Sprawozdanie

Grafika komputerowa

Laboratorium 3

Adrian Jurek

Rafał Król

L3

16.04.2021

Spis treści

[Część 1 – Projektowanie łazika marsjańskiego 3](#_Toc71715662)

[1. Klasa Wheel 3](#_Toc71715663)

[2. Klasa WheelLinks 5](#_Toc71715664)

[3. Klasa Body 8](#_Toc71715665)

[4. Klasa Rover 15](#_Toc71715666)

[5. Efekt końcowy 15](#_Toc71715667)

[Część 2 – Projektowanie otoczenia łazika. 18](#_Toc71715668)

[1. Tworzenie obiektów 18](#_Toc71715669)

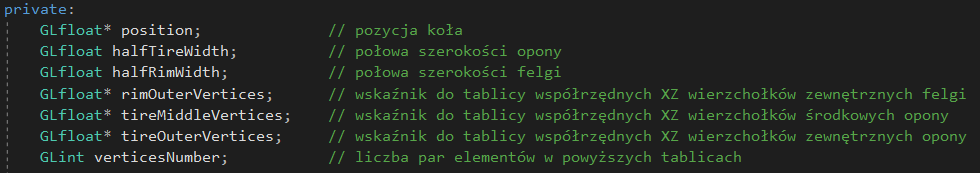
[2. Ładowanie i rysowanie stworzonych obiektów 19](#_Toc71715670)

[3. Efekt końcowy 25](#_Toc71715671)

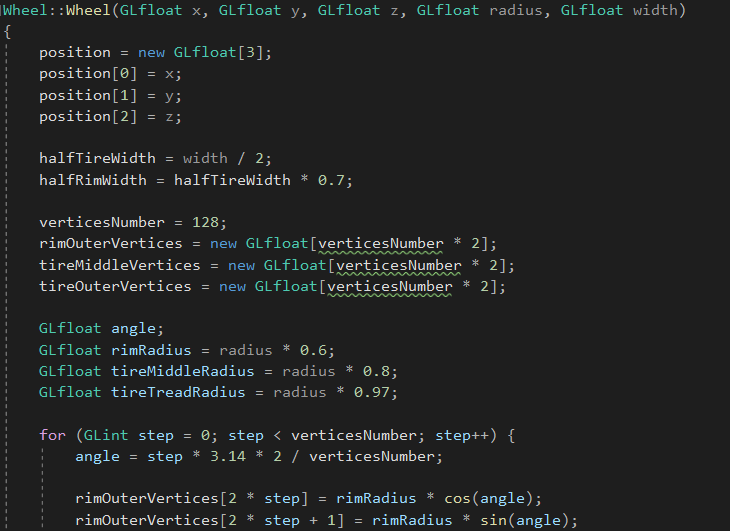
# Część 1 – Projektowanie łazika marsjańskiego

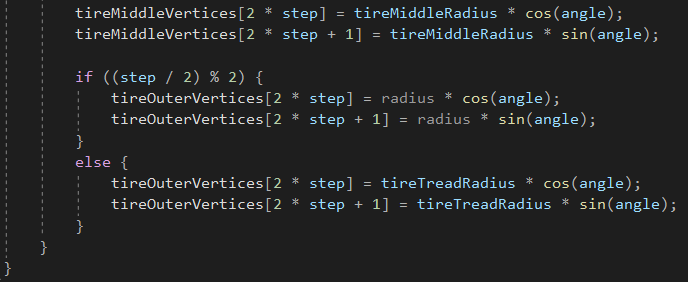
## Klasa Wheel

Reprezentuje pojedyncze koło łazika. Klasa posiada następujące pola:

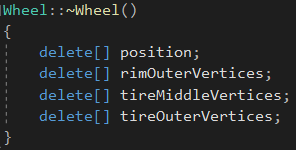


Konstruktor klasy Wheel: przyjmuje pozycję koła (pierwsze trzy parametry), jego promień oraz szerokość; ustawia odpowiednie pola klasy oraz alokuje i wypełnia tablice wierzchołków.

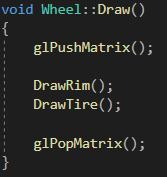


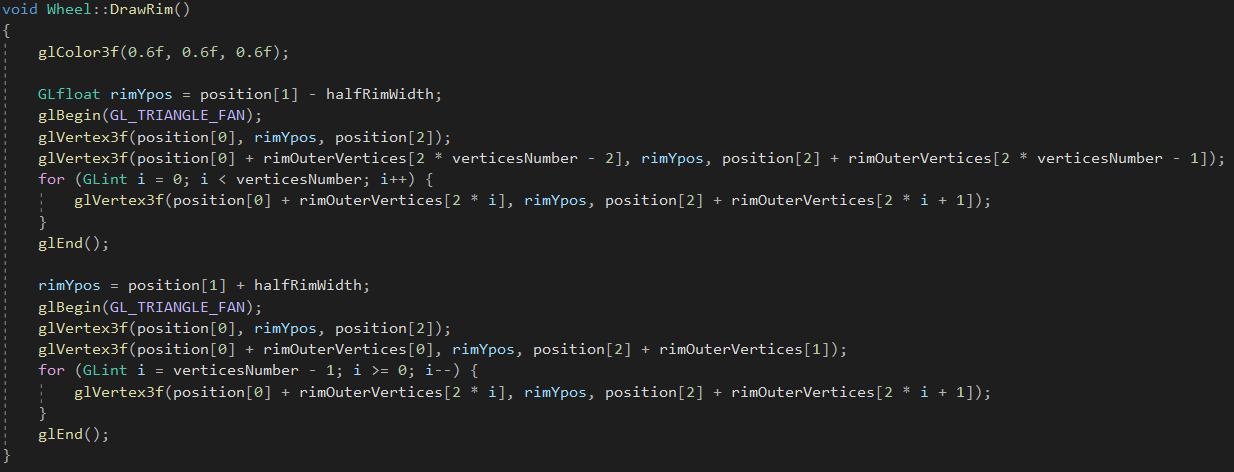


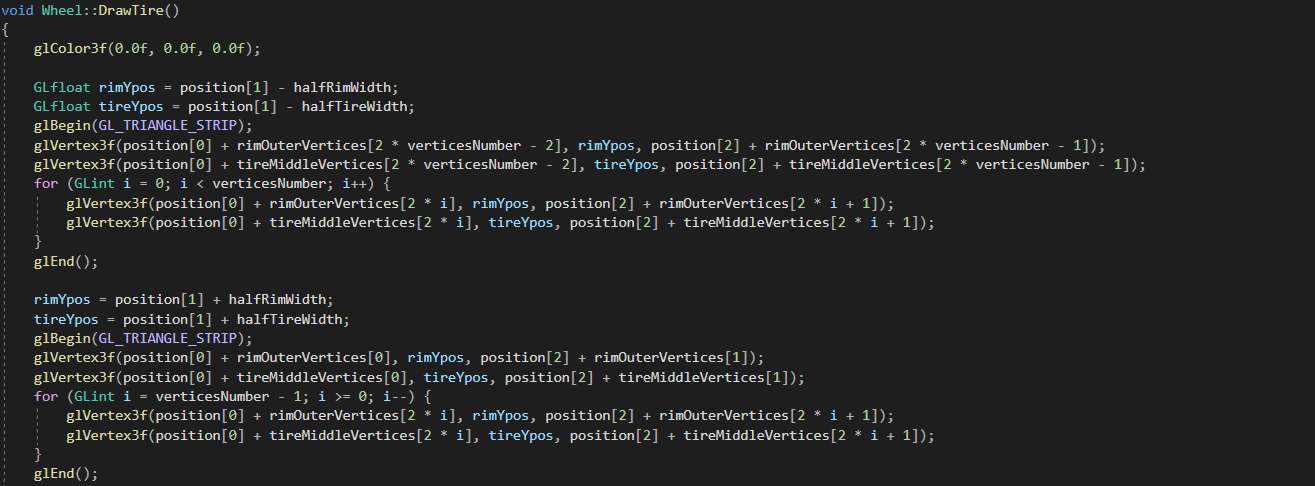
Destruktor klasy Wheel: zwalnia pamięć zaalokowaną przez tablice wierzchołków.

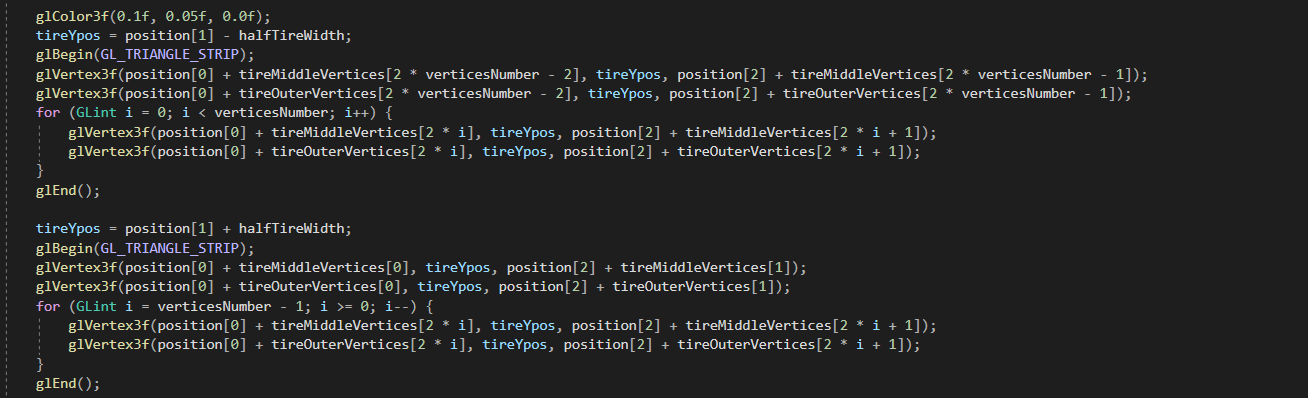


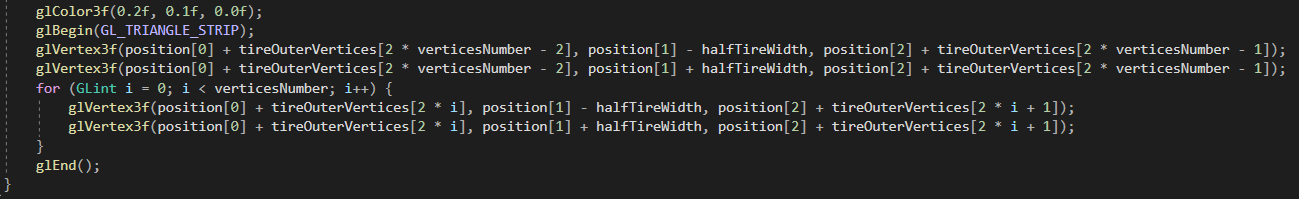
Publiczna metoda Draw: rysuje koło na ekranie. Metoda korzysta z dwóch metod prywatnych: DrawRim oraz DrawTire, które odpowiednio rysują felgę oraz oponę koła.



