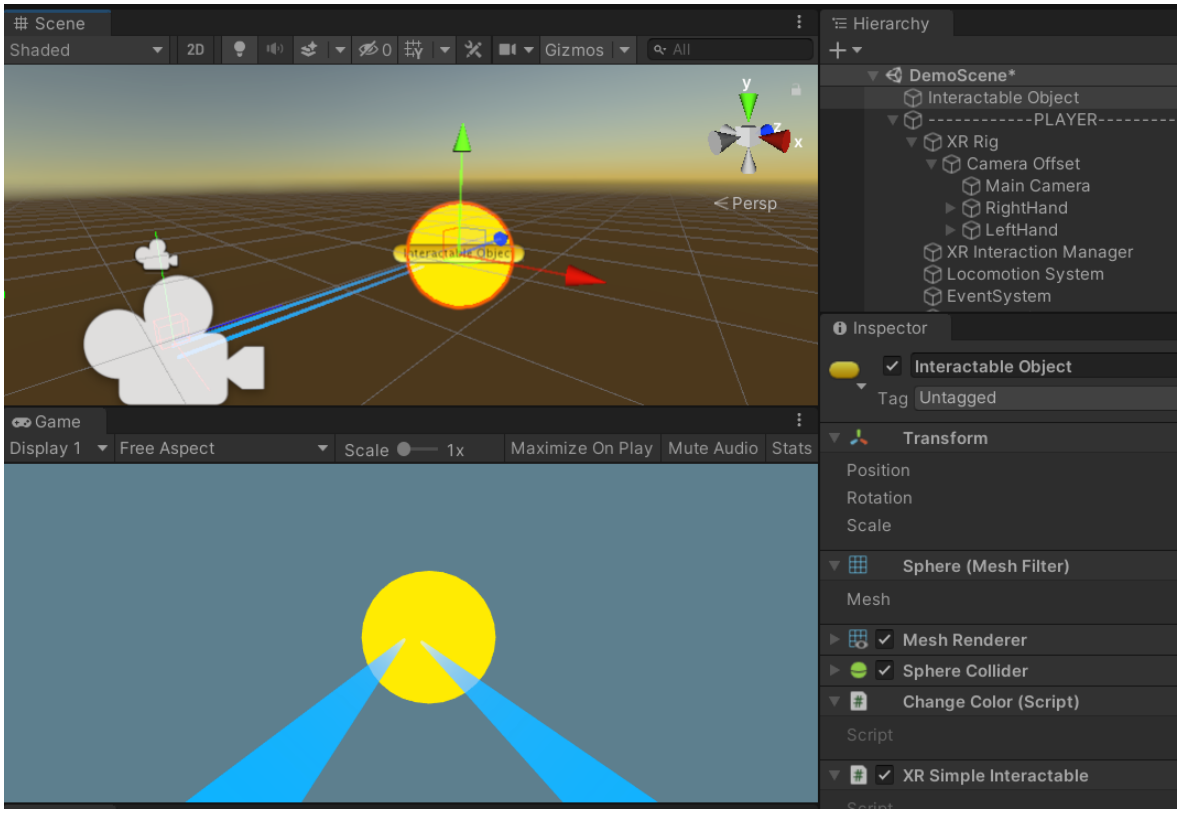
```
public class ChangeColor : MonoBehaviour
{
    public void OnHoverEnter()
    {
        GetComponent<Renderer>().material.color = Color.yellow;
    }
    public void OnHoverExit()
    {
        GetComponent<Renderer>().material.color = Color.white;
    }
}
```

Po stworzeniu klasy, na scenie należy umieścić obiekt, do którego zostanie przyczepiony wcześniej stworzony skrypt. W tym celu należy stworzyć sferę i następnie dodać do niej klasę *ChangeColor* oraz komponent *XRSimpleInteractable*, dostarczony w bibliotece *XR Interaction Toolkit*.



Rys. 7. Fragment komponentu *XRSimpleInteractable*

Komponent *XRSimpleInteractable* zawiera pola, które są *delegatami,* pozwalającymi na dodawanie do nich własnych funkcji. W prezentowanym przykładzie delegat *Hover Entered* zostanie wywołany, gdy kontroler zostanie wycelowany w obiekt, natomiast delegat *Hover Exited* zostanie wywołany, gdy kontroler przestanie być skierowany w stronę obiektu.

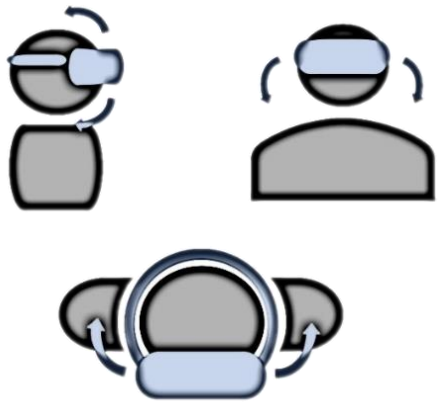


Rys. 8. Zrzut ekranu ukazujący fragment edytora Unity

Stworzona sfera oprócz dwóch wcześniej wspomnianych komponentów wymaga również innych komponentów m.in.: *Sphere Collider* umożliwiającego detekcję kolizji oraz *Mesh Renderer* odpowiedzialnego za renderowanie modelu 3D.

1.2 Środowisko sprzętowe

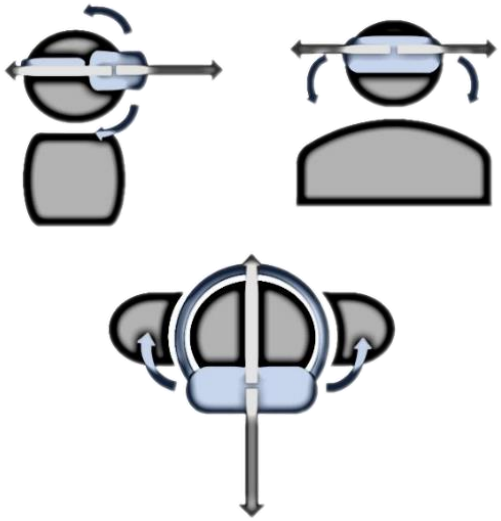
Najpopularniejszym urządzeniem pozwalającym doświadczyć wirtualnej rzeczywistości są gogle VR. Na rynku dostępnych jest wiele modeli od różnych producentów takich jak: Sony PlayStation VR, Valve Index VR, HTC Vive, HP Reverb oraz Facebook Oculus Quest. Gogle VR można podzielić na przewodowe oraz bezprzewodowe, znaczna większość dostępnych modeli jest przewodowa i wymaga do działania odpowiednio mocnego komputera lub konsoli. Można je również podzielić na dwie grupy pod względem rejestrowanych ruchów, gogle 3 DOF (*Three degrees of freedom),* trzy stopnie swobody, to takie, które rejestrują ruchy głową na boki i jej przechylenie, przykładem takich gogli są Samsung Gear VR.



Rys. 9. Reprezentacja 3 DOF[4]

Natomiast 6 DOF (*Six degrees of freedom),* sześć stopni swobody, oprócz śledzenia ruchów głową rejestrują również ruchy rękoma i kontrolerami oraz poruszanie się w przestrzeni np. pokoju. Przykładem takich gogli są Oculus Quest.

[4] https://www.mechatech.co.uk/journal/what-is-a-3dof-vs-6dof-in-vr, [dostęp: 13.05.2021r.].



Rys. 10. Reprezentacja 6 DOF[5]

Jako platformę docelową aplikacji zdecydowałem się wybrać Oculus Quest, których producentem jest Facebook. Mogą one działać zarówno w trybie bezprzewodowym, jak i w trybie przewodowym oraz rejestrują ruch 6 DOF.

Rys. 11. Zestaw Oculus Quest[6]

[5] https://www.mechatech.co.uk/journal/what-is-a-3dof-vs-6dof-in-vr, [dostęp: 13.05.2021r.].
[6] https://www.oculus.com/, [dostęp: 06.07.2021r.].

Gogle[7] zostały wydane w maju 2019 roku. Są wyposażone w procesor Qualcomm Snapdragon 835 z układem graficznym Adreno 540, który miał premierę w 2017 roku. Jest to procesor mobilny, który został użyty w takich *smartphonach* jak Samsung Galaxy S8 i Google Pixel 2. Oculus Quest został wyposażony w 4gb RAM oraz w zależności od wersji pamięciowej w 64 gb lub 128 gb pamięci wbudowanej, dodatkowo ma wbudowane głośniki stereo. Gogle zostały wyposażone w dwa panele OLED każdy o rozdzielczości 1440 × 1600 i posiadające odświeżanie obrazu na poziomie 72 Hz. Użytkownik ma do dyspozycji dwa kontrolery Oculus Touch. Każdy z kontrolerów składa się z 6 przycisków i joysticka. Przyciski znajdujące się z boku oraz na końcu kontrolera rejestrują, w jakim stopniu są wciśnięte. Z kolei pozostałe przyciski rejestrują jedynie dwa stany, wciśnięty lub nie.

