AKADEMIA GÓRNICZO-HUTNICZA

Im. Stanisława Staszica w Krakowie

Podstawy grafiki komputerowej

Projekt 38. Interpreter grafiki wektorowej

Arkadiusz Michalik
Eryk Jarocki
Piotr Pasternak

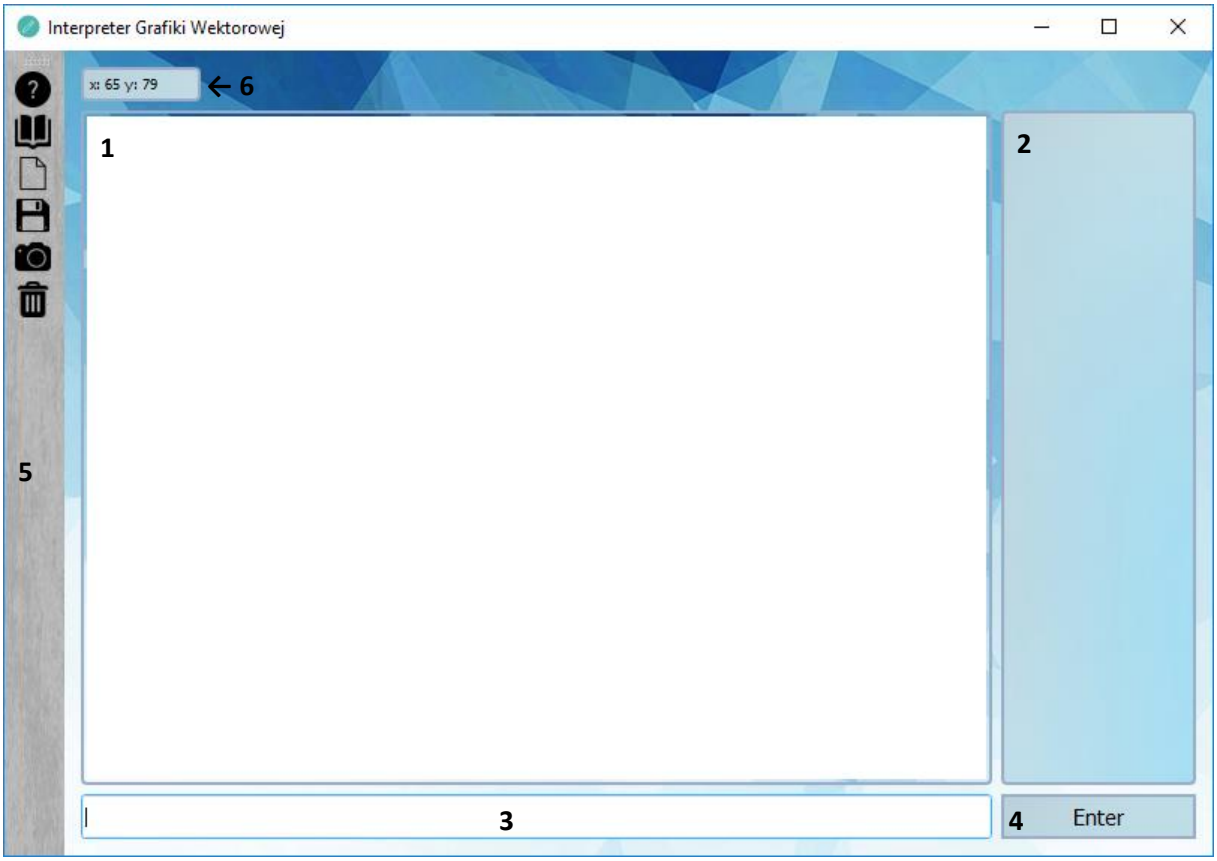
WFIIS

Spis treści:

1. Opis projektu.	Str 3.
2. Założenia wstępne przyjęte w czasie realizacji projektu	str 4.
3. Analiza projektu	str 4.
4. Podział pracy i analiza czasowa	str 7.
5. Opracowanie i opis niezbędnych algorytmów	str 8.
6. Kodowanie	str. 10.
7. Testowani	str. 12.
8. Wdrożenie, raport i wnioski	str. 15.

1. Opis projektu.

Projekt dotyczył stworzenia interpretera dwuwymiarowej grafiki wektorowej.
Do wykonania projektu użyto rozwiązania z biblioteki Qt (wersja 5.9.0 Community Licence) firmy The Qt Company, oraz programu Qt Creator (wersja 4.3.1) . Powyższa biblioteka pozwoliła na stworzenie szkieletu aplikacji (główne okna, oraz widżety składające się na aplikację).



Rysunek 1. Główne okno programu.

