Dokumentacja projektu Sceleton

Tymoteusz Chmielecki, Michał Ambroży, Krzysztof Łakomy

15.06.2020

Spis treści

1 Tytuł projektu

Naszym zadaniem było wykonanie projektu nr 36 - Animacja szkieletu. Tytuł naszego projektu to Sceleton.

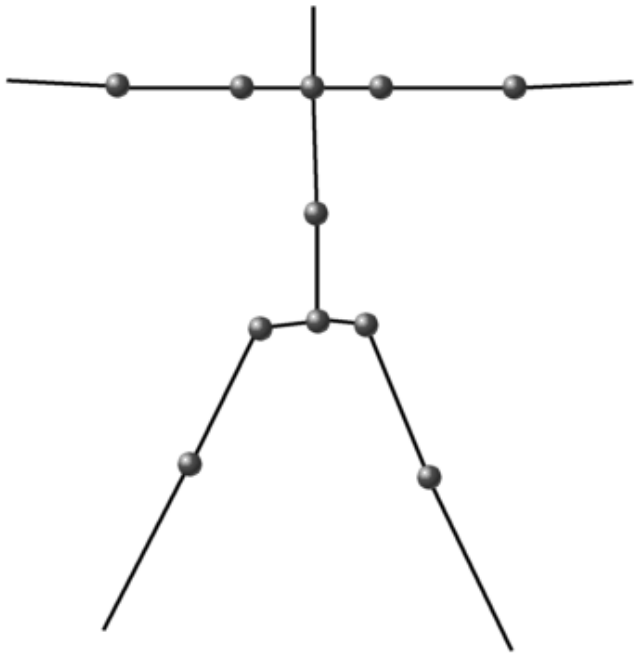
2 Opis projektu

2.1 Cel projektu

Celem projektu jest napisanie programu, który pozwala na animowanie szkieletu człowieka dzięki kontrolerom określającym kąty pomiędzy poszczególnymi segmentami szkieletu.

2.2 Wymagania podstawowe

Okno aplikacji podzielone jest na dwie części. Po lewej stronie widać szkielet, który można swobodnie obracać. Po prawej znajduje się zestaw suwaków, które pozwalają kontrolować naszego „ludzika". Szkielet składa się zaledwie z 14 segmentów (węzeł pomiędzy barkami rozdziela tylko szyję od kręgosłupa). Rysunek przedstawia przykładowy szkielet.



Rysunek 1: Schematyczne przedstawienie szkieletu człowieka

Każdemu węzłowi odpowiadają jeden lub dwa suwaki w zależności od tego ile dany staw posiada stopni swobody. Np. do sterowania zgięciem ręki w łokciu wystarcza jeden suwak, jednak już do sterowania stawem biodrowym potrzebne są dwa suwaki.

2.3 Wymagania rozszerzone

W wersji rozszerzonej program posiada tzw. „linię czasu". Użytkownik określa ile animacja będzie miała klatek i następnie może w dowolnym punkcie ustawić tzw. punkt kluczowy. Np. w pierwszym punkcie ustala, że kąt w łokciu wynosi 0 stopni zaś w punkcie drugim, że ten kąt wynosi 30 stopni. Jeśli teraz użytkownik naciśnie klawisz „Play" będzie mógł obserwować płynną zmianę ugięcia ręki w łokciu. Tak stworzoną animację użytkownik może zapisać w celu późniejszego wczytania. Powinna istnieć również możliwość zapisania animacji jako ciągu plików graficznych na dysku.

2.4 Uwagi dodatkowe

Interesujące byłoby przygotowanie kilku przykładowych animacji. Na przykład „bieg" czy, przysiady".

3 Założenia wstępne

3.1 Cel stworzenia projektu

Celem stworzenia oprogramowania jest wytworzenie aplikacji do animacji ruchów szkieletu.

3.2 Wymagania podstawowe

Oprogramowanie powinno umożliwiać sterowanie ruchów szkieletu. Mowa tu o kończynach, głowie i tułowiu. Ruchy powinne być płynne.

3.3 Wymagania systemowe

Program ma działać na wielu platformach systemowych, zarówno na systemach Windows, jak i na systemie Linux. Każda z pozostałych funkcjonalności powinna zostać wdrożona tak, aby funkcjonowały na każdym z wyżej wymienionych systemów.

