POLITECHNIKA GDAŃSKA | WYDZIAŁ FIZYKI TECHNICZNEJ I MATEMATYKI STOSOWANEJ

Imię i nazwisko studenta: Katarzyna Wojewoda
Nr albumu: 161599
Poziom kształcenia: Studia drugiego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Matematyka
Specjalność: Geometria i grafika komputerowa

## PRACA DYPLOMOWA MAGISTERSKA

Tytuł pracy w języku polskim: Projekt wirtualnego laboratorium

Tytuł pracy w języku angielskim: Project of virtual laboratory

Opiekun pracy: dr inż. Jakub Maksymiuk

Data ostatecznego zatwierdzenia raportu podobieństw w JSA:

**Streszczenie**

Celem tego projektu było stworzenie komputerowej gry edukacyjnej, w której użytkownik może wykonać pomiary wartości fizycznych oraz dokonać rzeczywistych obliczeń na ich podstawie. Program został zbudowany na silniku Unity3D, natomiast wszystkie modele 3D zostały przygotowane w programie Blender. W wirtualnym laboratorium użytkownik może wykonać doświadczenia fizyczne z zakresu dynamiki oraz mechaniki, których poziom został dostosowany dla uczniów szkół ponadpodstawowych. Badana jest zasada zachowania energii na równi pochyłej, współczynnik sprężystości linek przy pomocy siłomierza oraz wpływ zmian długości wahadła matematycznego na jego okres. Każdy eksperyment znajduje się w osobnym pokoju, po którym użytkownik może poruszać się przy pomocy klawiatury oraz myszy. Projekt może być dalej uzupełniany o dodatkowe parametry oraz o kolejne doświadczenia.

**Słowa kluczowe:** grafika komputerowa, modelowanie 3D, Blender, Unity 3D, technologie komputerowe

**Dziedzina nauki i techniki, zgodnie z wymogami OECD:** Nauki o komputerach i informatyka

**Abstract**

The goal of this project was to create a computer educational game in which the user can measure physical values and make real calculations based on them. This program was built in the Unity3D engine, while all 3D models were prepared in Blender. In the virtual laboratory, the user can perform physical experiments in the field of dynamics and mechanics, the level of which has been adapted for high school students. The user can study the law of conservation of energy on an inclined plane, the elasticity coefficient of the lines using a dynamometer and what impact on mathematical pendulums period does the change of its length have. Each experiment is located in a separate room, where the user can move around using the keyboard and mouse. The project may be further developed with more parameters and with additional experiments.

# Spis treści

# 1 Wstęp i cel pracy

Celem pracy było stworzenie projektu wirtualnego laboratorium fizycznego dla uczniów szkół podstawowych oraz jego implementacja w formie gry komputerowej opartej na silniku Unity 3D. Motywacją do przygotowania tego projektu była obowiązująca, w czasie planowania pracy, edukacja zdalna. Uniemożliwiła ona prowadzenie zajęć w szkołach, a w następstwie korzystanie z pracowni fizyczych.

Wykonywanie doświadczeń powinno być fundamentalnym elementem lekcji fizyki w szkole podstawowej, jak i ponadpodstawowej. Fizyka jest nauką przyrodniczą, więc poznawanie jej należałoby opierać na rzeczywistym obserwowaniu opisywanych zjawisk. Stąd projekt ten powstał, aby umożliwić wejście do, tym razem, wirtualnej pracowni. Co więcej, w trakcie lekcji trudno jest znaleźć czas na to, aby każdy z uczniów osobiście wykonał każde doświadczenie. Dodatkową zaletą progamu jest to, że może on być dostępny również poza zajęciami, aby umożliwić przeprowadzanie pomiarów w warunkach domowych. Dlatego też każde doświadczenie zawarte w tym projekcie posiada szczegółową instrukcję, wyjaśnienie badanych zjawisk oraz wyprowadzenie niezbędnych wzorów. Tak aby było zrozumiałe bez wsparcia ze strony nauczyciela.

Dodatkową motywacją była również chęć popularyzacji nauki wśród młodzieży, której zainteresowania obejmują w dużej mierze gry komputerowe. Na rynku znajduje się wiele gier edukacyjnych dla najmłodszych użytkowników, brakuje natomiast programów dla odbiorców posiadających już pewną wiedzę. Wirtualne laboratorium można uznać więc za pewnego rodzaju szkic rozwiązania, które mogłoby zapełnić tę lukę.

## 1.1 Założenia projektu i zadania do wykonania

Pierwotny projekt zakładał dostosowanie poziomu trudności doświadczeń dla uczniów szkół podstawowych. Po dokładniejszym przeanalizowaniu obecnie obowiązującej podstawy programowej, ostatecznie grupą odbiorców zostali uczniowie liceów oraz techników. Spowodowane było to również chęcią umożliwienia wykonywania rzeczywistych pomiarów i obliczeń na ich podstawie, co wprowadzane jest w późniejszych latach edukacji. Ponadto zmiana grupy docelowej pozwoliła na zaimplementowanie ciekawszych i atrakcyjniejszych funkcjonalności programu. Początkowo projekt zakładał, że zostanie przedstawione pięć różnych doświadczeń. To założenie zostało jednak zrewidowane z powodu ograniczeń czasowych. Ostatecznie liczba ta została ograniczona do trzech oraz skupiono się na dwóch działach fizyki - mechanice i dynamice. Zgodnie ze wstępnymi założeniami projekt miał być zbudowany na wirualnej rzeczywistości i dostępny przez okulary VR. Także to założenie zostało zmodyfikowane. Obrany został kierunek gry z widokiem trzecioosobowym z powodu ograniczeń sprzętowych końcowego użytkownika oraz będących do dyspozycji podczas przygotowywania projektu.

Poszczególne prace oraz metody rozwiązań zostaną opisane w kolejnych rozdziałach. Zaplanowane zostały następujące zadania:

- przygotowanie scenariusza,
- wykonanie niezbędnych modeli 3D,
- konfiguracja środowiska Unity,
- projekt aplikacji,
- implementacja eksperymentów,
- testy.

# 2 Projekt

## 2.1 Środowisko

Wszystkie modele zawarte w projekcie zostały przygotowane w programie do modelowania 3D Blender w wersji 2.90 [1]. Jest to oprogramowanie typu open source, na licencji GNU (General Public License) [2].

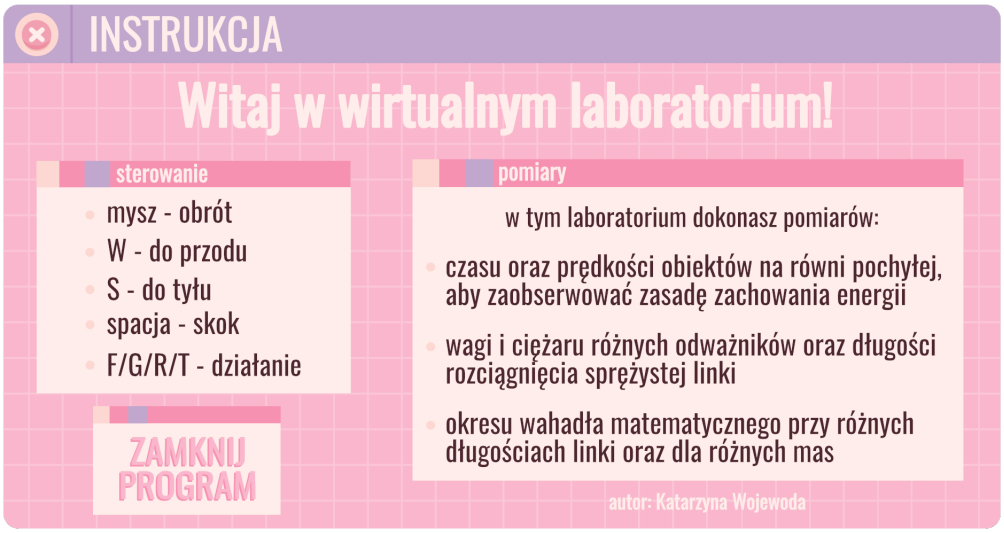
Do zbudowania programu wykorzystano zintegrowane środowisko do tworzenia gier komputerowych Unity w wersji 2019.2.0f1 [3]. Nazywane będzie dalej zamiennie silnikiem gry. Umożliwia ono tworzenie aplikacji na platformy mobilne, desktopowe, konsolowe, webowe oraz wirtualnej rzeczywistości. Licencja na wykorzystanie silnika w tym projekcie jest darmowa [4].

Kontrolowanie obiektów klasy **GameObject**, czyli wszystkich obiektów dodanych do danej sceny, odbywa się głównie przez dołączenie do nich skryptów. Scena zawiera wyszystkie elementy środowiska oraz interfejsu użytkownika niezbędne do zbudowania danego poziomu gry. Dodatkowo silnik umożliwia dodawanie niektórych funkcjonalności bez konieczności rzeczywistej implementacji kodu. Część parametrów wykorzystanych w skryptach również zmieniana jest z poziomu silnika. Unity wspiera dwa języki programowania - JavaScript oraz C#. W tym projekcie wybrano język C#. Natomiast do napisania niezbędnych skryptów wykorzystano zintegrowane środowisko programistyczne Visual Studio.

Wykorzystano również niektóre podstawowe moduły dostępne w silniku. Zastosowano między innymi **UnityUI**, który pozwolił na wprowadzenie funkcjonalności do interfejsu użytkownika (UI - *ang.* User Interface), moduł animacji umożliwiający symulację ruchu postaci oraz moduł fizyki, dzięki któremu wprowadzono siły działajace na obiekty umieszczone na scenie [5].

## 2.2 Instrukcja

Po włączeniu programu pojawia się pokój oraz postać stojąca obok stanowiska informacyjnego. Po naciśnięciu klawisza F, zgodnie z informacją w prawym dolnym rogu ekranu, wyświetli się powitalny interfejs użytkownika (Rysunek 1), wyjaśniający sterowanie w grze oraz umożliwiający zamknięcie programu. Poruszanie się po interfejsie następuje przy pomocy myszy. Wolną przestrzeń dostępną w pokoju startowym użytkownik może wykorzystać do zapoznania się ze sterowaniem postaci. Po zbliżeniu się do drzwi i naciśnięciu klawisza F można przejść do kolejnych pomieszczeń, w których znajdują się doświadczenia.




Rysunek 1: Fragment interfejsu użytkownika.

Ponieważ kolejne pokoje zostały przygotowane według tego samego schematu, zostanie teraz przedstawiona jedynie ogólna instrukcja zawierająca najważniejsze elementy, które się powtarzają. Dokładny sposób wykonywania oraz wyjaśnienie każdego z doświadczeń zostanie podane w następnych rozdziałach.

Po wejściu do kolejnego pokoju zalecane jest na początek podejście do stanowiska informacyjnego i zapoznanie się z zawartymi tam instrukcjami dotyczącymi działania danego doświadczenia oraz ze wzorami niezbędnymi do wykonania obliczeń. Zadaniem użytkownika jest dokonanie tych obliczeń na podstawie uzyskanych wyników doświadczeń. Następnie można skorzystać z dostępnych w pokoju sprzętów i dokonać pomiarów zgodnie z przeczytaną wcześniej instrukcją. Wszystkie doświadczenia opierają się na umieszczeniu odpowiedniego obiektu we właściwym miejscu. Większość interakcji ze sceną wykonywana jest poprzez naciśnięcie klawisza F lub G, kiedy jest to wskazane na dole ekranu. Natomiast wyświetlenie tabeli wyników następuje po naciśnięciu klawisza R. W pokojach, gdzie możliwe jest zmierzenie czasu, reset stopera następuje po naciśnięciu klawisza T. Możliwe jest dowolne dobranie badanych obiektów oraz ustawnienie niektórych parametrów doświadczeń. Nie ma konieczności wykonywania wszystkich pomiarów.