Główne okno programu składa się z:

1. Głównego obszaru roboczego (dalej ‘scena’),
2. Listy utworzonych obiektów,
3. Linii komend,
4. Klawisza zatwierdzającego komendę,
5. Przesuwalnego menu podręcznego,
6. Wskaźnika obecnej pozycji myszki na obszarze roboczym.

Docelowo aplikacja pozwala na komunikację z użytkownikiem przy użyciu komend interpretera. Komendy te przetwarzane są w obiekty wyświetlane na ekranie. Aplikacja pozwala na utworzenie zrzutu ekranu (format .png) stworzonego obrazu, oraz zapisanie postępu prac, a następnie ponowne ich wczytanie.

2. Założenia wstępne przyjęte w czasie realizacji projektu.

Na etapie wstępnej analizy problemu postanowiono zmodyfikować układ współrzędnych względem zadanego w założeniach projektowych (obecnie początek układu współrzędnych sceny znajduje się w lewym górnym rogu sceny, z dodatnią osią współrzędnych skierowaną w dół.)

Dodatkowo stworzony został przesuwalny pasek przycisków z podstawowymi opcjami (Pomoc, About, Open, Save, Screenshot, Clear) ułatwiający obsługę programu.

W programie stworzone zostało okno opisu aplikacji pomagające w podstawowej komunikacji z użytkownikiem oraz pozwalające zapoznać się ze sposobem tworzenia grafiki w programie.

Dodatkową opcją jest możliwość dodania na rysunku skalowalnej mapy bitowej.

Program został wyposażony we wszystkie polecenia sprecyzowane w części „Wymagania rozszerzone" karty projektowej, pozwala on na:
- Powiększanie obszaru
- Wyświetlane są współrzędne obrazka względem położenia myszy
- Zaimplementowano metody wspomagające obsługę spisu obiektów (obrót, skala, podświetlenie, usunięcie obiektu)

3. Analiza projektu

a) Specyfikacja danych wejściowych

Podstawowy format danych wejściowych składa się z ciągu poleceń wpisywanych w polu (3) – Linia komend okna interpretera.

Każda z wczytanych komend jest następnie sprawdzana pod względem składniowym czy nie występują w niej błędy (literówki etc.) i analizowana wstępnie do jednego z 16 rodzajów komendy (względem słowa kluczowego, pkt: 3.d).

Po wstępnym zakwalifikowaniu polecenia względem słowa kluczowego następuje proces weryfikacji liczby argumentów, ich formatu i wartości. W momencie gdy algorytm uzna całe polecenie za poprawne semantycznie następuje proces interpretacji polecenia w celu wykonania odpowiedniej aktywności (stworzenie obiektu, modyfikacja istniejącego, usunięcie obiektu, zapis, odczyt).

Dodatkowo istnieje możliwość wprowadzenia danych do programu poprzez wczytanie zestawu komend z pliku.

b) Opis oczekiwanych wyjściowych

Na podstawie danych z podpunktu 3.a powyższego opracowania program tworzy obiekty na scenie i wyświetla je użytkownikowi. Wspomniane elementy są obiektami klasy QGraphicsItem biblioteki Qt odpowiednio wyświetlane względem wartości podanych przez użytkownika. Możliwa jest modyfikacja obrazu poprzez odpowiednie komendy.

Program pozwala na zapis stworzonej już sceny do formatu .png.

Wszystkie wpisane komendy mogą zostać odtworzone po zapisaniu do pliku .ape.

c) Zdefiniowanie struktur danych

Podstawowym kontenerem przetrzymującym utworzone elementy jest list wbudowana w obiekt QGraphicsScene (klasa sceny użytkownika) oraz pomocnicza lista obsługująca wyświetlanie okna (2) – Lista obiektów.

Obiekt QGraphicsScene posiada wbudowaną listę typu QList<QGraphicsItem *> do której to dodawane są wszelkie elementy które winny zostać wyświetlone na obrazie. Operacje na wspomnianej liście pozwalają na modyfikację istniejących obiektów lub ich usunięcie.

Dodatkowo w programie skorzystaliśmy z tożsamej klasy w celu stworzenia Listy obiektów, poszerzonej o możliwość nadania nazwy elementu oraz jego ID.

Do zapisywania poprawnych komend posłużono się obiektem klasy QList<QString *> w którym to przetrzymywane są wszelkie komendy które pomyślnie przeszły proces walidacji. Tak stworzona lista jest następnie zapisywana do pliku, pozwalając na odtworzenie stanu bieżącego aplikacji.

d) Specyfikacja interfejsu użytkownika

Główne okno programu składa się z:

1. Głównego obszaru roboczego (dalej ‘scena’),
2. Listy utworzonych obiektów,
3. Linii komend,
4. Klawisza zatwierdzającego komendę,
5. Przesuwalnego menu podręcznego,
6. Wskaźnika obecnej pozycji myszki na obszarze roboczym.