3.4 Wymagania rozszerzone

Program powinien umożliwiać animację modelu, poprzez implementację osi czasu z punktem kluczowym. Konieczna jest również możliwość zapisywania animacji w formacie umożliwiające późniejsze wczytanie jej, jak i w postaci serii obrazów.

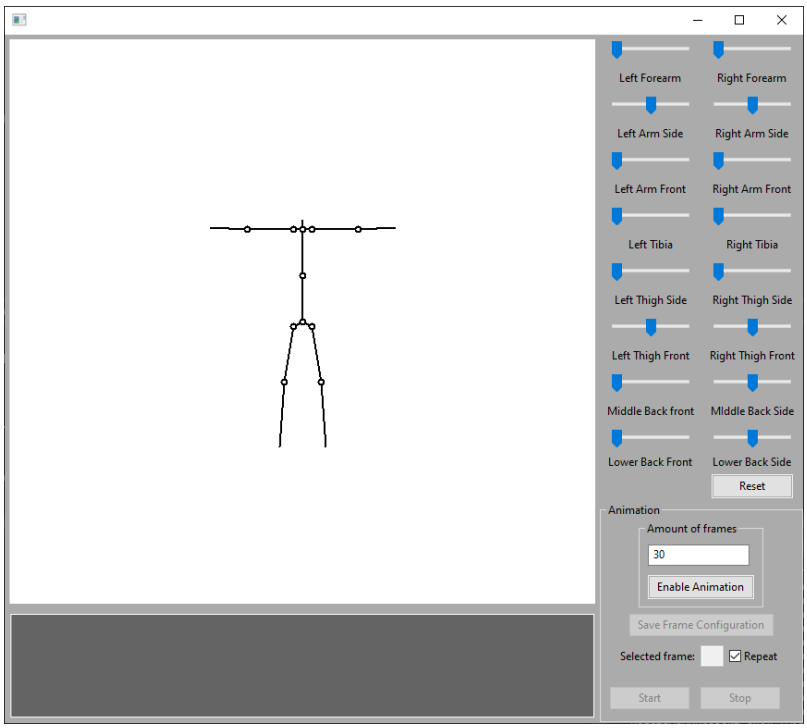
4 Analiza projektu

4.1 Opis ogólny

Oprogramowanie korzysta z biblioteki wxWidgets aby wyświetlić okno, kontrolki na ekranie oraz sam szkielet.

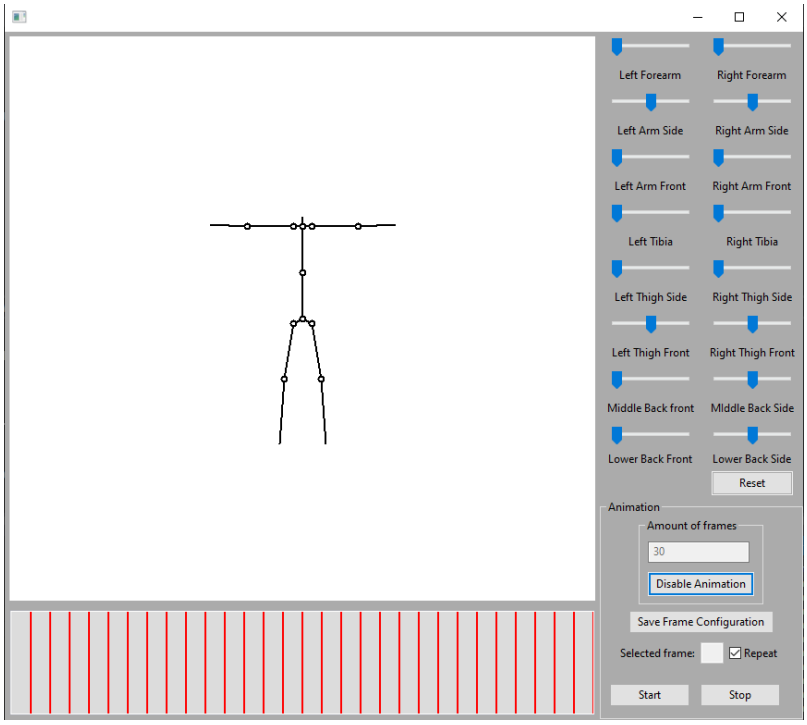
4.2 Specyfikacja interfejsu użytkownika

Intefejs użytkownika jest podzielony na dwie części w wariancie podstawowym oraz na trzy w wariancie rozszerzonym.



