1. Tytuł projektu i autorzy projektu

Tytuł projektu to: **Animacje**

Autorami projektu są: Judyta Choczewska, Klaudia Fil oraz Joanna Marcinkowska.

2. Opis projektu

Projekt polegał na napisaniu programu, który na podstawie danych pobranych z pliku, generowałby ciąg bitmap, który następnie byłby wyświetlany. Poza doborem poszczególnych kształtów plik tekstowy miałby wpływ na wielkość generowanego okna oraz czas wyświetlania każdej z ramek.

3. Założenia wstępne przyjęte w realizacji projektu

Założenia wstępne podstawowe:

Założono realizację programu, który powinien pobierać pliki tekstowe, a następnie linia po linii sczytywać i wykonywać szereg instrukcji. Pierwsza linia oznacza rozmiar generowanych bitmap, następnie podawany ma być numer klatki i odpowiadający mu czas trwania i polecenia odnośnie samego rysunku. Wśród tych ostatnich znajdują się: ustawienie rozmiaru i koloru pióra, ustawienie koloru wypełnienia figury, możliwość narysowania punktu, linii, prostokąta i elipsy. Figury ponadto powinny mieć znacznik czy są wypełnione kolorem, czy domyślnie są białe, przykładowa instrukcje:

- *PT x y* - rysuje punkty na zadanych współrzędnych
- *elipsa x1 y1 x2 y2 flaga* - rysuje elipsę o zadanych współrzędnych wypełnioną (flaga=1) lub nie (flaga=0)

Instrukcje rysujące poprzedzone są pełną nazwą znacznika lub jej skrótem (np. *ST*= *stop*). Poza samym rysowaniem możliwy byłoby zapisanie wszystkich wygenerowanych klatek do pliku.

Założenia wstępne rozszerzone:

W ramach rozszerzenia wzbogadziłyśmy szereg instrukcji rysujących w możliwość narysowania trójkąta, dodania tekstu na animację, oraz wczytywania i obsługi bitmap. Poza tym dodano możliwość zatrzymania i uruchomienia animacji w dowolnym czasie, oraz włączenia do niej muzyki, jeżeli takowa jest jej podporządkowana.

4. Analiza projektu

A. Specyfikacja danych wejściowych

Dane wejściowe narzucone były wewnątrz funkcji tj. plik tekstowy (.txt), audio (.wav) oraz obrazek (.png), ponieważ z góry były przyporządkowane konkretnym animacją. Jedną możliwością modyfikacji przez użytkownika był jej wybór.

B. Opis oczekiwanych danych wyjściowych

Dane wyjściowe to generowane klatka po klatce bitmapy wyświetlane na panelu aplikacji. Istnieje również opcja zapisania każdej animacji w pliku w formacie .bmp umożliwiająca zobaczenie każdej klatki osobno.

C. Definiowanie struktur danych

Dane (rysunki - funkcja Draw()) w naszym przypadku przechowywane są początkowo na bitmapie roboczej, którą po zakończeniu jednej klatki dodaje się do wektora przechowującego wxBitmap, dzięki tej operacji kolejne wyświetlenia są bardzo proste w implementacji oraz sam zapis do pliku jest maksymalnie uproszczony. Podobnie czasy klatek przechowywane są w wektorze, tylko typy int, to samo indeksowanie pozwala na równoległe prowadzenie ich.

D. Specyfikacja interfejsu użytkownika

Interfejs jest bardzo prosty w obsłudze, posiada zaledwie jeden przycisk odpowiedzialny za zapis do pliku, oraz rozwijane menu wyboru animacji podpisane statycznym tekstem. Poza tym wyodrębnione są dwa checkbox'y, które pozwalają użytkownikowi wpłynąć na włączenie/ wyłączenie animacji i muzyki do tej animacji. Większą część zajmuje panel, na którym wyświetlane są klatki. Wszystko klarownie podpisane.

E. Wyodrębnienie i zdefiniowanie zadań

Zadania można podzielić następująco:

- Wygenerowanie okna aplikacji
- Wybór odpowiedniego pliku tekstowego do animacji
- Wybór odpowiedniego pliku audio do animacji
- Wybór odpowiedniego pliku graficznego do animacji
- Generacja ciągu bitmap na podstawie 3 wcześniejszych podpunktów
- Wyświetlanie poklatkowe animacji
- Zapis do pliku serii klatek
- Tworzenie plików tekstowych (wykorzystując plik .c) - kilka

F. Decyzja o wyborze narzędzi programistycznych

Wykorzystane zostało środowisko Visual Studio, uznałyśmy, że wystarczająco dobrze poznałyśmy jego możliwości poprzez pracę w ostatnim semestrze. Pracowano na zasobach biblioteki wxWidgets, która od początku była brana jako jedyna pod uwagę, ze względu na bogate klasy i funkcje, a także przyjazną dokumentacje.