[7] https://developer.oculus.com/learn/oculus-device-specs/, [dostęp: 9.04.2021r.].

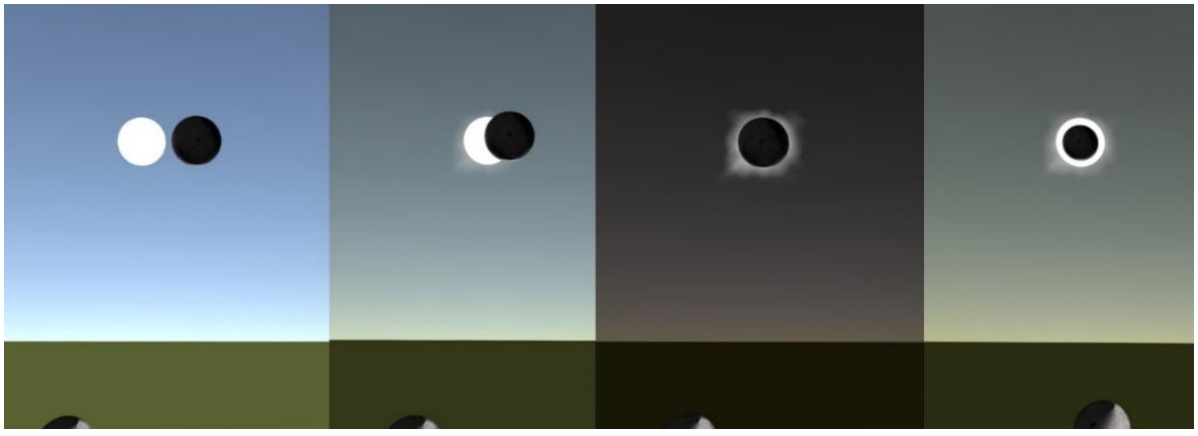
ROZDZIAŁ 2

PRZEDSTAWIENIE APLIKACJI

Gra składa się z sześciu różniących się od siebie scen: *Baza, Układ słoneczny – Animacja, Układ słoneczny – Symulacja, Grawitacja, Gwiazdozbiory* oraz *Zaćmienie Słońca*. Każdy z poziomów jest unikatowy i przedstawia graczowi inne zagadnienia z zakresu astronomii oraz fizyki. Celem poszczególnych poziomów jest ukazanie poruszanych zagadnień w przystępny sposób. Gracz ma do dyspozycji dwa kontrolery umożliwiające wchodzenie w interakcję z elementami otoczenia. Użytkownik ma również możliwość teleportowania się w dowolne miejsce na danym poziomie.

2.1 Zaćmienie słońca

Punktem odniesienia dla tej sceny był film na platformie Youtube autorstwa *Veritasium* pt. *Total Solar Eclipse (2017)*[8], w którym to przedstawiono przebieg całkowitego zaćmienia słońca. Scena została stworzona tak, aby jak najdokładniej odwzorować to zjawisko astronomiczne, co obejmuje m.in. stopniowe zapadanie zmroku oraz rzucanie przez Księżyc na Ziemię cienia.

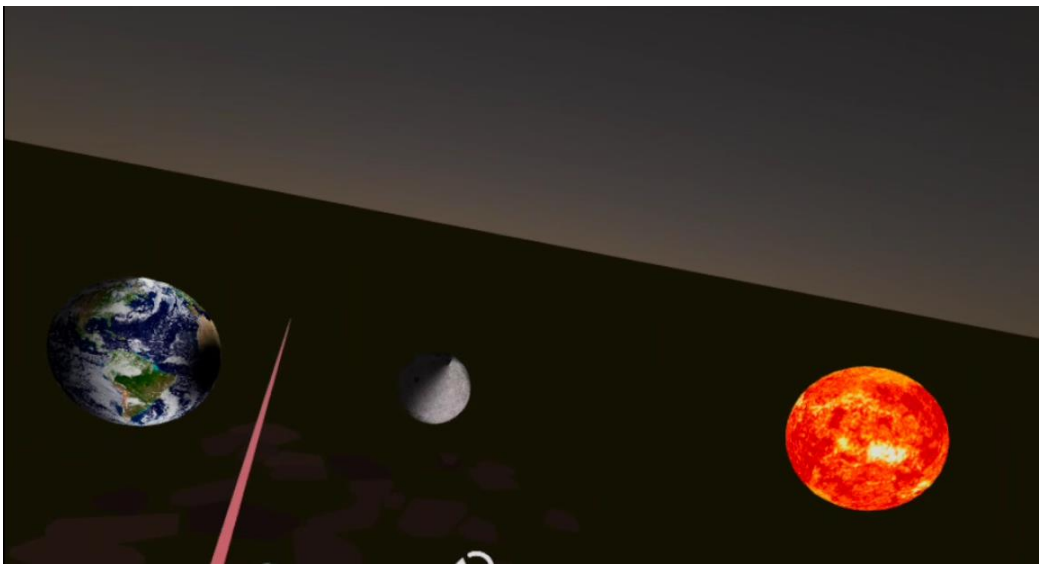


Rys. 12. Złożenie 4 kadrów z gry ukazujących zapadający zmrok oraz wygląd tarczy Słońca podczas zaćmienia

[8] https://youtu.be/G10m2ZZRH4U, [dostęp: 15.05.2021r.].

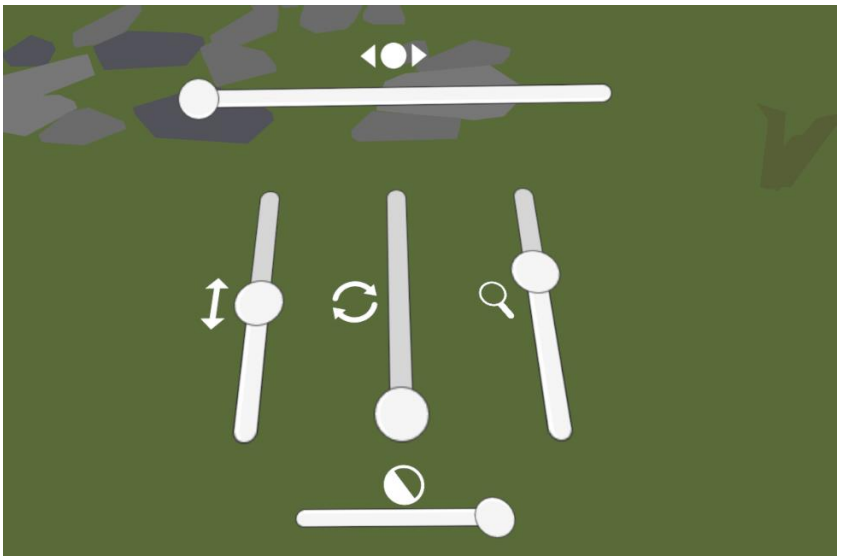
Rys. 13. Kadr z filmu *Total Solar Eclipse (2017)[3:08]*

Użytkownik ma do dyspozycji miniaturowy model odwzorowujący zaćmienie słońca, składający się z Ziemi, Księżyca oraz Słońca. W modelu nie ma zachowanych proporcji wielkości ciał niebieskich oraz odległości pomiędzy nimi. W momencie, gdy Księżyc nachodzi na tarczę Słońca, na modelu Ziemi pojawia się cień rzucany przez Księżyc. Model Księżyca widoczny na nieboskłonie jest sterowany przez użytkownika za pomocą joysticków na kontrolerach. Możliwość poruszania i obracania się na tej scenie jest wyłączona.

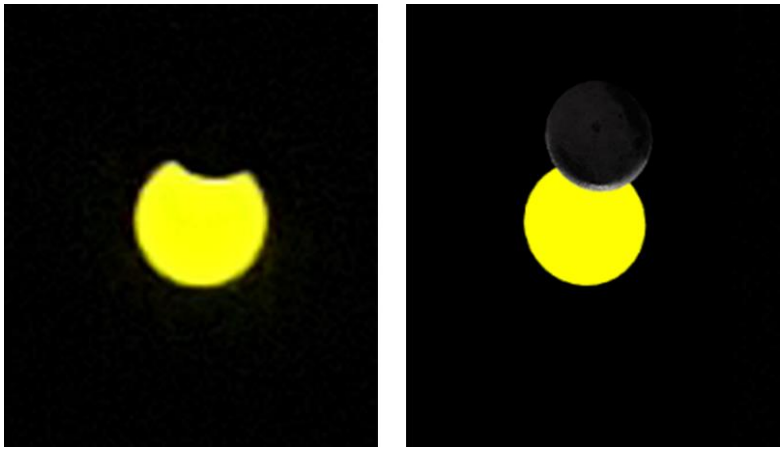


Rys. 14. Kadr z aplikacji przedstawiający model zaćmienia Słońca

Gracz ma do dyspozycji pięć suwaków, trzy pionowe suwaki umożliwiają dostosowanie modelu do swoich preferencji, odpowiadają za zmianę wielkości i odległości modelu od gracza oraz obracanie go wokół osi pionowej. Dwa poziome suwaki mają wpływ na wygląd zaćmienia Słońca, górny suwak pozwala na regulację odległości Księżyca od Ziemi, umożliwia to imitowanie niepełnego zaćmienia słońca, w którym to naturalny satelita Ziemi, nie zasłania całkowicie tarczy słońca. Dolny suwak jest odpowiedzialny za kolor nieba i Słońca.