Powyższy interpreter pozwala na użytkowanie przy użyciu następujących komend:

$$\boldsymbol{range\ x_1\ y_1\ x_2\ y_2} \rightarrow Przypisanie\ lewemu\ dolnemy\ narożnikowi\ obszaru\ roboczego\ współrzędnych\ (x_1\ , y_1), natomiast\ prawemu\ górnemu\ (x_2\ , y_2)$$

$$\boldsymbol{background\ c} \rightarrow Ustalenie\ koloru\ tła\ na\ zadany\ kolor\ 'c'$$

$$\boldsymbol{line\ x_1\ y_1\ x_2\ y_2\ c} \rightarrow Utworzenie\ na\ scenie\ linii\ od\ punktu\ (x_1\ , y_1), do\ punktu\ (x_2\ , y_2), oraz\ kolorze\ obramowania\ 'c'$$

$$\boldsymbol{rectangle\ x_1\ y_1\ x_2\ y_2\ c} \rightarrow\ Utworzenie\ na\ scenie\ prostokątu, o\ lewym\ górnym\ narożniku\ w\ punkcie\ (x_1\ , y_1)\ i\ prawym\ dolnym\ w\ (x_2\ , y_2),$$

$o\ obramowaniu\ w\ kolorze\ 'c'$

$\boldsymbol{circle\ x\ y\ r\ c} \rightarrow Utworzenie\ na\ scenie\ okręgu\ o\ środku\ w\ punkcie\ (x, y)$
$oraz\ promieniu\ r, o\ obramowniu\ w\ kolorze\ 'c'$

$\boldsymbol{ellipse\ x\ y\ r_x\ r_y\ c} \rightarrow Utworzenie\ na\ scenie\ elipsy, o\ środku\ w\ punkcie\ (x, y)$
$oraz\ pionowej\ i\ poziomej\ osi\ odpowiednio\ r_x\ i\ r_y, o\ obramowaniu\ w\ kolorze\ 'c'$

$\boldsymbol{arc\ x\ y\ r_x\ r_y\ b\ e\ c} \rightarrow Utworzenie\ na\ scenie\ łuku, opartego\ na\ elipsie$
$o\ środku\ w\ punkcie\ (x, y) oraz\ pionowej\ i\ poziomej\ osi\ odpowiednio\ r_x\ i\ r_y.$
$Łuk\ zaczyna\ się\ przy\ kącie\ b\ (wyrażony\ w\ stopniach), natomiast\ kończy\ się\ przy\ kącie\ e.$
$Kolor\ obramowania\ definiuje\ 'c'.$

$\boldsymbol{fill\ ID\ c} \rightarrow Wypełnienie\ obiektu\ o\ numerze\ ID\ kolorem\ 'c'$

$\boldsymbol{delete\ ID} \rightarrow Usuwa\ obiekt\ o\ podanym\ ID$

$\boldsymbol{move\ ID\ x\ y} \rightarrow Przesunięcie\ obiektu\ o\ podanym\ ID\ o\ wektor\ [x, y]$

$\boldsymbol{rotate\ ID\ x\ y\ \alpha} \rightarrow Obraca\ obiekt\ o\ wskazanym\ ID\ o\ kąt\ \alpha,$
$względem\ punktu\ o\ współrzędnych\ (x, y)$

$\boldsymbol{show\ ID} \rightarrow Podświetlenie\ obiektu\ o\ podanym\ ID\ na\ scenie$

$\boldsymbol{clear} \rightarrow Usunięcie\ wszystkich\ obiektów\ na\ scenie$

$\boldsymbol{write\ file} \rightarrow Zapisuje\ wszystskie\ obiekty\ wraz\ z\ aktualnymi$
$wartościami\ zakresy\ roboczego\ do\ pliku\ o\ nazwie\ [file].ape$

$\boldsymbol{read\ file} \rightarrow Wczytuje\ wszystkie\ obiekty\ zawarte\ w\ pliku\ o\ nazwie\ [file].ape$

$\boldsymbol{save\ w\ h\ file} \rightarrow Zapisanie\ aktualnego\ obrazka\ w\ postaci\ bitmapy$
$o\ szerokości\ w\ i\ wysokości\ h\ do\ pliku\ o\ nazwie\ [file]$

$\boldsymbol{pixmap\ x\ y\ s\ file} \rightarrow Wprowadzenie\ bitmapy\ z\ pliku\ w$
$punkcie\ o\ współrzędnych\ (x, y)\ i\ skali\ 's'$