Rysunek 2: Przykład interfejsu użytkownika

W prawej części obrazu interejsu widać kontrolki pozwalające na sterownie ruchem szkieletu znajdującego się po lewej stronie. Kontrolki te są suwakami regulującymi chciany przez nas kąt między kośćmi. W wariancie rozszerzonym, oprócz kontrolek w prawej części znajduje się grupa kontrolek służących do animowania ruchów szkieletu oraz kontroli nad osią czasu. Sama oś czasu znajduje się w lewej części interfejsu, w odzielonym pasku na dole.



Rysunek 3: Przykład interfejsu użytkownika z działającą osią czasu

4.3 Specyfikacja danych wejściowych

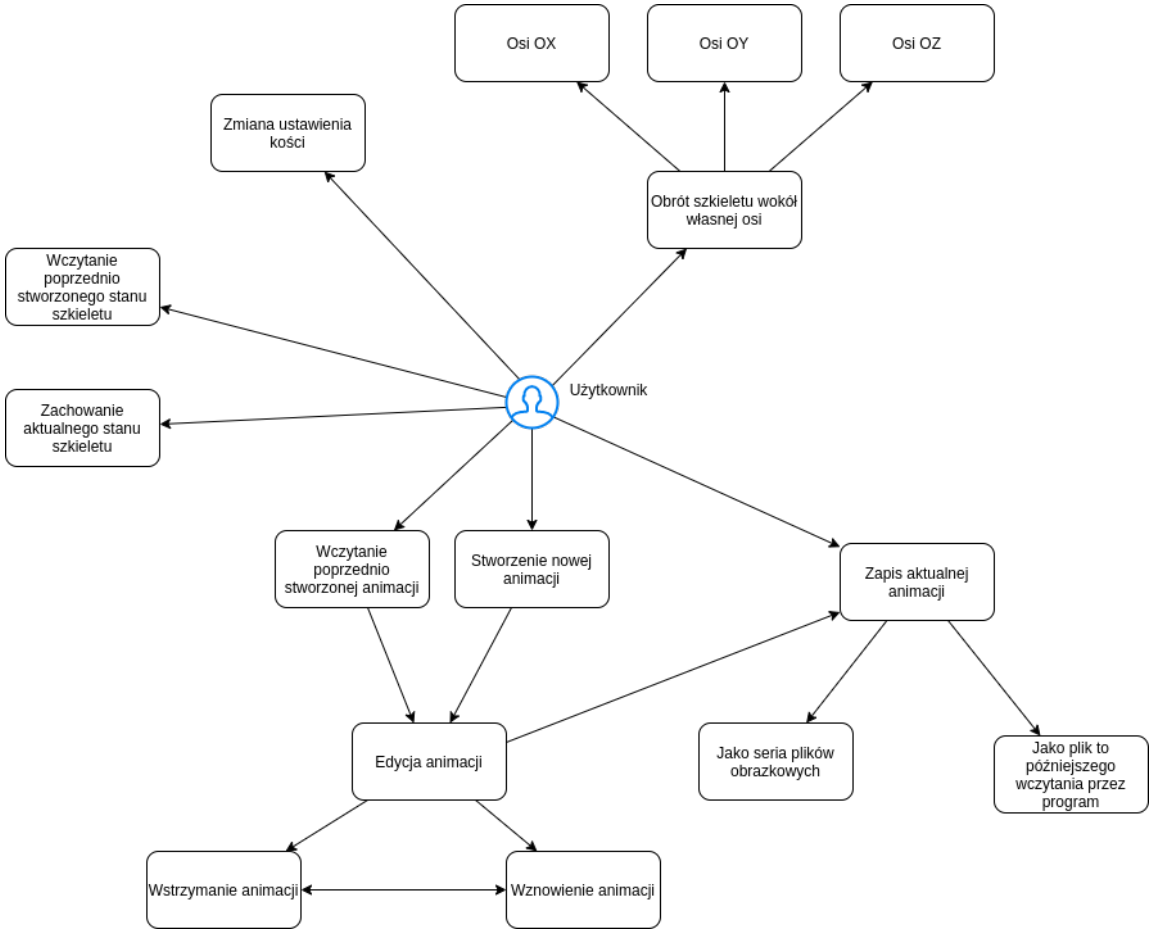
Danymi wejściowymi do projektu są kąty między kośćmi ustawionyme suwakami. W rozszerzonym wariancie jest to też plik z zapisaną konfiguracją szkieletu, lub całą animacją. Jeśli chodzi o suwaki, zakres jest ustalony i nie do przekroczenia. Trzeba jednak zachować ostrożność przy czytaniu plików wczytywanych z dysku, gdyż jest to potencjalny wektor ataku.

