

Wydział Matematyki i Nauk Informatycznych  
Politechniki Warszawskiej



## Snake 3D - projekt animacji

Mateusz Rymuszka

v.1.1

18 lutego 2017

# Spis treści

<b>1</b>	<b>Specyfikacja</b>	<b>2</b>
1.1	Opis biznesowy . . . . .	2
1.2	Wymagania funkcjonalne . . . . .	2
1.2.1	Przypadki użycia . . . . .	2
1.2.2	User stories . . . . .	4
1.3	Wymagania нефункционалне . . . . .	4
1.4	Harmonogram projektu . . . . .	5
1.5	Architektura rozwiązania . . . . .	6
<b>2</b>	<b>Dokumentacja końcowa (powykonawcza)</b>	<b>8</b>
2.1	Wymagania systemowe . . . . .	8
2.2	Wykorzystane biblioteki . . . . .	8
2.3	Instalacja oraz uruchomienie programu . . . . .	8
2.3.1	Wersja z kompilacją po stronie klienta . . . . .	8
2.3.2	Wersja bez kompilacji . . . . .	9
2.4	Instrukcja użycia . . . . .	10
2.4.1	Zmiana kierunku poruszania się węża . . . . .	10
2.4.2	Zmiana konfiguracji renderowania . . . . .	11
2.4.3	Dodatkowe informacje dla developerów . . . . .	12
2.5	Raport odstępstw od specyfikacji wymagań . . . . .	13
2.5.1	Zmiana biblioteki do obliczeń matematycznych . . . . .	13
2.5.2	Opuszczenie użycia loadera plików *.obj . . . . .	13
2.5.3	Odstępstwo od wydajności wyświetlania grafiki . . . . .	13

# 1 Specyfikacja

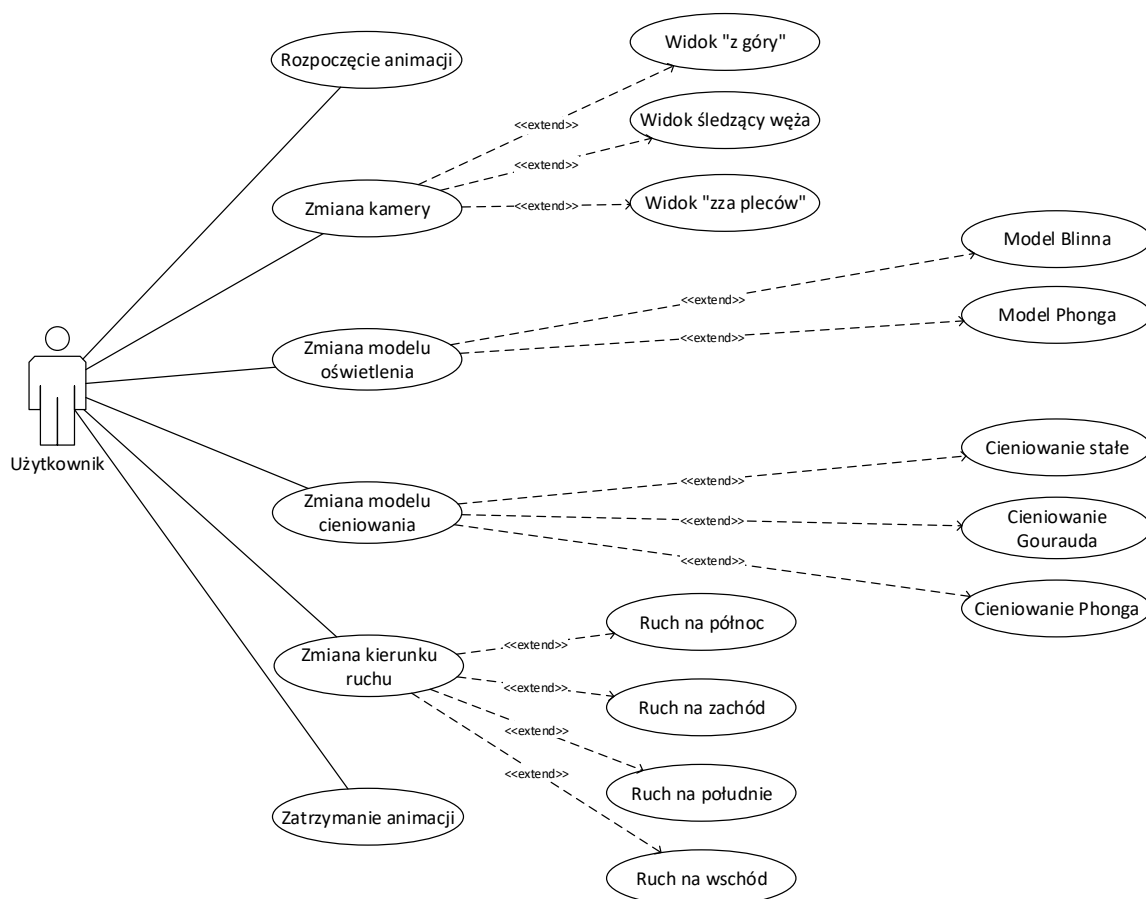
## 1.1 Opis biznesowy

Niniejszy projekt zakłada stworzenie programu opartego luźno na strukturze popularnej gry mobilnej Snake, umożliwiającego poruszanie wężem w przestrzeni trójwymiarowej. Ogólnym celem projektu jest symulacja potoków renderowania grafiki trójwymiarowej, poprzez odpowiednią prezentację obiektów w grafice 2D. W szczególności, planowane jest przedstawienie różnych modeli kamery, oświetlenia oraz cieniowania używanych w grafice trójwymiarowej oraz modelowanie obiektów i animacji trójwymiarowych na różne sposoby.

