WERSJA	DATA	ZMIANY
0.1	4.05.2024	Przygotowanie dokumentu
1.0	29.05.2024	Opisanie rozdziałów
1.1	4.06.2024	Dodanie diagramów UML

SYMULATOR RZUTU CIAŁEM

Autor: inż. Jakub Klęsk
Akademia Górniczo-Hutnicza

Spis treści

Lista oznaczeń

API	Application Programming Interface
GLU	OpenGl Utility
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
SDK	Software Development Kit
STL	Standard Template Library

1. Wstęp

Dokument dotyczy opracowania symulatora rzutu obiektem w przestrzeni trójwymiarowej. Podstawowym zadaniem tego oprogramowania jest symulacja zjawisk fizycznych takich jak np. Prawa Newtona oraz zobrazowanie ich w przestrzeni za pomocą interfejsu graficznego. Założeniem projektu jest stworzenie możliwości rzutu ciałem ustawiając parametry takie jak kąt rzutu i siła rzutu.

1.1 Wymagania systemowe (requirements)

Podstawowe założenia projektu:

1. Przygotowanie abstrakcyjnego opisu i materiałów potrzebnych do zaimplementowania algorytmów.
2. Określenie wymagań i opracowanie architektury podsystemu graficznego.
3. Określenie wymagań interfejsu użytkownika.
4. Implementacja podsystemu graficznego i jego API.
5. Testy podsystemu graficznego
6. Implementacja rzucanego ciała
7. Implementacja rzutu.
8. Implementacja zmiany siły rzutu w API.
9. Implementacja zmiany kąta rzutu w API.
10. Implementacja pozostania ciała na planszy po rzucie.
11. Testy końcowe.

2. Funkcjonalność

1) **Funkcjonalność interfejsu graficznego**
   a) Obrazowanie przestrzeni trójwymiarowej
   b) Generacja sfery (ciała rzucanego)
   c) Ustawienie siły oraz kątu rzutu
   d) Obracanie kamery
   e) Obrazowanie rzutu

2) **Funkcjonalność silnika fizyki**
   a) Symulacja zachowania ciała fizycznego w przestrzeni o zadanym przyśpieszeniu grawitacyjnym na zadaną siłę
   b) Obsługa zderzenia z powierzchnią płaską (symulacja zatrzymania)
   c) Generacja ciała fizycznego o zadanym rozmiarze przez programistę

3) **Funkcjonalność interfejsu użytkownika**
   a) Uruchomienie programu z argumentami umożliwiającymi ustawienie:
      -Siłę rzutu
      -Startu symulacji
      -Ustawienia kątu rzutu
      -Resetu symulacji (powrotu ciała do miejsca początkowego)
   b) Możliwość oglądnięcia symulacji rzutu
   c) Możliwość ponownego rzutu po skończonym poprzednim

3. Analiza problemu (*problem analysis*)

Projekt ten symuluje rzut kulą na odległość, uwzględniając takie parametry jak siła rzutu, kąt rzutu oraz odbicia kuli od powierzchni. W celu uproszczenia symulacji, zakładamy, że nie ma wpływu wiatru, a jedynymi siłami działającymi na kulę są siła ciężkości i siły wynikające z interakcji kuli z powierzchnią (odbić). Powierzchnią, na której odbija się kula, jest trawiasty teren, który modelujemy jako idealnie sprężysty.

Generacja sfery

W przestrzeni trójwymiarowej jednym z podstawowych kształtów jest sfera, która, podobnie jak każdy inny kształt, musi być zbudowana z płaskich trójkątów. Budowę sfery (a w zasadzie jej przybliżenia) najprościej rozpocząć od stworzenia wierzchołków na jej powierzchni . Do określenia współrzędnych punktów używa się współrzędnych sferycznych:

$$x = r\,cos\theta\,cos\varphi$$

$$y = r\,cos\theta\,sin\varphi$$

$$z = r\,sin\theta$$

Gdzie $\varphi \in [0,2\pi]$ - długość azymutalna, $\theta \in \left[-\frac{\pi}{2},\frac{\pi}{2}\right]$-odległość zenitalna, r -stały promień sfery

Współrzędne są liczone dla dyskretnych punktów (ich skończonej ilości) przez ustawienie odpowiedniego kroku zmiany kątów. Tak wygenerowane wierzchołki są umieszczane w tablicy wierzchołków.