$Dodatkowo\ zaimplementowano\ modyfikator\ \boldsymbol{all}\ pozwalający\ na\ wykonanie$
$zadania\ modyfikacji\ (move, rotate, show, delete, fill) na\ wszytskich\ utworzonych$
$uprzednio\ elementach$

e) Wyodrębnienie i zdefiniowanie zadań

W czasie analizy wstępnej projektu zdefiniowano potrzebę realizacji następujących zadań:

- Wybór IDE, oraz bibliotek niezbędnych do realizacji programu
- Stworzenie szkieletu okna (QT Creator), obsługa wskaźnika położenia myszy
- Stworzenie linii komend i analizatora wprowadzonego tekstu
- Walidacja poleceń
- Obsługa poleceń tworzenia obiektów na scenie (polecenia: ***rectangle, line, background, circle, ellipse, arc***)
- Obsługa poleceń modyfikacji obiektów na scenie (polecenia: **fill, delete, move, rotate, show, clear,** wraz z obsługą modyfikatora **all**)

- Obsługa poleceń zapisu, odczytu oraz zrzutu ekranu roboczego
- Testy programu
- Stworzenie dokumentacji projektowej

f) Wybór środowiska programistycznego i bibliotek

Po dogłębnej analizie różnic i korzyści płynących z zastosowania poszczególnych bibliotek języka CPP pozwalających na tworzenie i edycję grafiki (SFML, WxWidgets, Qt) zdecydowo o wyborze bibliotek Qt, wraz z użyciem Qt Creatora jako podstawowego narzędzia do tworzenia, debugowania oraz kompilowania programu projektowego. Biblioteka Qt zawiera wszystkie niezbędne nam klasy pozwalające na realizację założonych celów projektowych. Ponadto ważnym czynnikiem który spowodował wybór tej a nie innej biblioteki był, wspomniany wyżej, Qt Creator który pozwolił na zaoszczędzenie czasu przy tworzeniu podstawowych elementów interfejsu użytkownika.

4. Podział pracy i analiza czasowa

Podział pracy przy projekcie został rozłożony w następujący sposób:
- Stworzenie szkieletu okna (w tym przede wszystkim okna sceny) oraz linii komend. Niezbędniki do dalszego testowania i wprowadzania podstawowych funkcji programu. (1 dzień: Eryk Jarocki, Arkadiusz Michalik)

- Stworzenie analizatora tekstu (1 dzień: Eryk Jarocki, Arkadiusz Michalik)

Równolegle (okres tygodnia):
- Walidacja poleceń (Eryk Jarocki)
- Obsługa poleceń tworzenia obiektów na scenie (Arkadiusz Michalik, Piotr Pasternak)

- Obsługa poleceń modyfikujących istniejące obiekty (2 dni: Eryk Jarocki, Arkadiusz Michalik, Piotr Pasternak)
- Obsługa modyfikatora *all* (1 dzień: Arkadiusz Michalik)

- Obsługa poleceń zapisu, odczytu, zrzutu ekranowego (2 dni: Piotr Pasternak)

Następnie w okresie 3 dni przeprowadzane były testy istniejącego oprogramowania oraz proces poprawy istniejących błędów w pełnym trójosobowym składzie.

- Stworzenie dokumentacji (2 dni: Piotr Pasternak)

5. Opracowanie i opis niezbędnych algorytmów

Ze względu na rodzaj zadania projektowego nie było potrzeby tworzenia specjalnych algorytmów w celu rozwiązania problemów projektowych. Wszystkie algorytmy odpowiedzialne za dodawanie obiektów i rysowanie na scenie są zawartością biblioteki Qt i jako takie mogą zostać sprawdzone bezpośrednio w dokumentacji.

Ze względu na charakter interpretera, który to pozwalał na modyfikacje stworzonych już obiektów warto przybliżyć w jaki sposób w grafice komputerowej obsługiwane są transformacje obiektów przy użyciu macierzy transformacji:

Macierzą transformacji nazywamy zależność pomiędzy współrzędnymi punktu przed i po jego transformacji. Podstawowymi rodzajami transformacji są: translacja (przesunięcie o wektor), rotacja (obrót względem punktu), skalowanie.

Najczęściej macierz taka jest rzędu r+1 niż wymiar wektora współrzędnych.