## 1.2 Wymagania funkcjonalne

### 1.2.1 Przypadki użycia

Poniższy diagram UML ilustruje przypadki użycia dla użytkownika aplikacji - jedynego aktora naszej aplikacji. Przypadki te zostały dalej przedstawione w sposób bardziej szczegółowy w postaci tabeli.



Rysunek 1: Diagram przypadków użycia dla użytkownika aplikacji

<b>Nazwa</b>	<b>Opis</b>	<b>Odpowiedź systemu</b>
Rozpoczęcie animacji	Rozpoczęcie animacji - ruch porusza się po scenie w zadanym kierunku. Jest to domyślnie wywołana akcja.	Ruch węża po scenie oraz informacja w interfejsie o działaniu animacji
Zmiana kamery	Użytkownik wybiera jeden z dostępnych modeli kamery za pomocą odpowiedniej opcji na liście, która powoduje natychmiastową zmianę widoku.	Zmiana widoku na jeden z dostępnych: <ul style="list-style-type: none"> <li>• widok sceny z góry</li> <li>• widok śledzący węża</li> <li>• widok 'zza pleców'</li> </ul>
Zmiana modelu oświetlenia	Użytkownik wybiera jeden z dostępnych modeli oświetlenia za pomocą odpowiedniej opcji na liście, która powoduje natychmiastową zmianę oświetlenia.	Zmiana oświetlenia na jeden z dostępnych modeli: <ul style="list-style-type: none"> <li>• oświetlenie Blinna</li> <li>• oświetlenie Phong</li> </ul>
Zmiana modelu cieniowania	Użytkownik wybiera jeden z dostępnych modeli cieniowania za pomocą odpowiedniej opcji na liście, która powoduje natychmiastową zmianę sposobu cieniowania.	Zmiana cieniowania na jeden z dostępnych: <ul style="list-style-type: none"> <li>• cieniowanie stałe</li> <li>• cieniowanie Gourauda</li> <li>• cieniowanie Phong</li> </ul>
Zmiana kierunku poruszania	Za pomocą strzałek możemy zmienić kierunek poruszania się węża na północ, południe, wschód i zachód.	Skręcenie się węża oraz podążanie węża w nowym kierunku
Zatrzymanie aplikacji	Wąż przestaje się poruszać po scenie - scena jest jednak dalej widoczna. Można w tym czasie zmienić sposób wyświetlania sceny.	Natychmiastowa zatrzymanie się węża w obecnej pozycji, do momentu wznowienia ruchu.

Rysunek 2: Szczegółowy opis przypadków użycia

### 1.2.2 User stories

Poniżej przedstawię podstawowe historie użytkownika, które przedstawiają główne założenia programu.

1. Brak interakcji ze strony użytkownika  
Po uruchomieniu aplikacji, przedstawiony jest domyślny widok - widok sceny z góry, oświetlona przy użyciu modelu oświetlenia Phong'a oraz z cieniowaniem stałym. Wąż porusza się początkowo w kierunku północnym.
2. Użytkownik zmienia kierunek poruszania się  
Głowa węża zmienia swoje położenie o 90°, po poruszeniu się w miejscu 'głowy' tworzy się kolanko. Wąż przesuwa się w nowym kierunku, jednocześnie zachowując położenie środkowych części węża (głowa i ogon przesuwały się do przodu).
3. Wąż natrafia na ścianę sceny  
Wąż zachowuje się tak, jakby użytkownik tuż przed ścianą zmienił kierunek ruchu zgodnie z ruchem wskazówek zegara (np. dla kierunku północnego jest to wschód).
4. Użytkownik zmienia opcje renderingu  
Obraz prezentowany przez program jest zmieniany (na nowy, uwzględniający nową kamerę / oświetlenie / cieniowanie) tak szybko, jak zmiany zostaną wdrożone.

### 1.3 Wymagania niefunkcjonalne

Lp.	Obszar	Opis wymagania
1.	<b>Użyteczność</b>	Program powinien być kompilowalny do postaci binarnej zdolnej do uruchomienia na systemie Windows lub systemie z rodziny Linux.
2.		Program powinien posiadać przejrzysty interfejs użytkownika, obsługujący rozdzielczości od 1366x786px wzwyż.
3.	<b>Niezawodność</b>	Program nie może kończyć się w sposób nieprzewidziany przez twórcę, w szczególności musi być odporny na występowanie sytuacji wyjątkowych.
4.		Program musi być odporny na potencjalne niepożądane akcje wprowadzane przez użytkowników.
5.		Program musi wyświetlać animację zgodnie z określoną konfiguracją, w obszarze do tego zdefiniowanym.
6.	<b>Wydajność</b>	Aplikacja powinna być gotowa do wyświetlania animacji z prędkością minimum 6 fps (ramek na sekundę), przy czym ogólna średnia częstotliwość powinna wynosić minimum 16 fps.
7.		Dostępność i dynamika graficznego interfejsu użytkownika aplikacji nie mogą być ograniczane lub zależne od częstości wyświetlanych klatek w animacji.
8.	<b>Utrzymanie</b>	Do programu dołączona będzie prosta instrukcja obsługi.
9.		Program winien być napisany w sposób umożliwiający rozszerzanie go o dodatkowe funkcjonalności).
10.		Interfejs aplikacji winien jednoznacznie wskazywać na używaną konfigurację sceny.