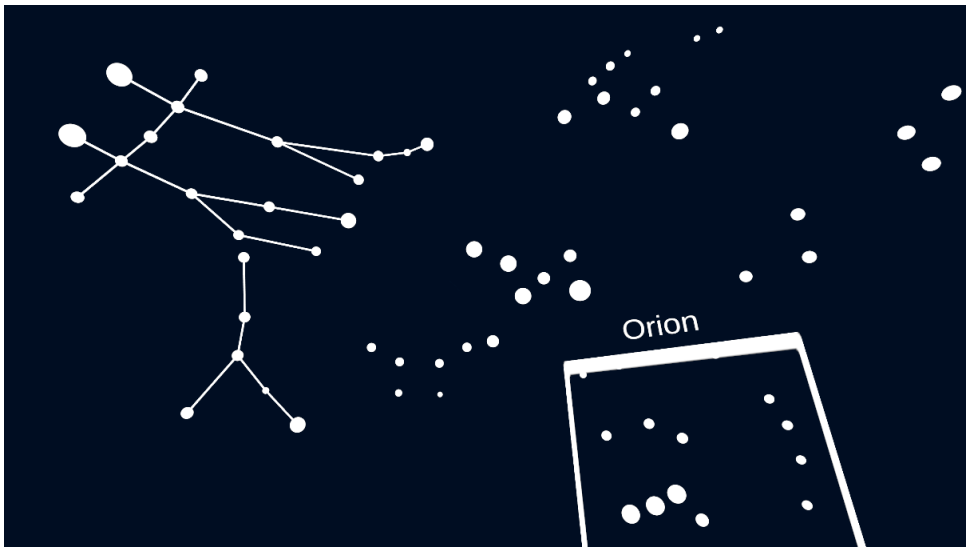
Rys. 15. Kadr z gry ukazujący suwaki do zmian właściwości modelu i zaćmienia Słońca



Rys. 16. Po lewej autorskie zdjęcie zaćmienia Słońca z dnia 10.06.2021r., po prawej kadr z aplikacji

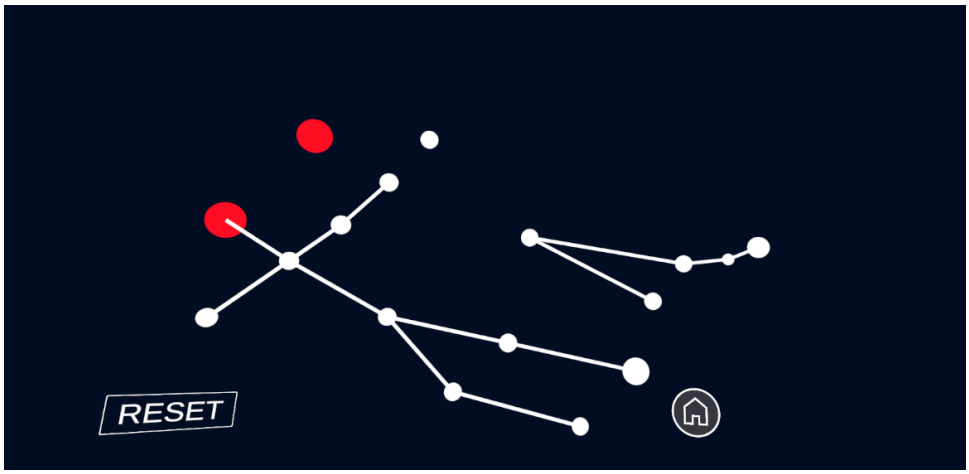
2.2 Gwiazdozbiory

Każdej nocy nad naszymi głowami znajdują się gwiazdy, które tworzą konstelacje zwane także gwiazdozbiorami. Mimo świadomości ich istnienia, niewiele ludzi zna ich kształty i nazwy. Ta scena ma na celu nauczenie użytkownika nazw oraz kształtów gwiazdozbiorów. Gracz na tym poziomie może przyciągnąć dowolny gwiazdozbiór oraz połączyć ze sobą gwiazdy. W celu przyciągnięcia, użytkownik musi nakierować laser w dowolną grupę i przyciągnąć ją za pomocą przycisku.



Rys. 17. Kadr z gry przedstawiający konstelacje w grze

Następnie za pomocą przycisku spustu może wybierać dwie gwiazdy, które jego zdaniem powinny być połączone. W rzeczywistości takie połączenia między gwiazdami nie istnieją, linie rysowane są tylko po to, aby gracz lepiej sobie mógł zwizualizować gwiazdozbiór.



Rys. 18. Kadr z gry ukazujący łączenie gwiazd konstelacji *Bliźnięta*

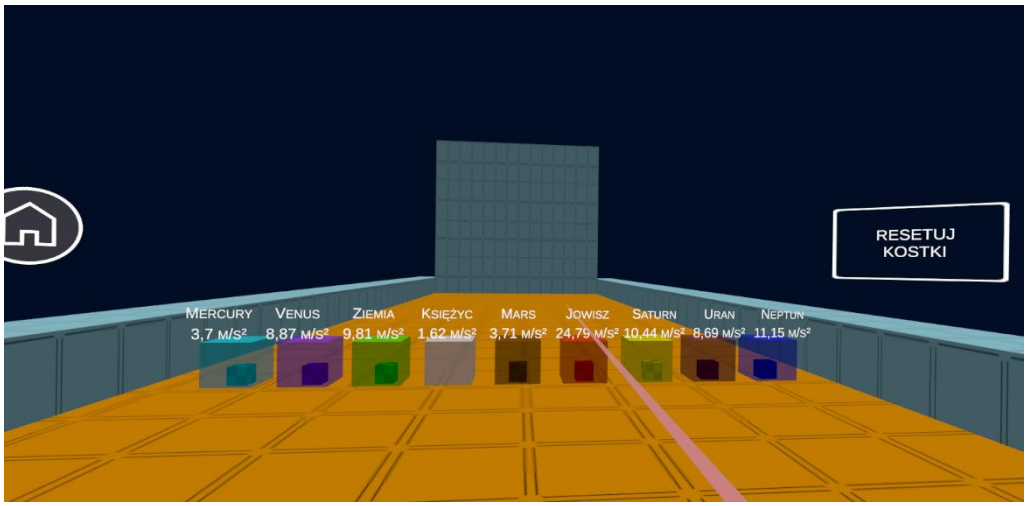
Stan gwiazdozbioru, kwestia tego, czy jest ułożony, czy nie, jest zapisywany i po ponownym włączeniu aplikacji stan jest odtwarzany. Jeśli gracz zechce, może zresetować stan poprzez kliknięcie przycisku *RESET*.

Pozycje konstelacji na nieboskłonie nie odpowiadają realnym położeniem na nocnym niebie, ale ich kształty i nazwy zostały oparte na prawdziwych układach przedstawionych w literaturze przedmiotu[9] i na portalach internetowych[10].

2.3 Grawitacja

Wszystkie obiekty we wszechświecie posiadające jakąkolwiek masę przyciągają do siebie inne obiekty obdarzone masą oraz one same są przyciągane. Im większa masa obiektu, tym bardziej przyciąga inne ciała[11]. W naszym układzie słonecznym obiektami posiadającymi największe masy są Słońce, planety i ich księżyce. W tej scenie chciałem dać możliwość doświadczalnego porównania między sobą grawitacji na różnych ciałach niebieskich w naszym układzie słonecznym. Użytkownik ma możliwość porównania grawitacji[12] pomiędzy: Merkurym, Wenus, Marsem, Ziemią, Księżycem, Jowiszem, Saturnem, Uranem i Neptunem. Na scenie znajduje się wiele stanowisk różniących się między sobą kolorem oraz napisami.

[9] J. Hermann, *Gwiazdy: Leksykon przyrodniczy*, Warszawa 1998.
[10] www.planetguide.net, www.constellation-guide.com, [dostęp: 09.03.2021r.].
[11] Halliday D., Resnick R., Walker J., Podstawy fizyki 2, PWN, Warszawa 2007, s. 27-29.
[12] https://phys.org/news/2016-01-strong-gravity-planets.html, [dostęp: 15.05.2021r.].