Dla operacji w trójwymiarowej przestrzeni macierz translacji przedstawia się następująco:

$$TRAN(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}, gdzie$$

$$[a,b,c] \rightarrow wektor\ przesunięcia\ względem\ osi\ X,Y,Z$$

Macierze skalowania w trójwymiarowej przestrzeni:

$$SCALE(S_x,S_y,S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, gdzie$$

$$[S_x,S_y,S_z] \rightarrow współczynniki\ skalowania\ względem\ osi\ X,Y,Z$$

Macierze rotacji dla przestrzeni trójwymiarowej (ze względu na prezentacje są one przedstawione jako trzy osobne macierze):

$Oś\ X$:

$$RotX(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$Oś\ Y$:

$$RotY(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta & 0 \\ 0 & 0 & 0 & 0 \\ -sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$Oś\ Z$: $$RotZ(\gamma) = \begin{bmatrix} cos\gamma & -sin\gamma & 0 & 0 \\ sin\gamma & cos\gamma & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

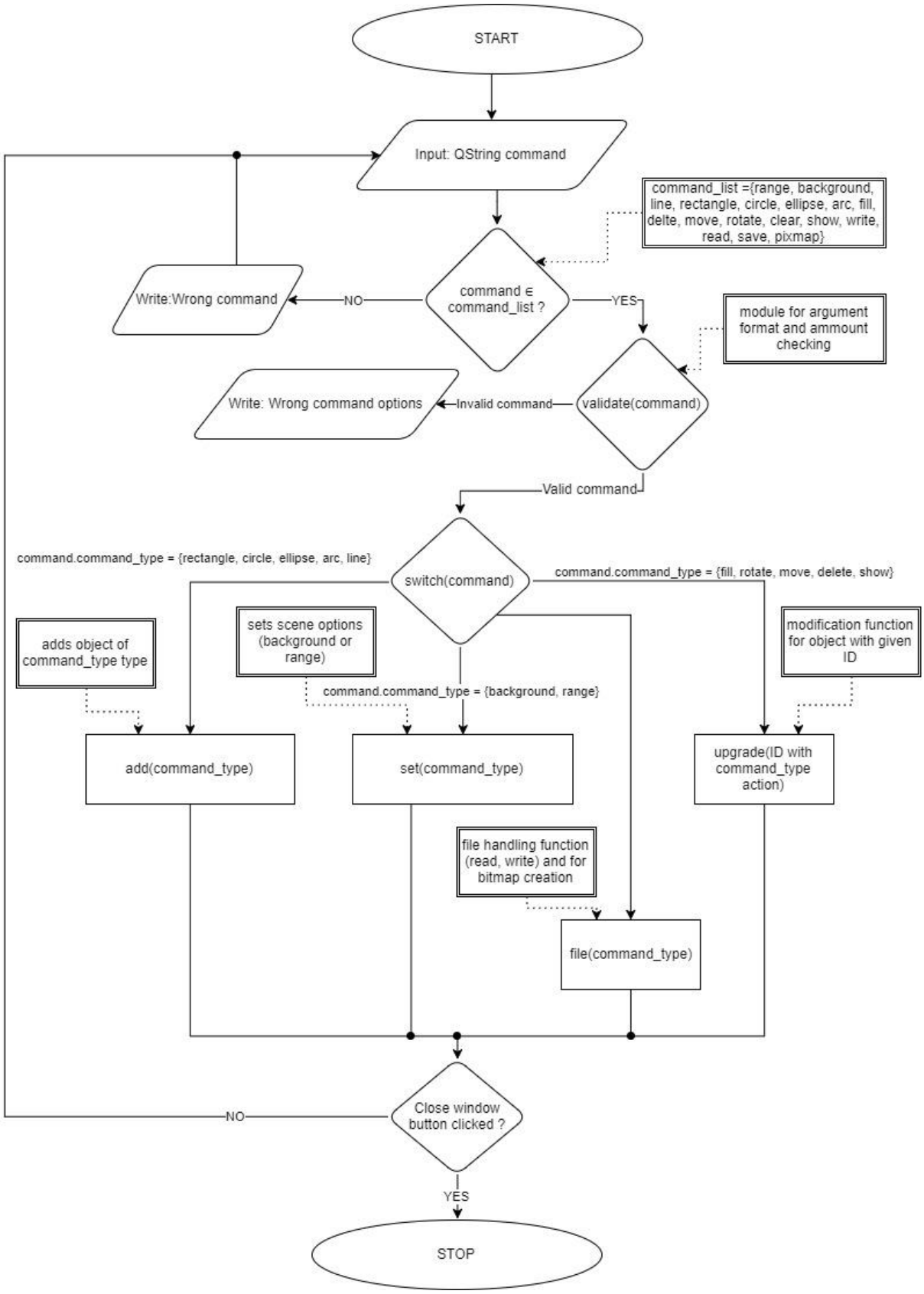
W powyższych $[\alpha, \beta, \gamma] \rightarrow kąty\ obrotu\ względem\ osi\ X, Y, Z$

Dodatkowo ze względu na częstą potrzebę zastosowania wielu transformacji na jednym obiekcie (punkcie) należy również wspomnieć o możliwości stworzeniu wynikowego wektora punktów:

Przykład w którym najpierw stosujemy operację skalowania a następnie translacji o wektor:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$, jak widać na załączonym przykładzie w momencie zaistnienia potrzeby dodania kolejnej transformacji, macierz transformacji takiej operacji jest mnożona (dokładana) z lewej strony iloczynu macierzy.