## 1.4 Harmonogram projektu

Harmonogram prac nad projektem został ujęty w poniższej tabelce. Przy planowaniu etapów oraz identyfikacji kamieni milowych uwzględnione zostały takie czynniki jak: stopień trudności poszczególnych zadań, dostępność czasowa wykonawcy projektu oraz potrzeba weryfikacji efektów w sposób zwinny z oceniającym projekt.

Cykl prac został podzielony na trzy etapy, każdy z nich zakłada realizację określonego celu:

ETAP 1 - Stworzenie działającego programu, renderującego grafikę 3D na jeden sposób (kamera z góry, oświetlenie Blinna oraz cieniowanie stałe) oraz podstawową logikę programu

ETAP 2 - Dodanie wszystkich wymaganych modeli kamery, oświetlenia i cieniowania

ETAP 3 - Uzupełnienie programu o animację oraz bardziej szczegółową logikę (skręty, czas reakcji) oraz ujednolicenie interfejsu użytkownika.

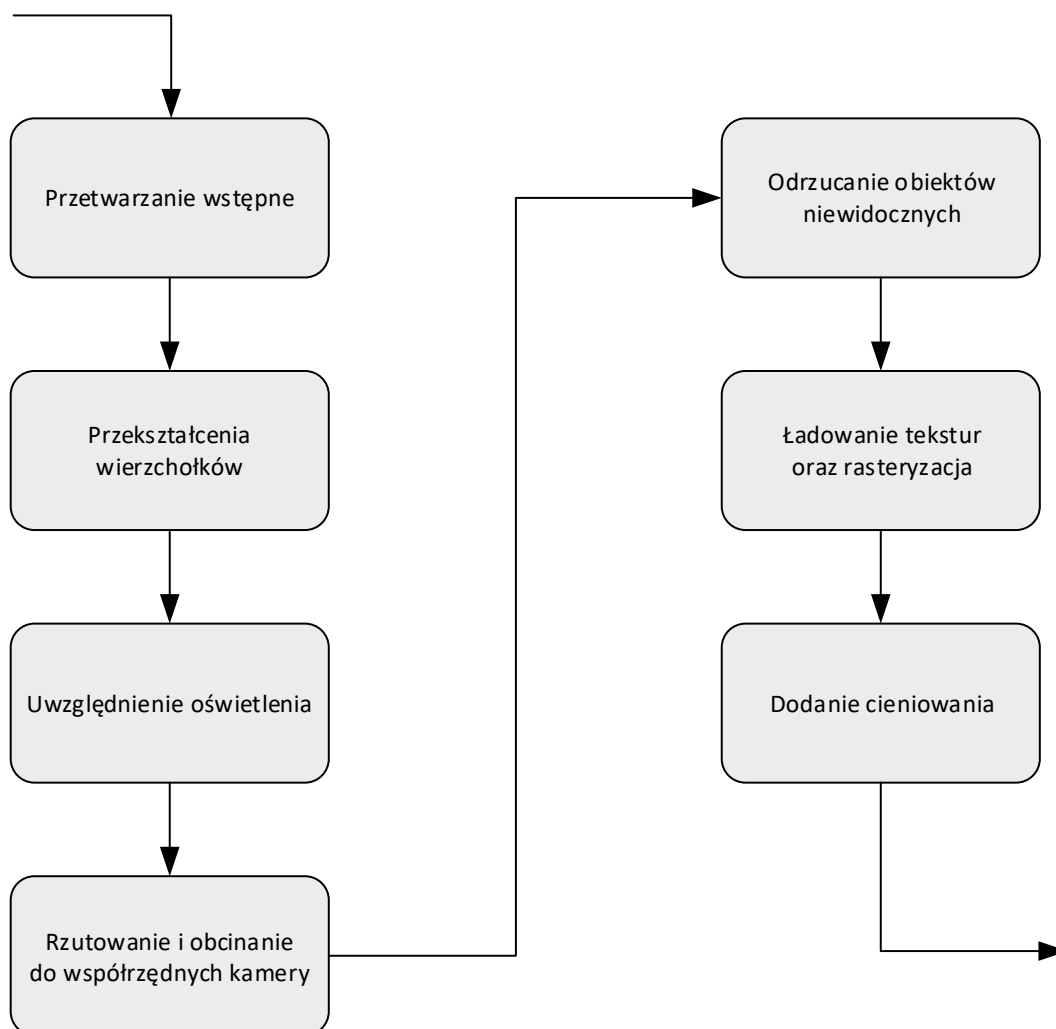
Po wykonaniu projektu zostanie dostarczona całkowita dokumentacja projektu.

Początek	Koniec	Czas	Zadanie
15.12.2016	18.12.2016	4 dni	Identyfikacja założeń projektu i przygotowanie specyfikacji przedwykonawczej
ETAP 1			
19.12.2016	25.12.2016	7 dni	Prace początkowe - zapoznanie się z bibliotekami oraz początek modelowania sceny
26.12.2016	4.01.2017	10 dni	Właściwe modelowanie sceny - stworzenie potoku renderingu (z jednym widokiem) oraz podstawowej logiki animacji
5.01.2017	-	-	Prezentacja projektu prowadzącemu przedmiot w jego początkowej fazie
ETAP 2			
5.01.2017	11.01.2017	7 dni	Implementacja kamer oraz różnych metod oświetlenia i cieniowania wraz z wdrożeniem zaleceń prowadzącego
11.01.2017	-	-	Prezentacja wtórna - weryfikacja poprawności wdrożonych rozwiązań i otrzymanie kolejnych uwag dot. wykonania
ETAP 3			
12.01.2017	18.01.2017	7 dni	Rozszerzenie programu o ruch oraz dodatkowe funkcjonalności związane z rozwojem logiki
18.01.2017	-	-	Prezentacja zasadnicza - weryfikacja zgodności wykonanego projektu z wymaganiami oraz uwagi końcowe
19.01.2017	26.01.2017	7 dni	Poprawki oraz modernizacje końcowe
27.01.2016	-	-	Zdanie projektu
27.01.2016	29.01.2016	2 dni	Przygotowanie dokumentacji wykonawczej
29.01.2016	-	-	Zdanie całości dokumentacji