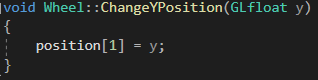






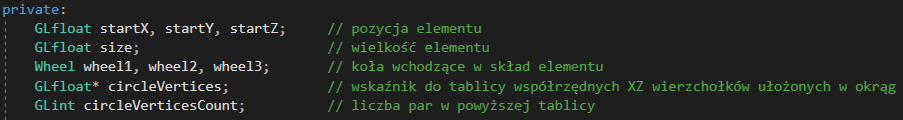


Publiczna metoda ChangeYPosition: zmienia pozycję koła w osi Y.

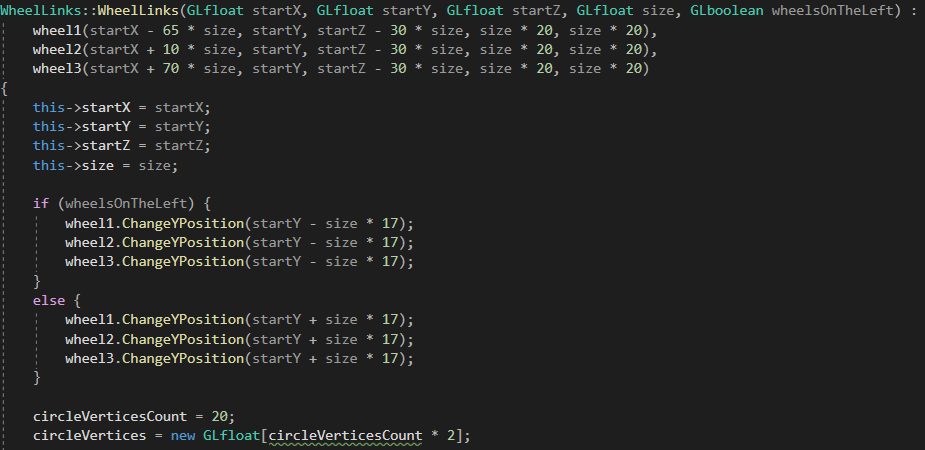


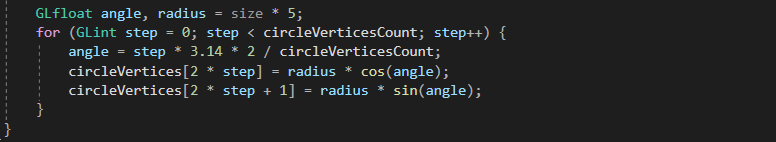
## Klasa WheelLinks

Klasa reprezentuje trzy boczne koła łazika, połączone ze sobą odpowiednimi ramionami. Klasa posiada następujące pola:

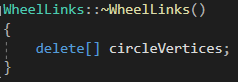


Konstruktor klasy WheelLinks: przyjmuje pozycję elementu (pierwsze trzy parametry), jego wielkość oraz zmienną określającą, czy koła mają być po lewej czy po prawej stronie; wywołuje konstruktory dla wszystkich kół składowych, ustawia odpowiednie pola klasy oraz alokuje i wypełnia tablicę wierzchołków ułożonych w okrąg.

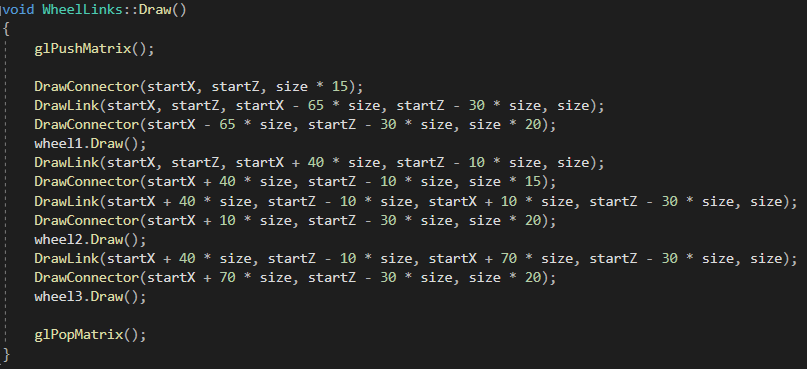


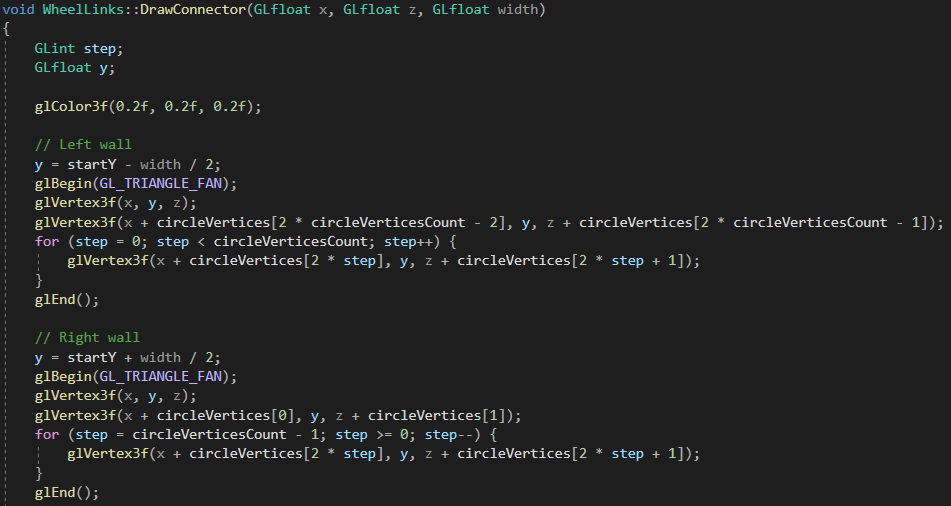


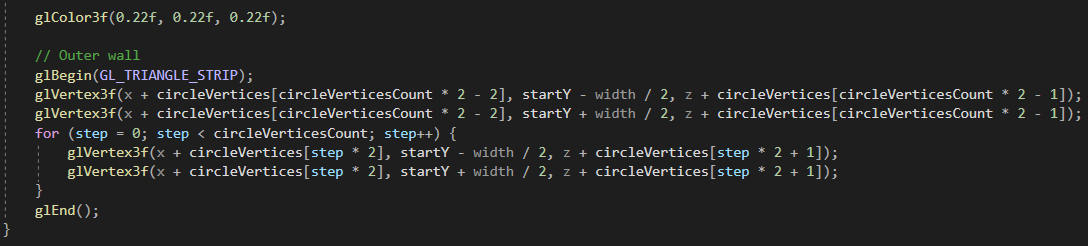
Destruktor klasy WheelLinks: zwalnia pamięć zaalokowaną przez tablicę wierzchołków.

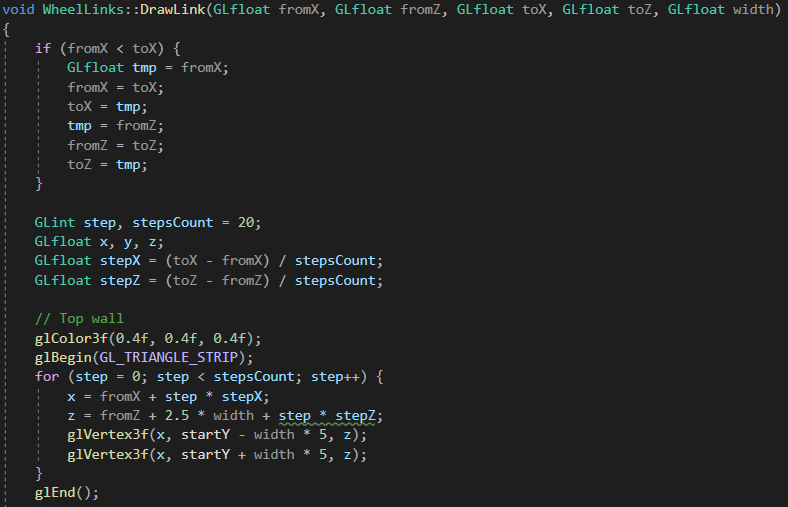


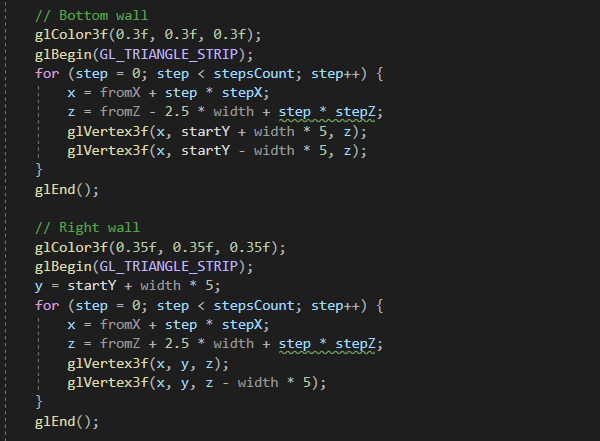
Publiczna metoda Draw: rysuje cały element na ekranie. Metoda używa dwóch prywatnych metod: DrawConnector i DrawLink. Pierwsza z nich rysuje cylindryczny łącznik, łączący ramiona i koła, druga z nich rysuje pojedyncze ramię.