Konieczne jest zmapowanie wierzchołków tak, aby tworzyły trójkąty. W tym celu brane są takie cztery wierzchołki, które tworzą czworokąt, tj. dwie pary wierzchołków – o takiej samej długości azymutalnej i o takiej samej odległości zenitalnej, a następnie tworzy się z nich dwa trójkąty. Indeksy wierzchołków tworzące trójkąty wpisuje się do tablicy indeksów.

Prawa fizyczne

1) **Ruch w rzucie ukośnym:**
   - Równania ruchu w dwóch wymiarach (x, y) dla rzutu ukośnego:

$$x(t) = v_0 \cos(\theta)\, t$$

$$y(t) = v_0 \sin(\theta)\, t - \frac{1}{2} g t^2$$

gdzie v_0 to początkowa prędkość nadania kuli.

   - Początkowa prędkość jest związana z siłą rzutu F i masą kuli m za pomocą drugiej zasady dynamiki Newtona:

$$v_0 = \frac{F}{m} \circ t_{kontakt}$$

gdzie $t_{kontakt}$ to czas kontaktu ręki z kulą podczas rzutu, który możemy założyć jako stałą wartość

2) **Maksymalna wysokość rzutu:**
   - Maksymalna wysokość *H* osiągana przez kulę jest dana równaniem:

$$H = \frac{(v_0 \sin(\theta))^2}{2g}$$

3) **Zasięg rzutu:**
   - Całkowity zasięg R rzutu, zanim kula pierwszy raz zetknie się z ziemią można obliczyć jako:

$$R = \frac{v_0^2 \sin(2\theta)}{g}$$

4) **Odbicia kuli:**
   - Przy założeniu idealnie sprężystego odbicia, prędkość pionowa po odbiciu jest równa prędkości przed odbiciem, ale skierowana w przeciwną stronę.
   - Prędkość pozioma pozostaje niezmieniona, gdyż zakładamy brak tarcia.

   Po odbiciu kula będzie kontynuowała ruch zgodnie z tymi samymi równaniami ruchu, ale z nowymi warunkami początkowymi ustalonymi przez prędkość po odbiciu

Analiza Dynamiki

1. **Czas wznoszenia do maksymalnej wysokości**

$$t_{up} = \frac{v_0 \sin(\theta)}{g}$$

2. **Całkowity czas lotu (do pierwszego kontaktu z ziemią):**

$$t_{total} = \frac{2v_0 \sin(\theta)}{g}$$

3. **Trajektoria po odbiciu:**
   - Po odbiciu, nowe warunki początkowe będą:

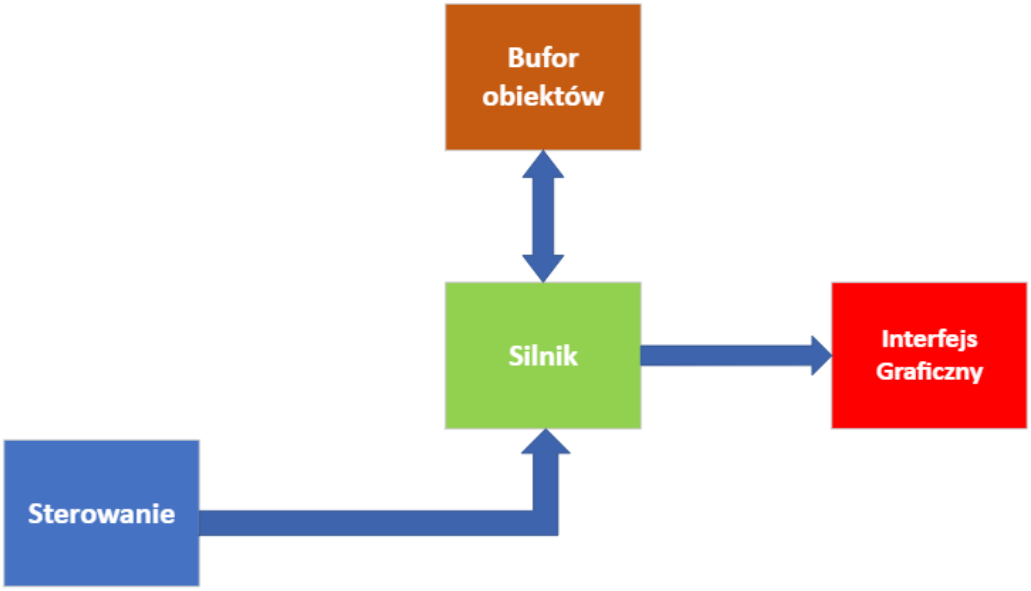
$$v_{x,new} = v_0 \cos(\theta)$$

$$v_{y,new} = -v_0 \sin(\theta)$$

   - Trajektoria bedzię symulowana z nowymi warunkami początkowymi

4. Projekt techniczny (*technical design*)

W projekcie symulacji rzutu kulą w OpenGL zastosowano podejście modułowe, które obejmuje kilka kluczowych elementów: strukturę danych dla kuli, stan maszyny, oraz ogólną architekturę systemu.

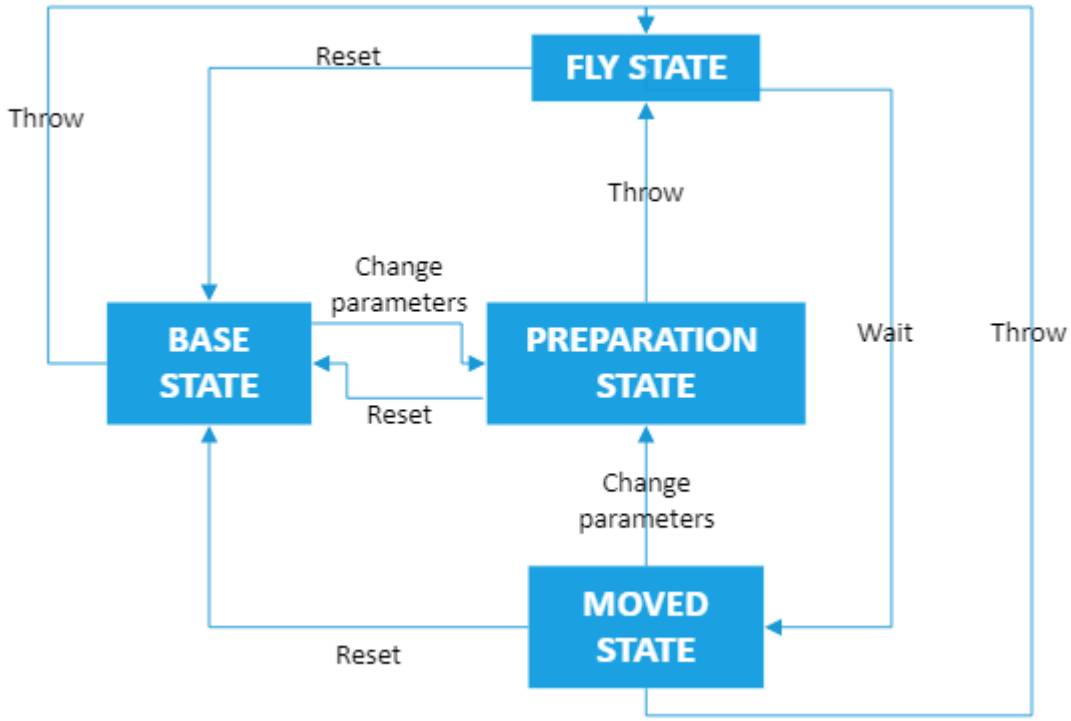


Rysunek 4-1. Budowa systemu

Architektura systemu składa się z kilku modułów, które współpracują ze sobą w celu realizacji symulacji.

Moduły Systemu:

- **Sterowanie**: Odpowiada za interakcję użytkownika (klawiatura, mysz).
- **Silnik**: Przetwarza logikę gry oraz fizykę ruchu kuli.
- **Bufor Obiektów**: Przechowuje dane dotyczące położenia i stanu obiektów.
- **Interfejs Graficzny**: Odpowiada za renderowanie sceny.



Rys.4.2. Przykładowa wizualizacja maszyny stanów

Stan Maszyny

W całej symulacji możemy wyróżnić 4 stany, które zarządzają róznymi etapami rzutu kulą.