Rysunek 3: Harmonogram prac nad projektem

## 1.5 Architektura rozwiązania

Architektura programu zakłada implementację prostego potoku renderowania grafiki trójwymiarowej. Poniższy diagram przedstawia uproszczony schemat potoku:



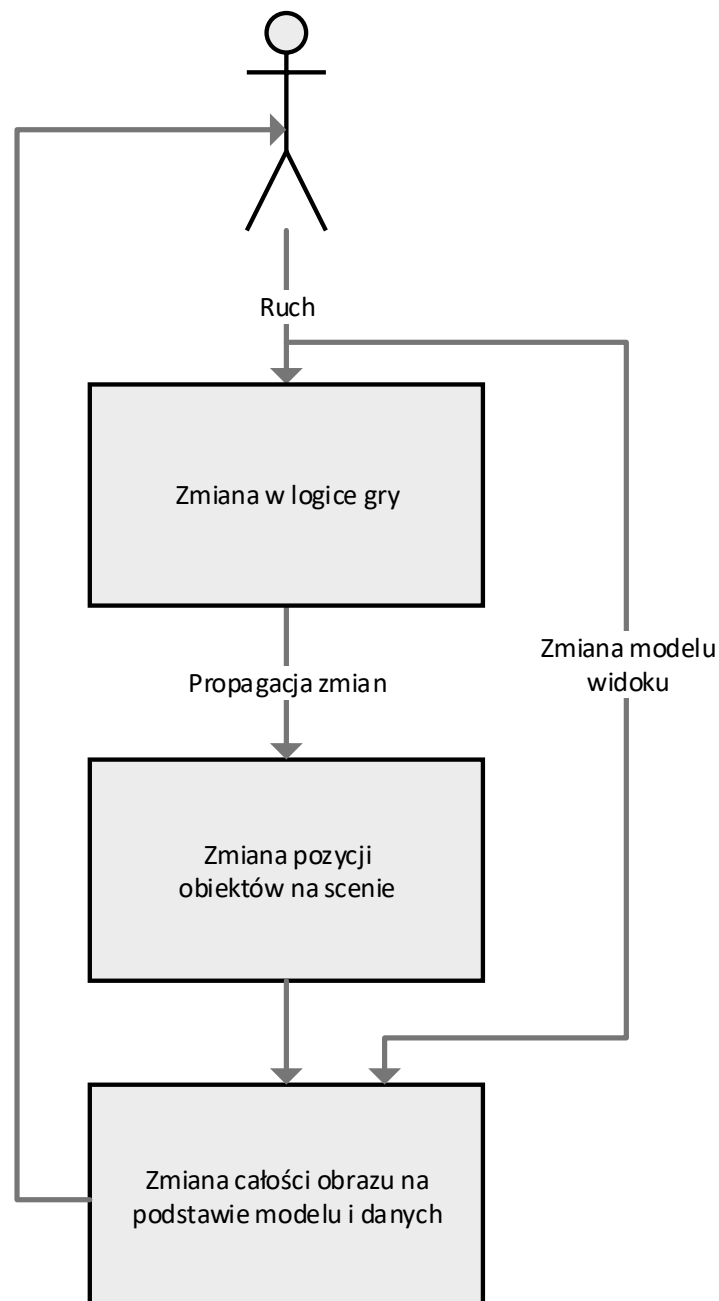
Rysunek 4: Ogólny schemat potoku renderowania w aplikacji

Wstępne plany zakładają stworzenie programu przy użyciu języka C++. Dodatkowo, planowane jest wykorzystanie bibliotek:

- SDL (Simple DirectMedia Layer) - bazowa biblioteka graficzna udostępniająca funkcjonalności związane z prezentacją grafiki 2D na urządzeniu
- Armadillo - biblioteka upraszczająca stosowanie operacji wymagających algebry liniowej w stopniu ponadpodstawowym
- tinyobjloader - prosty w użyciu parser obiektów modeli .obj, używany w przypadku ładowania bardziej wymagających kształtów

W szczególności, wszystkie elementy programu wymagane do transformacji modelu grafiki 3D na ekran zostaną stworzone wraz z projektem, w oparciu o wspomniane wyżej narzędzia.

W związku z tym, od projektu odseparowane zostaną logika animacji (model danych oraz kontroler aplikacji) oraz warstwa prezentacyjna. Ze względu na jednoznaczność odwzorowania położenia węża na mapie (tak, jak w tradycyjnej wersji gry), logika aplikacji będzie uwzględniać jedynie współrzędne w  $\mathbb{R}^2$ , natomiast dalsze obliczenia będą wykonywane przez obiekty potoku graficznego, na podstawie położenia punktu referencyjnego.



Rysunek 5: Przepływ zmian w aplikacji



## 2 Dokumentacja końcowa (powykonawcza)

### 2.1 Wymagania systemowe

Program został przystosowany do uruchamiania na systemie Windows w wersji 7 lub wyższej. Dodatkowo, program został przystosowany do kompilacji w środowisku programistycznym Visual Studio 2015. Do uruchomienia programu niezbędne jest dołączenie dynamicznej biblioteki SDL (skompilowanej do pliku `SDL2.dll`) - winna być ona umieszczona albo w lokalizacji uwzględnionej w zmiennej środowiskowej `PATH` lub w katalogu zawierającym skompilowany program.