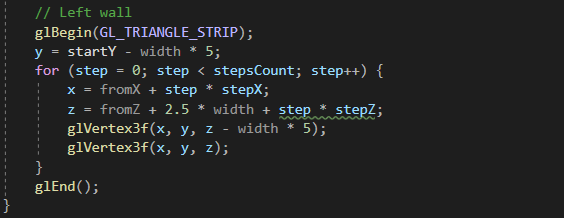








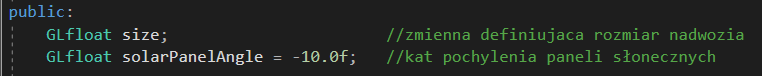


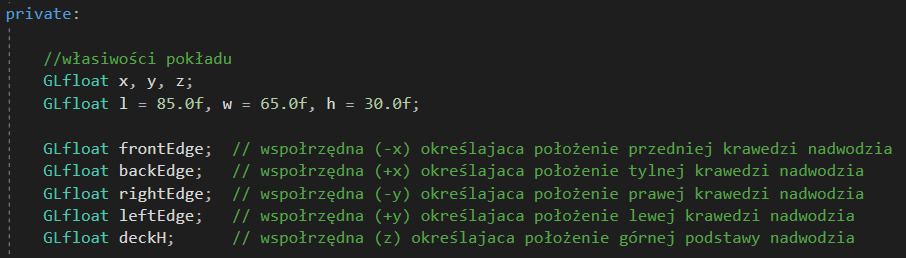


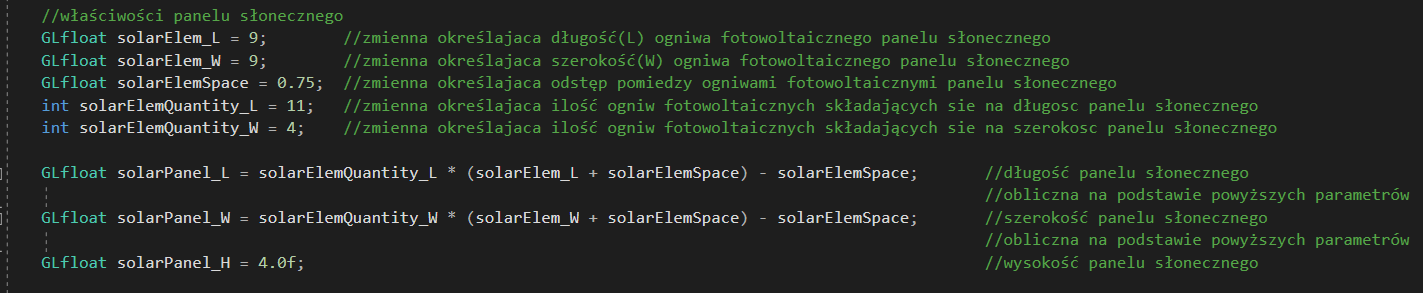
## Klasa Body

Klasa reprezentuje nadwozie łazika, które składa się z głównego elementu – pokładu, oraz dodatkowych elementów umieszczonych na nim. Położenie wszystkich elementów definiowane jest na podstawie współrzędnych i rozmiarów pokładu łazika.

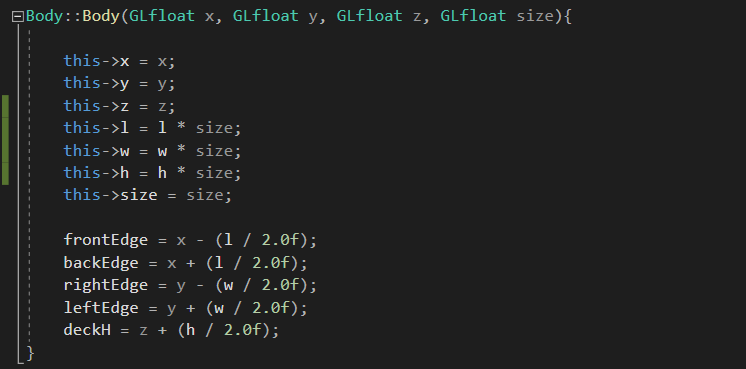
Klasa posiada następujące pola składowe:



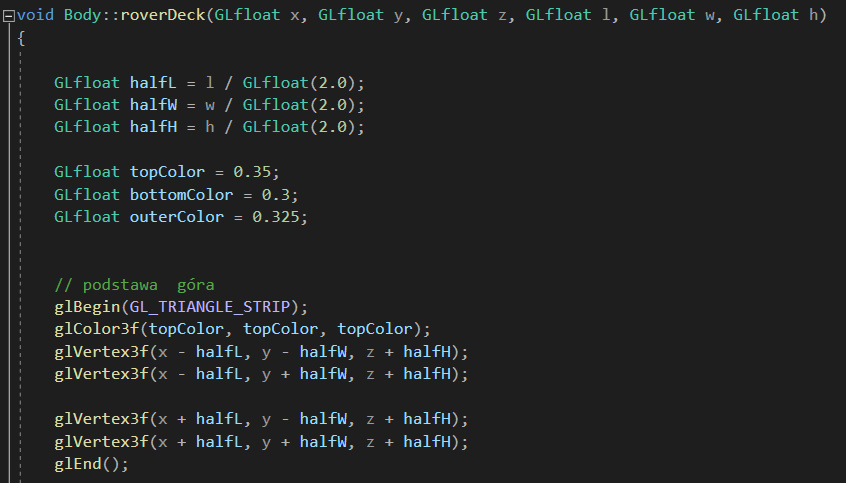




Konstruktor klasy Body: przyjmuje w parametrach (x, y, z) położenie pokładu łazika oraz rozmiar całego obiektu (size). Przypisuje wartości argumentów do pól składowych klasy oraz wyznacza współrzędne poszczególnych krawędzi pokładu łazika. Dodatkowo przemnaża rozmiary pokładu przez parametr size, po to aby nadwozie łazika mogło być renderowane w odpowiedniej skali.

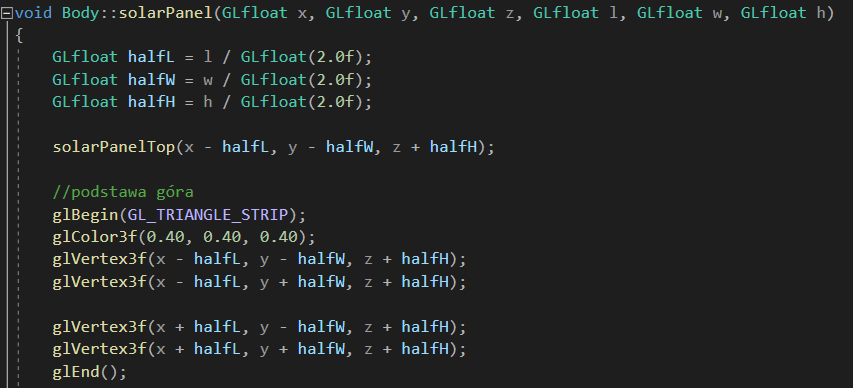


Prywatna metoda: roverDeck() tworzy prostopadłościan reprezentujący pokład łazika. W parametrach przyjmuje współrzędne(x, y, z) określające położenie pokładu oraz jego rozmiar(l, w, h)

Wszystkie ściany tworzone są według schematu jak dla podstawy górnej, z uwzględnieniem odpowiednich współrzędnych. 

Prywatna metoda: solarPanel() tworzy prostopadłościan reprezentujący panel słoneczny.

Wszystkie ściany tworzone są według schematu jak dla podstawy górnej, z uwzględnieniem odpowiednich współrzędnych. Dodatkowo aby górna podstawa odzwierciedlała ogniwa fotowoltaiczne poprzedzona jest wywołaniem metody solarPanelTop().



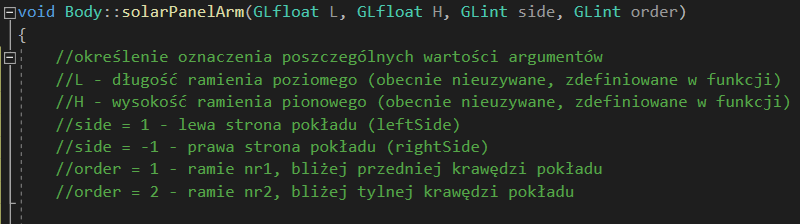
Prywatna metoda: solarPanelTop() tworzy prostokątną ścianę/teksturę reprezentującą ogniwa fotowoltaiczne panelu słonecznego.

W parametrach (x, y, z) przekazane są współrzędne pierwszego ogniwa fotowoltaicznego. Współrzędne kolejnych ogniw wyznaczane są w dwóch zagnieżdżonych pętlach for, które odpowiednio określają liczbę elementów składających się na szerokość panelu oraz jego długość. W danej iteracji współrzędna kolejnego elementu wyznaczana jest w następujący sposób:

pozycja\_kolejnego\_elem = pozycja\_aktualnego\_elem + rozmiar\_elem + odstęp\_miedzy\_elementami



Prywatna metoda: solarPanelArm() tworzy ramię łazika, zakończone wywołaniem metody soalrPanel() aby pozycja panelu słonecznego zgadzała się z pozycją ramienia.



Metoda ta jest bardzo złożona i zaprezentowanie całego jej kodu zajęło by kilka stron sprawozdania, dlatego postaram się jedynie wyjaśnić logikę stojąca za tworzeniem pojedynczego ramienia.