5. Podział pracy i analiza czasowa

Do pracy zabrałyśmy się już w połowie maja i określiłyśmy nasze oczekiwania co do projektu, które generalnie nie uległy zmianie podczas jego tworzenia. Jednak początkowe trudności związane z napisaniem głównej funkcji zajęły nam trochę czasu. Po zażegnaniu ich, mogłyśmy tworzyć pliki odpowiedzialne za animację.

Podział obowiązków:

- Judyta Choczewska - funkcja zapisująca bitmapy, poszukiwanie plików .png, część dokumentacji
- Klaudia Fil - funkcja generująca bitmapy, część dokumentacji
- Joanna Marcinkowska - funkcja dodająca muzykę do animacji, poszukiwanie plików audio, część dokumentacji

Ponadto każda z osób miała do napisania po 2 lub 3 animacje oraz wspólnymi siłami zajmowałyśmy się funkcją wyświetlającą klatki.

6. Opracowanie i opis niezbędnych algorytmów

Po wielu próbach i zbadanych możliwościach (próba wykorzystania funkcji bibliotecznych m.in. *wxMilliSleep()*) ograniczono działanie programu do dwóch zależnych od siebie algorytmów:

a) Algorytm generujący pojedyncze klatki

Najważniejszy w programie algorytm, który dzięki sczytywanym liniom z plików tekstowych generuje pożądane kształty o zadanych parametrach tworząc obraz. Algorytm posiłkuje się funkcją *std::string returnWord(int index, std::string str),* która umożliwia szybkie i łatwe wyłowienie pojedynczych słów (lub wartości), według wcześniej określonego schematu. Parametryzuje się nowy obiekt, którego typ ustalany jest w zależności od pierwszej wartości w nowej linii. Początkowo podawany jest rozmiar generowanych klatek, a następnie przy pomocy szeregu instrukcji *if()* uzupełnia się wektor odpowiedzialny za przechowanie wartości czasowych każdej z klatek lub rysunki na pojedynczej klatce animacji, którą również przekazuje się do wektora, ale odpowiedzialnego za przechowanie obrazków.

b) Algorytm wyświetlający animacje w funkcji czasu

Jest to algorytm odpowiedzialny za płynne wyświetlanie serii klatek, tworzy złudzenie ruchu, a co za tym idzie krótkie animacje. Uzależniając panel wyświetlający klatkę po klatce wykorzystano proste metody iteracyjne, które „wędrują” po wektorze przechowującym obrazki bez przerwy. Podczas włączenia czasu zwłoki przy przejściach wykorzystano drugi wektor z czasami, który prowadzono równolegle do elementów tablicy z obrazkami. Możliwe jest też zatrzymanie animacji, przy wyborze odpowiedniego przycisku, który pociąga za sobą wywołanie funkcji zatrzymującej stoper, przez co animacja zatrzymuje się w jednym miejscu i jest możliwe ponowne jej wznowienie od ostatniej wyświetlonej klatki.

Poza głównymi algorytmami stworzonymi na potrzeby działania samego okna, do poprawnego wykonywania zadania wykorzystano również krótkie algorytmy stworzone w