4.4 Oczekiwane dane wyjściowe

Danymi wyjściowymi w naszym projekcie jest zapisanie położenia szkieletu, oraz animacji jako całości.

4.5 Specyfikacja przypadków użycia

Oto diagram przedstawiający przypadki użycia oprogramowania przez użytkownika:



Rysunek 4: Digram przypadków użycia

4.6 Zdefiniowanie struktur danych

4.6.1 Format pliku animacji

Do zapisu animacji stosujemy plik CSV.

4.6.2 Zapis animacji jako ciąg obrazów

Do zapisu animacji obrazów tworzymy folder w którym zapisane zostają kolejne obrazy animacji kolejno ponumerowane.

4.7 Wyodrębnienie i zdefiniowanie zadań

Zadania można podzielić następująco:

4.7.1 Planowanie

1. Plan interfejsu użytkownika - plan wyglądu GUI na podstawie kryteriów danych w opisie zadania. Określenie koniecznych elementów
2. Plan struktury kodu - zaplanowanie struktury kodu, która będzie najlepiej odpowiadać poprzednio zdefiniowanenu interfejsowi oraz zadaniu.
3. Rozkład pracy - Bazując na poprzednio stworzonych planach rozdzielenie poszczególnych zadań wymieniowych w implementacji i dokumentacji na każdego z członków.

4.7.2 Implementacja

1. Implementacja interfejsu graficznego - zaprogramowanie wcześniej zdefiniowanego GUI w bibliotece wxWidgets.
2. Implementacja logiki programu - bazując na zaplanowanej strukturze kodu, implementacja wszystkich potrzebnych struktur danych oraz algorytmów potrzebnych do manipulacji szkieletu
3. Implementacja rozszerzonych wymagań dla projektu - po implementacji podstawowtych (koniecznych) elementów projektu, implementacja funkcjonalności rozszerzonych (podpunkt 2.3).
4. Połączenie frontendu i backendu Posiadając całą logikę i gotowy interfejs użytkownika, połącznie wszystkiego w funkcjonującą całość, t.j. zmiana odpowiednich paramterów i wywołanie odpowiednich funkcji w zależności od klikniętego elementu intefejsu.

4.7.3 Dokumentacja i testowanie

1. Plan dokumentacji ogólny dokumentacji - plan podpunktów oraz ich ogólnej treści
2. Szczegółowe uzupełnienie punktów dokumentacji - po skończeniu implementacji programu uzupełnienie szczegółowo części technicznej, analizy projektu oraz wniosków
3. Testowanie programu pod kątem normalnych sytuacji użytkowych - testowanie sprawności oraz niezawdności produktu przy użytkowaniu go w oczekiwany przez nas sposób.
4. Testowanie programu pod kątem nadzywczajnych sytuacji użytkowych - testy niezawodności produktu przy używaniu go w nieplanowany sposób.

4.8 Wybór narzędzi programistycznych

4.8.1 Język programowania

Wybór języka padł na C++. Język ten oferuje dużą szybkość dzięki kompilacji do kodu maszynowego. Równocześnie dzięki oferowanym przez niego abstrakcjom, kod jest bardzo czytelny i prosty w pisaniu. Jest to też język w pisaniu którego każdy z członków grupy miał spore doświadczenie.

4.8.2 Biblioteka graficzna

Do implementacji GUI zdecydowaliśmy się na bibliotekę wxWidgets. Jest ona stworzona z myślą o prostych i dobrze wygladających interfejsach użytkownika. Dodatkowymi atutami jest jej multiplatformowość oraz nasze wcześniejsze doświadczenie z nią dzięki laboratoriom z Podstaw Grafiki Komputerowej.

5 Podział pracy i analiza czasowa

5.1 Procentowy podział pracy

Dla wcześniej opisanych elementów projektu w niżej przedstawionej tabelce został przedstawiony procentowy udział każdego z członków grupy.