Rys. 19. Kadr z gry przedstawiający scenę *Grawitacja*

Użytkownik może podnosić, przyciągać do siebie oraz opuszczać kostki. Każda kostka reprezentuje inne przyśpieszenie grawitacyjne, kolor kostki jest uzależniony od tego, jakie ciało niebieskie reprezentuje, np. jeśli kostka ma kolor biało-czarny, będzie się zachowywać jak obiekt na powierzchni Księżyca.

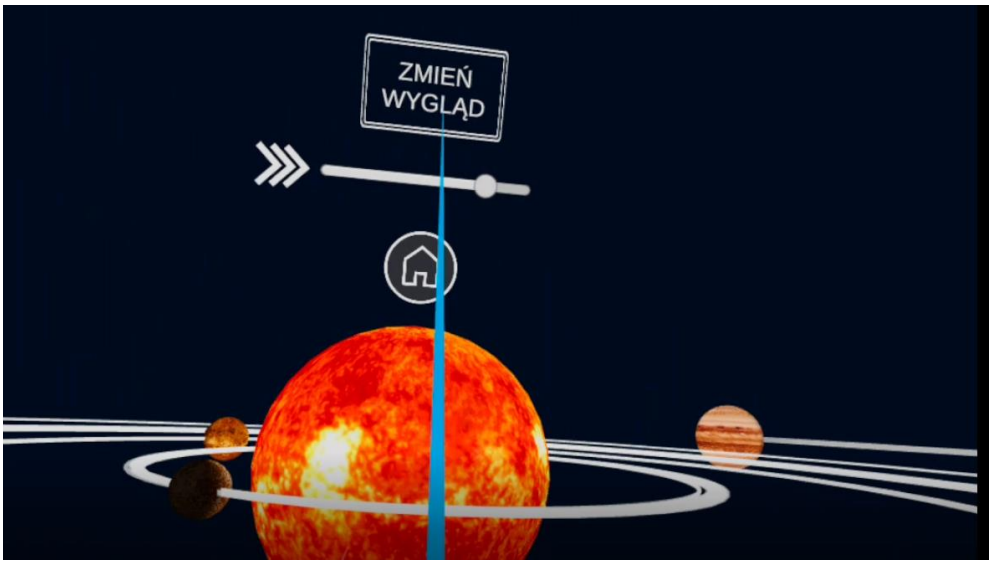
Gracz, biorąc do ręki biało-czarną kostkę reprezentującą Księżyc i drugą kostkę o kolorze czerwono-czarnym reprezentującym Jowisza, łatwo zauważy różnicę w tempie opadania kostek. Po podniesieniu obiektu joystick zmienia swoją funkcję i użytkownik może nim przyciągać lub obracać trzymany przedmiot.

2.4 Układ słoneczny – symulacja

W celu lepszego zrozumienia symulacji konieczne jest zaznajomienie się z najprostszą definicją Układu Słonecznego. Zgodnie ze Słownikiem Języka Polskiego jest to „*zespół ciał niebieskich złożony ze Słońca i obiegających dokoła niego ośmiu planet*”.[13]

W tej scenie użytkownikowi jest zaprezentowana symulacja fizyczna Układu Słonecznego. Pozwala to graczowi poznać kolejność planet, różnice w ich prędkościach orbitalnych oraz odległości od Słońca[14]. Białe linie reprezentują drogą przebytą przez każdą z nich.

[13] https://sjp.pwn.pl/sjp/Uklad-Sloneczny;2532335.html, [11.03.2021].
[14] https://space-facts.com/planet-orbits/, [16.03.2021r.].

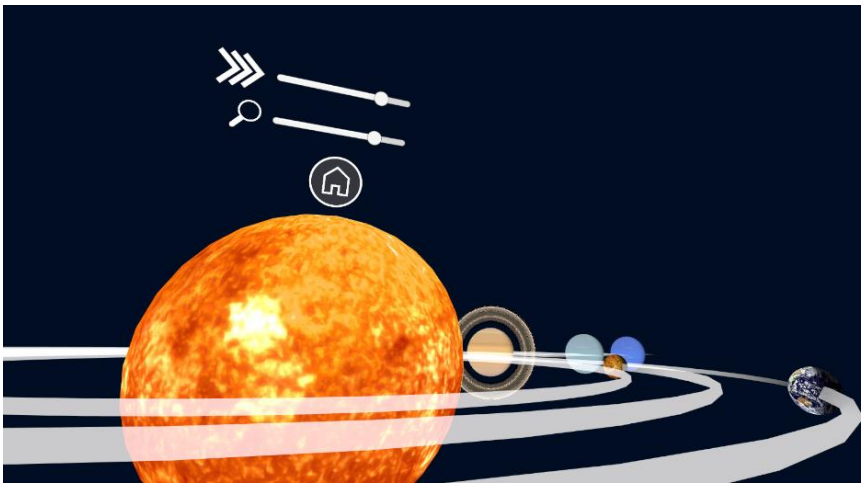


Rys. 20. Kadr z gry przedstawiający scenę *Układ Słoneczny – Symulacja*

Gracz może wejść w interakcję z układem poprzez suwak znajdujący się nad gwiazdą. Pozwala on na sterowanie prędkością symulacji. Drugą możliwą interakcją jest zmienienie sposobu prezentowania układu. Umożliwia ona zmianę odległości między planetami, dzięki czemu użytkownik ma lepiej zobrazowane porównanie prędkości poruszania się planet. Tę interakcję wywołuje się poprzez naciśnięcie przycisku *Zmień wygląd*.

2.5 Układ słoneczny – animacja

Ta scena jest podobna do omawianej poprzednio, natomiast różnica leży w sposobie realizacji Układu Słonecznego. Ten poziom został wykonany jako zwykła animacja, natomiast poprzedni opierał się na symulacji fizycznej ruchu. Cel jest taki sam: przedstawić użytkownikowi kolejność planet, różnice w prędkości obiegania Słońca oraz odległości od gwiazdy. W tej scenie oprócz możliwości kontrolowania prędkości animacji, użytkownik może również zmieniać jego wielkość.



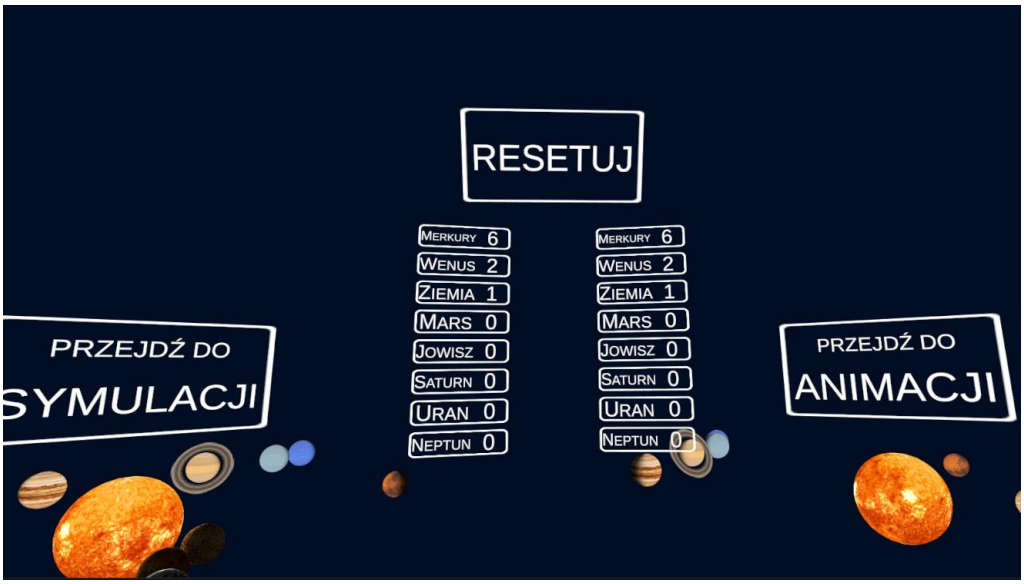
Rys. 21. Kadr z gry przedstawiający scenę *Układ Słoneczny – Animacja*

2.6 Baza

Ta scena jest najważniejszą sceną w aplikacji, bowiem to z tego miejsca użytkownik może przenieść się do omówionych wcześniej poziomów, bądź z powrotem – na wszystkich pozostałych poziomach jest przycisk pozwalający powrócić do *Bazy*.