Aby wyłączyć program należy ponownie udać się do pokoju startowego, włączyć interfejs dostępny pod stanowiskiem informacyjnym, a następnie kliknąć na słowa "ZAMKNIJ PROGRAM".

## 2.3 Dostępne doświadczenia

Projekt laboratorium zawiera trzy doświadczenia znajdujące się w osobnych pokojach. W każdym z nich można wykonać pomiary rzeczywistych wartości fizycznych. Wybór doświadczeń został oparty o podstawę programową liceum[6]. Dodatkowo przy opisie praw fizycznych i wyprowadzaniu wzorów kierowano się teorią zawartą w podreczniku szkolnym [7]. Do dyspozycji

użytkownika jest:

- badanie zasady zachowania energii na równi pochyłej,
- wyznaczanie współczynnika sprężystości linek przy pomocy siłomierza,
- obliczenie okresu wahadła matematycznego.

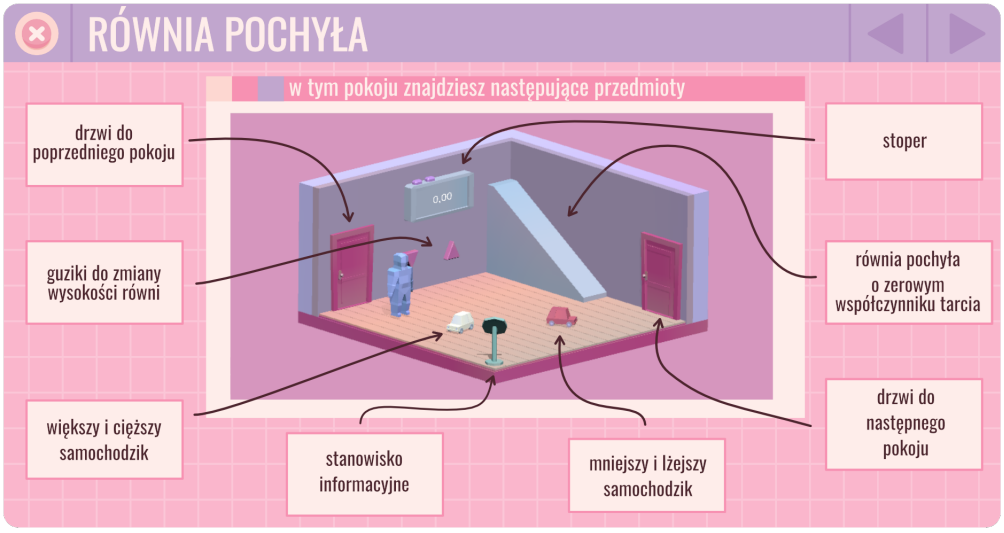
Zwrócono również uwagę na możliwości, jakie daje program komputerowy w porównaniu do eksperymentów wykonywanych w świecie rzeczywistym. Dzięki użyciu symulacji możliwe jest przeprowadzenie doświadczeń w "wyidealizowanym" środowisku, to znaczy z pominięciem na przykład tarcia oraz oporów powietrza. Każde z doświadczeń zostanie dokładniej pisane w następnych podrozdziałach. Ponieważ teoria zawarta w programie ograniczona do poziomu liceum ogólnokształcącego, z tego powodu pominięte zostaną dokładniejsze wyprowadzenia wzorów fizycznych.

## 2.4 Pokój 1 - równia pochyła

Doświadczenie w pierwszym pokoju laboratorium demonstruje zasadę zachowania energii. Użytkownik bada zachowanie obiektów na równi pochyłej.

**Zasada zachowania energii:** Energia całkowita układu izolowanego się nie zmienia.

Program umożliwia zmierzenie prędkości końcowej oraz czasu zjazdu dla dwóch samochodzików o różnych masach. Ponieważ pominięty został współczynnik tarcia (równi jak i badanych ciał), można zatem doświadczalnie sprawdzić, że spełniona jest zasada zachowania energii. Wszystkie elementy dostępne w tym pokoju zostały opisane w interfejsie użytkownika (Rysunek 2).




Rysunek 2: Fragment interfejsu użytkownika, pokój z równią pochyłą.

W przypadku równi pochyłej (bez tarcia) brane są pod uwagę dwa typy energii mechanicznej - energia kinetyczna oraz energia potencjalna grawitacji. Ciało w punkcie początkowym, to znaczy na szczycie równi przed rozpoczęciem ruchu (prędkość początkowa $v_0 = 0$), posiada energię potencjalną grawitacji

$$E_p = m \cdot g \cdot h$$

gdzie m - masa ciała, g - przyspieszenie grawitacyjne, h - wysokość równi.
Podczas zjazdu energia potencjalna ciała zamieniana jest na energię kinetyczną, która osiąga wartość maksymalną na dole równi

$$E_k = \frac{m \cdot v^2}{2}$$

gdzie m - masa ciała, v - prędkość.
Suma tych energii podczas całego ruchu jest stała

$$E_k + E_p = const$$

Więc można w ten sposób otrzymać wzór na wysokość równi, niezależny od masy poruszającego się ciała

$$h = \frac{v^2}{2 \cdot g}$$

Przebieg tego doświadczenia dla jednej wysokości równi jest następujący:

1. Położenie pierwszego samochodzika na równi pochyłej.
2. Zaobserwowanie ruchu.
3. Zresetowanie stopera.
4. Odłożenie pierwszego samochodzika.
5. Ponowne wykonanie tych czynności dla drugiego samochodzika.
6. Wyświetlenie tabeli wyników i wykonanie obliczeń.

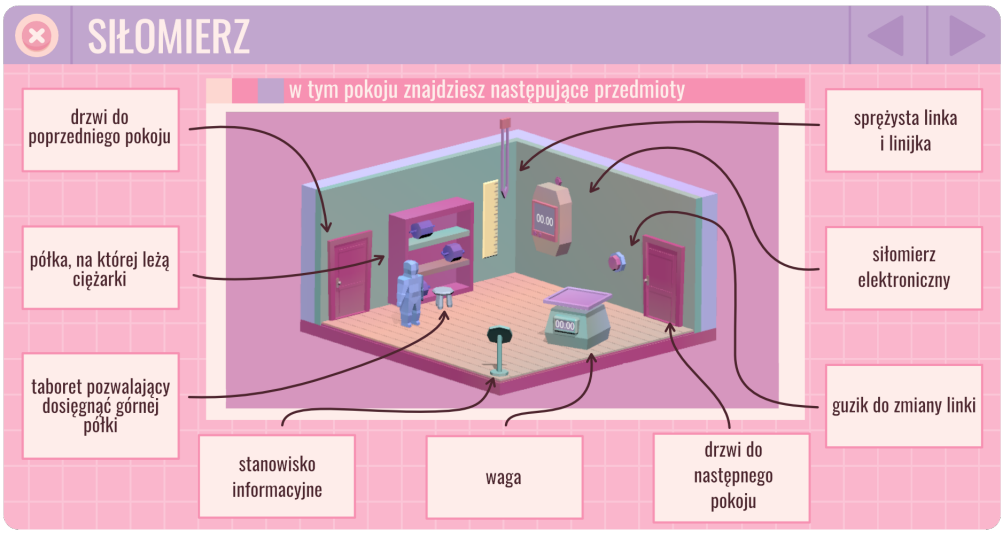
Użytkownik może wybrać trzy wysokości równi pochyłej, które są zmieniane przy pomocy guzików. Dla każdej z nich sposób pomiaru czasu oraz prędkości wygląda identycznie. Porównując wartości wyświetlane w tabeli wyników można zaobserwować, że dla danej wysokości równi pochyłej, niezależnie od masy badanego obiektu, czas zjazdu oraz prędkość końcowa będą takie same. Co jest zgodne z przewidywaniem na podstawie zasady zachowania energii. Dodatkowo na podstawie uzyskanych wyników oraz korzystając z wcześniej opisanych wzorów, użytkownik może obliczyć również dokładne wysokości tych równi.

## 2.5 Pokój 2 - siłomierz

W drugim pokoju użytkownik ma do dyspozycji siłomierz oraz trzy ciężarki o różnych masach. Na badanie siły ciężkości działającej na dane ciało przy pomocy siłomierza pozwala trzecia zasada dynamiki Newtona.

**III zasada dynamiki Newtona:** Oddziaływania ciał są zawsze wzajemne. Dwa ciała działają na siebie nawzajem siłami tego samego rodzaju, o tej samej wartości, lecz o przeciwnych zwrotach.

Zawartość pokoju pokazana jest na Rysunku 3. Doświadczenie można wykonywać wielokrotnie, ponieważ przy każdym ponownym wejściu do tego pokoju masy ciężarków są losowane z pewnego określonego przedziału. Możliwe jest wykonanie pięciu pomiarów siły ciężkości w dowolnej konfiguracji trzech ciężarków oraz zachodzącego przy tym rozciągnięcia dla każdej z linek.




Rysunek 3: Fragment interfejsu użytkownika, pokój z siłomierzem.

Schemat układu siłomierza oraz ciężarków wraz z siłami działającymi w nim znajduje się na Rysunku 4. Siła jest wielkością wektorową określającą oddziaływania między ciałami. Siła sprężystości wyraża się wzorem

$$F_S = k \cdot x$$

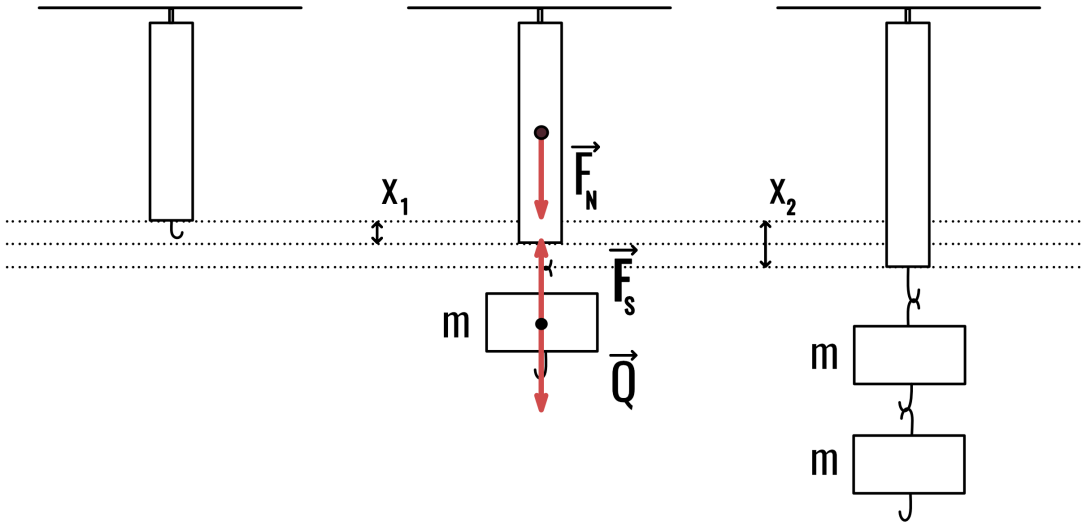
gdzie $k$ - współczynnik sprężystości, $x$ - odkształcenie. Natomiast ciężar

$$Q = m \cdot g$$

gdzie $m$ - masa, $g$ - przyspieszenie ziemskie.

