INTERPRETER GRAFIKII WEKTOROWEJ

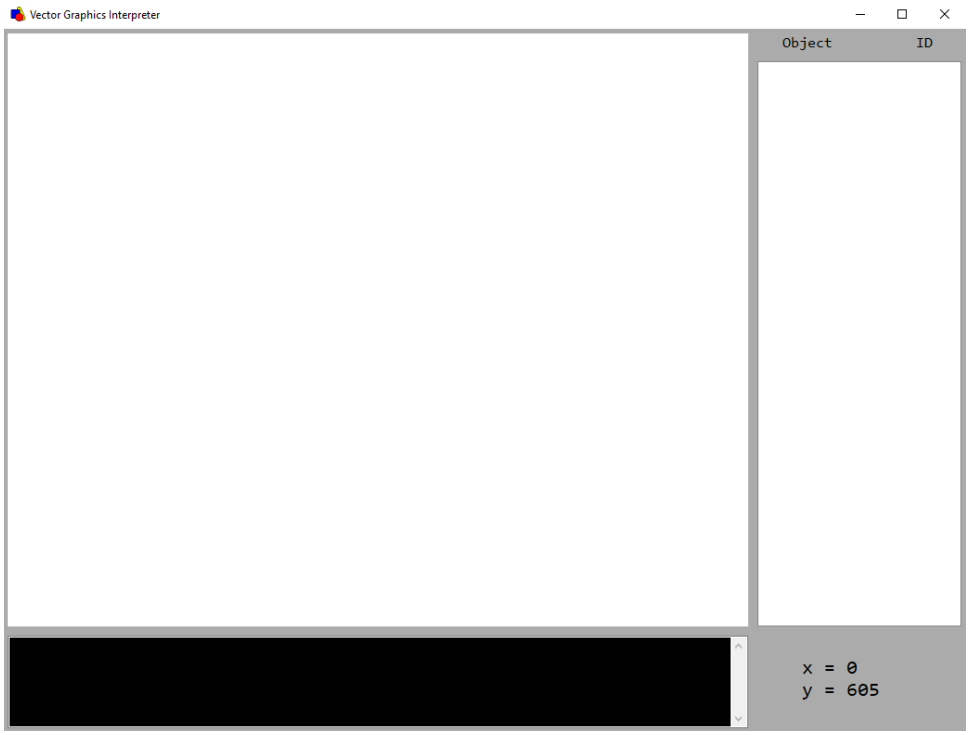
Projekt nr 38

Adam Łaba
Bartosz Balawender
Jan Wojdylak

15.06.2021

1. Opis projektu

W ramach projektu stworzyliśmy program będący interpreterem rysunków wektorowych. Główne okno składa się z trzech części, największą część stanowi panel, na którym rysujemy kształty, po prawej stronie znajduje się obszar, na którym wypisywane są informacje o kształtach, które zostały wyrysowane. W dolnej części znajduje się konsola, do której wpisujemy odpowiednie komendy, a w prawej dolnej wypisujemy aktualne położenie kursora myszki.



Rysunek 1. Główne okno program

2. Założenia wstępne przyjęte w realizacji projektu

Pierwszym i podstawowym założeniem było wykonanie projektu korzystając z biblioteki wxWidgets, która umożliwiła nam stworzenie głównego okna korzystając z wxBuilder.

Kolejnym założeniem było zwrócenie uwagi, aby operacje na obiektach nie pociągały ze sobą migotania całego obszaru roboczego oraz aby zmiana rozmiaru głównego okna wpływała jedynie na zmianę rozmiaru obszaru roboczego.

3. Analiza projektu

3.1.Specyfikacja danych wejściowych

1) Dane wejściowe naszego programu powinny składać się z ciągu poleceń wpisywanych "w czarnej" dolnej części naszego okna na "Rysunku 1"

2) Następnie każda wpisana do konsoli komenda jest sprawdzana pod względem błędów składniowych i literówek. Program sprawdza również czy w komendzie została podana poprawna liczba argumentów

3) Jeżeli podanie polecenie zostanie uznane za poprawne (w linii komend nie pojawią się żadne komunikaty świadczące o nieprawidłowościach w podaniu komendy) następuje interpretacja polecenia. (tworzenie obiektu, modyfikacja istniejącego obiektu, usunięcie obiektu, zapis oraz odczyt)