Opis stanów:

- **BASE STATE**: Stan początkowy, w którym kula znajduje się w pozycji wyjściowej.
- **PREPARATION STATE**: Użytkownik może ustawić parametry rzutu (siła i kąt).
- **FLY STATE**: Kula jest w trakcie lotu.
- **MOVED STATE**: Kula osiągnęła punkt docelowy, po czym następuje reset lub zmiana parametrów.

W programie znajduje się również struktura ‘Spherical’, która służy do reprezentacji położenia kamery w przestrzeni sferycznej:

```
struct Spherical
{
    float distance, theta, phi;
    Spherical(float gdistance, float gtheta, float gphi) : distance(gdistance), theta(gtheta), phi(gphi) { }
    float getX() { return 1.0 * cos(phi); }
    float getY() { return 1.0 * sin(theta); }
    float getZ() { return 1.0 * sin(phi); }
};
```

2. Opis realizacji (*implementation report*)

Cel Projektu

Celem projektu było stworzenie symulacji rzutu kulą w środowisku 3D, przy użyciu biblioteki OpenGL do renderowania grafiki oraz SFML do obsługi zdarzeń i interfejsu graficznego. Projekt miał na celu przedstawienie fizyki ruchu kuli, uwzględniając siłę i kąt rzutu oraz grawitację.

Etapy Realizacji:

1. Planowanie i Projektowanie

- Określenie wymagań projektu, w tym potrzebne funkcjonalności i interfejs użytkownika.

- Zaplanowanie struktury kodu, podział na moduły odpowiedzialne za różne aspekty symulacji (np. renderowanie, fizyka, interakcja użytkownika).

2. Środowisko Programistyczne

- Konfiguracja środowiska programistycznego z użyciem Visual Studio oraz instalacja niezbędnych bibliotek: OpenGL, SFML, ImGui.

- Utworzenie szablonu projektu, zawierającego podstawowe funkcje inicjalizacji i pętli głównej.

3. Implementacja Podstawowych Funkcji

- **Inicjalizacja OpenGL:** Ustawienie parametrów renderowania, wczytanie tekstur i konfiguracja oświetlenia.

- **Kamera:** Implementacja funkcji do poruszania i obracania kamery, aby umożliwić obserwację sceny z różnych perspektyw.

- **Rysowanie Sceny:** Definicja funkcji do renderowania kuli oraz płaskiej powierzchni z teksturą, dodanie pomocniczych linii oznaczających odległość.

4. Fizyka Ruchu Kuli

- Implementacja funkcji symulującej ruch kuli uwzględniając siłę i kąt rzutu, a także grawitację.

- Dodanie obsługi odbicia kuli od powierzchni, z uwzględnieniem strat energii przy odbiciu.

5. Interfejs Użytkownika

- Integracja z biblioteką ImGui w celu stworzenia interaktywnego interfejsu do sterowania parametrami symulacji.

- Dodanie suwaków do regulacji siły i kąta rzutu, przycisków do rozpoczęcia animacji oraz resetowania pozycji kuli.

6. Testowanie i Poprawki

- Przeprowadzenie testów funkcjonalnych w celu sprawdzenia poprawności działania symulacji oraz interakcji użytkownika.

- Optymalizacja kodu oraz poprawki błędów wykrytych podczas testowania.

Wyniki i Wnioski

Projekt zakończył się sukcesem, a stworzona symulacja pozwala na interaktywne obserwowanie ruchu kuli rzucanej z określoną siłą i pod określonym kątem. Implementacja realistycznej fizyki ruchu kuli oraz możliwość sterowania kamerą i parametrami rzutu pozwala użytkownikowi na lepsze zrozumienie wpływu różnych czynników na tor ruchu kuli.

Podsumowanie

Realizacja projektu pozwoliła na zdobycie praktycznych umiejętności w zakresie programowania grafiki 3D oraz symulacji fizycznych. Wykorzystanie bibliotek OpenGL, SFML i ImGui umożliwiło stworzenie zaawansowanego projektu z interaktywnym interfejsem użytkownika, który może być podstawą do dalszych, bardziej zaawansowanych symulacji.