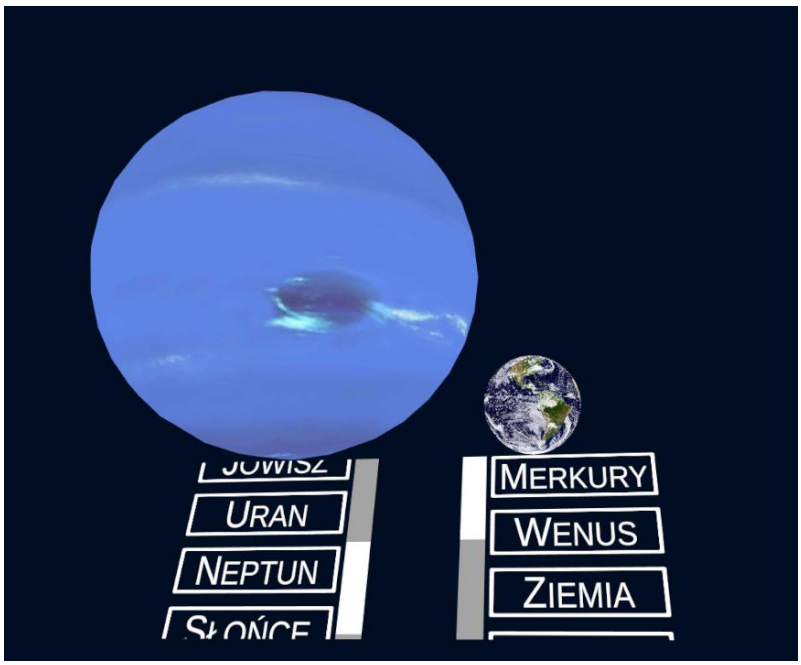
Rys. 22. Kadr z gry ukazujący przejścia do 3 kolejnych scen

Ma on również do zaoferowania dwa modele Układów Słonecznych, animacja oraz symulacja. Układy są ustawione blisko siebie, można więc śledzić poruszające się planety. Dodatkowo są pomiędzy nimi tablice z ilością pełnych obiegów wokół słońca każdej z planet.



Rys. 23. Kadr z gry przedstawiający porównanie Układów Słonecznych

Obok Układów Słonecznych znajduje się stanowisko, które służy do porównywania wielkości[15] oraz wyglądu[16] planet i Słońca (rysunek 24).



Rys. 24. Kadr z gry ukazujący porównanie planet

[15] https://nssdc.gsfc.nasa.gov/planetary/factsheet/, [dostęp: 16.03.2021r.]
[16] https://www.solarsystemscope.com/textures/, [dostęp: 28.10.2020r.]

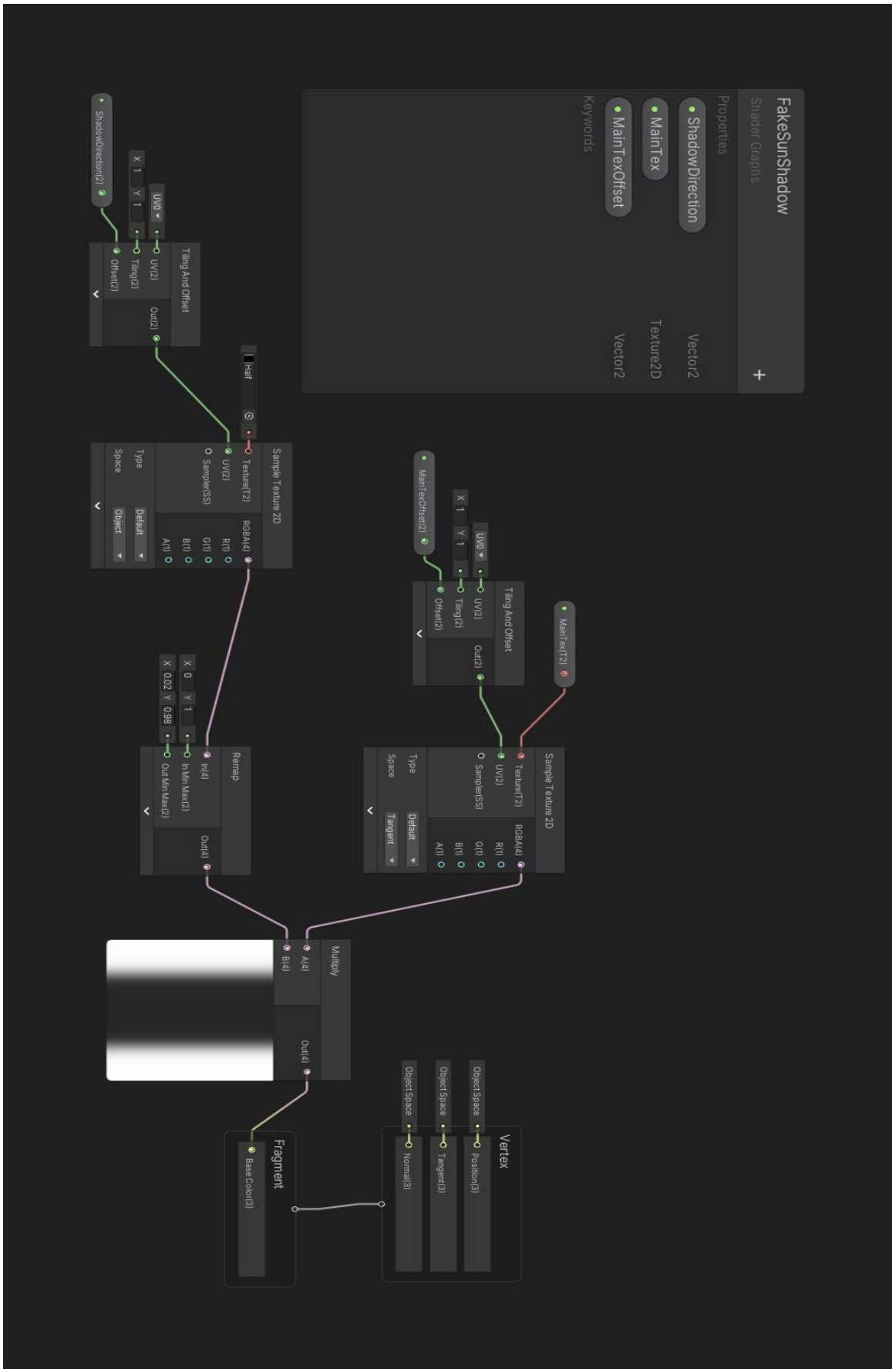
ROZDZIAŁ 3

IMPLEMENTACJA

Poniższy rozdział został poświęcony omówieniu szczegółów dotyczących scen, w jaki sposób one działają oraz jakie zostały wykorzystane techniki w celu uzyskania satysfakcjonującego rezultatu przy zachowaniu odpowiedniej płynności działania.