	Tymoteusz Ch	Michał A	Krzysztof Ł
Planowanie			
Plan interfejsu użytkownika	20 %	30 %	50 %
Plan struktury kodu	25 %	30 %	45 %
Rozdzielenie pracy	33 %	33 %	33 %
Implementacja			
Implementacja interfejsu graficznego	5 %	30 %	65 %
Implementacja podstawowej logiki programu	25 %	20 %	55 %
Implementacja rozszerzonej logiki programu	30 %	40 %	30 %
Testowanie i dokumentacja			
Plan dokumentacji	60 %	25 %	15 %
Szczegółowe uzupełnienie punktów dokumentacji	60 %	25 %	15 %
Testowanie pod kątem normalnych sytuacji użytkowych	25 %	50 %	25 %
Testowanie pod kątem nadzwyczajnych sytuacji	25 %	50 %	25 %

Tabela 1: Udział procentowy każdego z członków grupy w każdej częsci projektu

Udział procentowy został przybliżony jako funkcja czasu włożonego w dany element oraz ilości linii kodu / tekstu który został dodany w efekcie pracy.

5.2 Analiza czasowa

Łączny czas implementacji projektu szacowany jest na około 80 - 90h. Szczegółowy podział tego czasu na poszczególne zadania widać w tabelce poniżej

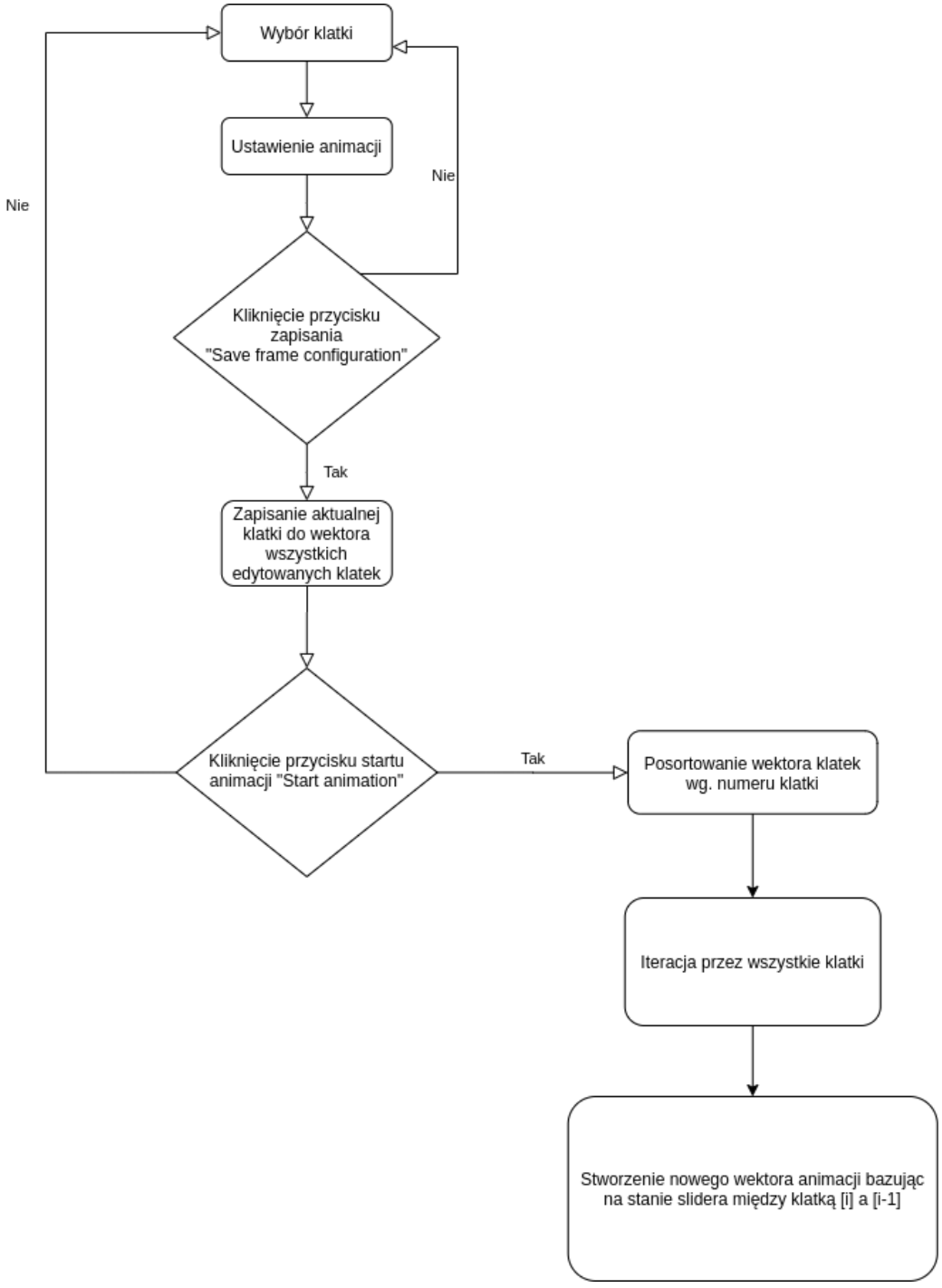
	Szacowana ilość godzin
Planowanie	
Plan interfejsu użytkownika	2 h
Plan struktury kodu	4 h
Rozdzielenie pracy	1 h
Implementacja	
Implementacja interfejsu graficznego	10 h
Implementacja podstawowej logiki programu	22 h
Implementacja rozszerzonej logiki programu	23 h
Testowanie i dokumentacja	
Plan dokumentacji	4 h
Szczegółowe uzupełnienie punktów dokumentacji	20 h
Testowanie pod kątem normalnych sytuacji użytkowych	2 h
Testowanie pod kątem nadzwyczajnych sytuacji	3 h
Suma	91 h