Każde ramię składa się z następujących elementów:

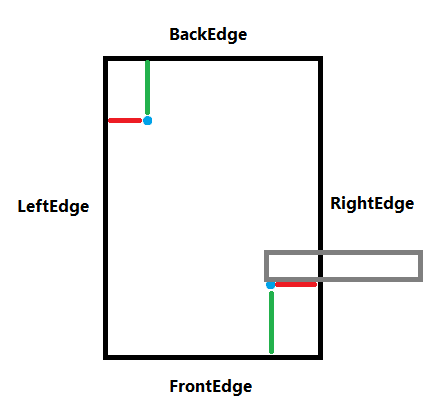
- podstawa ramienia (prostopadłościan)

- ramię poziome (prostopadłościan)

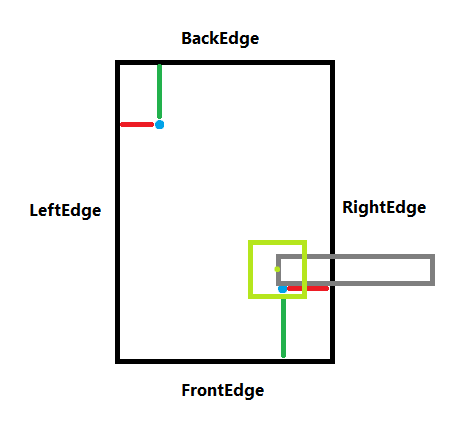
- ramię pionowe (prostopadłościan)

- uchwyt (połowa walca + prostopadłościan)

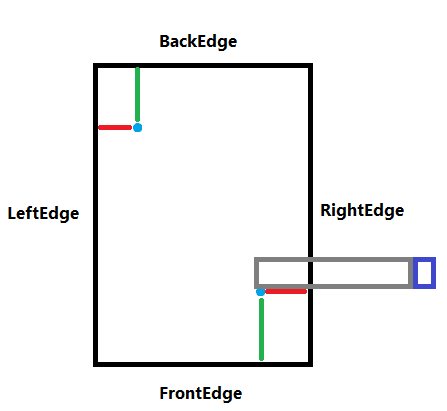
Wszystkie elementy pozycjonowane są względem pokładu łazika, oraz samych siebie, tak aby przy poruszaniu się łazika przesuwały się razem z nim.



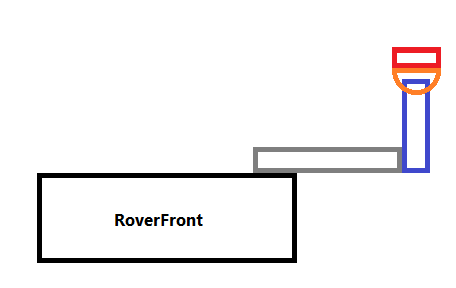
Ramię poziomie: pozycjonowane jest względem krawędzi łazika, poprzez określenie wewnątrz funkcji zmiennych odpowiadających za odstęp od krawędzi lewej lub prawej oraz za odstęp od krawędzi przedniej lub tylnej

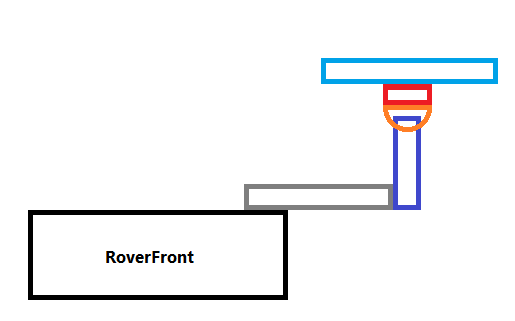


Podstawa ramienia: pozycjonowana jest względem początku ramienia



Ramię pionowe: pozycjonowane jest względem końca ramienia poziomego, zgodnie z rysunkiem



Uchwyt ramienia ustawiony jest względem końca ramienia pionowego obniżony o połowę promienia części półokrągłej uchwytu.

Metoda solarPanel() na samym końcu umiejscawia panel słoneczny względem końca chwytu.

Panel słonecznych oraz uchwyt znajdujące się na samej górze ramienia pionowego są położone w przesuniętym układzie współrzędnych po to aby można było zmieniać kich kąt nachylenia. Do tego celu została wykorzystana kombinacja funkcji

glPushMatrix();

glTranslatef(x,y,z),

glRotatef(angle, 1, 0, 0);

. armHandleCylinderPart … //cześć kodu tworząca cylindryczna część uchwytu

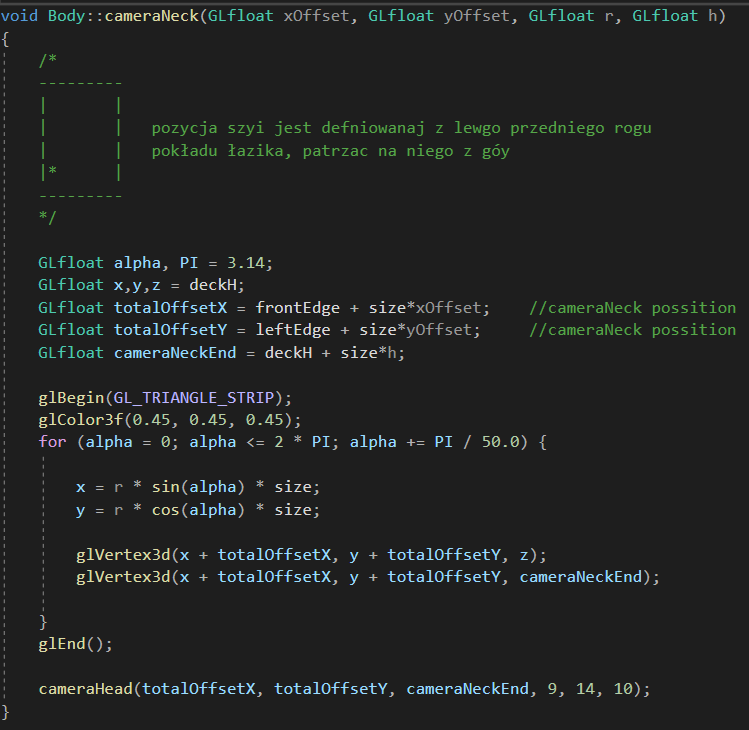
. armHandleCuboidPart … //część kodu tworząca prostopadłościenną część uchwytu

. soalrPanel(x,y,z); //funkcja tworząca panel słoneczny

glPopMatrix();

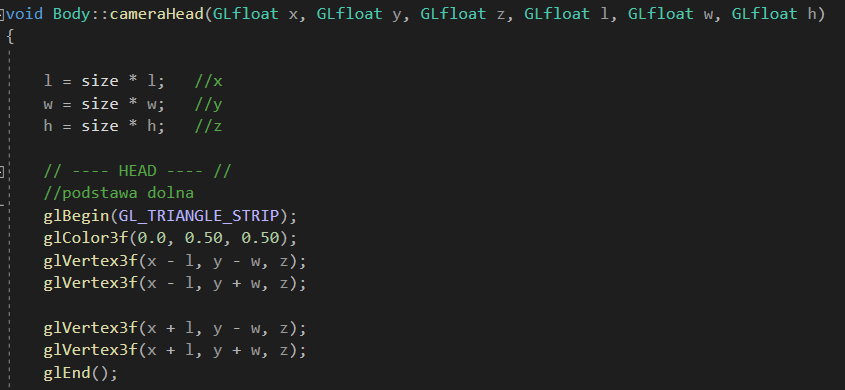
Prywatna metoda: cameraNeck() tworzy szyję kamery łazika.

Metoda ta rysuję walec na pokładzie łazika o zadanych pramateriach(r, h) względem lewego przedniego rogu łazika o zadanym przesunięciu(xOffest, Offset). Na końcu wywołuję metodę cameraHead(), która tworzy kamerę.

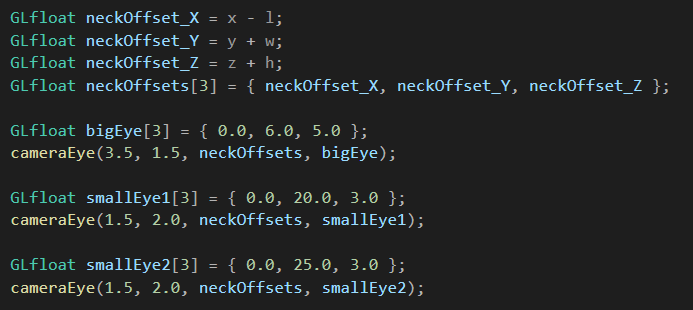


Prywatna metoda: cameraHead() tworzy prostopadłościan reprezentujący kamerę łazika. Przyjmuje parametry (x, y, z), które są środkiem zakończenia szyi łazika oraz (l,w,h) czyli wymiary kamery.

Wszystkie ściany tworzone są według schematu jak dla podstawy dolnej, z uwzględnieniem odpowiednich współrzędnych.

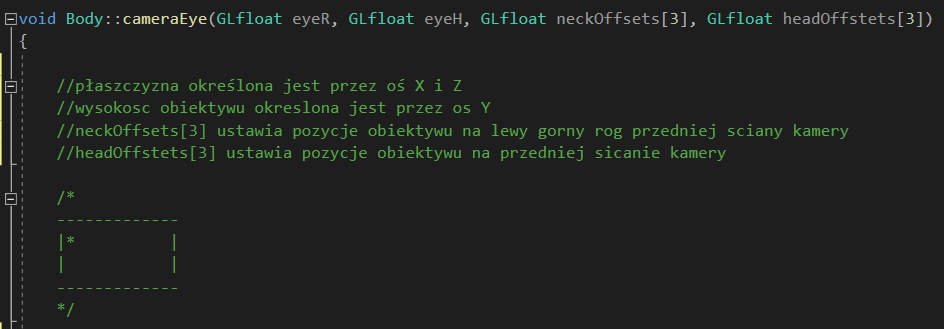


Po stworzeniu głowy metoda cameraHead() wywołuje metodę cameraEye(), która tworzy obiektywy/czujniki na przedzie kamery.