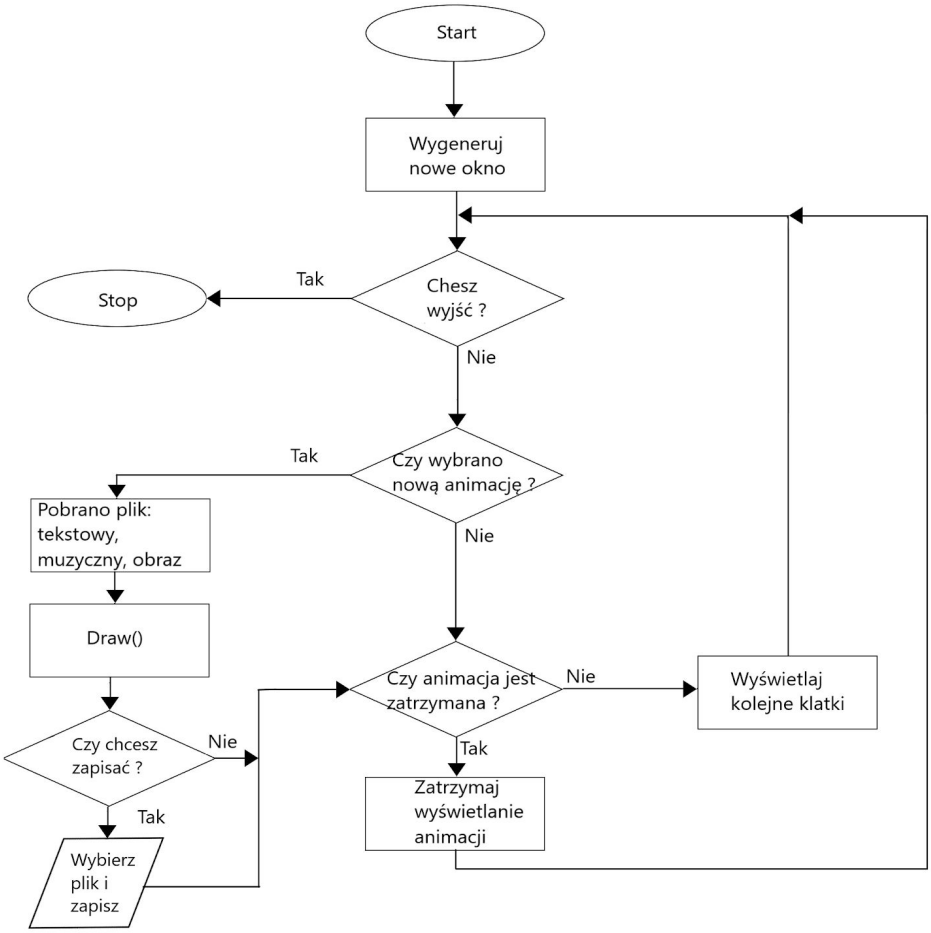
języku C na potrzeby kreowania plików tekstowych. Algorytmy te opierały się na głównej pętli, która w każdym obiegu generowała i zapisywała do pliku tekst potrzebny do stworzenia jednej klatki zależny od autora. Były one generowane na zasadzie prób i błędów.

7. Kodowanie

a) Opis programu

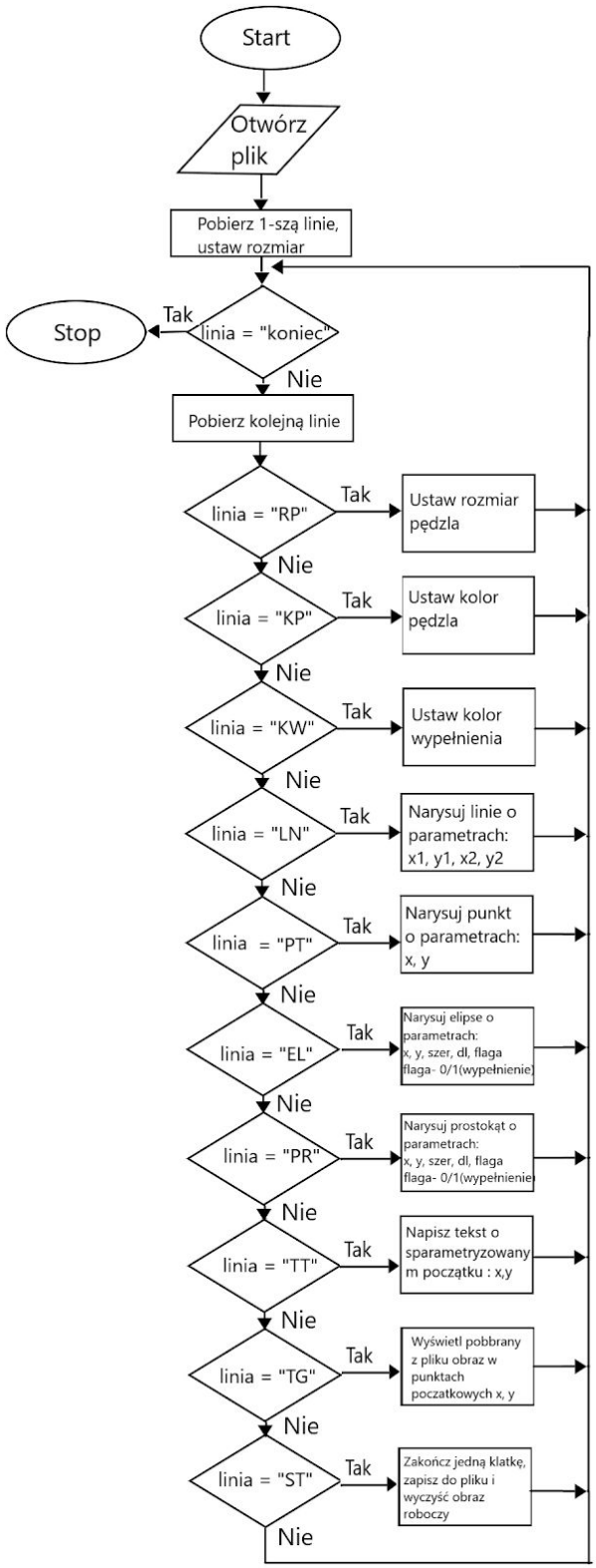
Program został stworzony w celu wyświetlania animacji poklatkowej z ciągu wcześniej wygenerowanych bitmap na podstawie plików tekstowych. Opiera się na dwóch klasach, z których jedna jest wygenerowana przy pomocy wxFormBuilder, a druga dziedziczy po niej wprowadzając odpowiednie metody, które pozwalają na wykonanie wcześniej założonych możliwych operacji.