Domyślnie, po zbudowaniu projektu dołączona biblioteka DLL powinna zostać skopiowana do katalogu z plikiem wykonywalnym. Jest ona także zawarta w archiwum razem ze skompilowanym plikiem wykonywalnym.

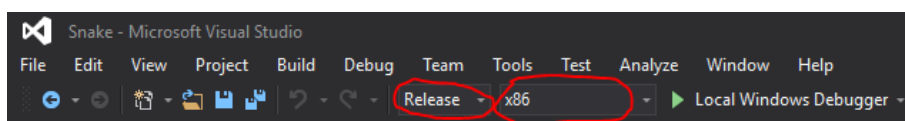
### 2.2 Wykorzystane biblioteki

Lp.	Nazwa i wersja	Komponenty	Opis	Licencja
1.	GLM (OpenGL Mathematics) v.0.9.8.3	lib/include/glm/	Nagłówkowa biblioteka do obliczeń matematycznych przydatnych w programowaniu graficznym	Licencja MIT
2.	SDL v.2.0.5 (Simple DirectMedia Layer)	lib/include/SDL2/ lib/SDL2.lib lib/SDL2main.lib lib/SDL2test.lib SDL2.dll	Wieloplatformowa biblioteka dostarczająca niskopoziomą obsługę graficzną aplikacji	Licencja zlib

### 2.3 Instalacja oraz uruchomienie programu

#### 2.3.1 Wersja z kompilacją po stronie klienta

1. Przed instalacją należy się upewnić, że spełnione są wszystkie wymagania systemowe, tzn.:
  - system operacyjny Windows jest w wersji 7 lub wyższej,
  - zainstalowane jest środowisko programistyczne Visual Studio 2015 wraz ze składnikami Visual C++ Redistributable.
2. Wypakować zawartość dostarczonego archiwum `snake_src.zip` do wybranej przez siebie lokalizacji (za pomocą programu do obsługi archiwów, np. 7-Zip).
3. Otworzyć solucję `Snake.sln` w programie Visual Studio 2015 z katalogu głównego.



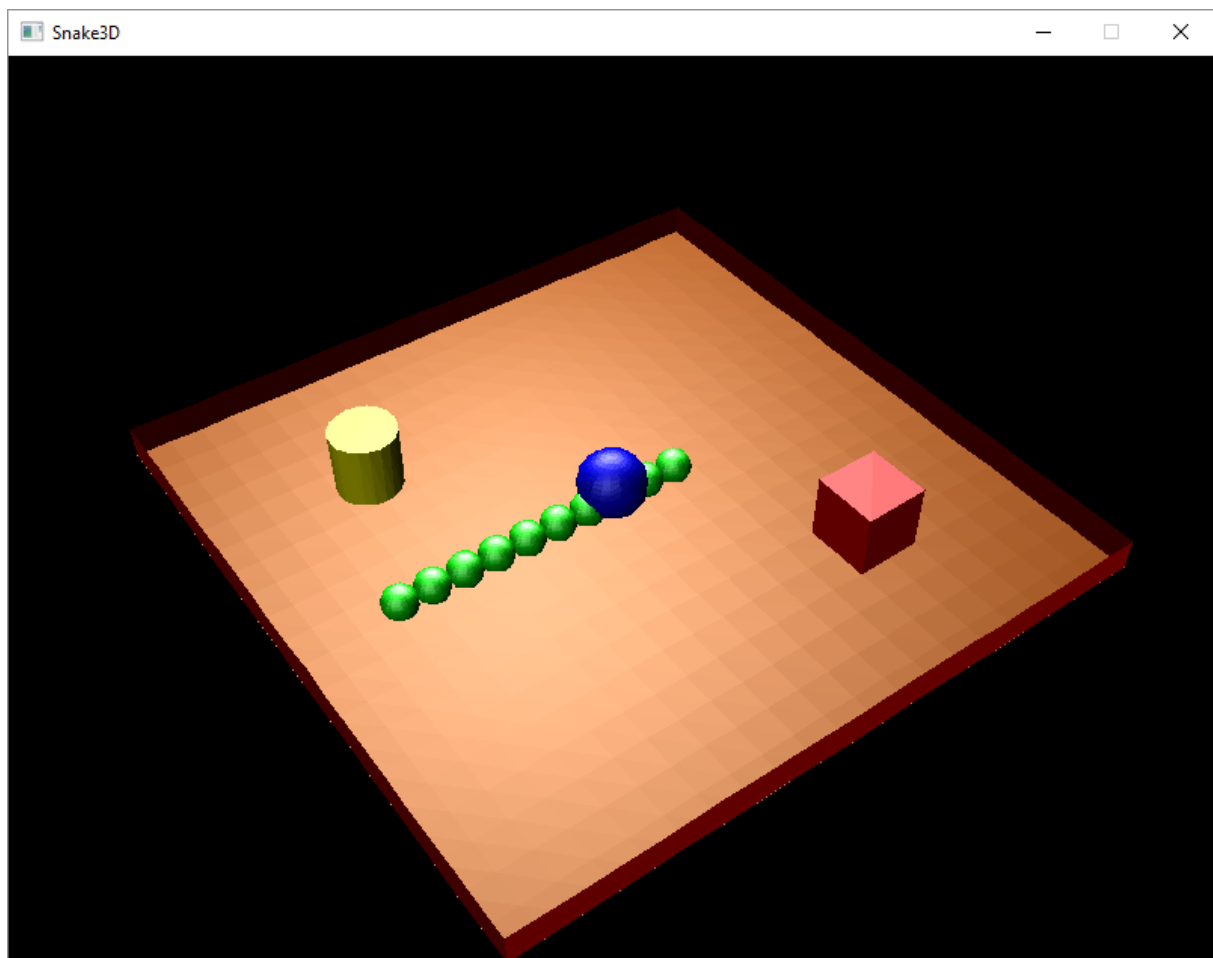
Rysunek 6: Konfiguracja wersji do kompilacji programu