6. Kodowanie



Rysunek 2. Uproszczony schemat blokowy programu.

Opis funkcji stworzonych w programie:

$$void\ MainWindow::rotateSceneItem(QGraphicsItem*\ item, int\ x, int\ y, int\ angle)$$

Metoda klasy MainWindow, której celem jest rotacja obiektu typu QGraphicsItem na scenie wokół punktu (x,y) o zadany kąt, nie zwracająca żadnej wartości. Wykorzystuje funkcje wbudowane w bibliotekę Qt pozwalające na ustawienie kąta dla zadanego obiektu.

$$void\ MainWindow::return\_pen\left(QGraphicsRectItem*\ rect, QPen\ tmp_{pen}\right)$$

Metoda klasy MainWindow, której celem jest ustawienie pióra odpowiedzialnego za sposób rysowania linii i kontur obiektu QGraphicsRectItem. Metoda ta nie zwraca żadnej wartości.

$$void\ MainWindow::on\_myPushButton\_clicked()$$

Metoda klasy MainWindow, która nie posiada żadnych argumentów i jest wywoływana w trakcie naciśnięcia klawisza „Enter". Jest ona jedną z najważniejszych funkcji programu gdyż odpowiada za obsługę linii komend. Zawarty w niej switch wyłapuje komendę po czym następuję dodawanie obiektu do naszego programu lub obsługę już istniejących obiektów poprzez ich modyfikację. Nie zawraca żadnej wartości.

$$void\ MainWindow::resizeEvent(QResizeEvent\ *)$$

Metoda klasy MainWindow, odpowiedzialna za zmianę rozmiarów okna zachowując odpowiednie dopasowanie obiektów w obszarze roboczym, jej argumentem jest wskaźnik na QResizeEvent (listener eventu skalowania okna), nie zwraca żadnej wartości.

$$bool\ is\_hex\_notation(std::string\ const\&\ s)$$

Funkcja sprawdzająca występowanie notacji szesnastkowej w zadanym stringu, pobierająca referencję na std::string i zwracająca typ bool czyli prawdę lub fałsz w odpowiednim przypadku.

$$void\ MainWindow::on\_myClearScreenButton\_clicked()$$

Metoda klasy MainWindow, której celem jest obsługa przycisku czyszczenia sceny poprzez wyczyszczenie obszaru roboczego z wszystkich obiektów oraz usunięcie ich z listy. Nie pobiera żadnego argumentu oraz nie zwraca żadnej wartości.

$$void\ MainWindow::on\_mySaveToFileButton\_clicked()$$

Metoda klasy MainWindow, które zapisuje zrzut ekranu do pliku w postaci .png. Nie pobiera argumentu, jak i również nie zwraca żadnej wartości.

$$void\ MainWindow::ProvideContextMenu(const\ QPoint\ \&pos)$$

Metoda klasy MaindWindow, która pozwala na utworzenie menu po kliknięci prawym przyciskiem myszy na obiekt w liście obiektów. Uzyskane menu pozwala na dostęp do prostych funkcji na obiektach w tym: rotacje, wypełnienie, usunięcie obiektu oraz zmianę skali w przypadku pixmapy. Pobiera ona referencję na QPoint (w celu ustalenia który element został wybrany), natomiast nie zwraca żadnej wartości.

$$void\ MainWindow::on\_myListWidget\_itemClicked(QListWidgetItem\ *\ item)$$

Metoda klasy MainWindow, powodująca podświetlenie obiektu na kolor złoty przez 0.5s po naciśnięciu lewego przycisku myszy na obiekt w liście Widgetów. Pobiera ona wskaźnik na QListWidgetItem oraz nie zwraca żadnej wartości.

$$void\ MainWindow::mouseMoveEvent(QMouseEvent\ *\ event)$$

Metoda klasy MainWindow, odpowiedzialna za wskazywanie obecnej pozycji myszki na obszarze roboczym. Pobierająca wskaźnik na QMouseEvent (listener obsługi myszki) oraz nie zwracająca żadnej wartości.

$$void\ MainWindow::on\_actionFaqButton\_triggered()$$

Metoda klasy MainWindow, odpowiedzialna za obsługę przycisku „FAQ", powoduje otwarcie okna z opisem podstawowych poleceń programu, nie zwraca wartości

$$void\ MainWindow::on\_actionExplainButton\_triggered()$$

Metoda klasy MainWindow, odpowiedzialna za obsługę przycisku otwierającego okno „Podstawowych informacji o programie", zawierającego zdjęcia programu oraz opis przestrzeni roboczej, tworzy objekt typu MyTabWidget, nie zwraca żadnej wartości.