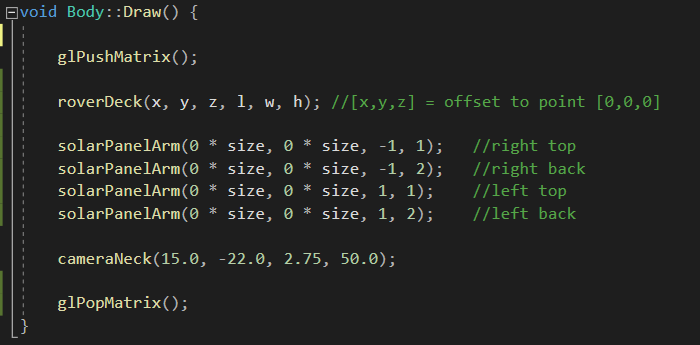


Prywatna metoda: cameraEye() tworzy walec reprezentujący obiektyw kamery łazika.

Znaczenie argumentów opisane jest na zdjęciu. Poza tym metoda ta zawiera 3 typowe fragmenty kodu odpowiedzialne za rysowanie walca.

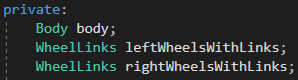


Publiczna metoda: Draw() wywołująca poszczególne metody prywatne odpowiedzialne za rysowanie poszczególnych elementów nadwozia łazika.

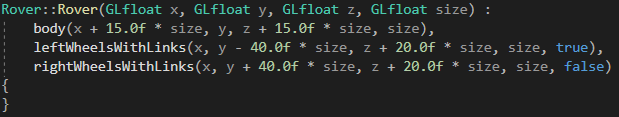


## Klasa Rover

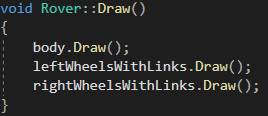
Klasa reprezentuje cały łazik. Klasa posiada następujące pola składowe:



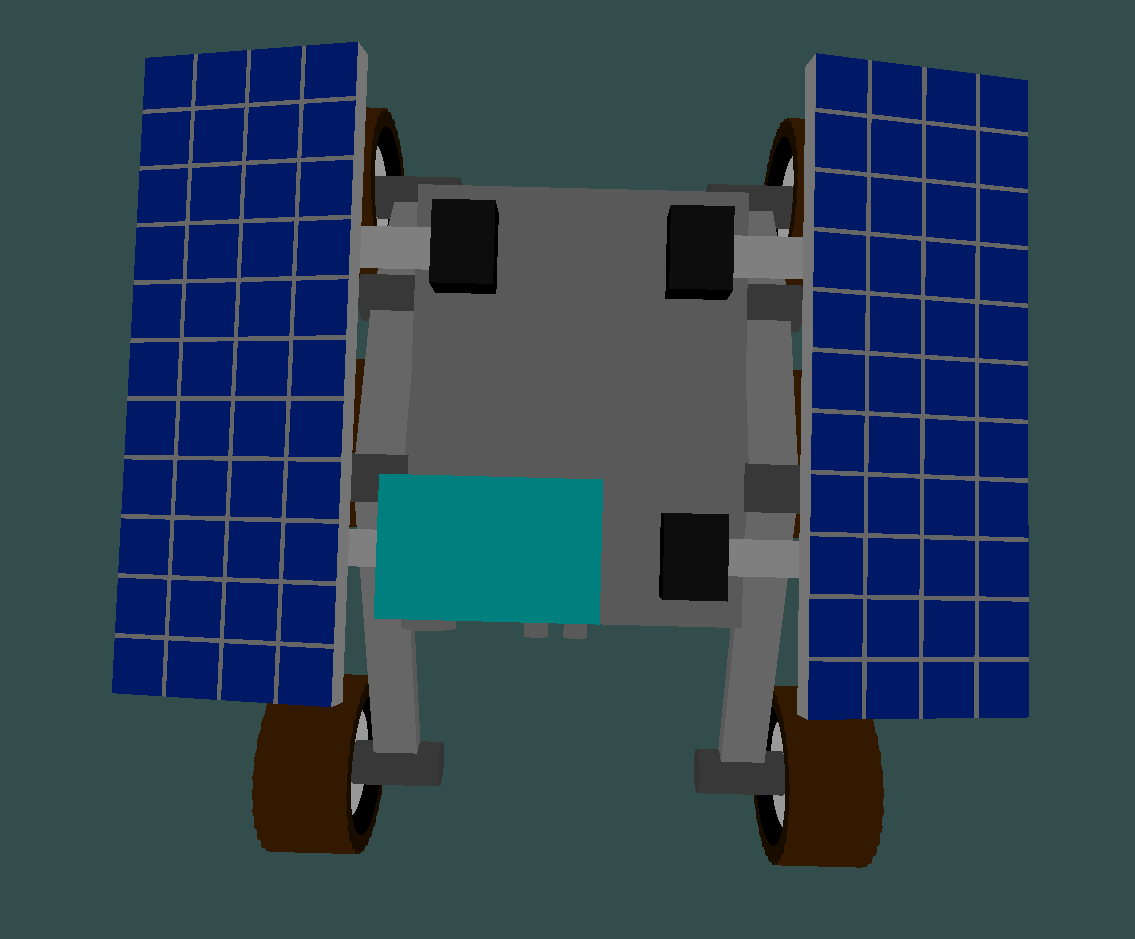
Konstruktor klasy Rover: wywołuje konstruktory pól składowych.



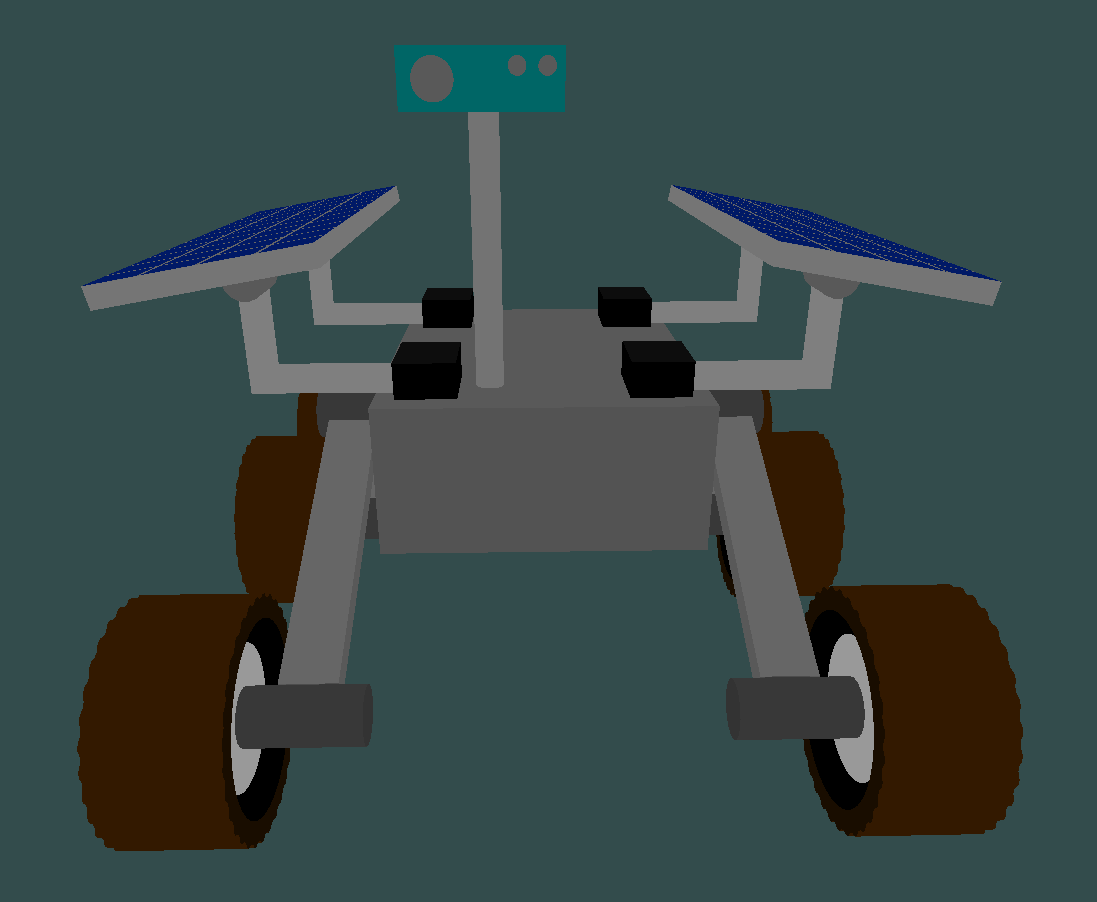
Publiczna metoda Draw: wywołuje metody Draw pól składowych.