4. Skompilować program w wersji Release (możliwa jest także kompilacja w wersji Debug, jednak wydajność będzie znacznie mniejsza) dla platform 32-bitowych. Należy:
  - Wybrać z pierwszej rozwijanej listy opcję *Release* oraz z drugiej rozwijanej listy opcję *x86*.
  - Wybrać z paska menu *Build > Build Solution*.
5. Po kompilacji program powinien być możliwy do uruchomienia, np. bezpośrednio z Visual Studio 2015 (poprzez wybór z menu *Debug > Start without debugging*).

### 2.3.2 Wersja bez kompilacji

1. Przed instalacją należy się upewnić, że spełnione są wszystkie wymagania systemowe:
  - system operacyjny Windows jest w wersji 7 lub wyższej.
2. Wypakować zawartość dostarczonego archiwum `snake_exe.zip` do wybranej przez siebie lokalizacji (za pomocą programu do obsługi archiwów, np. 7-Zip).
3. Uruchomić program `Snake.exe` zawarty w katalogu głównym (upewnić się wcześniej, że w katalogu znajduje się biblioteka `SDL2.dll`).

Po uruchomieniu programu powinien się wyświetlić domyślny widok:



Rysunek 7: Domyślny widok aplikacji zaraz po uruchomieniu programu





## 2.4 Instrukcja użycia

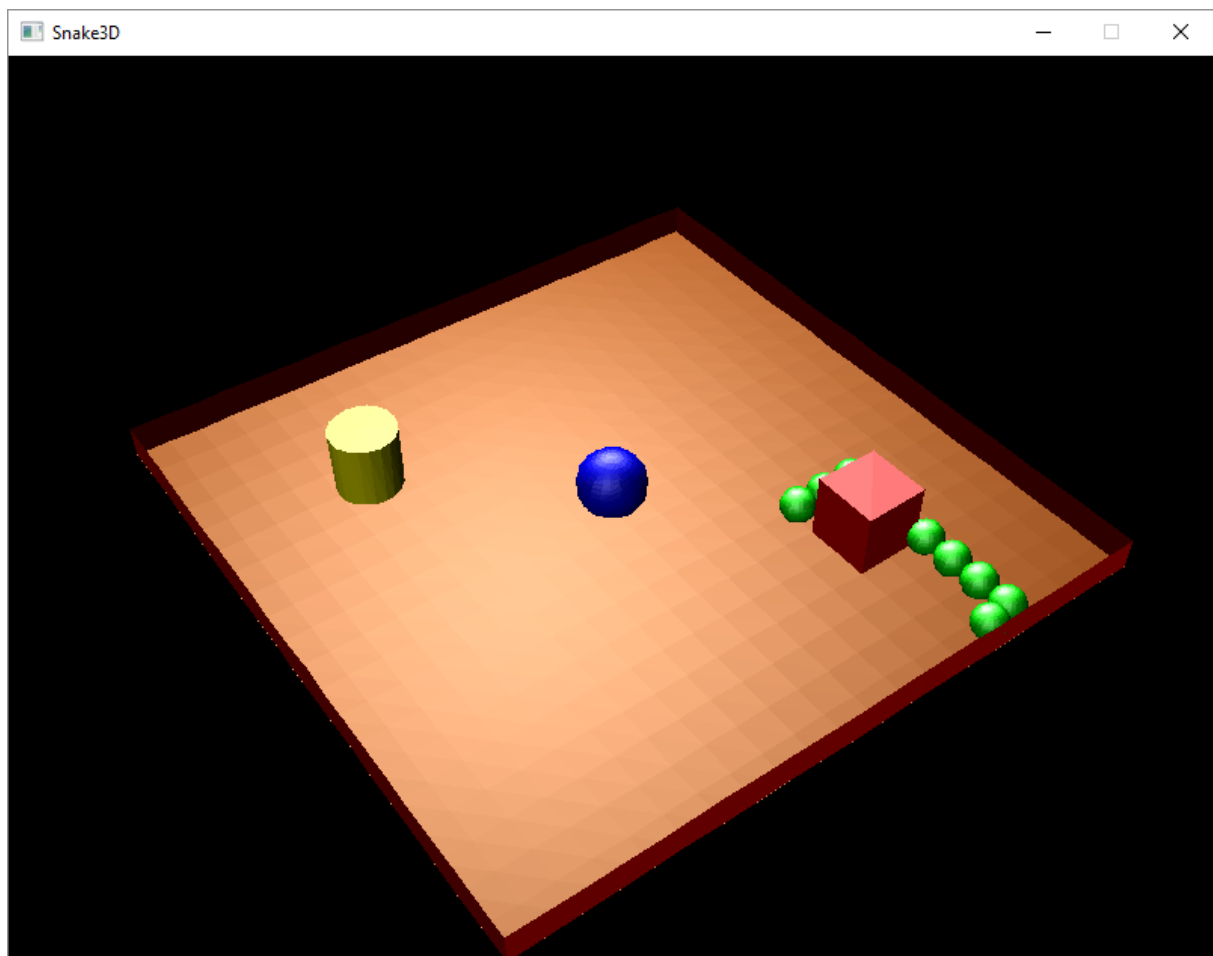
Program domyślnie wyświetla animację poruszającego się węża, którego czoło znajduje się na środku planszy i porusza się w kierunku północnym. Prędkość animacji zależy bezpośrednio od mocy obliczeniowej procesora, jednak nie powinna przekraczać 100 przesunięć na sekundę.