b) Schemat blokowy



Rys 1. Schemat blokowy działania programu (uproszczony-bez procedury Draw())

c) Schemat najważniejszego algorytmu - procedura Draw()



Rys 2. Schemat blokowy działania procedury Draw()

Schemat na powyższym rysunku został uproszczony w blokach warunkowych wykorzystano jedynie skróty w celu zwiększenia przejrzystości grafu.

d) Szczegółowy opis klas funkcji i zmiennych:

Klasy:

1. **MyFrame** - klasa zapisana w pliku *GUI.h* wygenerowana przy pomocy wxFormBuilder odpowiedzialna za stworzenie interfejsu aplikacji oraz połączenie poszczególnych przycisków z funkcjami

1.1. Funkcje

· Wszystkie funkcje z tej klasy są wirtualne, ich odpowiedniki zostaną omówione na podstawie klasy GUIMyFrame, która dziedziczy po tej klasie.

1.2. Zmienne

- wxPanel* m_panel
  – panel, na którym wyświetlane są animacje
- wxButton* m_buttonSave
  – przycisk, który powiązany z odpowiednią funkcją zapisuje serie klatek
- wxStaticText* m_staticText
  – statyczny tekst, dla upiększenia „Animacje:”
- wxChoice* m_choiceAnimation
  – menu wyboru spośród dostępnych animacji
- wxCheckBox* m_checkBoxMusic
  – checkbox umożliwiający włączenie lub wyłączenie muzyki, jeżeli takową opcje animacja ma
- wxCheckBox* m_checkBoxStopAnimation
  – checkbox umożliwiający zatrzymanie animacji i ponowne jej uruchomienie
- wxTimer m_timer
  – timer potrzebny przy wyświetlaniu klatek, które są od niego uzależnione

2. **GUIMyFrame** – klasa dziedzicząca po MyFrame

2.1. Funkcje

- virtual void FormUpdate(wxUpdateUIEvent& event)
  – funkcja wirtualna odpowiada za aktualizację okna na podstawie wybranych opcji
- virtual void OnButtonClickSave(wxCommandEvent& event)
  – funkcja wirtualna odpowiada za zapis do katalogu aktualnie wyświetlanej animacji klatka po klatce, którą po prostu sczytuje z wektora z *xwBitmap*
- virtual void OnChoiceAnimation(wxCommandEvent& event)
  – funkcja wirtualna odpowiada za aktualizacje nazwy zmiennej przechowującej plik tekstowy, załadowanie obrazka czy tekstu, jeżeli taki dana animacja posiada oraz wywołuje funkcję Draw(), która aktualizuje wektor bitmap i finalnie uruchamia timer, który pozwala na wyświetlenie animacji, możliwe jest również jego zatrzymanie
- virtual void OnCheckBoxMusic(wxCommandEvent& event)
  – funkcja wirtualna odpowiada za uruchomienie muzyki w zależności od wyboru animacji, oraz czy ta animacja takową opcję ma, zależna od zaznaczenia checkboxa
- virtual void OnCheckBoxStopAnimation(wxCommandEvent& event)
  – funkcja wirtualna odpowiada za zatrzymanie/wznowienie animacji, co za tym idzie muzyki w zależności od zaznaczenia checkboxa
- virtual void OnTimer(wxTimerEvent& event)
  – funkcja wirtualna odpowiedzialna za zmianę aktualnego numeru wyświetlanej bitmapy, działająca w sposób symetryczny dla animacji, pierwsze wyświetlają się klatki w kolejności rosnącej, następnie w malejącej i tak nieskończenie długo
- void Draw()
  – funkcja rysująca do wektora bitmap poszczególne klatki, jej koncepcja została przedstawiony w punkcie 6. od strony technicznej warto zauważyć, że operuje ona przy pomocy obiektu *wxMemocyDc* na podpiętej pod niego *wxBitmap* uzupełniając rysunek w każdym przebiegu pętli *while()* w zależności od instrukcji pobranych z pliku *wxTextFile,* uzupełnia też rozmiary panelu i długości trwania klatek w wektorze z czasami; kończy się instrukcją z pliku „KONIEC”