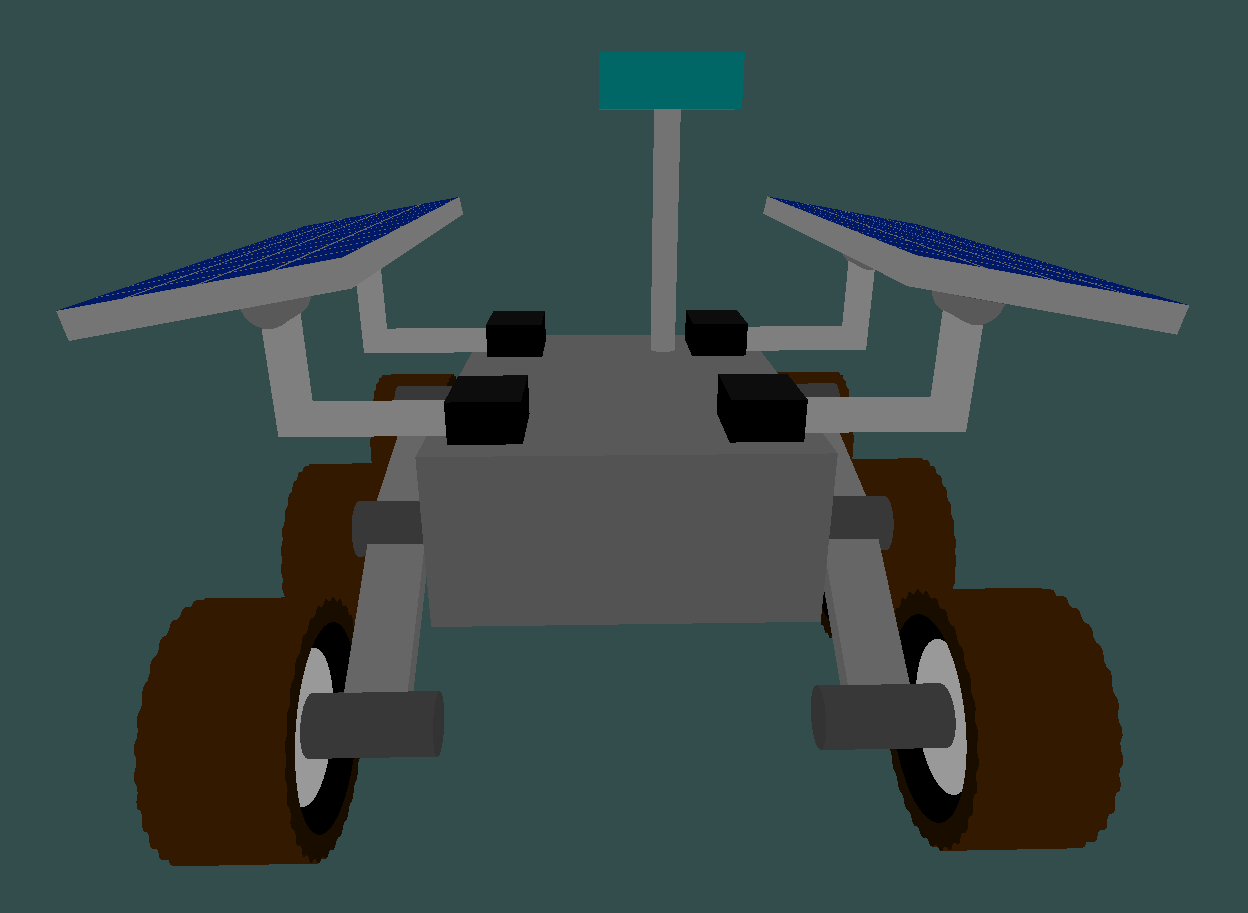
## Efekt końcowy



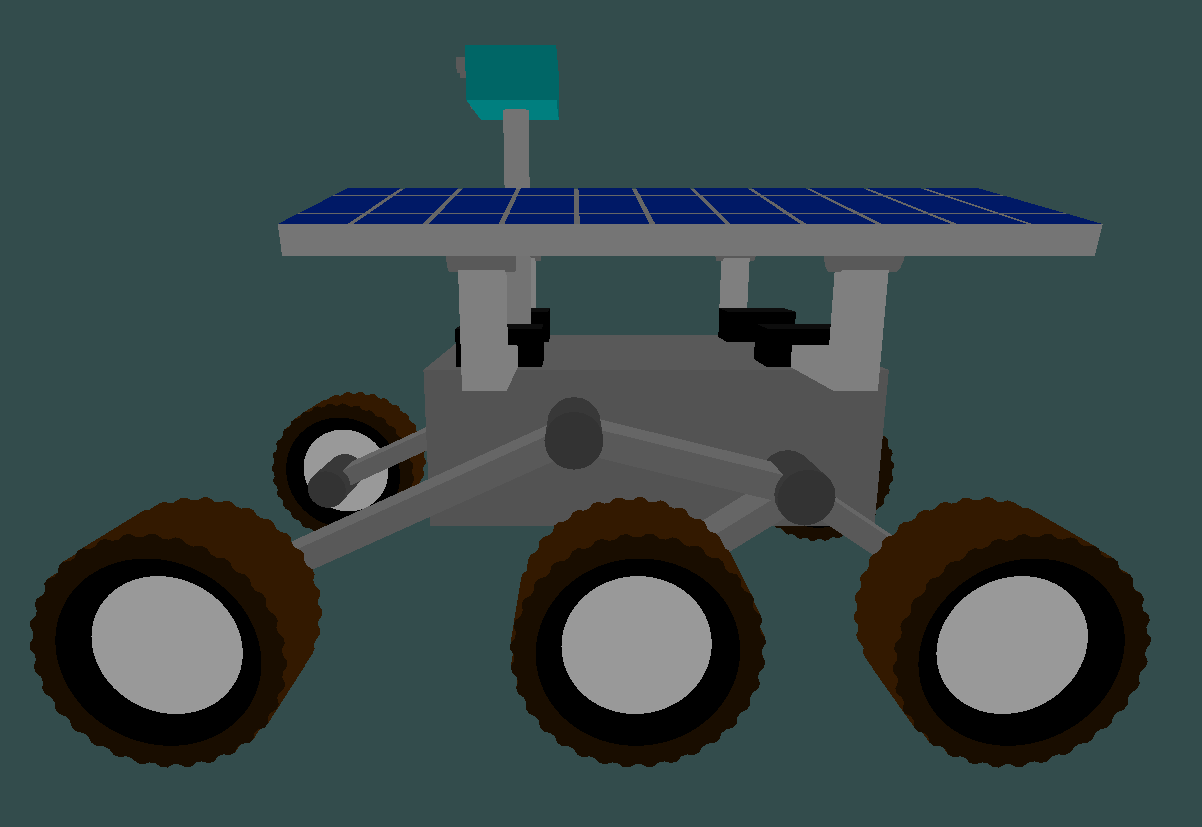
*Widok z góry*

**

*Widok z przodu*

**

*Widok z tyłu*

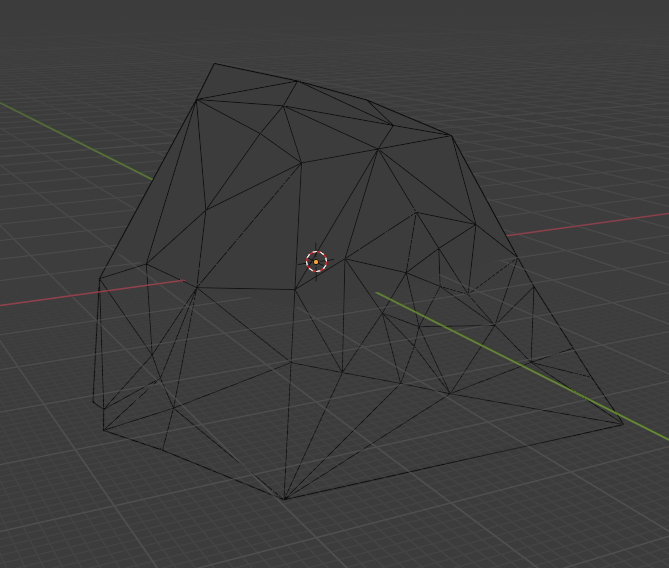
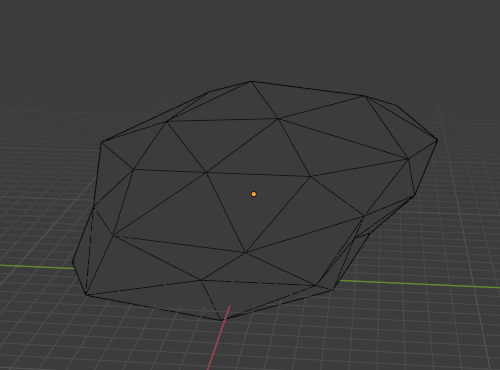
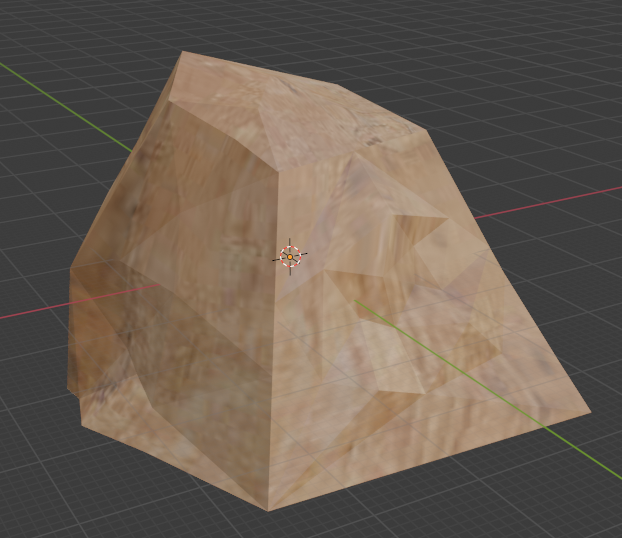
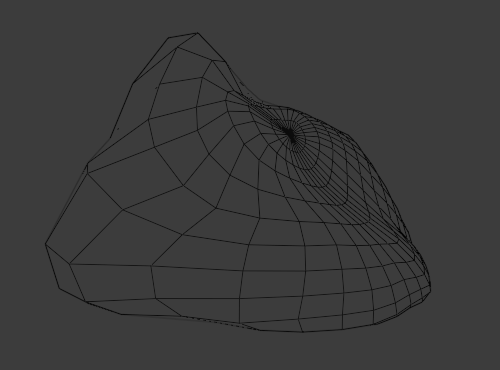
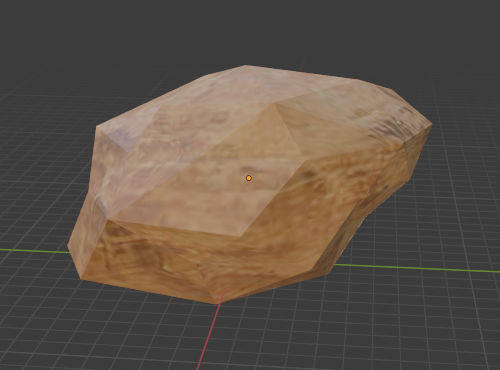
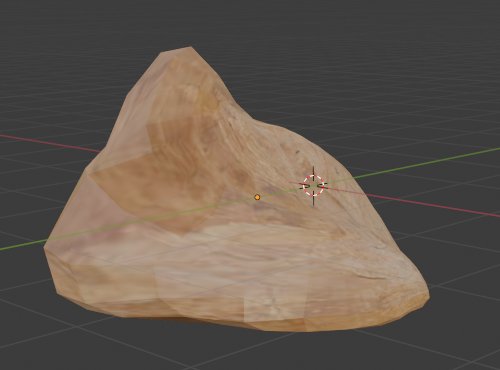
**