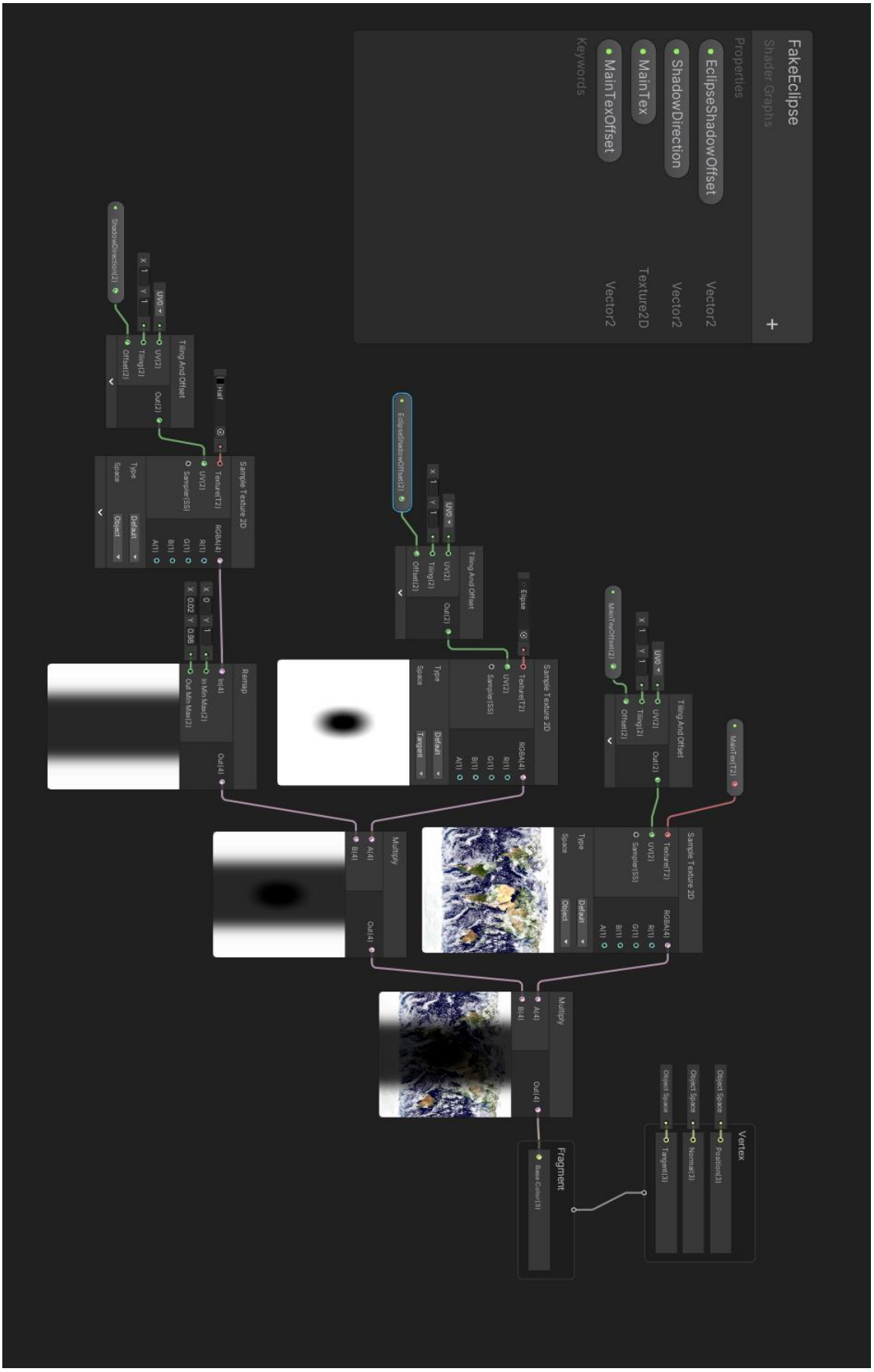
3.1 Optymalizacja światła

Planety w zaprezentowanych Układach Słonecznych sprawiają wrażenie jakby były oświetlane przez Słońce w centrum układu. Do uzyskania takiego efektu nie mogło być wykorzystane dynamiczne źródło światła, ponieważ jego użycie za bardzo obciążało podzespoły gogli. Uniemożliwiło to uzyskanie stabilnych 72 klatek na sekundę. W związku z czym został wykorzystany shader stworzony w Shader Graph ukazany na rysunku 25.



Rys. 25. Zdjęcie ukazujące implementacje shadera *FakeSunShadow*

W scenie *Zaćmienie Słońca* został ukazany dynamicznie zmieniający się cień na powierzchni Ziemi, który jest rzucany przez Księżyc podczas zaćmienia Słońca. Przez zbyt małą wydajność Oculus Quest nie można było zastosować dynamicznego światła. Dlatego do imitacji tego zjawiska został zmodyfikowany shader *FakeSunShadow*. Została dodana tekstura elipsy, która miała imitować cień rzucany przez Księżyc na powierzchnię Ziemi, której położenie jest kontrolowane przez zmienną *EclipseShadowOffset*.



Rys. 26. Zdjęcie ukazujące implementacje shadera *FakeEclipse*

Drugą częścią rozwiązania było stworzenie skryptu odpowiedzialnego za kontrolowanie pozycji cienia na modelu Ziemi. W tym celu została stworzona metoda *EclipseShadowUpdate()* wchodząca w skład klasy *SolarEclipseModelController*, której fragment został zaprezentowany w tabeli 2. Funkcja na podstawie pozycji Księżyca i punktu, w którym następuje pełne zaćmienie Słońca, oblicza, jak bardzo Księżyc zakrywa tarczę Słońca. Finalnie funkcja *SetVector()* przypisuje obliczony wektor przesunięcia tekstury do zmiennej *_EclipseShadowOffset*.

Tabela 2. Fragment klasy *SolarEclipseModelController*

```
void EclipseShadowUpdate()
{
    Vector2 direction = (SolarEclipseController.Instance.perfectEclipsePosition –
                            SolarEclipseController.Instance.moon.position).normalized;

    if (SolarEclipseController.Instance.deltaPosition <
                                    SolarEclipseController.Instance.triggerDistance)
    {
        float calculated = SolarEclipseController.Instance.deltaPosition /
                            SolarEclipseController.Instance.triggerDistance * 0.5f;

        float calculatedY = calculated * direction.y;

        calculatedY = Mathf.Clamp(calculatedY, -.5f, .5f);

        earthMaterial.material.SetVector("_EclipseShadowOffset",
                            new Vector2(calculated * -direction.x, calculatedY));
    }

    else
    {
        earthMaterial.material.SetVector("_EclipseShadowOffset",
                    new Vector2(0.5f * direction.x, 0.5f * Mathf.Sign(direction.y)));
    }
}
```

Rysunek 27. przedstawia porównanie w ilości klatek na sekundę. Po lewej przy użyciu shadera *FakeEclipse*, po prawej z wykorzystaniem dynamicznego światła.

Rys. 27. Kadry z gry ukazujący różnice w ilości wyświetlanych klatek na sekundę

3.2 Narzędzie do tworzenia konstelacji

Każda konstelacja składa się z następujących danych: nazwa konstelacji, ilość gwiazd, pozycja gwiazd, wielkość gwiazd, połączenia między gwiazdami, wysokość nazwy konstelacji, pozycja na nieboskłonie oraz środek i wielkość obrysu. Do serializacji tych danych została stworzona klasa *SOConstelationBase* (tabela 3.) wykorzystująca *SciptableObject*, czyli wbudowany w Unity kontener na dane.

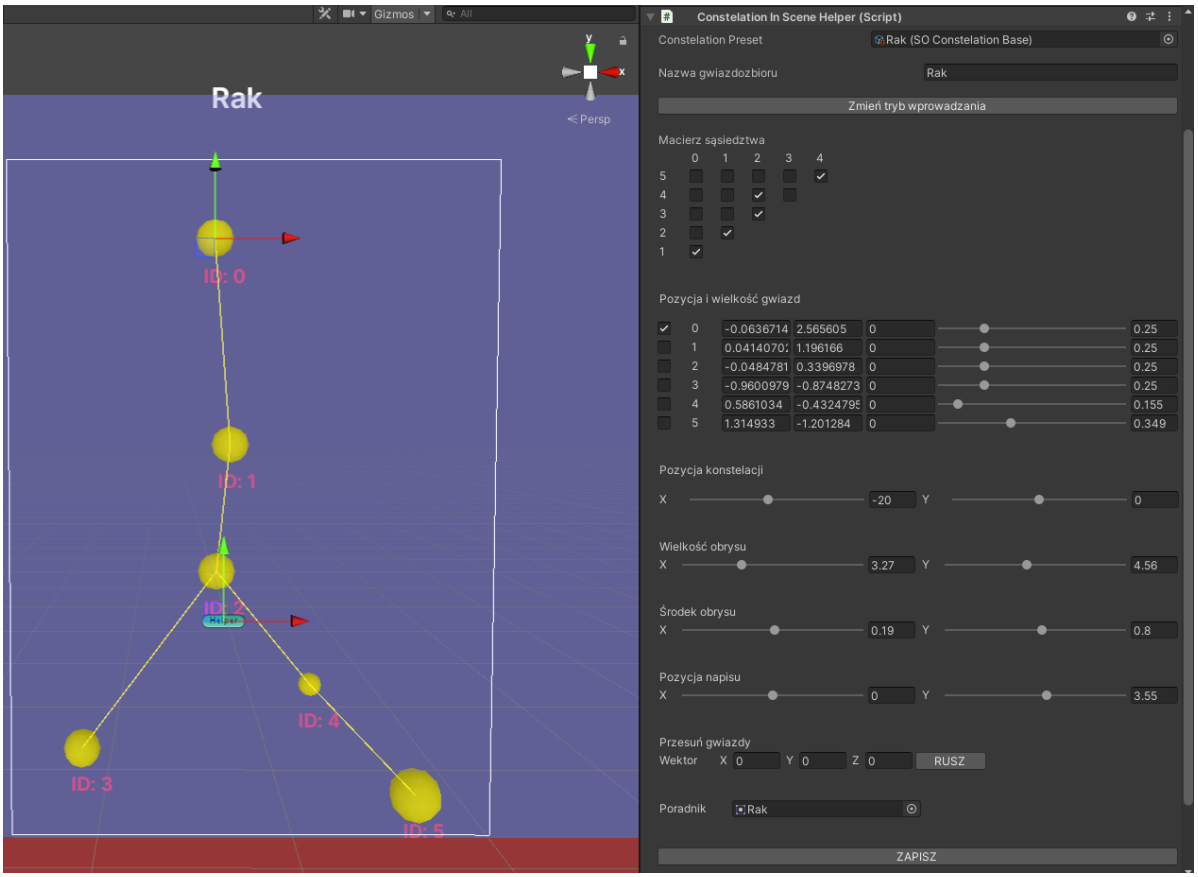
Tabela 3. Klasa *SOConstelationBase*

```
public class SOConstelationBase : ScriptableObject
{
    public string conName;
    public bool[] adjMatrix;
    public Node[] nodes;
    public int size;
    public Vector2 skyPosition;
    public Vector2 boundCenter;
    public Vector2 boundSize;
    public Vector2 titleHeight;
    public Sprite guideImage;
}
```

Domyślne okno *Inspector* pozwalające na modyfikowanie powyższych danych było mało czytelne i uniemożliwiało oglądanie zmian na bieżąco. Unity pozwala na tworzenie własnych narzędzi do edytora oraz rozwijanie już istniejących. W tym celu zostały stworzone klasy *ConstelationInSceneHelper* oraz rozszerzająca ją klasa

ConstelationInSceneHelperEditor. Druga klasa nadpisuje domyślny wygląd okna *Inspector* oraz umożliwia podgląd zmian na bieżąco.