- void Repaint()

  – funkcja spinająca wszelkie działania *wxTimera*, która w zależności od niego uruchamia animację oraz ustawia parametry rozmiaru panelu i okna pobrane z pliku

2.2. Zmienne

- enum m_animation {start = 0, gitarzysta = 1, perkusista = 2, trebacz = 3, tancerz = 4, krzysztof = 5, niebo = 6}

  - zmienna wyliczeniowa przechowująca opcje animacji, nazwy dla uproszczenia kodu
- int m_chosenAnimation;

  - przechowuje aktualny numer animacji wybranej z menu
- bool m_isNew;

  - zmienna logiczna zależna od pytania: Czy zmieniono wybór animacji?
- wxString m_nameTextFile;

  - zmienna przechowująca nazwę pliku tekstowego zależna od wyboru animacji
- bool m_isStopActived;

  - zmienna logiczna zależna od pytania: Czy animacja jest zastopowana?
- bool m_isMusicActived;

  - zmienna logiczna zależna od pytania: Czy muzyka jest włączona?
- wxSound m_sound;

  - zmienna przechowująca dźwięk dla wybranej animacji
- wxBrush m_brushColor;

  - zmienna przechowująca zmodyfikowany kolor pędzla dla jednej animacji
- int m_penSize;

  - zmienna przechowująca zmodyfikowany rozmiar pędzla dla jednej animacji
- wxPen m_pen;

  - zmienna przechowująca zmodyfikowany kolor i rozmiar pędzla dla jednej animacji
- wxString m_tekst;

  - zmienna przechowująca tekst wyświetlany na animacji, jeżeli ta taką opcję posiada
- wxFont m_fontTextOnAnimation;

  - zmienna przechowująca zmodyfikowaną czcionkę tekstu wyświetlanego

- int m_panelX
  - zmienna przechowująca rozmiar poziomy panelu sczytany z pliku
- int m_panelY
  - zmienna przechowująca rozmiar pionowy panelu sczytany z pliku
- wxBitmap *m_image;
  - zmienna przechowująca obrazek wyświetlany na animacji, jeżeli ta taką opcję ma
- wxImageHandler * m_imageHandler;
  - zmienna do przetrzymywania obrazu wyświetlanego na animacji
- bool m_isImage;
  - zmienna logiczna zależna od pytania: Czy ta animacja ma mieć obrazek?
- std::vector<wxBitmap> m_allAnimation;
  - zmienna typu wektor przechowujący bitmapy z aktualnie wybranej animacji
- std::vector<int> m_timeFrames;
  - zmienna typu wektor przechowujący czasu z aktualnie wybranej animacji
- int m_frame;
  - zmienna przechowująca numer aktualnie wyświetlanej ramki
- int m_numberOfFrames;
  - zmienna przechowująca ilość klatek danej animacji

8. Testowanie

Testy aplikacji prowadzone były na bieżąco, początkowo zależało nam na poprawnym wyświetleniu chociaż jednej klatki, następnie zajęto się generowaniem zwłoki przy wyświetlaniu, końcowo zapisem animacji do pliku. Po wykonaniu opcji podstawowych przeszłyśmy do dodania stopu i muzyki. Ostatecznie najwięcej czasu poświęciłyśmy na tworzenie plików z animacjami, do których doszłyśmy przez próby i błędy.

9. Wdrożenie, raport i wnioski

Program spełnia wszystkie nasze początkowe założenia. Wykonane zostało 7 animacji, a ponadto program można w bardzo prosty sposób rozszerzyć o kolejne uzupełniając w odpowiednich miejscach kod. Jedynym problemem jest chwilowe zawieszenie podczas generowania dłuższych serii bitmap. Można byłoby spróbować operować na innych strukturach lub wyświetlić odpowiedni komunikat np. “Proszę czekać”.