3. Opis wykonanych testów (*testing report*) - lista buggów, uzupełnień, itd.

Kod usterki	**Data**	**Autor**	**Opis**	**Stan**
#TEST1	02.06.2024	Jakub Klęsk	Test poprawności działania	Działa prawidłowo

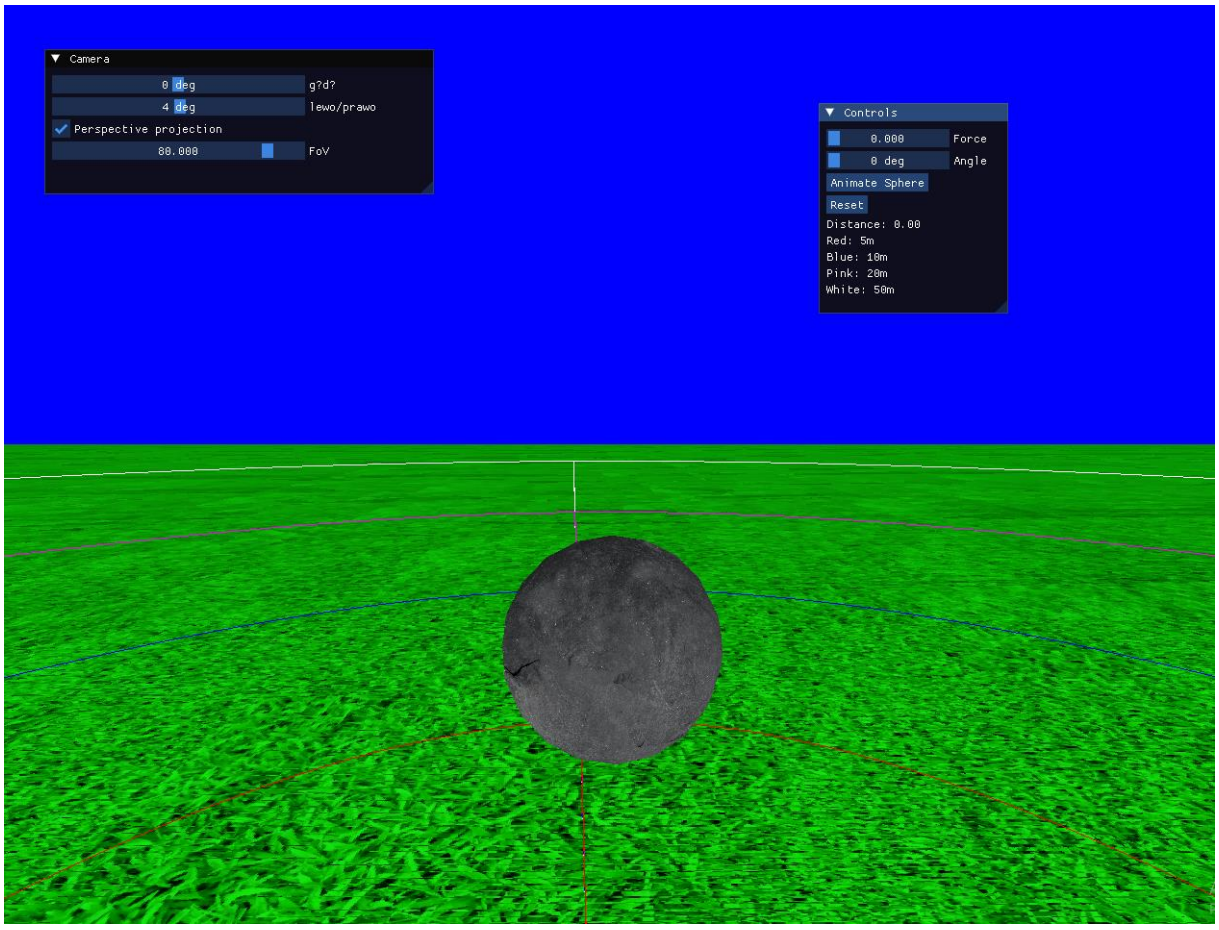
4. Podręcznik użytkownika (*user's manual*)

Uruchomienie programu

Oprogramowanie należy uruchamiać na 64-bitowej platformie Microsoft Windows 10 z obsługą OpenGL w wersji 2.0 lub nowszej. Do uruchomienia konieczne są odpowiednie biblioteki współdzielone, które są załączone wraz ze skompilowanym programem.

Po włączeniu symulacji pojawiają się dwa okna. Jedno z konsolą CMD, drugie z interfejsem graficznym.