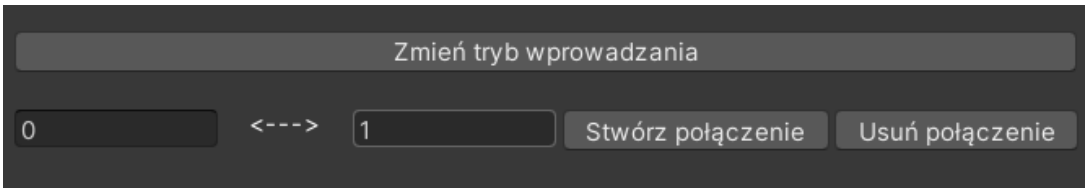
Do stworzenia logiki połączeń między gwiazdami została wykorzystana znana z teorii grafów macierz sąsiedztwa[17]. Macierz sąsiedztwa widoczna na rysunku 28. została stworzona dzięki możliwości dostosowania wyglądu okna *Inspector*.



Rys. 28. Zdjęcie z edytora Unity ukazujące budowę komponentu *ConstelationInSceneHelper*

Przy większej ilości gwiazd w konstelacji, macierz sąsiedztwa może stać się nieczytelna, dlatego istnieje alternatywna metoda tworzenia oraz usuwania połączeń dostępna po kliknięciu przycisku *Zmień tryb wyświetlania*. W miejscu macierzy sąsiedztwa pojawiają się dwa pola, w które należy wpisać wybranie numery gwiazd. Po wpisaniu poprawnych numerów, aktywują się dwa przyciski, *Stwórz połączenie* oraz *Usuń połączenie*.

[17] Cormen T.H., Leiserson C.E., Rivest R.L., Wprowadzenie do algorytmów Warszawa 1997 s. 526-529.



Rys. 29. Zdjęcie z edytora Unity ukazujące fragment komponentu *ConstelationInSceneHelper*

3.3 Symulacja fizyczna Układu Słonecznego

Do stworzenia symulacji fizycznej naszego Układu Słonecznego zostało wykorzystane *Prawo powszechnego ciążenia*, które opisuje, w jaki sposób dowolne dwa obiekty posiadające masę oddziałują na siebie. Równanie pozwalające obliczyć siłę oddziaływania grawitacyjnego wyraża się wzorem[18]:

$$\vec{F} = \frac{G*M*m}{R^2}\vec{r}_i \qquad (1)$$

gdzie: G – to stała grawitacyjna [$N \cdot m^2/kg^2$]

M, m – ciężary obiektów [N]

R – odległość między obiektami [m]

$\vec{r}_i$ – kierunek i zwrot, w którym obiekt jest przyciągane

Wzór 1 został wykorzystany w klasie *SimGravityObject* (tabela 4.), w metodzie *UpdateVelocityPlanet()*. W pierwszej kolejności należy obliczyć odległości między obiektami i podnieść ją do kwadratu. Następnym krokiem jest wyliczenie kierunku oddziaływania siły i wprowadzenie obliczonych wyników do równania grawitacyjnego. W celu możliwości kontrolowania szybkości odgrywania symulacji obliczona wartość jest mnożona przez krok czasowy. Aby utrzymać stabilność symulacji, planety nie oddziałują między sobą, ponieważ w przeciwnym razie planety wypadałyby ze swoich kołowych orbit. Następną metodą w omawianej klasie jest *UpdatePosition(),* której zadaniem jest poruszenie obiektu poprzez dodanie do aktualnej pozycji, obliczonej wartości nowej prędkości.

[18]D. Halliday, R. Resnick, J. Walker, *Podstawy fizyki 2*, Warszawa 2007, s. 27-29.

Tabela 4. Klasa *SimGravityObject*

```
public class SimGravityObject : MonoBehaviour
{
    public BodyType bodyType = BodyType.Planet;
    public Rigidbody rb;
    public SimGravityObject relativeTo;
    public Vector3 initialVelocity;
    public Vector3 velocity;

    private void Awake()
    {
        rb = GetComponent<Rigidbody>();
        velocity = initialVelocity;
    }

    public void UpdateVelocity(SimGravityObject[] allBodies)
    {
        if (bodyType == BodyType.Static)
            return;

        foreach (var otherBody in allBodies)
        {
            if (otherBody.bodyType == BodyType.Static)
            {
                if (otherBody != this)
                {
                    float sqrDistance = (otherBody.rb.position - rb.position).sqrMagnitude;

                    Vector3 forceDirection = (otherBody.rb.position –
                                                            rb.position).normalized;

                    Vector3 acceleration = Constant.constG * otherBody.rb.mass /
                                                        sqrDistance * forceDirection;

                    velocity += acceleration * SimController.Instance.timeStep;
                }
            }
        }
    }

    public void UpdatePosition()
    {
        rb.MovePosition(rb.position + velocity * SimController.Instance.timeStep);
    }
}
```

Metody *UpdatePosition()* i *UpdateVelocity()* są wywoływane przez klasę *SimController* będącą kontrolerem całej symulacji. Wraz z początkiem sceny, należy wyszukać wszystkie obiekty posiadające klasę *SimGravityObject* i utworzyć z nich tablicę. Do obliczeń związanych z fizyką, Unity zaleca wykorzystanie metody *FixedUpdate()* (tabela 5.), która jest niezależna od ilości wyświetlanych klatek i wynosi stałe 72 Hz. W ciele funkcji najpierw jest wykonywana iteracja po tablicy wywołując metodę *Move()*, a następnie ponownie przechodząc przez tablice jest aktualizowana pozycja poprzez wywołanie metody *UpdatePosition()*.

Tabela 5. Fragment klasy *SimController*

```
private SimGravityObject[] bodies;

private void FixedUpdate()
{
    for (int i = 0; i < bodies.Length; i++)
    {
        bodies[i].UpdateVelocity(bodies);
    }

    for (int i = 0; i < bodies.Length; i++)
    {
        bodies[i].UpdatePosition();
    }
}
```

Bez nadania prędkości inicjalizującej ruch, planety wpadały by wprost w Słońce. W tym celu została stworzona klasa *SimCalculateAcceleration,* która oblicza wymaganą prędkość. Prędkość, która zapewnia orbitę kołową, jest znana jako *Pierwsza prędkość kosmiczna*, wyrażana wzorem[19]:

$$v = \sqrt{\frac{G*M}{R}} \tag{2}$$

gdzie: G – to stała grawitacyjna [$N \cdot m^2/kg^2$]

M – ciężar obiektu [N]

R – odległość od obiektu [m]

Wzór 2. wykorzystano w metodzie *Force()* (tabela 6.), obliczona prędkość jest dodatkowo przemnożona przez wektor kierunkowy, który jest prostopadły do kierunku siły grawitacji. W rezultacie planeta ma prędkość umożliwiającą jej ruch po stabilnej orbicie wokół Słońca.

Tabela 6. Fragment klasy *SimCalculateAcceleration*

```
public void Force()
{
    float sqrDst = (otherBody.position - rb.position).magnitude;

    Vector3 forceDir = (otherBody.position - rb.position).normalized;

    float acceleration = Mathf.Sqrt(Constant.constG * otherBody.mass / sqrDst);
```

[19] https://zpe.gov.pl/a/predkosci-kosmiczne/DWfNHXUxB, [dostęp: 02.06.2021r.].

```
    Vector3 initialDir = new Vector3(forceDir.z, 0, -forceDir.x).normalized;

    GetComponent<SimGravityObject>().initialVelocity = initialDir * acceleration;
}
```

3.4 Animacja Układu Słonecznego

Orbity planet ukazane w aplikacji są koliste, dlatego do stworzenia animacji został zastosowany wzór na prędkość w ruchu jednostajnym dla ruchu po okręgu[20], który prezentuje się następująco:

$$v = \frac{2*\Pi}{T} \tag{3}$$

gdzie: T – czas potrzebny na okrążenie Słońca [s]

Wzór 3. został wykorzystany w klasie *SolarSystemOrbitController* w metodzie *UpdatePosition()* (tabela 7.). Wynik działania jest mnożony przez zmienną umożliwiającą sterowanie szybkością animacji. Następnie obliczona wartość jest dodawana do zmiennej *angle*, która oznacza, ile radianów wokół Słońca wykonała planeta.

Położenie na okręgu można wyrazić z wykorzystaniem funkcji trygonometrycznych *sinus* oraz *cosinus.* W tym celu są inicjowane zmienne *x* oraz z, które przechowują informację o położeniu planety w osi x oraz w osi z.