Z trzeciej zasady dynamiki możemy przyrównać do siebie te dwie siły i w ten sposób otrzymać wzór na współczynnik sprężystości, który użytkownik będzie mógł wyznaczyć w tym doświadczeniu

$$k = \frac{m \cdot g}{x}$$




Rysunek 4: Schemat działania siłomierza. $\overrightarrow{Q}$ - siła ciężkości, $\overrightarrow{F_S}$ - siła sprężystości, $\overrightarrow{F_N}$ - siła naciągu działająca na siłomierz, $m$ - masa ciężarka, x - rozciągnięcie ($x_2 = 2x_1$).

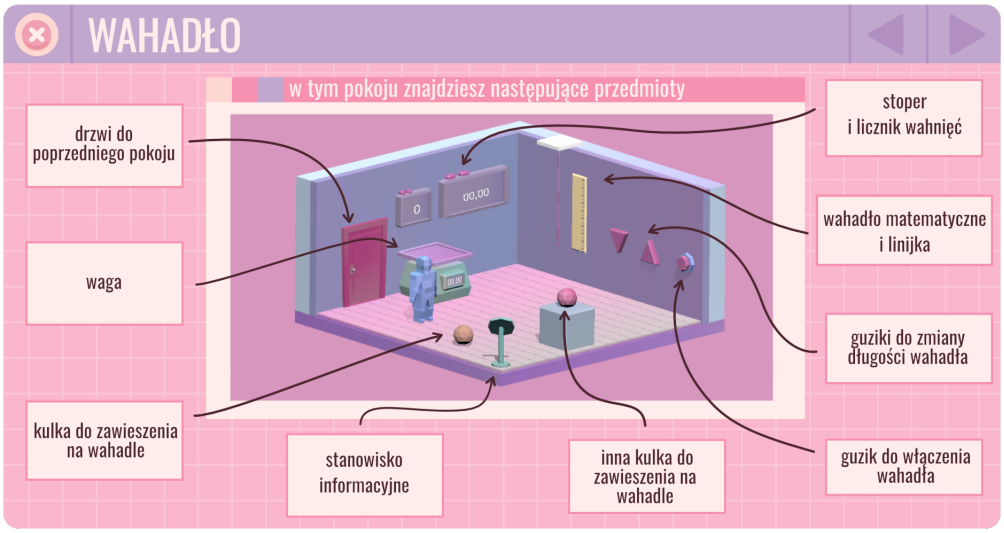
Przebieg przykładowego doświadczenia dla jednego ciężarka jest następujący:

1. Podniesienie ciężkarka z półki.
2. Położenie ciężarka na wadze i zważenie go.
3. Podniesienie ciężarka.
4. Zawieszenie ciężarka na lince.
5. Dodanie wyniku pomiaru z siłomierza do tabeli.
6. Odłożenie ciężarka.
7. Zamiana linki naciśnięciem na guzik.
8. Ponowne podniesienie ciężarka i zawieszenie go na lince.
9. Dodanie drugiego wyniku do tabeli.
10. Wyświetlenie tabeli wyników i wykonanie obliczeń.

Wszystkie pomiary wykonuje się analogicznie jak opisano powyżej. Otrzymanie przynajmniej jednego wyniku pozwala na obliczenie współczynnika sprężystości danej linki przy pomocy wcześniej wprowadzonych wzorów. Wyznaczenie masy ciężarków nie jest niezbędne, daje natomiast użytkownikowi pewność poprawności wskazań siłomierza. Wykonując to doświadczenie użytkownik zapoznaje się z zasadą działania sprzętu pomiarowego jakim jest siłomierz.

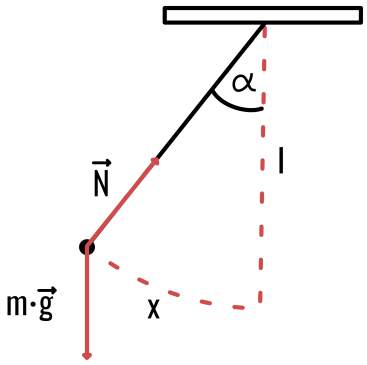
## 2.6 Pokój 3 - wahadło

Wykorzystując możliwość pominięcia oporów ruchu, w trzecim doświadczeniu badane jest wahadło matematyczne. Okres drgań takiego wahadła będzie zależeć od jego długości, a nie od masy zawieszonej na jego końcu. Zgodnie z planem pokoju z Rysunku 5, użytkownik może wybrać trzy długości linki, które są zmieniane skokowo po naciśnięciu odpowiedniego guzika, oraz dwie kulki o różnych masach. Tak jak w poprzednim pokoju, masy te są losowane z pewnego określonego przedziału przy każdym załadowaniu się sceny. Aby obliczyć okres drgań, mierzony będzie czas potrzebny na wykonanie danej liczby wahnięć.




Rysunek 5: Fragment interfejsu użytkownika, pokój z wahadłem.

**Wahadło matematyczne (proste):** Wahadło matematyczne to ciało o masie punktowej $m$, zawieszone na cienkiej, nieważkiej, nierozciągliwej nici o długości $l$, wychylone z położenia równowagi o kąt $\alpha$ (Rysunek 6).




Rysunek 6: Schemat wahadła matematycznego.

Siła wypadkowa w ruchu harmonicznym ma następującą postać

$$F = m \cdot g \cdot \sin \alpha$$

gdzie dla małych kątów (mniejszych niż 7°) można zapisać $\sin \alpha = \alpha$. Dalej z drugiej zasady dynamiki Newtona

$$m \cdot g \cdot \alpha = m \cdot a$$
$$a = g \cdot \alpha$$

Wykorzystując wzór na przyspieszenie w ruchu po okręgu oraz na $\alpha$

$$a = -\omega^2 \cdot x$$
$$\alpha = \frac{x}{l}$$
$$\omega = \sqrt{\frac{g}{l}}$$

gdzie $\omega$ - prędkość kątowa.
Stąd można otrzymać wzór na okres drgań wahadła matematycznego

$$\omega = \frac{2\pi}{T}$$
$$T = 2 \cdot \pi \cdot \sqrt{\frac{l}{g}}$$

Łatwo zauważyć, że okres wahadła nie zależy od masy zawieszonego ciała, a jedynie od jego długości.

Dokładniej - im wahadło będzie krótsze, tym wykona więcej wahnięć w jednostce czasu. Ponieważ zachodzi również

$$T = \frac{n}{t}$$

gdzie n - liczba wahnięć, t - czas.

Wykonanie przykładowych pomiarów wygląda następująco:

1. Podniesienie kulki.
2. Położenie kulki na wadze i zważenie jej.
3. Podniesienie kulki.
4. Zawieszenie kulki na wahadle.
5. Włączenie wahadła naciśnięciem guzika.
6. Wykonanie pomiarów.
7. Zresetowanie licznika wahnięć.
8. Zatrzymanie i wyzerowanie stopera.
9. Odłożenie kulki.
10. Wyświetlenie tabeli wyników i wykonanie obliczeń.

W tym doświadczeniu można zmierzyć czas ruchu wahadła dla pięciu wahnięć i w ten sposób wyznaczyć jego okres oraz długość. Dodatkowo użytkownik może zaobserwować, że okres ten nie zależy od zawieszanej masy. Dlatego w tym doświadczeniu jej pomiar przy pomocy wagi jest niezbędny do zrozumienia istoty badanego zjawiska.

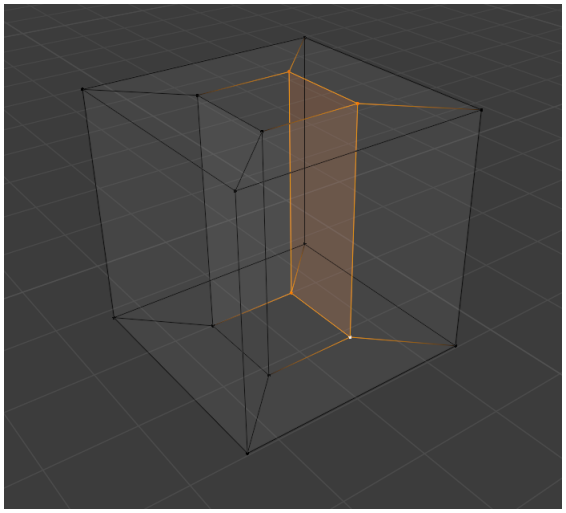
# 3 Grafika

## 3.1 Podstawowy model 3D

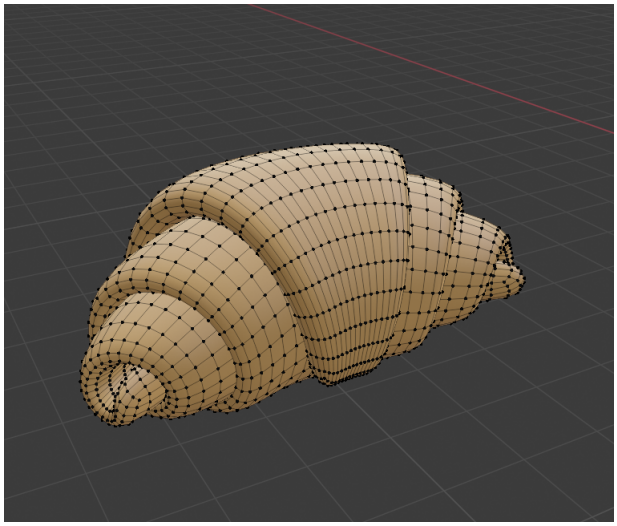
Aby osiągnąć względnie realistyczną symulację rzeczywistego laboratorium, w programie została zastosowana grafika trójwymiarowa. Wykorzystanych zostało około 35 różnych modeli. Zasady modelowania 3D są takie same dla każdego obiektu, niezależnie od jego wielkości ani ilości szczegółów.

### 3.1.1 Elementy modelu

Dowolnie skomplikowany model 3D składa się z trzech podstawowych elementów. Są to wierzchołki, krawędzie oraz ściany. Dokładniej, wierzchołki łączone krawędziami tworzą ściany, które mają kształt wielokątów. Tak otrzymaną siatkę można dalej modyfikować i deformować w celu otrzymania większej ilości szczegółów. Przykładowy model znajduje się na Rysunku 7. Dla porównania, na Rysunku 8 najduje się obiekt o bardziej skomplikowanej geometrii, gdzie kropki wskazują położenia wierzchołków.



Rysunek 7: Siatka prostej bryły w programie Blender z zaznaczoną ścianą, 4 krawędziami oraz 4 wierzchołkami.