Aby zatrzymać/wznović ruch węża, należy wcisnąć klawisz `Space`. Domyślną metodą zamknięcia programu jest kliknięcie przycisku ZAMKNIJ na pasku tytułu.

### 2.4.1 Zmiana kierunku poruszania się węża

Wąż może zmieniać kierunek poruszania się, co będzie skutkowało powstaniem "kolanka", i zmianą kierunku poruszania się czoła, tak jak w popularnej grze Snake. W celu zmiany kierunku poruszania się, należy wcisnąć klawisz strzałki odpowiadający kierunkowi, w który ma się skierować czoło węża:

-  - północ
-  - zachód
-  - południe
-  - wschód



Rysunek 8: Przykładowy efekt zmiany kierunku poruszania się węża


Ponadto, wąż nie powinien wykraczać poza granice widocznej planszy. W przypadku napotkania granicy planszy, czoło węża powinno skręcić zgodnie z ruchem wskazówek zegara. Ściślej rzecz ujmując:

- napotykając ścianę północną, wąż skręca na wschód,
- napotykając ścianę zachodnią, wąż skręca na północ,
- napotykając ścianę południową, wąż skręca na zachód,
- napotykając ścianę wschodnią, wąż skręca na południe.

Wąż nie powinien zatrzymywać się ani skręcać przy napotykanii na obiekty umieszczone na planszy - powinien przejść przez nie w sposób nienaruszający jego integralności.

#### 2.4.2 Zmiana konfiguracji renderowania


Program umożliwia użytkownikowi zmianę widoku (kamery), modelu oświetlenia (dla cieniowania płaskiego oraz Gourauda) oraz sposobu cieniowania trójkątów.


Zmiana widoku odbywa się poprzez wciśnięcie klawisza . W efekcie, kamera będzie zmieniana na następną w ustalonej niżej kolejności. Spośród kamer do wyboru mamy:

- kamerę statyczną, umieszczoną w jednym punkcie i skierowaną stale w jednym kierunku,
- kamerę statyczną, umieszczoną w jednym punkcie, która zmienia kierunek patrzenia w zależności od położenia czoła węża,
- kamerę podążającą za wężem (tzw. kamerę zza pleców).

Do wyboru mamy także zmianę modelu oświetlenia pomiędzy:

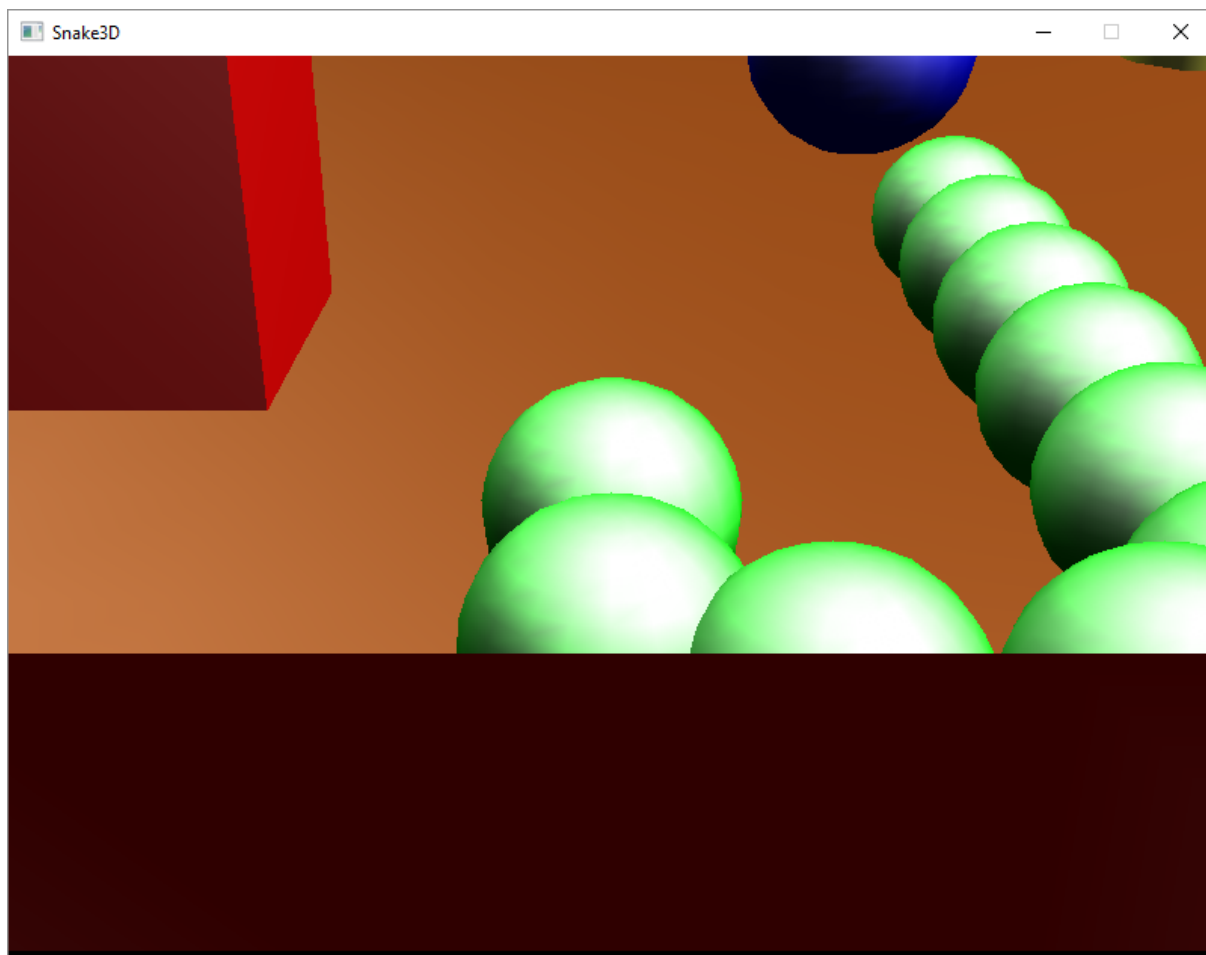
- modelem oświetleniowym Phong
- modelem oświetleniowym Blinn-Phong