4)Komendy mogą zostać wprowadzone poprzez wczytanie uprzednio zapisanych komend z pliku tekstowego

3.2.Opis oczekiwanych danych wyjściowych

Oczekujemy, że nasz program będzie tworzył obiekty i wyświetlał je użytkownikowi na scenie. Obiektami nazywam kształty, które reprezentują grafikę wektorową w 2D (elipsę, łuk, koło, prostokąt oraz linie). Narysowany obraz można modyfikować poprzez użycie odpowiednich komend. Chcemy również, aby utworzony obraz poprzez użycie odpowiedniej komendy zapisywał się do bitmapy oraz o zapis do pliku .txt wszystkich użytych do tej pory komend.

3.3.Zdefiniowanie struktur danych

1) vector - wszystkie wyrysowywane obiekty przechowywane są w wektorze – "std::vector". Wektor naszych kształtów jest odpowiednio modyfikowany w zależności od zmian jakie wprowadzimy na danych obiektach. (np. komendy: rotate, move)

2) Point – w naszym projekcie utworzyliśmy klasę Point do przechowywania punktu o współrzędnych (x, y)

3.4.Specyfikacja interfejsu użytkownika

Główne okno naszego programu składa się z:

- Obszaru roboczego, który umożliwia wyświetlanie rysowanych obiektów 2D
- Czarny obszar znajdujący się na samym dole to nasza konsola (linia komend). Konsola pozwala nam na wpisanie następujących komend:
    - **range x1 y1 x2 y2 –** komenda ta zmienia współrzędne obszaru roboczego, przypisuje lewemu dolnemu narożnikowi współrzędne (x1, y1), natomiast prawemu górnego (x2, y2)
    - **background c** - zmienia kolor tła na c
    - **line x1 y1 x2 y2 c** - rysuje linię od punktu (x1, y1) do punktu (x2, y2) i o zadanym kolorze c
    - **rectangle x1 y1 x2 y2 c** - rysuje prostokąt, którego lewy dolny narożnik znajduje się w punkcie (x1, y1), a prawy górny w punkcie (x2, y2), c jest jego kolorem obramowania
    - **circle x y r c** - rysuje okrąg o środku w punkcie (x, y) i promieniu r i obramowaniu o kolorze c
    - **ellipse x y rx ry c** - rysuje elipsę o środku w punkcie (x, y) oraz poziomej i pionowej półosi równej rx i ry i obramowaniu o kolorze c
    - **arc x y rx ry b e** - rysuje fragment łuku opartego na elipsie o środku w punkcie (x, y) i półosiach rx i ry. Łuk rozpoczyna się przy kącie b (wyrażonym w stopniach i liczonym przeciwnie do ruchu wskazówek zegara) a kończy przy kącie e, o kolorze c
    - **fill id c** - wypełnia obiekt o identyfikatorze id kolorem c
    - **delete id –** usuwa obiekt o identyfikatorze id
    - **move id x y –** przesuwa obiekt o identyfikatorze id o wektor (x, y)
    - **rotate id x y a –** obraca obiekt o identyfikatorze id o kąt a (wyrażony w stopniach, w kierunku przeciwnym do ruchu wskazówek zegara) wokół punktu o współrzędnych (x, y)
    - **show id - z**aznacza na krótko (np. pół sekundy) obiekt o identyfikatorze id.
    - **clear -** usuwa wszystkie obiekty i wykonuje komendę: range 0 0 1 1
    - **write –** zapisuje obecny stan panelu do pliku
    - **clearcmd -** czyści konsolę i przenosi użytkownika do nowej linii
    - **read file -** wczytuje wszystkie obiekty zawarte w pliku file oraz ustawia zakres roboczy na wartości z pliku, który wybierzemy z dysku
    - **save w h** - zapisuje aktualny obrazek w postaci bitmapy o szerokości w i wysokości h do pliku graficznego wybranego przez nas z dysku

W każdej z powyższych komend użycie słówka “all” zamiast identyfikatora “id” spowoduje wykonanie danej komendy na wszystkich obiektach, na których jest to możliwe. Przekazanie koloru przy tworzeniu figur jest opcjonalne, domyślnie program przyjmuje czarny jako kolor obramowania i przeźroczyste wypełnienie.