Klasa $MyTabWidget$ → obsługa nowego okna po wykonaniu metody MainWindow::on_actionExplainButton_triggered(), tworzy nowe okno z opisem interfejsu użytkownika z użyciem bitmap.

Klasa $MyArcObject$ → klasa obsługująca rysowanie na scenie obiektu typu „łuk". Konstruktor klasy przyjmuje wartości całkowite: (x, y) położenia początkowego łuku, wartości całkowite półosi wygięcia (r_x,r_y), oraz wartości całkowite kątów: (α,β) początkowego oraz końcowego. Klasa ta wykorzystuje narzędzie QPainter biblioteki Qt w celu narysowania obiektu na scenie.

7. Testowanie

Testowanie gotowego programu zostało podzielone na następujące segmenty:
a) Testowanie interfejsu użytkownika
b) Testowanie tworzenia obiektów sceny
c) Testy walidatora poleceń interpretera
d) Testowanie poleceń modyfikacji obiektów
e) Testy podstawowych operacji wejścia- wyjścia obsługi plików (zapis bieżącego stanu, zrzuty ekranowe)

ad. a) Testy interfejsu użytkownika polegały na modyfikacji wielkości okna oraz nieustannym śledzeniu wielkości poszczególnych elementów programu pod względem ich wyglądu, rozmieszczenia oraz skali. Na tym etapie podjęto decyzję o utworzeniu dodatkowego rozwiązania przesuwającego się zasobnika z podstawowymi użytkowalnościami programu (elem. 5 Interfejsu użytkownika). Etap ten został połączony ze szczegółowymi testami menu elementu 2. – Listy utworzonych obiektów. Przeprowadzono sprawdzenie podstawowych funkcji listy (podświetlenie obiektu, skalowanie, usunięcie, zmiana koloru) dla każdego rodzaju obiektu. Na tym etapie wykryto problem zwracania złych wartości położenia kursora myszki względem sceny i został on natychmiastowo naprawiony.

ad. b) Testowanie tworzenia obiektów sceny polegało na wprowadzeniu pełnych zestawów komend do interpretera dla różnych wartości testowych. Testy były prowadzone dla wartości granicznych mogących powodować nieścisłości w działaniu programu.
Przykładowe polecenia użyte w procedurze testowej to:

rectangle 100 100 50 50 red; / Wartości x2 y2 mniejsze od x1 y1
rectangle 100 100 100 100 red; / Wartości położeń wierzchołków równe sobie
circle 50 50 -20 red; / Ujemny promień okręgu
ellipse 200 200 -20 -30 black; / Ujemne mimośrody elipsy
line -100 -100 -50 -50 black; / Tworzenie obiektu poza obszarem roboczym
arc 400 400 20 20 -20 -40; / Ujemne wartości kątów dla łuku
rectangle 400 400 500 500 121212; / Podanie koloru w zapisie heksadecymalnym bez '#'

Po analizie powyższych przypadków i im pokrewnych zdecydowano się na wprowadzenie mechanizmu walidacji wpisywanych komend. Należy zauważyć, że wszystkie powyższe komendy ostatecznie są traktowane jako poprawne (łącznie z ujemnymi wartościami promienia które powodują rysowanie obiektu w przeciwnym kierunku, niż wartość wskazywana przez zmienną promienia.)

Na powyższym etapie zdecydowano się przeprogramować sposób rysowania łuku ze względu na problemy z wyświetlaniem w/w dla pewnych wartości. Stworzona została osobna klasa MyArcObject pozwalająca na tworzenie oraz modyfikację obiektów łukowych.

ad. c) Testy walidatora polegały w głównej mierze na podawaniu danych w niepoprawnym formacie i/lub zbyt dużej/małej liczby argumentów.
Przykładowe polecenia procedury testowej:

rectangle 100 100 50 red; / brak współrzędnej
rectangle 50 50 100 100; / brak definicji koloru
rectangle a 50 50 50 green; / niepoprawny format współrzędnej
delete [indeks ostatniego elementu +1]; / próba usunięcia nieistniejącego jeszcze obiektu

Dla wszystkich powyższych błędów składniowych komend, program został wyposażony w system walidacji która wypisuje komunikat o błędzie. Należy zauważyć, że ze względu na możliwość definicji koloru wypełnienia (bądź obramowania) za pomocą notacji heksadecymalnej jak i również przy wykorzystaniu angielskich nazw kolorów, zdecydowano się o wprowadzeniu domyślnego koloru czarnego dla niepoprawnych danych zmiennej 'c' -> koloru.