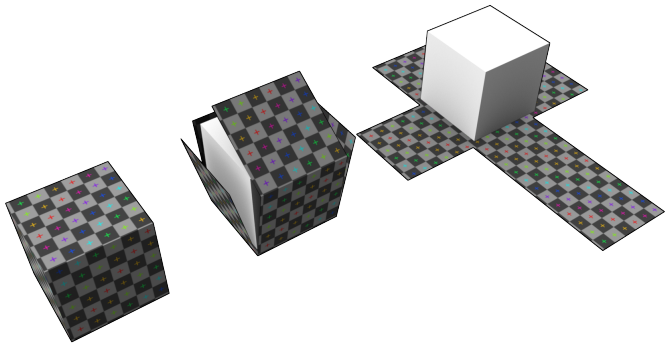
Rysunek 8: Siatka modelu rogalika w programie Blender

Sam proces modelowania 3D rozpoczyna się od prymitywu geometrycznego, czyli od prostej figury geometrycznej, z której poprzez deformacje można dalej budować bardziej skomplikowane obiekty. W zależności od wybranego oprogramowania oraz modelowanego przedmiotu, może to być na przykład prostokąt, sześcian, kula lub graniastosłup. Następnie przesuwając pojedyncze wierzchołki, krawędzie i ściany oraz dodając kolejne prymitywy, buduje się pożądaną geometrię. Można również dzielić wielokąty siatki na mniejsze, W celu optymalizacji czasu pracy nad modelem w programie Blender stosuje się również odpowienie modyfikatory, takie jak na przykład modelowanie z odbiciem lustrzanym. Dokładny opis wszystkich dostępnych modyfikatorów można znaleźć w dokumentacji programu [8]. Przy dość prostej i intuicyjnej metodzie budowania modeli, obliczenia wykonywane przez program są zdecydowanie bardziej złożone. Z tego powodu duże i rozbudowane modele posiadające wiele szczegółów wykonuje się na komputerach o odpowiednio dużej mocy obliczeniowej.

W tym projekcie wszystkie modele zostały przygotowane w stylu low-poly. To znaczy z wykorzystaniem jak najmniejszej ilości wielokątów (*ang.* low - nisko/mało, *ang.* poly - wielokąt), tak aby obiekt mimo to nadal przypominał rzeczywisty przedmiot. Ten sposób modelowania może być zwykłym wyborem stylistycznym albo, tak jak w tym przypadku, dzięki ograniczeniu geometrii modeli otrzymujemy program, który nie będzie nadmiernie obciążający dla karty graficznej oraz procesora komputera.

### 3.1.2 Teksturowanie

Mapowanie UV pozwala na przeniesienie kolorów z dwuwymiarowego obrazu na wielokąty modelu. Obraz ten nazywany jest teksturą UV. Przykład znajduje się na Rysunku 9. Litery U oraz V oznaczają osie dwuwymiarowej płaszczyzny tekstury. Typowe oznaczenia osi X, Y oraz Z zarezerwowane są dla przestrzeni trójwymiarowej, w której znajduje się model. Współrzędne UV mogą zostać wygenerowane dla każdego wierzchołka siatki.



Rysunek 9: Przykładowe mapowanie tekstury.
https://upload.wikimedia.org/wikipedia/commons/f/fe/Cube_Representative_UV_Unwrapping.png

Proces nakładania tekstury obejmuje przypisanie odpowiednich pikseli do powierzchni modelu mapowanej na teksturę. Może to zostać wykonane automatycznie przez program. Kiedy mapowany model ma bardziej skomplikowaną geometrię należy przygotować go ręcznie. W tej metodzie wskazuje się miejsca, w których siatka wielokątów ma zostać przecięta, tak aby po rozwinięciu jej na łaszczyznę UV, tekstury na modelu łączyły się w sposób ciągły, dający wrażenie realistycznych obiektów.

Tak jak zostało wspomniane wcześniej, końcowy program został zoptymalizowany pod kątem komputerów o słabszym procesorze. Tekstury w wysokiej rozdzielczości przygotowane osobno dla każdego modelu wymagałyby odpowiednio dużej mocy obliczeniowej. Z tego powodu wykorzystane zostały jedynie dwie proste tekstury, na których w odpowiednich miejscach zostały rozwinięte siatki modeli wykorzystanych w projekcie. Obrazy te mają wymiary 6x6 pikseli oraz każdy z nich zawiera jedynie ubogą paletę kolorów dla zachowania spójności kolorystyki projektu. Przykład rozwinięcia modelu postaci na płaszczyznę UV znajduje się na Rysunku 10.

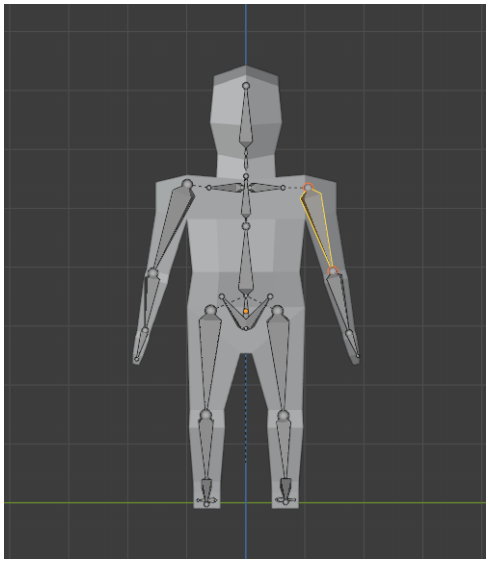
Rysunek 10: Mapowanie tekstury modelu postaci.

## 3.2 Model 3D postaci

Wykonanie modelu, do którego będzie dołączona animacja, następuje prawie identycznie jak opisano w poprzednim podrozdziale. W celu zagęszczenia siatki należy jednak dodać dodatkowe wielokąty w okolicy połączeń, które będą się zginać. W przypadku modelu postaci będą to na przykład okolice łokci oraz kolan. Dzięki temu będzie można uzyskać płynną animację.

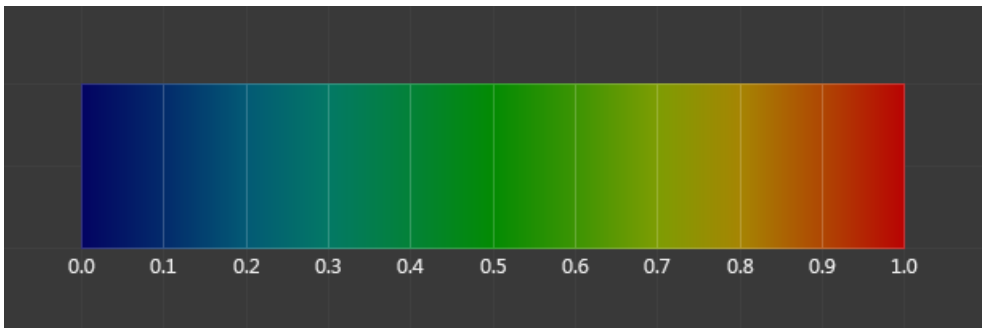
### 3.2.1 Szkielet

Aby przygotować model 3D do animacji ruchu, należy dołączyć do niego szkielet, czyli wykonać tak zwany rigging. Na początek do modelu dodawane są kości, które łączą się w łańcuchy kinematyczne. Rysunek 11 przedstawia postać ze szkieletem i zaznaczoną na pomarańczowo kością ramienia. Rigging niektórych modeli również może zostać wykonany automatycznie przez program Blender. Należy wtedy jednak zwrócić uwagę, które połączenia będą niezbędne. W tym modelu pominięty został szkielet twarzy oraz pojedynczych palców dłoni i stóp. W produkcjach takich jak filmy animowane lub wysokorealistyczne gry komputerowe rigging wykonuje się bardziej szczegółowo. Aby osiągnąć płynne animacje, modele o skomplikowanej geometrii potrzebują większej liczby kości oraz połączeń między nimi. Jest to również bardziej wymagające od strony sprzętowej. W takim wypadku do samego dodawania szkieletów przypisana jest konkretna osoba lub zespół.

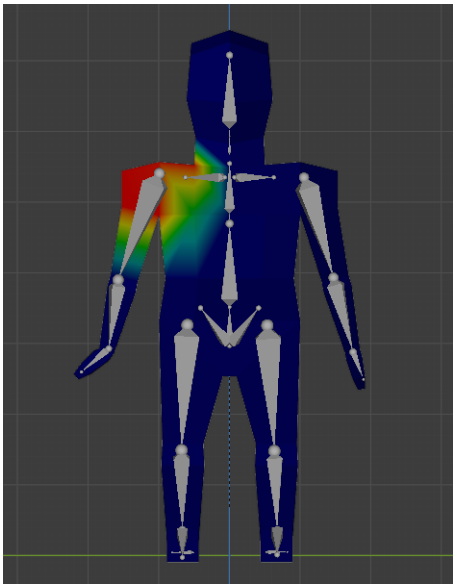


Rysunek 11: Model 3D postaci z dodanym szkieletem.

Kiedy rigging został wykonany oraz wszystkie kości są ustawione w odpowiedniej hierarchii, należy przejść do dodania map wag (Rysunek 12) odpowiednich części siatki wielokątów. Mapa wag określa jaki wpływ na dany fragment modelu będzie miał ruch danej kości. Wielokąty oznaczone na czerwono będą bardziej wrażliwe na zmiany położenia. Ten proces jest szczególnie ważny dla bardziej skomplikowanych modeli, które są podatne na błędy animacji. Przykład nałożenia fragmentu map wag znajduje sie na Rysunku 13.



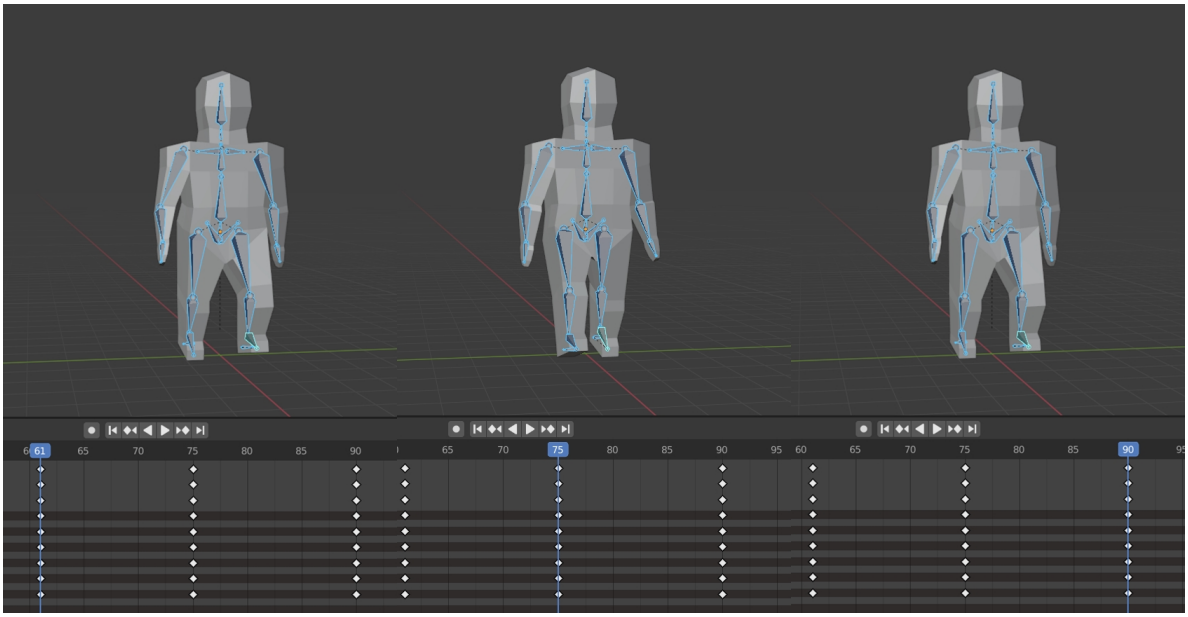

Rysunek 12: Spektrum kolorów mapy z odpowiadającymi wagami.
https://docs.blender.org/manual/en/latest/_images/sculpt-paint_weight-paint_introduction_color-code.png



Rysunek 13: Przykładowy fragment procesu nakładania map wag.