- Po prawej stronie wyświetlana jest lista obiektów (Obiekt + jego ID)
- W prawym dolnym rogu można odczytać współrzędne na jakich obecnie znajduję się myszka na obszarze roboczym

3.5.Wyodrębnienie i zdefiniowanie zadań

Po wstępnej analizie projektu i omówieniu początkowych celów zdefiniowaliśmy potrzebę realizacji następujących zadań:

1) Wybór niezbędnych bibliotek potrzebnych do realizacji projektu
2) Stworzenie szkieletu głównego okna programu, obsługa wskaźnika położenia myszy, linii komend

3) Walidacja poleceń

4) Stworzenie klas obiektów kształtów oraz implementacja poszczególnych funkcji odpowiedzialnych za wyrysowanie kształtów

5) Obsługa poleceń służących do modyfikacji i usuwania wyrysowanych obiektów

6) Obsługa poleceń zapisuj i odczytu dla pliku tekstowego oraz zapis do bitmapy (.bmp) oraz dialogów dla każdego obiektu

7) Sprawdzenie poprawności działania programu

8) Ostatnie poprawki w projekcie

9) Napisanie dokumentacji projektowej

3.6.Wybór środowiska programistycznego i bibliotek

Nasz projekt został napisany w języku C++ w oparciu o bibliotekę wxWidgets w środowisku Windows. Projekt był kompilowany przy użyciu programu Microsoft Visual Studio. Dodatkowo, w celu ułatwienia współpracy z biblioteką wxWidgets korzystaliśmy z programu WxFormBuilder.

4. Podział pracy i analiza czasowa

Praca nad projektem trwała około 4 tygodnie, spotkania odbywały się co tydzień w celu przedyskutowania zrealizowanych celów, omówienia błędów i innych problemów oraz ustalenia celów na kolejny tydzień.

Tydzień 1.

- Omówienie początkowych celów
- Zaprojektowanie hierarchii klas i ustalenie założeń dla każdej z nich
- Stworzenie szkieletu głównego okna programu w aplikacji wxBuilder (Adam)

Tydzień 2.

- Napisanie klas: Panel, Shape oraz Point (Janek)
- Implementacja klas: Line, Rectangle (Bartek)
- Zrealizowanie okna konsoli oraz walidacja danych wejściowych (Adam)
- Implementacja komend wyrysowujących kolejne obiekty

Tydzień 3.

- Napisanie klas: Ellipse oraz Circle (Bartek)
- Implementacja komend odpowiedzialnych za modyfikacje obiektu oraz komendę range (Janek i Bartek)
- Stworzenie i implementacja dialogów (Adam)

Tydzień 4.

- Napisanie komend zapisujących i wczytujących obiekty (Janek i Adam)
- Zapis do bitmapy oraz rozszerzenie funkcji klas dialogów (Adam)
- Usuwanie błędów związanych z rysowaniem obiektów (Bartek)
- Testowanie i wykonywanie ostatecznych poprawek
- Dokumentacja

5. Opracowanie i opis niezbędnych algorytmów

Głównym celem było poprawne zaimplementowanie algorytmów służących do modyfikacji wyrysowanego obiektu.

W przypadku rotacji o kąt α skorzystaliśmy z równań opisujące nowe położenie punktu (x,y):

$$x' = x * \cos(\alpha) - y * \sin(\alpha)$$
$$y' = y * \cos(\alpha) + x * \sin(\alpha)$$

Skalowanie odbywa się poprzez przemnożenie współrzędnych przez ustalone wartości Sx i Sy

$$x' = x * Sx$$
$$y' = y * Sy$$

Skalowanie wykorzystywane jest do aktualizacji położenia w przypadku zmiany rozmiaru głównego okna lub podczas wykonywania komendy range, wartości Sx i Sy ustalane są na podstawie wzoru

$$Sx = w_1 / w_2$$
$$Sy = h_1 / h_2$$

Gdzie w_1 i h_1 to szerokość i wysokość głównego okna, a w_2 i h_2 to szerokość i wysokość obiektu klasy panel.