Aby przełączać między obydwoma modelami, należy nacisnąć przycisk .

Możemy także zmienić sposób wypełniania trójkątów. Do tego celu służy klawisz , który będzie po kolei zmieniał sposób cieniowania. Do dyspozycji użytkownika są metody:

- cieniowania płaskiego
- cieniowania Gourauda
- cieniowania Phong

Należy tutaj nadmienić, że zmiana modelu oświetlenia w przypadku cieniowania Phong nie ma wymiernego efektu, ze względu na zdefiniowany jednoznacznie sposób wypełniania wierzchołków (tzn. niekorzystający z wyliczonych kolorów w wierzchołkach trójkąta).



Rysunek 9: Kamera zza pleców, z modelem Blinna-Phonga i cieniowaniem Gourauda

### 2.4.3 Dodatkowe informacje dla developerów

Okienko aplikacji nie posiada możliwości zmiany jego rozmiaru - jest ono ustalane w momencie kompilacji. Aby bezpośrednio zmienić rozmiar okna, należy w pliku `main.cpp` zmienić parametry konstruktora dla obiektu klasy `Application`.

Możliwa jest także zmiana wymiarów planszy, domyślnej jednostki odległości oraz "gładkości" rysowanych brył w momencie kompilacji. W tym celu należy zmienić odpowiednie parametry metody `OnInit` obiektu klasy `Application`. Domyślna jednostka odległości powinna jednak być liczbą całkowitą parzystą, aby wąż poprawnie zachowywał się w pobliżu granic planszy.

Domyślną długością węża jest 10 oraz na planszy znajdują się obiekty kostki, walca i kuli o szerokości dwa razy dłuższej niż promień segmentu węża. Dodatkowo, są one ustawione na przeciwdiagonali naszej planszy. Kolory poszczególnych komponentów są takie, jak zaprezentowano na umieszczonych zdjęciach. Dodano także dwa światła: punktowe nad sceną oraz kierunkowe.

Aby zmienić konfigurację sceny, należy użyć odpowiednich metod klas `Renderer` oraz `SceneCreator`, w sposób analogiczny do zaprezentowanego w konfiguracji domyślnej.

## 2.5 Raport odstępstw od specyfikacji wymagań

### 2.5.1 Zmiana biblioteki do obliczeń matematycznych

<b>DOTYCZY</b>	Architektura rozwiązania
<b>ZMIANA</b>	Zmiana użytej biblioteki do obliczeń matematycznych w grafice trójwymiarowej z armadillo na GLM
<b>UZASADNIENIE</b>	Biblioteka armadillo jest nastawiona głównie na użycie do obliczeń numerycznych oraz o szerszym spektrum zastosowań. Z kolei GLM jest biblioteką podobną do języka GLSL (języka używanego w pisaniu shaderów do OpenGL) oraz lżejszą od armadillo. Jej użycie pozwala na prostszą i bardziej intuicyjną implementację potoku renderowania 3D.

### 2.5.2 Opuszczenie użycia loadera plików \*.obj

<b>DOTYCZY</b>	Architektura rozwiązania
<b>ZMIANA</b>	Rezygnacja z użycia loadera tinyobjloader służącego do ładowania plików modeli 3D Wavefront OBJ
<b>UZASADNIENIE</b>	Projekt w końcowej fazie korzysta jedynie z prostych modeli, złożonych z siatek geometrycznych reprezentujących proste bryły geometryczne (prostopadłościany, graniastosłupy, walce, kule). Wobec tego, użycie loadera do zewnętrznych modeli jest zbędne.

### 2.5.3 Odstępstwo od wydajności wyświetlania grafiki

<b>DOTYCZY</b>	Wymagania нефункциональные - wydajność
<b>ZMIANA</b>	Zaniedbanie wymagania dotyczącego prędkości wyświetlania grafiki w określonej liczbie klatek na sekundę
<b>UZASADNIENIE</b>	Głównym celem projektu jest produkcja potoku renderowania, wykonywanego na centralnej jednostce obliczeniowej. W tym przypadku, wydajność będzie mocno zależna od użytej architektury (prędkość taktowania procesora, użyta architektura procesora itp.). W tym przypadku, nie można jednoznacznie zagwarantować jednakowej wydajności na większości używanych sprzętów. Jednakże, na większości nowszych procesorów program będzie działał w miarę płynnie, z wydajnością zbliżoną do podanej w wymaganiach.

## Literatura

- [1] Kotowski P., *Slajdy z wykładów z przedmiotu Grafika komputerowa 1*
- [2] Dokumentacja biblioteki SDL2 <https://wiki.libsdl.org/>
- [3] Dokumentacja biblioteki GLM <http://glm.g-truc.net/0.9.8/api/index.html>