Ad d.) Testowanie poleceń modyfikacji obiektów polegało na stworzeniu pojedynczego egzemplarza każdego rodzaju obiektu, a następnie wykonaniu na nim predefiniowanych i dozwolonych dla danego typu obiektu, operacji.

Przykładowe dane testowe:

fill 1 azure; / Gdzie obiekt 1 był łukiem
show 1; / Podświetlenie obiektu o ID 1.
rotate 2 200 200 20; / Rotacja okręgu względem jego środka
rotate 2 400 300 30 ; / Roracja okręgu względem punktu, który nie jest jego środkiem

Ważnym elementem powyższych testów było sprawdzenie zachowania obiektów przy wykonaniu paru różnych rodzajów transformacji (rotacji, translacji, skalowania), np.:

rectangle 400 400 500 500 red; / Utworzenie prostokąta o ID 1.
move 1 30 30; / Przesunięcie powyższego o wektor [30;30]
rotate 1 450 450 45; / Obrót prostokąta o 45° względem swojego środka
move 1 -20 -20; / Przesunięcie obróconego już prostokąta o wektor [-20;-20]
rotate 1 300 300 20; / Ponowny obrót obiektu o kąt 20° względem punktu (300; 300)

Po wykonaniu powyższych instrukcji natrafiono na problem związany z aplikowaniem wielu operacji transformacji na jednym obiekcie. Po dogłębnej analizie problemu związanego z translacją punktów sceny do układu współrzędnych obiektu problem został zidentyfikowany i rozwiązany.

Ad e.) Testowanie podstawowych operacji wejścia-wyjścia polegało głownie na analizie działania komend ***save, write, read*** oraz przycisków menu podręcznego odpowiedzialnych za powyższe operacje.
Przykładowe komendy:

write stan; / Utworzenie pliku stan.ape z aktualnym stanem programu
read stan; / Wczytanie uprzednio utworzonego pliku stan.ape
save 400 500 obrazek.png; / Utworzenie zrzutu ekranu do pliku o nazwie ‘obrazek.png’ o rozdzielczości 400 500

Powyższe testy wykazały problem związany ze skalowaniem zrzutu ekranu do wartości podanych przez użytkownika. Problem polegający na potrzebie ustawienia odpowiednich opcji obiektu QPrinter został zidentyfikowany i rozwiązany przy użyciu dokumentacji biblioteki Qt.

Przy wykonywaniu testów posłużono się narzędziem biblioteki Qt o nazwie qDebug(), które na bieżąco pozwalało monitorować wartości przyjmowane przez zmienne utworzonych obiektów sceny.

8. Wdrożenie raport i wnioski

Po przeprowadzeniu procedury testowej program (wraz z instrukcją użytkowania) został udostępniony osobom zainteresowanym (członkowie rodzin), w celu zapoznania się z opiniami na temat interfejsu użytkownika i doświadczeniach płynących z obsługi programu.

Raport dotyczący wykonania:

Z punktów wyszczególnionych w karcie projektowej udało się zaprojektować następujące:
- Stworzenie trzyczęściowego okna programu
- Integracja obszaru sceny z obszarem listy utworzonych obiektów
- Skalowanie okna
- Komendy: ***range, background, line, rectangle, circle, ellipse, arc, fill, delete, move, rotate, show, clear, write, read, save***
- Modyfikator all wraz z wyjątkami (brak możliwości wypełniania obiektów nie posiadających pola)
- Stworzenie komendy do wprowadzania bitmapy z pliku (***pixmap***)
- Wyświetlanie współrzędnych kursora we współrzędnych obrazka
- Obsługa dodatkowych funkcjonalności listy obietków (podświetlenie, podręczne menu z możliwością usunięcia, wypełniania czy obrotu obiektów)

W dalszych etapach rozwoju projektu planuje się:

- Usprawnienie działania komendy ***range*** pozwalającej na zmianę współrzędnych środka układu współrzędnych.
- Wprowadzenie możliwości wypełnienia obiektu teksturą. Na obecnym etapie nie udało się uzyskać zadowalających rezultatów, które pozwoliłyby na dodanie w/w funkcjonalności do ostatecznej wersji projektu.
- Modyfikacja kodu programu w celu zapewniania większej czytelności (OOP).