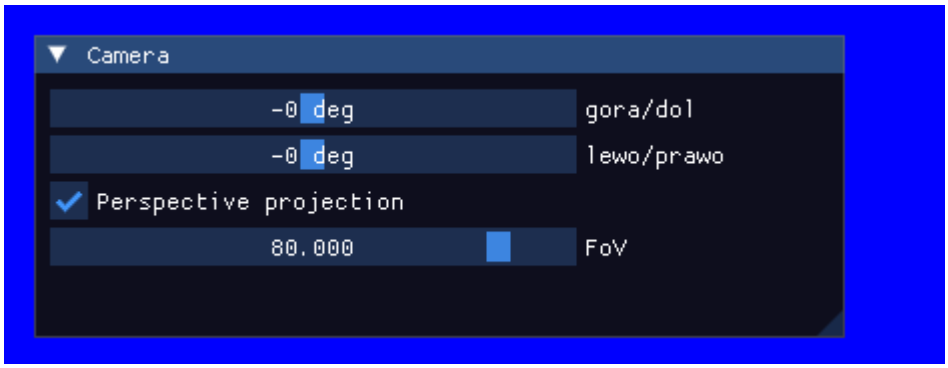
Zajmujemy tylko się drugim. Okno przedstawione na rysunku 7.1 jest oknem interfejsu graficznego.



Rysunek 4.-1. *Okno interfejsu graficznego*

Są tutaj dwa panele

- jeden odpowiadający za ustawienie kamery(Rys.7.2)
- drugi odpowiadający za rzut(Rys7.3).