Tabela 7. Fragment klasy *SolarSystemOrbitController*

```
public float radius = 1f;
public float speed = 1f;

private float angle;
private float calculatedSpeed = 1f;
private float x;
private float z;

public void UpdatePosition()
{
    if (speed == 0f)
        return;

    calculatedSpeed = (2f * Mathf.PI) / (1f / speed) *
                                        SolarSystemController.Instance.speedMultiplier;
```

[20] D. Halliday, R. Resnick, J. Walker, *Podstawy fizyki 1*, Warszawa 2006, s. 71.

```
    angle += calculatedSpeed * Time.deltaTime;

    x = Mathf.Cos(angle) * radius;

    z = Mathf.Sin(angle) * radius;

    transform.localPosition = new Vector3(x, transform.localPosition.y, z);

    orbitsCount = Mathf.FloorToInt(angle / (Mathf.PI * 2f));
}
```

Metodę *UpdatePosition()* zawartą w tabeli 7., wywołuje klasa *SolarSystemController* w metodzie *FixedUpdate()* (tabela 8.), poprzez iterację po każdym obiekcie w tablicy *bodies*.

Tabela 8. Fragment klasy *SolarSystemController*

```
private SolarSystemOrbitController[] bodies;

private void FixedUpdate()
{
    for (int i = 0; i < bodies.Length; i++)
    {
        bodies[i].UpdatePosition();
    }
}
```

3.5 Porównanie symulacji z animacją

W scenie *Baza* symulacja fizyczna oraz animacja są ustawione obok siebie, pomiędzy nimi znajduje się tabela z informacją, ile razy każda z planet obiegła Słońce od momentu wczytania się sceny. Dla każdego z Układów Słonecznych sposób liczenia pełnych obiegnięć jest inny.

Porównanie ma na celu sprawdzenie zgodności stworzonego modelu fizycznego z rzeczywistymi danymi[21] dotyczących obiegu planet wokół Słońca, na podstawie których została stworzona animacja.

Dla planet z animacji Układu Słonecznego ilość obiegnięć Słońca jest obliczana poprzez podzielenie aktualnego kąta, wyrażonego w radianach pomiędzy planetą a Słońcem, przez

[21] https://nssdc.gsfc.nasa.gov/planetary/factsheet/, [dostęp: 16.03.2021r.]

liczbę 2 * Π, czyli pełny okrąg wyrażony w radianach. Ostatecznie by liczyć jedynie pełne okrążenia wokół Słońca wynik ilorazu jest zaokrąglany w dół.

Tabela 9. Fragment metody *UpdatePosition()* z klasy *SolarSystemOrbitController*

```
orbitsCount = Mathf.FloorToInt(angle / (Mathf.PI * 2f));
```

Do zliczania okrążeń w symulacji fizycznej Układu Słonecznego została stworzona klasa *SimCalculateOrbitsCount* (tabela 10.) oraz wykorzystano wbudowaną w Unity funkcję *Physics.Raycast()*[22]. Umożliwia ona detekcję kolizji z innymi obiektami poprzez wystrzeliwanie promienia. Jako argumenty przyjmuje długość, zwrot i długość wystrzeliwanego promienia oraz maskę szukanego obiektu. Jeśli prosta trafi w odpowiedni obiekt- w naszym przypadku Słońce, funkcja zwraca prawdę, w innym przypadku fałsz. W celu uniknięcia kilkukrotnego naliczenia jednego pełnego okrążenia została stworzona zmienna *alreadyHit*.

Tabela 10. Klasa *SimCalculateOrbitsCount*

```
public class SimCalculateOrbitsCount : IOrbitsCount
{
    public LayerMask layerMask;
    private bool alreadyHit = true;

    void FixedUpdate()
    {
        if (Physics.Raycast(transform.position, Vector3.left, Mathf.Infinity, layerMask))
        {
            if (alreadyHit == false)
                orbitsCount++;
            alreadyHit = true;
        }
        else
        {
            alreadyHit = false;
        }
    }
}
```

[22]https://docs.unity3d.com/ScriptReference/, [dostęp: 16.05.2021r.]

3.6 Proces testowania

Do debugowania i testowania aplikacji wykorzystałem możliwość podłączenia Oculus Quest do komputera za pomocą przewodu USB-C. Edytor Unity wykryje podłączone urządzenie i po kliknięciu przycisku *Play* zacznie przesyłać obraz do gogli.

Alternatywnym sposobem testowania aplikacji jest wykorzystanie oficjalnej darmowej paczki do Unity *XR Device Simulator*[23]. Jest ona symulatorem gogli VR, który umożliwia testowanie aplikacji bez konieczności podłączania urządzenia do komputera. Ma taką samą funkcjonalność jak gogle VR, pozwala na obracanie kamerą i w pełni działające odpowiedniki kontrolerów.

[23]https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.10/api/UnityEngine.XR.Interaction.Toolkit.Inputs.Simulation.XRDeviceSimulator.html, [dostęp: 13.03.2021r.]

ZAKOŃCZENIE

Celem niniejszej pracy inżynierskiej było stworzenie aplikacji VR na urządzenie Oculus Quest, która przedstawiałaby zagadnienia z obszaru fizyki i astronomii w prosty i czytelny sposób. Napisana aplikacja daje użytkownikowi do dyspozycji 6 scen, każda przedstawiająca inne elementy. Gra jest w pełni interaktywna, użytkownik może m.in. podnosić i rzucać obiektami, zmieniać skalę i prędkość.

Wybrane środowisko Unity było dobrym wyborem, ponieważ zapewniło mi wszystkie potrzebne narzędzia do stworzenia kompletnej aplikacji VR. Dodatkowo podczas tworzenia projektu, Unity zaktualizowało udostępnione biblioteki, co umożliwiło wprowadzenie kilku nowych elementów.

Problematyczna okazała się wydajność urządzenia Oculus Quest, która była niewystarczająca w scenach wymagających dynamicznych źródeł światła. Dzięki zaimplementowaniu własnych shaderów udało mi się jednak przezwyciężyć to ograniczenie i stworzyć scenę ukazującą zjawisko zaćmienia Słońca.

W przyszłości planuję rozwinąć aplikację poprzez dodanie nowych scen ukazujących nowe zagadnienia z innych obszarów nauki, nie tylko z fizyki i astronomii oraz dodanie krótkiego wprowadzenia do gry. Ponadto chciałbym ją udostępnić w oficjalnym sklepie Oculus, by każdy chętny mógł ją przetestować za darmo.

BIBLIOGRAFIA

Cormen T.H., Leiserson C.E., Rivest R.L., *Wprowadzenie do algorytmów* Warszawa 1997 s. 526-529.

Halliday D., Resnick R., Walker J., *Podstawy fizyki 1*, Warszawa 2006.

Halliday D., Resnick R., Walker J., *Podstawy fizyki 2*, Warszawa 2007.

Herrman J., *Leksykon Przyrodniczy Gwiazdy*, Warszawa 1998.

NETOGRAFIA

https://assetstore.unity.com/, [dostęp: 2.06.2021r.].

https://developer.oculus.com/learn/oculus-device-specs/, [dostęp: 9.04.2021r.].

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.10/api/UnityEngine.XR.Interaction.Toolkit.Inputs.Simulation.XRDeviceSimulator.html, [dostęp: 13.03.2021r.].

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@1.0/manual/index.html, [dostęp: 2.06.2021r.].

https://docs.unity3d.com/ScriptReference/, [dostęp: 16.05.2021r.].

https://nssdc.gsfc.nasa.gov/planetary/factsheet/, [dostęp: 16.03.2021r.].

https://phys.org/news/2016-01-strong-gravity-planets.html, [dostęp: 19.05.2021r.].

https://sjp.pwn.pl/sjp/Uklad-Sloneczny;2532335.html, [11.03.2021].

https://space-facts.com/planet-orbits/, [dostęp: 16.03.2021r.].

https://unity.com/products/unity-platform, [dostęp: 25.04.2021r.].

https://www.constellation-guide.com/constellation-list/, [dostęp: 09.03.2021r.].

https://www.mechatech.co.uk/journal/what-is-a-3dof-vs-6dof-in-vr, [dostęp: 13.05.2021r.].

https://www.mordorintelligence.com/industry-reports/virtual-reality-market, [dostęp: 8.06.2021r.].

https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-an-eclipse-58, [dostęp: 10.03.2021r.].

https://www.oculus.com/, [dostęp: 06.07.2021r.].

https://www.planetguide.net/, [dostęp: 09.03.2021r.].

https://www.solarsystemscope.com/textures/, [dostęp: 28.10.2020r.].

https://youtu.be/G10m2ZZRH4U, [dostęp: 15.05.2021r.].

https://zpe.gov.pl/a/predkosci-kosmiczne/DWfNHXUxB, [dostęp: 2.06.2021r.].