### 3.2.2 Animacje

Rigging modeli jest niezbędny w procesie animacji. Dana część szkieletu pozwala na płynne przesuwanie grup wierzchołków połączonych z tą kością przy pomocy mapy wag. W tym projekcie postać posiada dwie animacje - przemieszczania się oraz "stania w miejscu", która zostaje uruchomiona w przypadku dłuższej nieaktywności. Obie zostały przygotowane w programie Blender przy wykorzystaniu osi czasu. Ręcznie dodawane są jedynie klatki kluczowe, a następnie silnik renderujący interpoluje pozycje wierzchołków pomiędzy nimi. Im bardziej szczegółowy będzie model, tym będzie wymagał większej liczby ręcznie ustawianych klatek. Przy prostym modelu 3D animowanie ruchu polega na ustawieniu go, przy pomocy dodanego wcześniej szkieletu, w wybranej pozycji i zapisaniu jej na początku oraz na końcu wybranego przedziału czasu. Następnie pomiędzy nimi dodawana jest poza w odbiciu lustrzanym. Po zaimportowaniu modelu do silnika gry, czas może zostać zapętlony i w ten sposób zostaje uzyskana symulacja ruchu. Przykład ustawień modelu do animacji przemieszczania się dla 30 sekundowego przedziału znajduje się na Rysunku 14. Animacja "stania w miejscu" została wykonana w analogiczny sposób.



Rysunek 14: Model postaci w ustawieniu do animacji.

## 3.3 Scena w silniku Unity

Program został podzielony na cztery poziomy, gdzie każdy z nich znajduje się na osobnej scenie. Warto zwrócić uwagę na ustawienie kierunków osi w silniku. W odróżnieniu od typowego kartezjańskiego układu współrzędnych, w Unity układ jest lewoskrętny z osią Y skierowaną do góry. Stąd grawitacja, symulowana na ziemską, działać będzie w kierunku przeciwnym do wersora osi Y, czyli w dół.

### 3.3.1 Modele w silniku i budowanie sceny

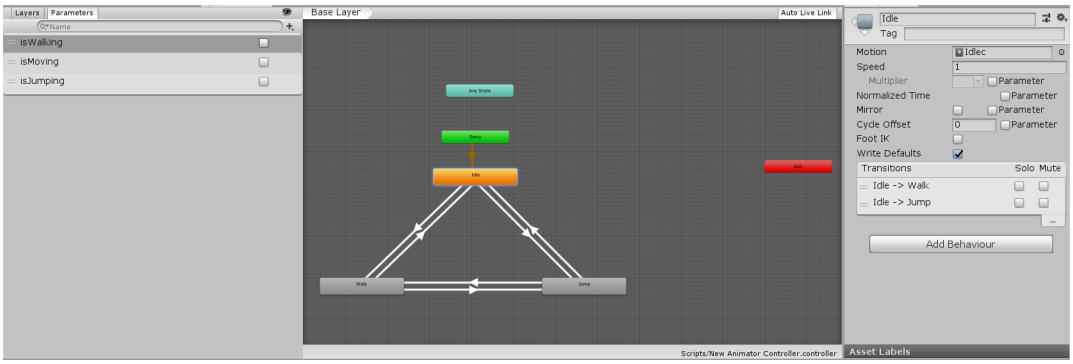
Gotowy model można zaimportować do silnika. Jest oczywiście możliwe również dodawanie prymitywnych modeli bezpośrednio w programie. Unity przyjmuje z pliku:

- pozycję, rotację i skalę,
- środek ciężkości i nazwy elementów,
- siatkę z wierzchołkami, wielokątami oraz trójkątami,
- mapę UV,
- szkielet i animacje.

Zaimportowany model można dodać do sceny przeciągając go z folderu zasobów (wszystkich komponentów gry). Wtedy ustala się jego położenie oraz skalę. Przyjmuje się, że 1 jednostka w Unity to 1 metr świata rzeczywistego. Na tej podstawie silnik wykonuje wszelkie obliczenia fizyczne. W przypadku tego projektu ważniejsze niż dokładne wymiary obiektów były proporcje między nimi, co należało wziąć pod uwagę podczas implementacji funkcjonalności.

Kolejnym istotnym krokiem jest dodanie granic kolizji. Dalej będą one nazywane **Colliderami**. **Collidery** są niewidoczne w czasie gry oraz nie muszą mieć kształtu geometrii obiektu. Bez ich zastosowania silnik nie rozróżnia siatki modelu jako przeszkody i możliwe jest wystąpienie takich błędów jak na przykład przechodzenie przez ściany. **Box Collider** ma domyślnie kształt sześcianu, który można dostosować do geometrii obiektu. Wykorzystany został na przykład do określenia granic kolizji podłogi, co umożliwia poruszanie się postaci i stanowi odniesienie dla sił grawitacji. **Collidery** mają również zastosowanie przy interakcjach z obiektami sceny. Wybranie opcji **is Trigger** sprawia, że jest możliwe wejście w zaznaczony obszar, dodatkowo wykonując przypisane do niego zdarzenie. Zostanie to dokładniej wyjaśnione w kolejnych rozdziałach.

Do obiektu postaci dodano jeszcze przygotowane wcześniej animacje. W silniku można podzielić oś czasu na dane sekwencje ruchu oraz zapętlić czas ich wykonywania. Następnie należy dołączyć komponent odpowiedzialny za animacje, który nosi nazwę **Animator Controller**. Przy jego pomocy ustala się relację pomiędzy dostępnymi animacjami. Ustawienie dla tego projektu znajduje się na Rysunku 15. Dalej za animacje ruchu postaci będzie odpowiadał skrypt korzystający z parametrów komponentu typu **Animator Controller**.



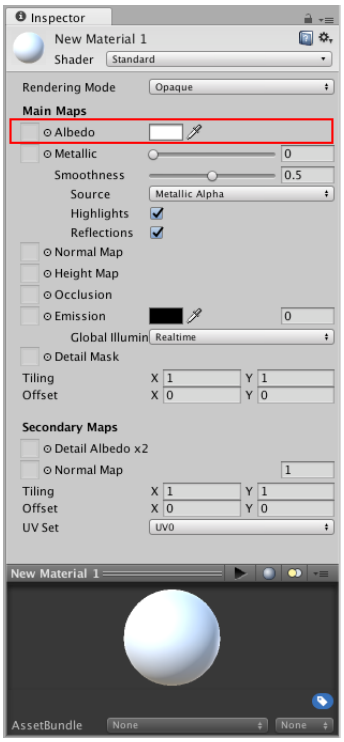

Rysunek 15: **Animator Controller** w silniku Unity.

### 3.3.2 Materiały

Do renderowania sceny Unity wykorzystuje trzy powiązane ze sobą elementy:

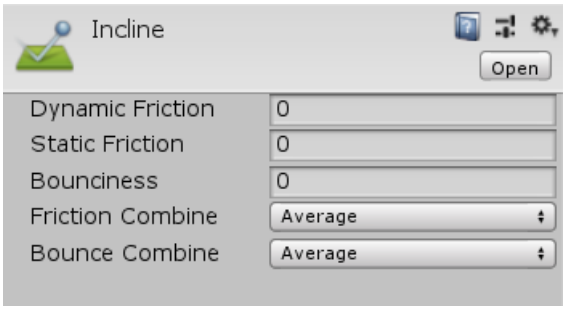
- **Materials** definiujące sposób renderowania danej powierzchni. Posiadają odniesienie do tekstur, własności fizycznych oraz koloru. Dostępne parametry zależą od wykorzystywanych Shaderów.
- Shadery to krótkie skrypty zawierające obliczenia oraz algorytmy opisujące właściwości renderowanych pikseli w zależności od światła i ustawień obiektu typu **Material**.
- Tekstury to obrazy w formie bitmap. Shader przypisany do danego obiektu **Material** wykorzystuje teksturę przy obliczaniu koloru modelu. Zazwyczaj przypisuje się je do obiektu **Material** jako kolor podstawowy (Albedo). Można przy ich pomocy określać również na przykład chropowatość powierzchni.

Ustawienia obiektów typu **Material** w tym projekcie ograniczyły się do przypisania tekstury do koloru podstawowego (Rysunek 16). Zastosowano wbudowany **Standard Shader** [9], pomijając pisanie tego skrypu ręcznie. Łączy on różne typy shaderów w jeden, w efekcie czego te same obliczenia światła używane są we wszystkich obszarach sceny, zapewniając realistyczny i spójny render. Można go wykorzystać do wszystkich rodzajów obiektów **Material** symulujących na przykład kredę, ceramikę lub drewno, które nazywa się powierzchniami twardymi, w kontraście do materiałów nietwardych, takich jak woda.




Rysunek 16: Ustawienia obiektu **Material** w silniku Unity (na czerwono kolor podstawowy).
https://docs.unity3d.com/2018.3/Documentation/uploads/Main/StandardShaderParameterAlbedoColor.png

Oprócz zwykłych obiektów typu **Material** wykorzystano również **Physical Material** - Materiał Fizyczny. Stosuje się go do ustawienia konkretnych wartości współczynników tarcia oraz sprężystości. Dodawany jest nie bezpośrednio do modelu, ale do **Collidera** stowarzyszonego z danym obiektem. Dzięki niemu można symulować tarcie pomiędzy różnego rodzaju tworzywami. W tym projekcie dodano go do równi pochyłej z wartościami zerowymi wszystkich parametrów. Otrzymano w ten sposób spełnioną zasadę zachowania energii bez strat na opory ruchu. Ustawienia **Physical Material** znajdują się na Rysunku 17.

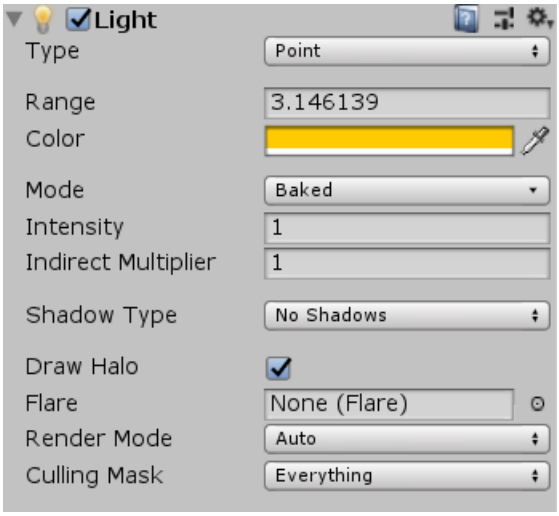



Rysunek 17: Materiał Fizyczny w silniku Unity.

### 3.3.3 Oświetlenie

Dzięki zastosowaniu mało skomplikowanych modeli, oświetlenie sceny może być renderowane w czasie rzeczywistym (*ang.* Realtime lighting) bez nadmiernego obciążenia komputera. Oznacza to, że światło padające na obiekty w grze odświeżane jest w każdej klatce. Jest to szczególnie przydatne dla geometrii, która będzie się poruszać. Do każdej ze scen dodano dwa źródła oświetlenia typu kierunkowego - imitującego światło słoneczne, które nie zależy od odległości od obiektu. Pozwala ono na osiągniecie wiarygodnych cieni bez wskazania skąd dokładnie pochodzi światło. Jedno z tych źródeł, o słabszej mocy i ustawione "od góry", służy do generalnego rozjaśnienia sceny i nie powoduje powstawania cieni. Natomiast drugie, mocniejsze i skierowane "od przodu", pozwala obiektom na rzucanie cieni twardych, to znaczy bez rozmazanych krawędzi, co pasuje do ogólnej stylizacji projektu.