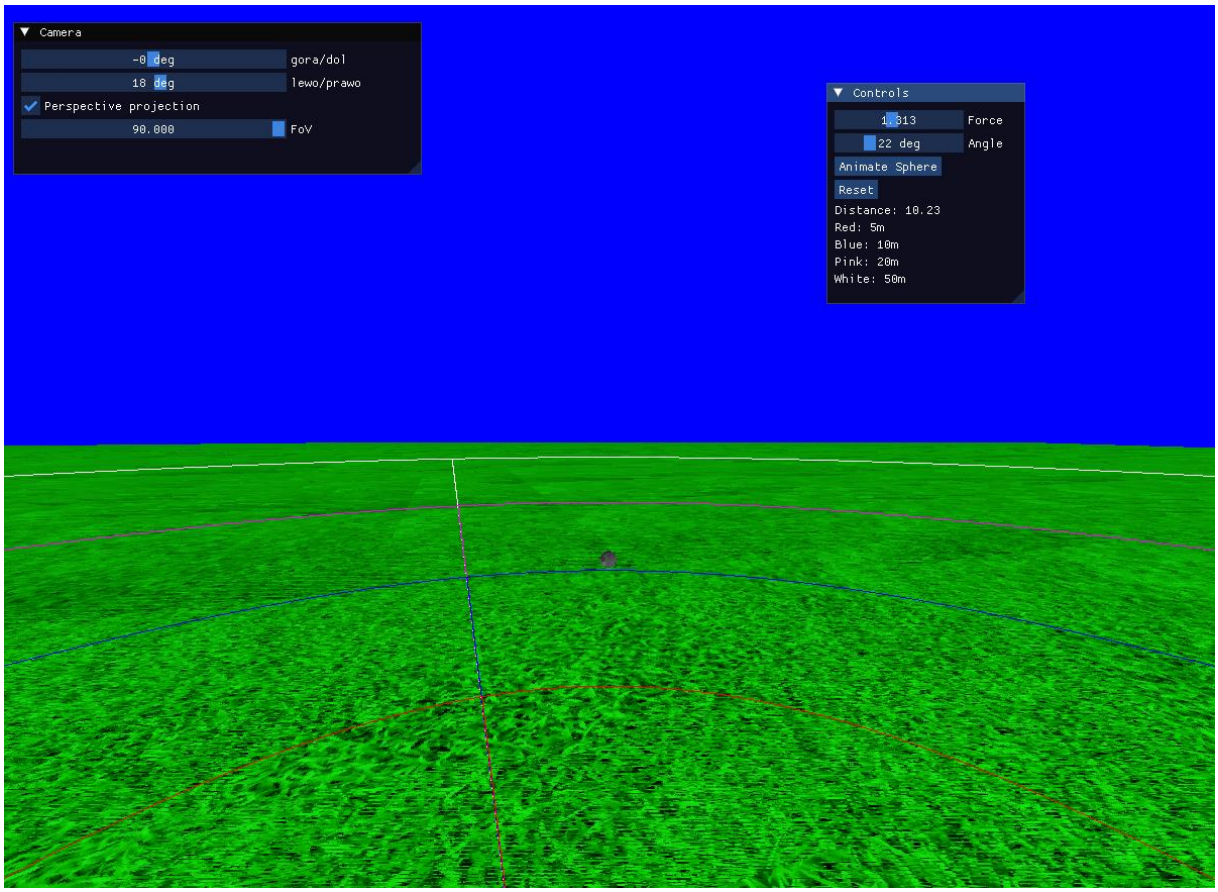


Rys.7.2 Panel obsługujący kamere

Za pomocą tego panelu można decydować w którym kierunku poleci kula. Odpowiada za to kontrolka “lewo/prawo” pozostałe dwie kontrolki odpowiadają za ustawienie kamery. Kontrolka “gora/dol”, jak nazwa wskazuje pozwala obracać kamere w płaszczyźnie pionowej. Kontrolka “FoV” odpowiada za zmiane pola widzenia

Rys.7.3 Panel obsługujący rzut

Za pomocą tego panelu można ustawiać parametry rzutu. Kontrolka “**Force**” odpowiada za siłę rzutu, kontrolka “**Angle**” odpowiada za kąt rzutu, kontrolka “**Animate Sphere**” zaczyna symulacje rzutu, a kontrolka “Reset” odpowiada za powrót kuli do pozycji początkowej. Poniżej jest również napisany dystans jaki dzieli pozycje kamery od kuli.

