1. Cel ćwiczenia:

Zapoznanie z programowaniem grafiki przy użyciu shader'ów, zastosowanie swobodnego poruszania kamery oraz kontroli szybkości działania pętli głównej.

2. Wstęp:

W celu przeniesienia ruchu myszy na świat trójwymiarowy konieczne jest wyznaczenie położenia kursora myszy w bieżącej klatce i wyznaczenie o ile zmieniła położenie względem poprzedniej. Kolejną ważną rzeczą jest ustalenie stałej ilości klatek na sekundę niezależnie od szybkości sprzętu.

3. Free look

Pierwszą rzeczą, którą powinno się wykonać jest przechwycenie kursora myszy przez okno gry oraz jego ukrycie. W bibliotece SFML można to wykonać poleceniami:

```
window.setMouseCursorGrabbed(true); //przechwycenie kursora myszy w oknie ----------
window.setMouseCursorVisible(false); //ukrycie kursora myszy ---------------------
```

W celu skanowania ruchu myszy należy wykorzystać zdarzenie MouseMoved:

Np.

```
case sf::Event::MouseMoved:
    ustawKamereMysz(uniView, time.asMicroseconds(), window);
break;
```

Do nawigacji będą też wykorzystywane klawisze, dlatego w celu sprawdzania klawiatury można użyć osobnej funkcji w pętli głównej:

np.

```
ustawKamereKlawisze(uniView,time.asMicroseconds());
```

W celu kontroli aktualnego nachylenia kamery potrzebne będą dwa kąty

```
double yaw =-90; //obrót względem osi Y
double pitch=0; //obrót względem osi X
```

a. wyznaczyć zmianę pozycji myszy względem ostatniej klatki,

b. uaktualnić wartości Yaw i Pitch dla kamery,

c. Nałożyć ograniczenia co do ruchu kamery,

d. Wyznaczyć nowy wektor kierunku.

Wyznaczenie lokalnej pozycji kursora myszy:

```
sf::Vector2i localPosition = sf::Mouse::getPosition(_window);
```

Wyznaczenie zmiany położenia i zapamiętanie ostatniej pozycji:

```
double xoffset = localPosition.x - lastX;
double yoffset = localPosition.y - lastY;
lastX = localPosition.x;
lastY = localPosition.y;
```

Aktualizacja kątów ustawienia kamery:

```
xoffset *= sensitivity;
yoffset *= sensitivity;
yaw += xoffset;
pitch -= yoffset;
```

Nałożenie ograniczeń na maksymalne wartości kątów i wyznaczenie nowych wartości wektora pochylenia kamery(tzw. kąty Eulera):

```
if (pitch > 89.0f)
pitch = 89.0f;

if (pitch < -89.0f)
pitch = -89.0f;
glm::vec3 front;
front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
front.y = sin(glm::radians(pitch));
front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
cameraFront = glm::normalize(front);
```

Tworzenie aktualnej macierzy widoku:

```
glm::mat4 view;
view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
glUniformMatrix4fv(_uniView, 1, GL_FALSE, glm::value_ptr(view));
```

Obsługa klawiszy również się zmieni, klawisze strzałek lewo i prawo nie będą już służyły do obrotu kamery ale do ruchu prostopadłego do wektora widoku tzw. strefę. Aby to uzyskać należy określić wektor prostopadły do wektora widoku i pionu kamery. Można wykorzystać iloczyn wektorowy:

```
cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
```

4. Szybkość wykonywania pętli głównej

Aby zapewnić stałą szybkość wykonywania pętli programu konieczne jest obliczenie współczynnika, mnożonego przez wektory ruchu obiektów (również kamery).

Potrzebny jest czas wykonywania pętli głównej. Biblioteka SFML udostępnia odpowiednie funkcje.

Trzeba dodać plik nagłówkowy:

```
#include <SFML/System/Time.hpp>
```

Następnie przed pętlą główną utworzyć obiekty Clock i Time:

```
sf::Clock clock;
sf::Time time;
```

W pętli głównej programu pobieramy czas wykonywania pętli:

```
time = clock.getElapsedTime();
clock.restart();
```

Uzyskany czas łącznie z przyjętym współczynnikiem wyznacza szybkość poruszania kamery:

```
float cameraSpeed = 0.000002f*_time;
```

5. Ćwiczenie:

Należy przygotować projekt pozwalający na swobodne poruszanie kamery po scenie z użyciem klawiatury i myszy, uniezależnić szybkość działania programu od szybkości komputera (można to sprawdzić poprzez ograniczenie ilości klatek na sekundę funkcją: `window.setFramerateLimit(20);`). Wyświetlić ilość klatek na sekundę w pasku tytułowym okna.