W pokoju drugim oraz trzecim dodane zostało światło typu punktowego (Rysunek 18). Pojawia się przy zbliżaniu się do przedmiotów, które można podnieść. Jest ono przypisane jako komponent danego obiektu i włączanie go odbywa się przy pomocy skryptu. Światło punktowe, jak wskazuje nazwa, emituje światło z jednego miejsca we wszystkie kierunkach. Jego zasięg ograniczony jest przez pewną kulę o zadanym promieniu. Moc tego rodzaju oświetlenia jest odwrotnie proporcjonalna do kwadratu odległości od źródła. Opcja **Draw Halo** pozwala na uzyskanie wrażenia podświetlenia przedmiotu. Zastosowane światło punktowe nie rzuca cieni.




Rysunek 18: Światło punktowe w silniku Unity.

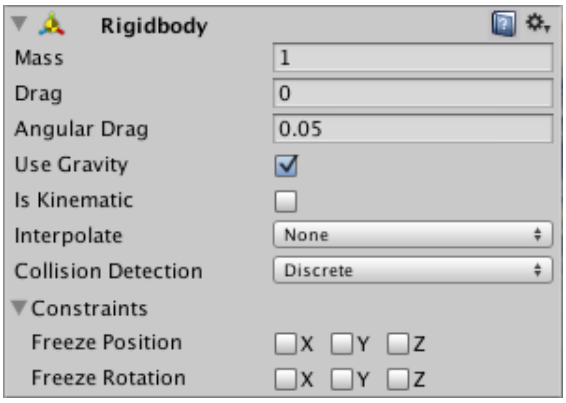
# 4 Implementacja

Poza wykonaniem modeli, projekt został w całości zbudowany w Unity. Nie zostały oczywiście wykorzystane wszystkie możliwości tego silnika. Opisane zostaną jedynie zastosowane metody oraz istotne, z punktu widzenia projektu, rozwiązania.

## 4.1 Silnik

Główną korzyścią wynikającą z wykorzystania silnika był zaimplementowany w nim **EventSystem**. Odpowiada on za przetwarzanie i obsługiwanie wszystkich wydarzeń w grze. Przyjmuje również dane wejściowe otrzymywane przez program, takie jak na przykład naciskane klawisze klawiatury lub ruchy myszy. Wykorzystanie konkretnych funkcji **EventSystemu** zostanie opisane w kolejnych rozdziałach.

Oprócz opisanych wcześniej **Colliderów** i **Physical Materials** z modułu fizyki w silniku Unity, wykorzystano również komponent **Rigidbody**, czyli bryłę sztywną. Obiekty posiadające przypisane **Rigidbody** będą w realistyczny sposób reagować na działające na nie siły. Własności tego komponentu znajdują się na Rysunku 19. Mogą być ustawiane także z poziomu skryptu. Ustalenie masy danego obiektu również odbywa się przez **Rigidbody**. Własności **Drag** oraz **Angular Drag** odpowiadają za opory powietrza, jakie będa miały wpływ na ten obiekt. W tym projekcie ustawiono wartość 0. Wykorzystano również opcję **Is Kinematic**, która, jeżeli jest zaznaczona, unieruchamia ciało. Można także ograniczać ruch oraz obroty jedynie w kierunku wybranych osi, za co odpowiadają obie opcje **Freeze**. Siły grawitacji będą działać w tym silniku jedynie na obiekty posiadające komponent **Rigidbody**.




Rysunek 19: **Rigidbody** w silniku Unity.

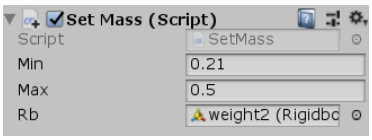
Przy pomocy **UnityUI** kontrolowane są wszystkie obiekty tekstowe umieszczone na scenie, takie jak na przykład wskazania przyrządów pomiarowych. Kolejne części interfejsu dodaje się jako komponenty obiektu typu **Canvas**. Jest to prostokątny obszar, który pozwala na wygodne skalowanie i pozycjonowanie elementów UI. W tym projekcie wykorzystano dwa typy renderowania takich obiektow. **Screen Space - Overlay** umieszcza interfejs jako nakładkę na scenę widoczną na ekranie użytkownika. Ten sposób umożliwia wygodne zmiany rozdzielczości UI. Ja-

ko drugi typ renderowania interfejsu zastosowano renderowanie **World Space**. W tym wypadku **Canvas** traktowany jest jako obiekt na scenie, tak jak modele 3D, i można go umieszczać w wybranym przez siebie miejscu. Ten typ interfejsu nazywa się również interfejsem diegetycznym. Oprócz obiektów tekstowych dodano również obiekty typu **Button**. Przyciski wykorzystują funkcję **EventSystem (On Click)**, który pozwala na interakcje użytkownika z interfejsem przez kliknięcie w odpowiednim miejscu ekranu. Działania wykonywane po naciśnięciu guzika wprowadzane są na poziomie skryptu.

Każdy z pokoi jest w tym projekcie osobną sceną - poziomem gry. Za organizację scen odpowiada **SceneManager**. Pozwala on na tworzenie nowych poziomów i ustalenie ich hierarchii przypisując scenom odpowienie indeksy. Dzięki temu otwieranie kolejnych scen odbywa się za pomocą jednego skryptu, który opiera się na ładowaniu sceny z indeksem o jeden większym od indeksu sceny aktywnej.

## 4.2 Sposób implementacji

Podczas planowania funkcjonalności programu, wzięto pod uwagę powtarzalność niektórych rozwiązań. Oczywistym jest, że skrypt obsługujący na przykład poruszanie się może być wykorzystany w każdym pokoju. Wszystkie zastosowane rozwiązania zostały podzielone na osobne foldery dotyczące każdej ze scen oraz folder wspólny dla, między innymi, wspomnianego wcześniej skryptu ruchu. Stąd implementację w projekcie można określić jako modularną. Niektóre skrypty przypisane są do wielu obiektów. Ich parametry natomiast zmieniane są z poziomu silnika. Na Rysunku 20 pokazany został przykład dla losowania masy ciężarka. Wartości brzegowe przedziału masy zostały przypisane do **Rigidbody** konkretnego obiektu. Pozwala to zaoszczędzić czas w trakcie implementacji kodu oraz ograniczyć pamięć jaką zajmuje gotowa gra.




Rysunek 20: Przykład skryptu dostępnego z poziomu silnika.

## 4.3 Zakresy parametrów

Większość parametrów, od których zależą wyniki doświadczeń, zmieniana jest skokowo. Opisany będzie dokładniej przypadek równi pochyłej. Problem ten został rozwiązany przez umieszczenie na scenie wszystkich trzech modeli o różnych wysokościach. W zależności od opcji, którą wybierze użytkownik przy pomocy strzałek, tylko jedna z nich będzie aktywna i widoczna. Każdy z tych obiektów posiada jako komponent skrypt obsługujący pomiar czasu zjazdu, który zwraca wartości jedynie dla danej równi. W ten sam sposób przeprowadzana jest zmiana długości wahadła w trzecim eksperymencie. W drugim pokoju dostępne są dwie linki o różnych współczynnikach sprężystości. Skrypt przypisany do dynamometru oblicza ich rozciągnięcie biorąc wartość współczynnika jedynie aktywnej (widocznej na scenie) linki. Przełączenie aktywności

tych obiektów wykonuje użytkownik przy pomocy przycisków na scenie. Dostępne współczynniki k wynoszą odpowiednio 1000 $\frac{N}{m}$ oraz 1500 $\frac{N}{m}$.

Tak jak zostało wspomniane wcześniej, masy ciężarków w doświadczeniu drugim oraz trzecim losowane są z zadanych przedziałów. Wykorzystana została do tego funkcja Random.Range

*Random.Range(min, max)*

Zwraca ona losową liczbę typu *float*, zmiennoprzecinkową, z domkniętego przedziału o podanych wartościach brzegowych. W przypadku ciężarków z doświadczenia drugiego, losowane są liczby pomiędzy 1 kg a 15 kg, podzielone odpowiednio na mniejsze przedziały w zależności od wielkości obiektu. Natomiast kulki z doświadczenia drugiego mają masę między 0,02 kg a 0,5 kg. W pierwszym pokoju nie jest wykonywany pomiar masy badanych obiektów, gdyż zgodnie z fizyką zjawiska jej wartość nie wpływa na ruch. Wartość ta w tym wypadku nie jest losowana i została dobrana tak, aby zachować płynność animacji zjazdu z równi. Stąd samochodziki mają odpowiednio 1000 kg oraz 1500 kg. Przyspieszenie grawitacyjne we wszystkich doświadczeniach zostało ustalone jako $9,81\frac{m}{s^2}$.

Otrzymanie poprawnych wyników doświadczeń nie było trywialne z powodu niedostosowania skali modeli do skali świata rzeczywistego, na podstawie którego silnik fizyczny wykonuje obliczenia. Dlatego podczas implementacji i wykonywania testów wartości liczbowe parametrów były zmieniane wielokrotnie, stąd ze zwzględu na łatwy dostęp większość parametrów jest zmiennymi typu publicznego, czyli dostępnego z poziomu silnika.

# 5 Algorytmy

W tym rozdziale zostaną opisane wybrane algorytmy zastosowane w projekcie. Jak zostało wspomniane wcześniej, skrypty zostały napisane w języku C# z zastosowaniem środowiska Visual Studio. Wszystkie mają taką samą strukturę jak pokazana poniżej.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ScriptName : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
      ...
    }

    // Update is called once per frame
    void Update()
    {
      ...
    }
}
```

MonoBehaviour jest klasą bazową, z której pochodzi każdy skrypt Unity. Funkcja *Start* wywoływana jest w momencie włączenia sceny, natomiast *Update* wywoływana jest co klatkę. Domyślna prędkość odtwarzania w Unity wynosi 24 FPS (*ang.* Frames Per Second - klatka na sekundę). Dalej skrypty będą przedstawione w formie pseudokodu.

## 5.1 Mechanika sterowania

W ogólności, poruszanie się obiektów pod wpływem sił w silniku Unity umożliwia komponent **Rigidbody**. Aby pominąć implementowanie fizyki kolizji wokół niego, do obiektu postaci dodaje się **CharacterController** [10]. Dzięki temu można uzyskać ruch wywołując jedną funkcję *Move*, która jest zawarta w dołączonym skrypcie. W tym miejscu obsługiwana jest również animacja poruszania się.