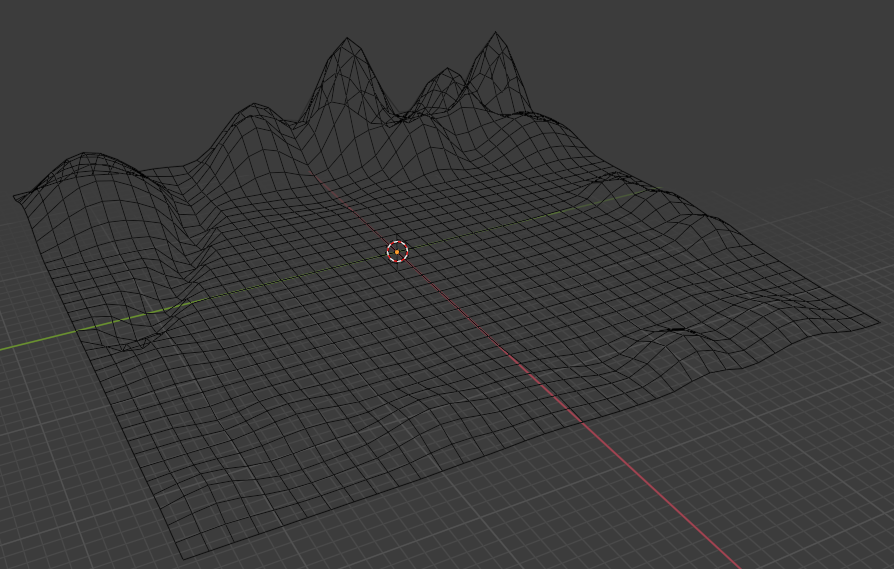
*Widok z boku*

# Część 2 – Projektowanie otoczenia łazika.

## Tworzenie obiektów

Otoczenie łazika, na którego składają się minimum dwie przeszkody (skały) oraz teren po którym będzie się poruszał zostały zrealizowane w programie graficznym Blender.

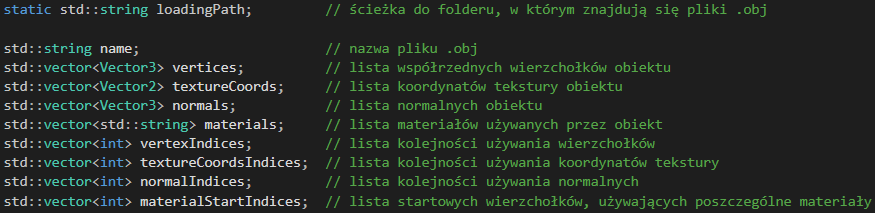
   



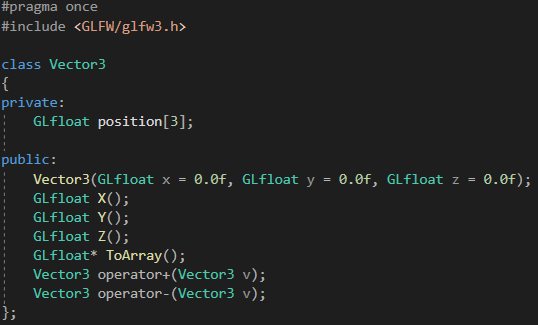
W celu stworzenia powyższych obiektów zostały wykorzystane podstawowe narzędzia programu Blender takie jak: moduł do modelowania brył oraz moduł do rzeźbienia w bryle. Prezentowane modele 3D powstały na skutek modyfikacji gotowych siatek obiektów typu mesh sphere oraz mesh grid. Ponieważ generowane automatycznie siatki nie są wystarczająco szczegółowe, dlatego konieczne było zwiększenie liczby wierzchołków, aby móc uzyskać efektowne kształty brył. W tym celu wykorzystany został dodatek Subdivision Surface, który dzieli powierzchnię o zadaną liczbę razy. W wyniku tego siatka staję się gęstsza, co przekłada się na większą dokładność obiektów oraz bardziej gładką powierzchnię. Niestety później w fazie testowania okazało się, że niektóre ze stworzonych obiektów posiadają zbyt dużą liczbę wierzchołków i musiały zostać zmodyfikowane ze względu na niską wydajność programu/gry. Tekstury wykorzystane w projekcie zostały pobrane z Internetu jako pliki z rozszerzeniem .jpg. Podczas tworzenia powyższych obiektów wyzwaniem było uzyskanie zadowalających aspektów wizualnych. Myślę jednak, że efekt końcowy prac prezentowanych na zrzutach ekranu można uznać za dobry.

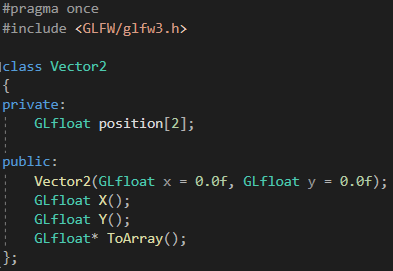
## Ładowanie i rysowanie stworzonych obiektów

Na początek, została utworzona klasa Object. Każdy ładowany plik .obj będzie instancją tej właśnie klasy. Prywatne pola składowe klasy:

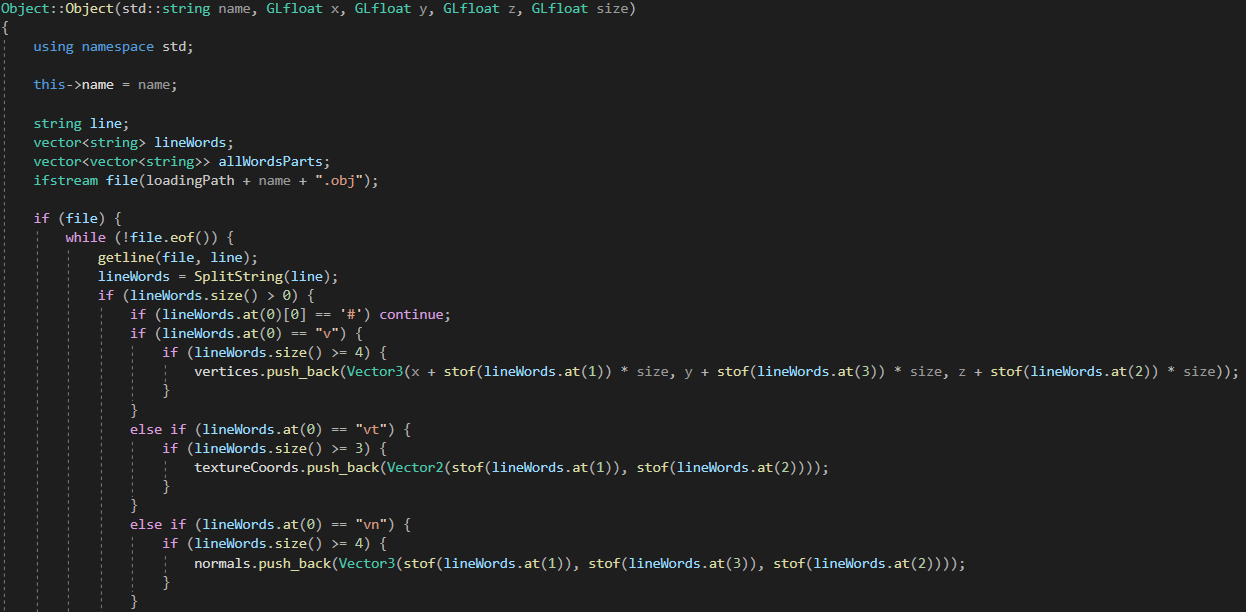


Na powyższym rysunku można zauważyć, że są używane klasy Vector3 oraz Vector2. Są to klasy pomocnicze stworzone na potrzeby realizacji tego zadania i reprezentują wektory odpowiednio trój- i dwuwymiarowe. Ich pliki nagłówkowe wyglądają następująco:



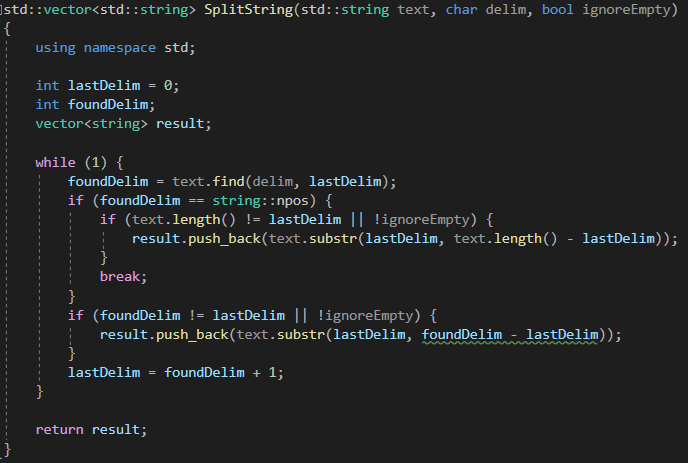


Klasa Object zawiera konstruktor oraz dwie publiczne metody. Konstruktor, jako parametry przyjmuje nazwę pliku .obj, startową pozycję obiektu oraz jego wielkość. W konstruktorze następuje inicjalizacja pól składowych na podstawie odczytanego pliku .obj. Pierwsze słowo z każdej linii pliku wskazuje, jakie informacje zawiera dana linia, więc w zależności od tego słowa ustawiane są wartości odpowiednich pól składowych klasy.