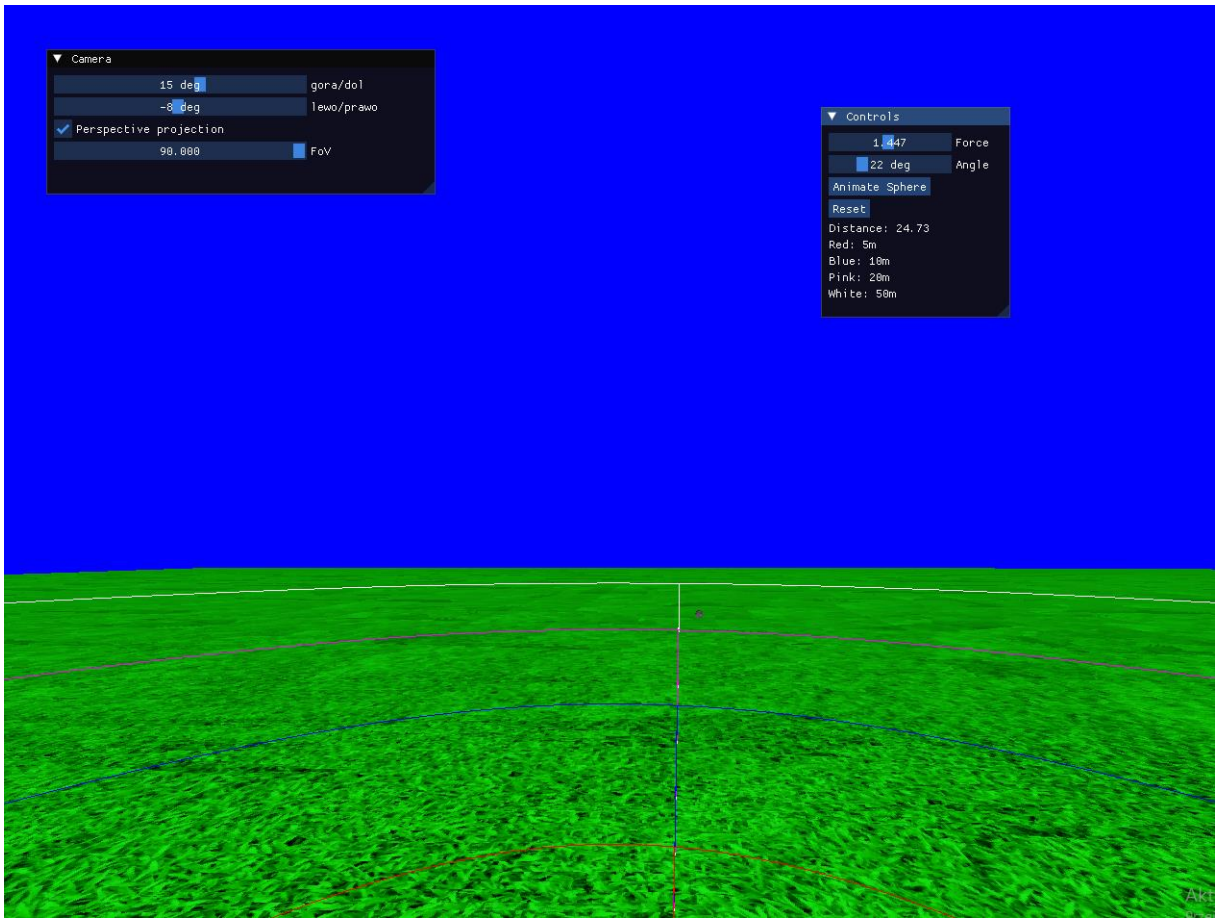


Rys.7.4 Widok programu po przykładowym rzucie

Po rzucie są dwie możliwości:

-ponowny rzut ze zmianą lub zostawianem parametrów Rys.7.5,

-zresetować rzut Rys.7.6,



Rys.7.5 Przykładowy widok po ponownym rzucie i zmianie parametrów

5. Metodologia rozwoju i utrzymania systemu (*system maintenance and deployment*)

Aby zapewnić skuteczny rozwój i utrzymanie prostego systemu symulacji rzutu kulą w OpenGL, przyjęto uproszczoną, iteracyjną metodologię. Proces rozpoczęto od zdefiniowania podstawowych wymagań funkcjonalnych, takich jak możliwość ustawienia siły i kąta rzutu oraz realistyczne odbicie kuli od powierzchni.

Kolejnym krokiem było projektowanie systemu, obejmujące wybór narzędzi (OpenGL i SFML) oraz podstawowe struktury kodu. Następnie przystąpiono do implementacji, rozwijając projekt w krótkich iteracjach, aby łatwo wprowadzać poprawki i nowe funkcje. W trakcie implementacji tworzono i testowano moduły, takie jak fizyka ruchu kuli i rendering grafiki.

Testowanie systemu odbywało się na bieżąco, poprzez manualne sprawdzanie poprawności działania poszczególnych funkcji. Błędy były naprawiane na bieżąco, a zmiany w kodzie wprowadzane stopniowo, aby zachować stabilność projektu.

Po zakończeniu implementacji przeprowadzono końcowe testy systemowe, aby upewnić się, że wszystkie funkcje działają zgodnie z oczekiwaniami. System wdrożono na lokalnym środowisku, a dokumentacja użytkownika i techniczna została uaktualniona, aby odzwierciedlać końcowy stan projektu.

Utrzymanie systemu polega na regularnym monitorowaniu jego działania oraz wprowadzaniu niezbędnych aktualizacji i poprawek. W miarę potrzeby, projekt można rozbudowywać o nowe funkcje, opierając się na zebranym feedbacku i nowych pomysłach.

Podsumowując, prosty i iteracyjny proces rozwoju, testowania i wdrażania pozwolił na skuteczną realizację projektu symulacji rzutu kulą, zapewniając jego stabilne działanie i łatwość wprowadzania przyszłych modyfikacji.

Bibliografia

[Stroustrup, Sommervile, McConnel, Hunt & Thomas]

[1] Bronsztejn I.N., Siemiendiajew K.A., Musiol G., Mühlig H.: Nowoczesne kompendium matematyki, PWN, 2004.

[2] Cyganek B., Siebert J.P.: An Introduction to 3D Computer Vision Techniques and Algorithms, Wiley, 2009.

[3] Mitra S.K.: Digital Signal Processing. A Computer-Based Approach, 2001.

[4] Oppenheim A.V., Schafer R.W.: Discrete-Time Signal Processing, Prentice Hall, 1989.

[5] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P.: Numerical Recipes in C. The Art of Scientific Computing. Second Edition. Cambridge University Press, 1999.

[6] Rao K.R.: Fast Fourier Transform: Algorithms and Applications, Signals and Communication Technology, Springer Science & Business Media, 2010.