```
Wymagane komponenty: CharacterController, Animator

funkcja Start:
    z obiektu weź komponent CharacterController
    z obiektu weź komponent Animator

funkcja Update:
    jeżeli (characterController.isGrounded i playerVelocity.y < 0)
      wtedy    (playerVelocity.y = 0f)
    wywołaj funkcję Move
    wywołaj funkcję Rotate

    jeżeli (Input.GetButtonDown("Jump") i characterController.isGrounded)
        wtedy (playerVelocity.y += Mathf.Sqrt(jumpHeight * -2.0f * gravityValue))

    playerVelocity.y += gravityValue * Time.deltaTime
    characterController.Move(playerVelocity * Time.deltaTime)

funkcja Awake:
    Cursor.lockState = CursorLockMode.Locked
    Cursor.visible = false

funkcja Rotate:
    horizontalRotation = Input.GetAxis("Mouse X")
    verticalRotation = Input.GetAxis("Mouse Y")
    transform.Rotate(0, horizontalRotation * mouseSensitivity, 0)

funkcja Move:
    verticalMove = Input.GetAxis("Vertical")
    jeżeli (characterController.isGrounded)
      wtedy (verticalSpeed = 0)
      w przeciwnym razie (verticalSpeed -= gravity * Time.deltaTime)
    Vector3 gravityMove = new Vector3(0, verticalSpeed, 0)
    Vector3 move = (-1) * transform.right * verticalMove;
    characterController.Move(speed * Time.deltaTime * move + gravityMove *
    ↪ Time.deltaTime)
    animator.SetBool("isWalking", verticalMove != 0)
```

W funkcji *Start* pobierane są wartości odpowiednich komponentów obiektu, do którego dołączony jest powyższy skrypt. Następnie *Update* co klatkę odświeża położenie postaci. Komenda *isGrounded* sprawdza, czy **CharacterController** dotykał podłoża podczas ostatniego ruchu. Jeżeli postać nie wykonuje skoku, prędkość w kierunku osi Y jest zerowana. Postać podskoczy po naciśnięciu klawisza spacji. W tym miejscu przekazywane są również wartości do komponentu **CharacterController**. Zgodnie ze wzorem, droga w ruchu jednostajnym jest równa iloczynowi prędkości i czasu. Funkcja *Awake* jest wywoływana przed funkcją *Start* i działa na całą scenę. Wyłącza ona w tym wypadku kursor myszy, ponieważ nie jest wykorzystywany w trakcie wykonywania doświadczeń. Funkcja *Rotate* odpowiada za obrót postaci przy pomocy komendy *transform*. Input, czyli dane wejściowe, pobierane są z ruchu myszy. Następnie funkcja *Move* obsługuje pozostały ruch. W tym wypadku input pobierany jest z klawiszy W i S z klawiatury, które odpowiadają za poruszanie się do przodu i do tyłu. Ruch otrzymywany jest poprzez obliczanie kolejnego położenia modelu komendą *transform*. W tej funkcji, przy pomocy animatora, odbywa się również przełączenie animacji na poruszanie się.

## 5.2 Sposób interakcji ze sceną

Wszystkie interakcje z obiektami dostępnymi na scenie odbywają się przez wejścia w odpowiedni obszar oraz naciśnięcie wskazanego klawisza na klawiaturze. Jak zostało wspomniane wcześniej, miejsce danego zdarzenia wyznaczają **Collidery** z zaznaczoną opcją **is Trigger**. Oznacza to, że nie będą wyznaczać granic fizycznej kolizji, ale określać miejsca wystąpienia danego zdarzenia. Poniżej zostanie przedstawiony przykład dla wyświetlenia interfejsu użytkownika. Pozostałe działania zostały zaimplementowane na tej samej zasadzie.

```
isInTrigger = false

funkcja Start:
    UiObject.SetActive(false);

funkcja OnTriggerEnter(Collider other):
    jeżeli (!other.CompareTag("Player"))
         wtedy (return)
    isInTrigger = true;

funkcja OnTriggerExit(Collider other):
    jeżeli (!other.CompareTag("Player"))
         wtedy (return)
    isInTrigger = false;
    UiObject.SetActive(false)
```

```
funkcja Update:
    jeżeli (isInTrigger)
        wtedy (jeżeli (Input.GetKeyDown(KeyCode.F)
                    wtedy (UiObject.SetActive(true)))
```

Zmienna typu bool isInTrigger na początku jest ustawiona jako false. W funkcji *Start* na początku wyłączany jest obiekt interfejsu. Nie jest to krok wymagany, ponieważ moża zmienić jego aktywność z poziomu silnika, minimalizuje to jednak możliwość wystąpienia błędów. Funkcja *OnTriggerEnter* sprawdza, czy **Collider** obiektu z odpowiednim tagiem znajduje się w triggerze. Jeżeli tak, to zmieniana jest wartość zmiennej isInTrigger. Natomiast funkcja *OnTriggerExit* przełącza ją z powrotem na false, jeżeli w obszarze znajduje się właściwy Collider. Wyłącza również obiekt interfejsu. W momencie kiedy isInTrigger ma wartość true i jeśli użytkownik naciśnie klawisz odpowiedni przycisk (w tym przypadku klawisz F) zostanie aktywowany interfejs. W zależności od danej funkcjonalności może być w tym momencie wykonane inne zdarzenie. Obsługę akcji można również zaimplementować wykorzystując same funkcje *OnTriggerEnter* oraz *OnTriggerExit*, powoduje to natomiast wiele błędów i niechcianych efektów.

Poniżej znajduje się lista wszystkich dostępnych interakcji, które może wykonać użytkownik:

(a) Pokój startowy:

- włączenie instrukcji
- przełączenie do następnej sceny

(b) Pokój pierwszy - równia pochyła:

- przełączenie do poprzedniej sceny
- włączenie instrukcji
- położenie samochodzika na równi
- odłożenie samochodzika na miejsce
- zresetowanie stopera
- podwyższenie równi pochyłej
- obniżenie równi pochyłej
- pokazanie tabeli wyników
- przełączenie do następnej sceny

(c) Pokój drugi - siłomierz:

- przełączenie do poprzedniej sceny
- włączenie instrukcji
- podniesie ciężarka z półki
- odłożenie ciężarków na miejsce
- położenie ciężarka na wadze
- podniesienie ciężarka z wagi
- zawieszenie ciężarka na siłomierzu
- dodanie wyniku do tabeli
- zamiana linki
- pokazanie tabeli wyników
- przełączenie do następnej sceny

(d) Pokój trzeci - wahadło matematyczne:

- przełączenie do poprzedniej sceny
- włączenie instrukcji
- podniesie kulki
- odłożenie kulki na miejsce
- położenie kulki na wadze
- podniesienie kulki z wagi
- zawieszenie kulki na wahadle
- wydłużenie linki
- skrócenie linki
- włączenie wahadła
- zresetowanie stopera
- pokazanie tabeli wyników

## 5.3 Metoda pomiaru czasu

W projekcie wykorzystano dwa pomiary czasu. Mierzony jest czas zjazdu samochodzika z równi pochyłej oraz czas danej liczby wahnięć wahadła matematycznego. Oba zostały zaimplementowane w oparciu o obiekty wchodzące w zasięg triggera, przy czym stoper w pierwszym doświadczeniu kończy pomiar w momencie kiedy samochodzik opuści równię. Natomiast w drugim przypadku działa nieprzerwanie, aż do zatrzymania przez użytkownika, a pomiar dodawany jest to tabeli wyników po pięciu wahnięciach wahadła. Ponieważ dla obu stoperów implementacja jest podobna, zostanie przedstawiony dokładnej jedynie skrypt obsługujący zjazd z równi

pochyłej.

```
funkcja Start:
    time0 = startTime;
    timerText.text = startTime.ToString("F2");

funkcja OnTriggerEnter(Collider other):
    jeżeli (other.CompareTag("car"))
        wtedy (return)
    isInTrigger = true

funkcja OnTriggerExit(Collider other):
    isInTrigger = false;
    jeżeli (!other.CompareTag("car"))
    wtedy  (time_1.text = startTime.ToString("F2"))

funkcja Update:
    jeżeli (isInTrigger == true)
        wtedy (startTime += Time.deltaTime
               timerText.text = startTime.ToString("F2"))
    jeżeli (isInTrigger == false)
        wtedy (jeżeli (Input.GetKeyDown(KeyCode.T))
                   wtedy (timerText.text = time0.ToString("F2")
                          startTime = time0))
```

Na początku w funkcji *Start* zmiennej time0 zostaje przypisana wartość czasu startowego, który wynosi 0. Wartość ta jest wyświetlona na stoperze. W momencie wejścia **Collidera** dołączonego do obiektu z odpowiednim tagiem, tak jak przy poprzednim skrypcie, w funkcji *OnTriggerEnter* isInTrigger zostanie zmienione na true. Funkcja *OnTriggerExit*, oprócz zmiany wartości zmiennej isInTrigger, dodaje wynik pomiaru do tabeli. Na stoperze wyświetlony będzie czas wyjścia z triggera. Jeżeli badany obiekt znajduje się w triggerze, funkcja *Update* co klatkę zmienia wartość czasu wyświetlanego na stoperze. *Time.deltaTime* to czas, który upłynął od ostatniej klatki. W tym momencie zmienna startTime jest czasem rzeczywistym. Natomiast jeżeli w triggerze równi pochyłej nie znajduje się żaden obiekt ze wskazanym tagiem, a użytkownik naciśnie na klawiaturze klawisz T, startTime przyjmie z powrotem wartość 0, co zostanie również wyświetlone na stoprze. W ten sposób mogą być wykonywane kolejne pomiary. Jeżeli użytkownik nie zresetuje wskazań stopera, czas będzie liczony od momentu jego zatrzymania.

## 5.4 Przykładowy skrypt - ruch wahadła

W trzecim doświadczeniu badane jest wahadło matematyczne. Porusza się ono jednostajnie i nie działają na nie żadne siły oporu. Poniżej znajduje się fragment skryptu odpowiadający za ten ruch.

```
funkcja FixedUpdate:
 jeżeli (measure == true)
   wtedy (timer += Time.fixedDeltaTime
          jeżeli (timer > 1f)
            wtedy (phase++
                   phase %= 4
                   timer = 0f)
          switch (phase)
            warunek 0: transform.Rotate(0f, 0f, speed * (1 - timer))
            warunek 1: transform.Rotate(0f, 0f, -speed * timer)
            warunek 2: transform.Rotate(0f, 0f, -speed * (1 - timer))
            warunek 3: transform.Rotate(0f, 0f, speed * timer))
 jeżeli (measure == false)
    wtedy (transform.rotation = Quaternion.identity)
```

Funkcja *Update* zawiera instrukcje wykonywane wraz z renderowaniem się sceny, natomiast funkcja *FixedUpdate* ma częstotliwość odświeżania zgodną z silnikiem fizycznym. Następuje to, zgodnie z domyślnymi ustawieniami, 50 razy na sekundę. Powinna być stosowana podczas implementacji przykładania sił, momentów obrotowych lub innych funkcji związanych z fizyką. Dlatego też czas jest obliczany przy pomocy *fixedDeltaTime*, a nie samego *DeltaTime*. Ruch wahadła odbywa się wzdłuż osi Z i zaimplementowany został metodą switch-case, gdzie w zależności od danej fazy ruchu:

- warunek 0 - obrót dla prędkości od maksimum do 0;
- warunek 1 - obrót dla prędkości od 0 do maksimum w przeciwnym kierunku;
- warunek 2 - obrót dla prędkości od maksimum do 0 w przeciwnym kierunku;
- warunek 2 - obrót dla prędkości od 0 do maksimum.

Kiedy wykonywanie pomiarów zostanie zakończone, obiekt wraca do punktu początkowego. Do implementacji obrotów w przestrzeni trójwymiarowej można wykorzystać kwaterniony. Kwaternion jest wektorem postaci [x y z w] w $\mathbb{R}^4$ z odpowiednio określonymi działaniami. Obrót opisany jest w następujący sposób (gdzie RotationAngle podane jest w radianach)

x = RotationAxis.x * sin(RotationAngle / 2)
y = RotationAxis.y * sin(RotationAngle / 2)
z = RotationAxis.z * sin(RotationAngle / 2)
w = cos(RotationAngle / 2).

Komenda *Quaternion.identity* odpowiada za przekształcenie tożsamościowe, czyli obrót o kąt $0°$. To przekształcenie nazywane jest również jedynką kwaternionową i ma postać $1+0i+0j+0k$, gdzie $i^2 + j^2 + k^2 = -1$.

## 5.5 Przykładowy skrypt - podnoszenie obiektów

Poniższy skrypt dołączony jest do wszystkich obiektów, które można podnieść. Są to ciężarki w doświadczeniu drugim oraz kulki w doświadczeniu trzecim. Wymagane jest, aby obiekty te posiadały komponent **Rigidbody**.

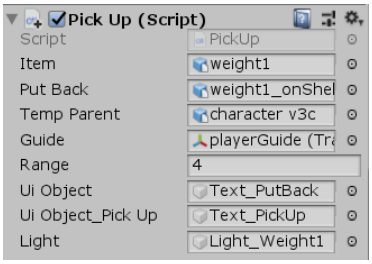
---