Tabela 2: Szacowany wkład czasowy w poszczególne części projektu

6 Opracowanie i opis niezbędnych algorytmów

6.1 Algorytm animacji

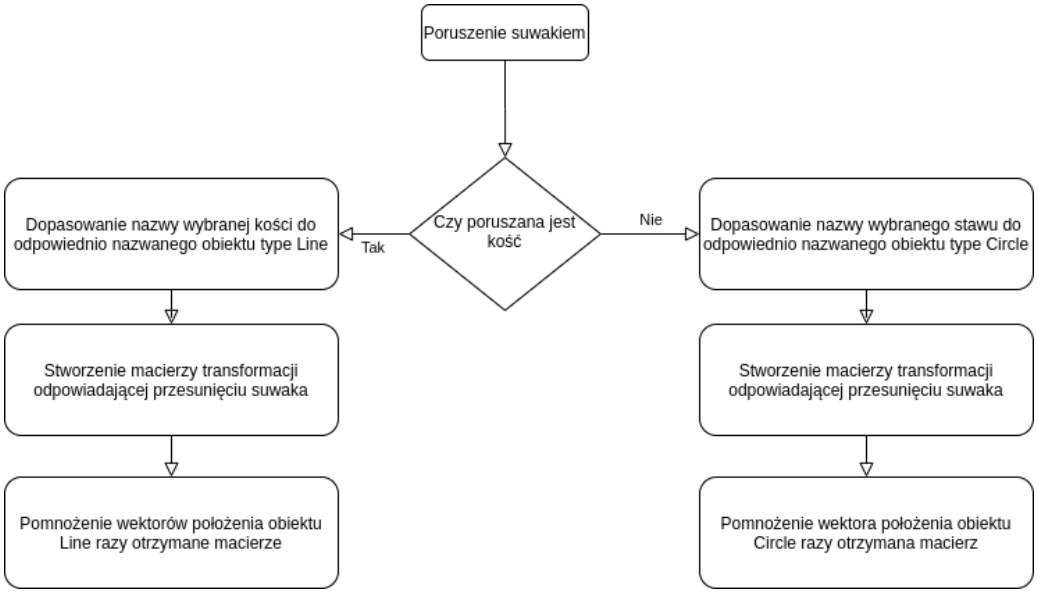
Oto diagram przedstawiający wybrany przez nas algorytm obsługi animacji:



Rysunek 5: Digram obsługi animacji

TODO Pierwszym etapem jest wybór klatki na osi czasu. To generalnie algorytm jest prosty jak konstrukcja cepa Na osi czasu wybierasz w której klatce co ustawiasz I zapisuje się to na przycisk I jak zaczynasz animacje przyciskiem start To wtedy sobie sortuje te zapisana tablice wedle numeru klatki Tworzę nowy wektor Animacji Który jest tworzony na podstawie Klatki tej i poprzedniej