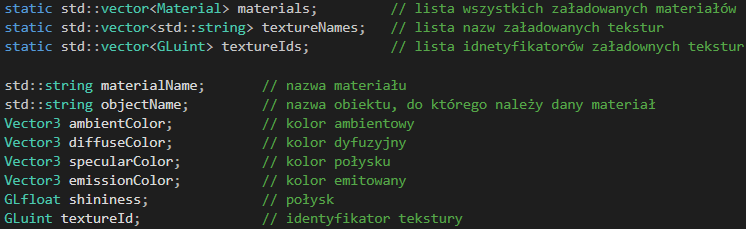




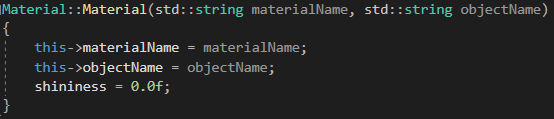
W konstruktorze używana jest pomocnicza funkcja SplitString. Przyjmuje ona łańcuch znaków oraz znak rozdzielający słowa, a zwraca vector zawierający te słowa. Dodatkowo, dzięki ostatniemu parametrowi można określić, czy puste słowa (o długości 0) powinny być zignorowane w zwracanym wyniku. Domyślną wartością znaku rozdzielającego to spacja, a ignorowanie pustych słów jest domyślnie włączone. Kod funkcji:



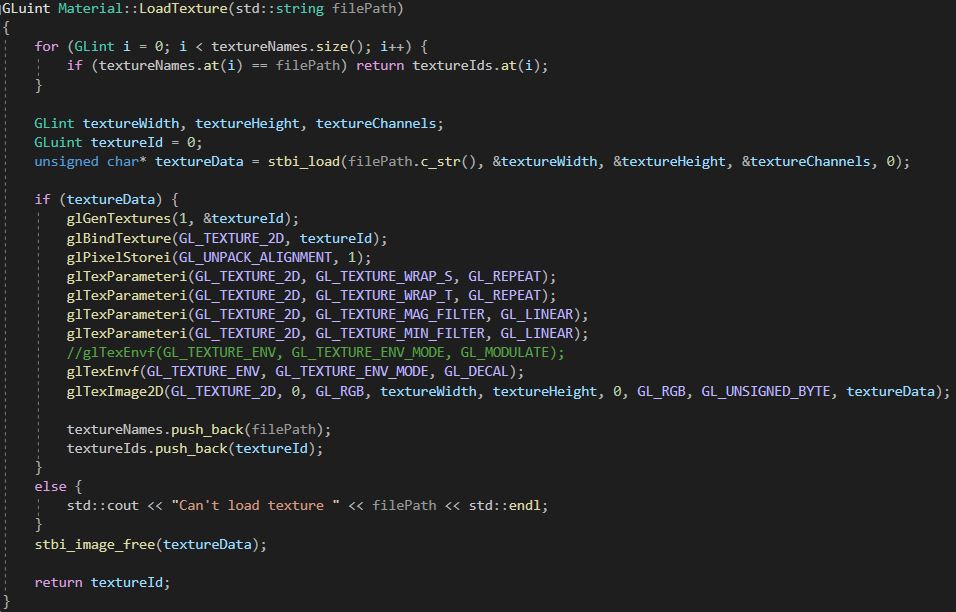
W konstruktorze klasy Object została również użyta metoda z klasy Material. Jest to druga z głównych stworzonych klas i reprezentuje materiał używany przez obiekt. Prywatne pola składowe klasy Material:



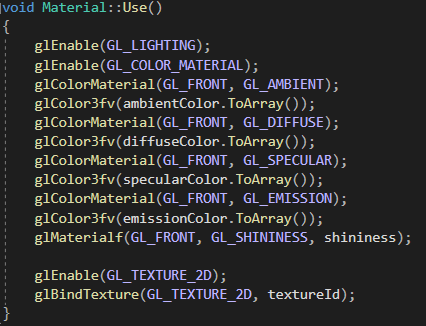
W konstruktorze klasy Material przypisywane są wartości do podstawowych pól składowych:



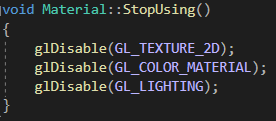
Metoda LoadTexture odpowiada za ładowanie tekstury z podanego pliku oraz przesłanie jej do pamięci karty graficznej. Zanim jednak tekstura zostanie załadowana, sprawdzane jest, czy nie została ona już wcześniej załadowana, aby uniknąć zwielokrotnienia jednej tekstury w pamięci. Metoda zwraca identyfikator załadowanej tekstury. Odczyt pliku graficznego został zrealizowany z pomocą funkcji stbi\_load z biblioteki stb\_image: <https://github.com/nothings/stb/blob/master/stb_image.h>



Metoda Use włącza wszystkie efekty (wraz z teksturą) danego materiału:

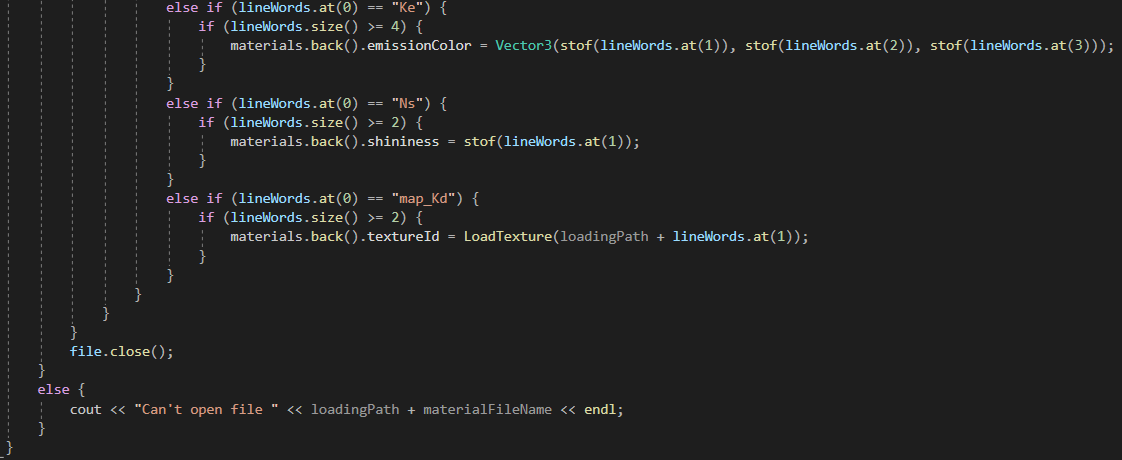


Statyczna metoda StopUsing wyłącza efekty materiału:

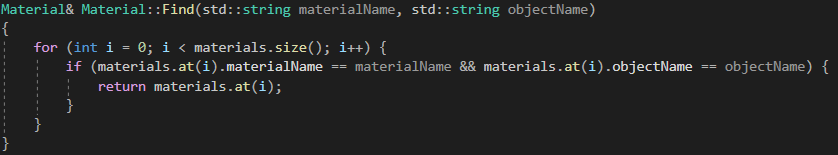


Statyczna metoda Add odczytuje plik .mtl i tworzy obiekty klasy Material z odpowiednimi właściwościami. Podobnie jak w przypadku plików .obj, w pliku .mtl pierwsze słowo danej linii określa, jakie informacje zawiera dana linia. Każdy dodany materiał może być jednoznacznie zidentyfikowany za pomocą kombinacji swojej nazwy oraz nazwy obiektu, który używa tego materiału.

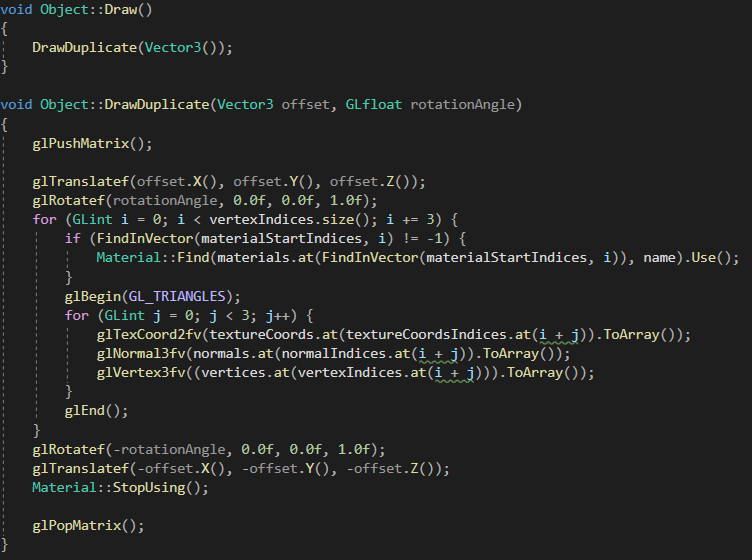




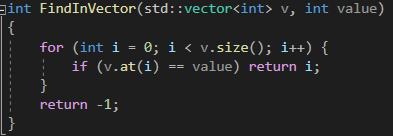
Statyczna metoda Find zwraca referencję do materiału, znalezionego na podstawie jego nazwy oraz nazwy obiektu.



Klasa Object posiada jeszcze dwie publiczne metody: Draw i DrawDuplicate. Obie odpowiadają za wyrysowanie obiektu na scenie. Pierwsza z nich rysuje obiekt w miejscu zdefiniowanym podczas jego inicjalizacji. Druga zaś pozwala określić przesunięcie obiektu względem jego oryginalnej pozycji oraz kąt obrotu.



Wewnątrz metody DrawDuplicate używana jest pomocnicza funkcja FindInVector. Dla podanych jako parametry vector’a liczb całkowitych oraz liczby całkowitej ona indeks, na którym znajduje się podany element w vectorze, lub -1, gdy takowy nie występuje.



## Efekt końcowy