```
funkcja Start:
    item.GetComponent<Rigidbody>().useGravity = true
    UiObject.SetActive(false)
    UiObject_PickUp.SetActive(false)

funkcja Update:
    Light.SetActive(false);
    UiObject_PickUp.SetActive(false);
    jeżeli (carrying == false i (guide.transform.position -
    ↪ transform.position).sqrMagnitude < range * range i
    ↪ tempParent.transform.childCount <4)
        wtedy(Light.SetActive(true)
              UiObject_PickUp.SetActive(true)
              jeżeli (Input.GetKeyDown(KeyCode.F))
                  wtedy(UiObject_PickUp.SetActive(false)
                        wywołaj funkcję Pickup
                        carrying = true
                        Light.SetActive(false)))
        w przeciwnym razie jeżeli (carrying == true)
                  wtedy (UiObject.SetActive(true)
                        jeżeli (Input.GetKeyDown(KeyCode.G))
                            wtedy(wywołaj funkcję Drop
                                  carrying = false
                                  UiObject.SetActive(false)))

funkcja Pickup:
    item.GetComponent<Rigidbody>().useGravity = false
    item.GetComponent<Rigidbody>().isKinematic = true
    item.transform.position = guide.transform.position
    item.transform.rotation = guide.transform.rotation
    item.transform.parent = tempParent.transform
```

```
funkcja Drop:
    item.GetComponent<Rigidbody>().useGravity = true
    item.GetComponent<Rigidbody>().isKinematic = false
    item.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePosition
    item.transform.parent = null
    item.transform.position = putBack.transform.position
    item.transform.rotation = putBack.transform.rotation
```




Rysunek 21: Ustawienia wartości zmiennych skryptu umożliwiającego podnoszenie przedmiotów dla przykładowego obiektu.

Każdy obiekt posiada komponent **Transform**, który określa jego pozycję względem współrzędnych x, y i z, obrót wokół osi X, Y i Z mierzony w stopniach, oraz skalę względem tych osi. Stąd obliczenia położeń i obrotów odbywają się przy pomocy komendy *transform*, która zmienia te wartości z poziomu skryptu. Na Rysunku 21 znajdują się wartości zmiennych zastosowanych we wcześniejszym skrypcie, które ustalane są z poziomu silnika. Akcje wywołane w funkcji *Start* nie są konieczne, ale dają dodatkową pewność właściwych wartości parametrów. Na początku włączana jest siła grawitacji działająca na **Rigidbody** danego obiektu oraz dezaktywowane są elementy interfejsu - polecenia "podnieś" i "odłóż". W funkcji *Update* sprawdzane jest czy obiekt jest podniesiony. W tym wypadku zasięg zdarzenia nie jest wyznaczany przez **Collider**, a przez zadaną wartość zmiennej range. Instrukcja

$$(guide.transform.position - transform.position).sqrMagnitude < range * range$$

jest najbardziej wydajną metodą sprawdzania odległości od obiektu w sposób ciągły. Jako zmienną guide przyjęty został obiekt punktowy umieszczony w ręce modelu postaci. *sqrMagnitude* zwraca długość danego wektora. Warunek $tempParent.transform.childCount < 4$ został dodany w celu uniknięcia błędu przenoszenia więcej niż jednego ciężarka, ponieważ odbywa się to poprzez przypisanie go jako komponent postaci. Akcja podniesienia zostanie wykonana w przypadku naciśnięcia przez użytkownika klawisza F. Jeżeli obiekt jest w danym momencie niesiony, można go odłożyć naciskając klawisz G. Ponieważ w przypadku tego zdarzenia nie jest badana odległość, powoduje to również powrót na miejsce początkowe każdego obiektu, do którego dodany jest powyższy skrypt. Funkcja *Pickup* odpowiada za transformację współrzędnych położenia i obrotu. Dodatkowo na obiekt przestaje bezpośrednio działać grawitacja i zostaje on przeniesiony we wskazany punkt w przestrzeni, ustalony względem modelu postaci. W funkcji *Drop* obiekt odkładany jest do punktu początkowego wskazywanego przez obiekt przypisany

do zmiennej putBack. Jest to zduplikowany model ciężarka umieszczony w odpowienim miejscu, a następnie dezaktywowany. Pobierając wartości z komponentu **Transform** tego obiektu, ponownie została wykorzystana wcześniej wspomniana transformacja. Następnie zablokowany zostaje ruch i obroty we wszystkich osiach. Dodatkowo obiekt przestaje być komponentem postaci.

# 6 Podsumowanie

Zaprezentowane w tej pracy wirtualne laboratorium zostało w całości przygotowane przez jedną osobę. W środowisku zajmującym się rozwijaniem gier komputerowych tego typu projektem zajmują się zazwyczaj duże zespoły programistów, grafików, jak i game designerów. Przedstawiony program można uznać za funkcjonujący prototyp, a założenia (mimo kilku zmian) za spełnione. Pozostawiona została za to ogromna przestrzeń na udoskonalenia i rozwijanie go.

## 6.1 Udoskonalenie grafiki

Wybór stylu w jakim przygotowana została grafika motywowany był głównie przez ograniczenia czasowe na przygotowanie modeli jak i ograniczenia sprzętowe. Obliczanie prostszych geometrii jest dla komputera zdecydowanie mniej obciążające, a wzięta została pod uwagę również moc urządzeń dostępnych w salach lekcyjnych. Możliwe jest natomiast dopracowanie grafiki w obecnej formie, tak aby bardziej przypominała realistyczną oraz bardziej dostosowaną do współczesnego użytkownika. Oczywiście przy większej ilości dostępnego czasu można by przygotować dodatkowe modele przedmiotów dekoracyjnych w celu zapełnienia pustych przestrzeni i zwiększenia zainteresowania odbiorcy. Wybór kolorystyki również może być otwarty do dyskusji z potencjalnym użytkownikiem końcowym lub może być udostępniona opcja wyboru koloru.

## 6.2 Rozbudowanie fizyki doświadczeń

Kolejnym krokiem rozwijania projektu mogłoby być rozbudowanie zawartej w nim fizyki. Szczególnie biorąc pod uwagę parametry dostępne w doświadczeniach. Ciekawym pomysłem mogłaby być zmiana wartości przyspieszenia grawitacyjnego. Umożliwiłoby to na przykład pomiary w środowisku zbliżonym do warunków panujących na Księżycu. Co pozwoliłoby jeszcze bardziej zwiększyć atrakcyjność programu, ponieważ byłyby to pomiary możliwe do wykonania jedynie w laboratorium wirtualnym.

Obecnie dostępne zmiany parametrów wykonywanych doświadczeń można łatwo zagęścić o kolejne skoki. Wymagałoby to jedynie przygotowania kilku prostych modeli 3D oraz dodania miejsca na pomiary w tabelach wyników. Rozszerzenie sposobu wyboru parametru na pewien określony przedział mogłoby być bardziej skomplikowane. Chociaż ułatwieniem byłoby zastosowanie jednego modelu, to musiałby być on skalowany przy pomocy kodu, co nie zawsze daje poprawne efekty wizualne. Dodatkowo należałoby zmienić metodę zapisywania wyników do tabeli. Jednakże jest to możliwe do wykonania.

## 6.3 Dodanie kolejnych doświadczeń

Jak zostało wspomniane wcześniej, wszystkie funkcjonalności były dodawane w sposób modularny. Natomiast pokoje zawierają podobne elementy i są zbudowane na tym samym modelu pomieszczenia, na który nałożone zostały różne tekstury. Ustawienia interfejsu użytkownika,

kamery oraz oświetlenia można eksportować pomiędzy scenami. Dzięki temu można szybko dołączyć kolejne eksperymenty. Należałoby jedynie przygotować niezbędne modele i ewentualne dodatkowe skrypty.

W tej wersji projektu, z powodu ograniczenia czasowego, skupiono sie na dwóch działach fizyki. W laboratorium można wykonać doświadczenia jedynie z dynamiki oraz mechaniki. Starano się minimalizować ilość potrzebnych skryptów i implementować je jak najbardziej uniwersalnie, aby można je było wykorzystać do różnych obieków. Na przykład licznik czasu został napisany w sposób umożliwiający dostosowanie do danych pomiarów. Korzystając z dostępności obiektów typu **Physical Material**, zgodnie z podstawą programową, można dodać zderzenia wraz z wyznaczaniem masy lub prędkości jednego z ciał, korzystając z zasady zachowania pędu. Biorąc pod uwagę możliwości silnika, można rozważać rozbudowanie projektu o dodatkowe działy takie jak prąd stały lub magnetyzm. Jako kolejne doświadczenie możnaby zaproponować na przykład badanie napięć w układzie ogniw połączonych szeregowo.

Repozytorium projektu znajduje się pod adresem: *https://github.com/katwojewoda/magisterka*

Program do pobrania znajduje się pod adresem: *https://github.com/katwojewoda/wirtualne-laboratorium*

# Literatura


[1] Blender 2.92 Manual
https://docs.blender.org/manual/en/2.90/index.html

[2] Blender licencja
https://www.blender.org/about/license/

[3] Unity User Manual (2019.2)
https://docs.unity3d.com/2019.2/Documentation/Manual/index.html

[4] Unity licencje
https://store.unity.com/compare-plans?currency=USD

[5] Unity User Manual (2019.2), Lista pakietów
https://docs.unity3d.com/2019.2/Documentation/Manual/pack-build.html

[6] Podstawa programowa liceum
https://podstawaprogramowa.pl/Liceum-technikum/Fizyka

[7] Ludwik Lehman, Witold Polesiuk, Grzegorz F. Wojewoda, *FIZYKA. PODRĘCZNIK. KLASA 1. ZAKRES PODSTAWOWY. REFORMA 2019.* Wydawnictwa Szkolne i Pedagogiczne, Warszawa 2019

[8] Blender 2.92 Manual, Modelowanie, Modyfikatory
https://docs.blender.org/manual/en/latest/modeling/modifiers/index.html

[9] Unity User Manual (2019.2), Shader Standardowy
https://docs.unity3d.com/2019.2/Documentation/Manual/shader-StandardShader.html

[10] Unity User Manual (2019.2), CharacterController
https://docs.unity3d.com/2019.2/Documentation/Manual/class-CharacterController.html