6.2 Algorytm obsługi suwaka



Rysunek 6: Digram obsługi animacji

7 Programowanie

7.1 Specyfikacja techniczna

7.1.1 Środowisko, język, biblioteki

Kod aplikacji jest napisany w języku C++ przy pomocy Visual Studio. Visual Studio to zintegrowane, wieloplatformowe środowisko graficzne (IDE) ukierunkowane na tworzenie aplikacji w języku C++. Do implementacji GUI użyto biblioteki wxWidgets. Skupia się ona na dostarczeniu prostego i intuicyjnego interfejsu do programowania prostych interfejsów użytkonika (GUI). Pozwala na rysowanie prostych linii, oraz dostarcza elementy interfejsu takie jak suwaki, guziki oraz checkboxy. Jest ona również wieloplatformowa. Oprogramowanie jest kompletnie samowystarczalne i wymagana jest jedynie kompilacja w celu uruchomienia na dowolnym systemie operacyjnym (np. Windows, Linux).

7.2 Klasy

Oto klasy użyte przez nas w tym projekcie:

7.2.1 Vec

Klasa przedstawiająca trójwymiarowy wektor. Zawarte w niej funkcje

1. Vec - kontruktor przyjmujący trzy parametry typu double oraz domyślny.
2. Set - funkcja zmieniająca wartości wektora

7.2.2 Matrix

Klasa przedstawiająca macierz o wymiarach 4x4. Zawarte w niej funkcje:

1. Matrix - kontruktor przyjmujący 0 paramterów, inicjujący macierz.

7.2.3 Line

Klasa przedstawiająca trójwymiarówa linię. Przechowuje jako paramtery swój początek i koniec w postaci obiektów Vector (klasa 7.1.1) oraz swoją nazwę (używane do przechowywania nazwy kości).

7.2.4 Circle

Klasa przedstawiająca trójwymiarówa kulę. Przechowywuje jako parametry swój środek jako obiekt Vector (klasa 7.1.1), swoją nazwę (używane do zapisana nazwy stawu) oraz swój promień (stała 3.0).

7.2.5 MyApp

Rozszerza klasę wxApp.

7.3 Funkcje niezależne

1. mat_vec_multiply - funkcja mnożąca macierz (klasa 7.1.2) razy wektor (klasa 7.1.1).
2. mat_mat_multiply - funkcja mnożąca dwie macierze (klasa 7.1.2).
3. rotate_x - funkcja zwracająca macierz (klasa 7.1.2) rotacji o dany kąt w płaszczyźnie x.
4. rotate_y - funkcja zwracająca macierz (klasa 7.1.2) rotacji o dany kąt w płaszczyźnie y.
5. rotate_z - funkcja zwracająca macierz (klasa 7.1.2) rotacji o dany kąt w płaszczyźnie z.
6. translate - funkcja zwracająca macierz (klasa 7.1.2) translacji w osiach x, y i z.

8 Testowanie

8.1 Testowanie zwyczajnych sytuacji

W celu testowania sytuacji wynikających z normalnych czynności, osoby trzecie nie biorące w nim udziału zostały poproszone o parę minut użytkowania. Były w tym czasie obserwowane i każdy błąd był notowany. W ten sposób został znaleziony błąd przy obsłudze osi czasu.

8.2 Testowanie nadzwyczajnych sytuacji

W celu testowania sytuacji nadzwyczajnych programowi został podany zbyt długi i źle sformatowany plik z zapisaną animacją. Spowodowało to crash programu i w następstwie ryzyko bezpieczeństwa. Błąd został następnie naprawiony.

9 Wdrożenie, raport i wnioski

9.1 Wnioski

Po skończeniu projektu naturalne są wnioski wynikające z jego implementacji. Najważniejsze z nich zostają wymienione poniżej:

1. Początkowe założenia i plany nie zakładały problemów technicznych takich jak niezgodność bibliotek, lub różnic między środowiskami. Jest to coś na co trzeba zwrócić uwagę przy przyszłym planowaniu.

2. Podział pracy zmienił się w stosunku to pierwszych planów. Powodowało to konieczność płynnej zamiany zadań, co mogłoby być problemem w grupie gdzie nie każdy posiada wszystkie konieczne umiejętności.

3. Testowanie zaimplementowanych funkcjonalności pozwala na poprawę błędów oraz sprawia, że oprogramowanie jako całość staje się lepsze. Z tego powodu jest bardzo ważnym elementem cyklu produktu.