Jedną z operacji jakie możemy wykonać na obiekcie jest przesunięcie go o wektor, zgodnie ze wzorem:

$$x' = x + Tx$$
$$y' = y + Ty$$

Proces rysowania obiektów polega na wykonaniu operacji rotacji, przesunięcia oraz skalowania dla każdego punktu obiektu. Dla zmodyfikowanych punktów wywołujemy metody dostępne w bibliotece wxWidgets, dla linii, prostokąta i koła są to odpowiednie metody DrawLine, DrawPolygon, DrawCircle.

W przypadku elipsy i łuku nie mieliśmy możliwości skorzystania z gotowych funkcji. Elipsa oraz fragment łuku tworzone są ze zbioru punktów, który tworzy krzywą. Kolejne punkty wyliczane są z równań:

$$x = x_0 + w * \cos(\alpha)$$
$$y = y_0 + h * \sin(\alpha)$$

Gdzie:
x_0, y_0- współrzędne środka elipsy
w, h - szerokość i wysokość elipsy

Łuk jest rysowany podobnie jak elipsa z jedną różnicą, że łuk przyjmuje wartości kąta początku i końca łuku, w przypadku elipsy jest to odpowiednio 0 i 360 stopni.

6. Kodowanie

W naszym projekcie rozdzieliśmy wszystkie obiekty na osobne klasy, a wszystkie klasy zostały opisane w osobnych plikach. Podstawą do rysowania obiektów są klasy Shape oraz Panel. Pierwsza z nich jest klasą podstawową dla każdego z kształtów, które mamy zamiar wyrysowywać. Druga natomiast odpowiedzialna jest za przetrzymywanie aktualnych współrzędnych naszego Panelu roboczego.

Szczegółowa dokumentacja w formacie Doxygen umieszczona jest w kodzie projektu. Poniżej przestawione zostaną najważniejsze zmienne i metody utworzonych przez nas klas.

Klasa Panel

Jest to klasa, w której zawarte są aktualne współrzędne naszego obszaru roboczego oraz kolor tła Zmienne znajdujące się w klasie Panel:

- ***Point*** *m_leftDownPoint* – zmienna przechowywująca współrzędne (x, y) lewego dolnego rogu panelu
- ***Point*** *m_rightUpPoint* – zmienna przechowywująca współrzędne (x, y) prawego górnego rogu panelu
- ***wxColour*** *m_backgroundColor* – zmienna przechowywująca kolor tła, domyślnie przyjmuje wartość **wxColour(255, 255, 255)**

Metody klasy Panel:

- ***Panel****(double x1 = 0, double y1 = 0, double x2 = 799.0, double y2 = 641.0)* - konstruktor ustawiający początkowe współrzędne naszego panelu
- ***void*** *updateCoordinates(double x1, double y1, double x2, double y2)* jest to metoda aktualizująca współrzędne naszego Panelu na te podane jako parametry funkcji, używana podczas wywoływania komendy **range x1 y1 x1 y2**

Klasa Shape

Jest to klasa bazowa, na której opiera się rysowanie obiektów. Zmienne klasy Shape przechowują:

- ***int*** *m_id* – zmienna przechowująca ID obiektu
- ***double*** *m_transformX = 0* – zmienna przechowująca współrzędną x wektora przesunięcia obiektu
- ***double*** *m_transformY = 0* – zmienna przechowująca współrzędną y wektora przesunięcia obiektu
- ***double*** *m_rotateX =0* – zmienna przechowująca współrzędną x punktu, wokół którego obracamy nasz obiekt
- ***double*** *m_rotateY = 0* – zmienna przechowująca współrzędną y punktu, wokół którego obracamy nasz obiekt
- ***double*** *m_rotateAngle = 0* – zmienna przechowująca kąt o jaki obiekt powinien zostać obrócony
- ***wxColor*** *m_color* – zmienna przechowująca kolor obramowania obiektu
- ***wxColor*** *m_fillColor* – zmienna przechowująca kolor wypełnienia obiektu
- ***std::string*** *m_name* – zmienna przechowująca nazwę obiektu

W Shape znajdują się metody wirtualne dziedziczone przez klasy pochodne (Line, Rectangle, Circle, Ellipse oraz Arc). Najważniejszą z metod jest:

- ***void*** *virtual draw(wxBufferedDC * dc, double w, double h, Panel panel)* - jest to metoda wirtualna, która później jest przeciążana w każdej z klas reprezentujących rysowane kształty. Jako parametru funkcji poza "dc" podajemy również rozmiary okna aplikacji w(width) i h(heigth) potrzebne do skalowania punktu oraz panel, aby mieć dostęp do aktualnych współrzędnych obszaru roboczego (transformacja punktu względem współrzędnych lewego dolnego rogu panelu).
  Ponieważ obiekty domyślnie rysowały się "od góry do dołu" (punkty były transformowane względem lewego górnego rogu obszaru roboczego). Aby uzyskać zamierzony efekt podczas rysowania (rysowanie obiektów transformując względem lewego dolnego rogu) każdego z kształtów zmienialiśmy każdą współrzędną y punktu (x, y) na wyrażenie (x, h-1-y). Taka zmiana pozwoliła nam na wyrysowanie obiektów zgodnie z założeniami

Klasy pochodne służące do rysowania kształtów

Każdy obiekt do wyrysowania przedstawiony w osobnej klasie. Utworzyliśmy klasy: ShapeLine, ShapeRectangle, ShapeCircle, ShapeEllipse oraz ShapeArc. Wszystkie klasy mają przeładowaną metodę ***void*** *virtual draw(wxBufferedDC * dc, double w, double h, Panel panel)* dziedziczoną po klasie Shape oraz metode:

- ***std::string*** *getParameters()* - jest to metoda używana w zapisie do pliku (wywołanie komendy *write*) zwracająca nazwę funkcji, parametry potrzebne do wyrysowania obiektu oraz kolor jego obramowania

Klasy zawarte w katalogu CommandValidator

W danym katalogu zawierają się klasy których zadaniem jest walidacja danych wprowadzanych przez komendę przez użytkownika. W przypadku wprowadzenia złych danych na konsolę zostaje wypisany błąd informujący użytkownika co zrobił nie tak. Dodatkowo, co może nie być oczywiste, sprawdzane są zależności danych punktów (np. dla prostokąta $x_1 < x_2$, $y_1 < y_2$) oraz sprawdzane jest, czy obiekt w całości znajduje się w panelu.

Naszą główną klasą zarządzającą działanie całego programu jest klasa VectorGraphicsInterpreterGUI. Znajdująca się w niej metoda consoleOnTextEnter zajmuje się przetworzeniem danych wejściowych oraz wywołaniem odpowiednich metod, które odpowiadają komendom wpisanym przez użytkownika. Poniżej krótko opisane zostało kodowanie każdej z metod.

commandRange() – ustawia rozmiar obiektu klasy Panel

commandBackground() - ustawia kolor tła obiektu klasy Panel

commandLine() - tworzony jest nowy obiekt typu ogólnego Shape na podstawie parametrów zwróconych przez obiekt m_commandValidator, linii zostaje przypisana nazwa oraz zostaje dodana do wektora m_shapes

Pozostałe metody zajmujące się rysowaniem kolejnych kształtów zajmują się analogicznie jest metoda commandLine()

commandFill() - przyjmuje parametry z m_commandValidator, sprawdza, czy komenda została wywołana dla każdego obiektu czy tylko jednego oraz czy można wypełnić dany obiekt i w zależności od tego wywołuje metodę na obiekcie typu Shape setFillColour()

CommnadDelete() – usuwa obiekty z wektora m_shapes w przypadku, gdy w komendzie wystąpiło slowo all lub szuka obiektu o danym id i go usuwa

commandMove() – wywołuje metodę z klasy Shape transform() z parametrami przekazanymi od m_commandValidator, przekazane parametry odnoszą się do stanu początkowego linii za każdym razem, kiedy komenda jest używana

commandRotate() – wywołuje metodę z klasy Shape rotate() z parametrami przekazanymi od m_commandValidator, podobnie jak w komendzie move przekazane parametry cały czas odnoszą się do stanu początkowego

commandShow() - podświetla na krótki czas (0.5 s) obiekt o danym id/wszystkie obiekty. Podświetlenie opiera się na przekolorowani zewnętrznego koloru kształtu na czerwień oraz na rozjaśnieniu koloru wypełnienia na dany czas

commandClear() - usuwa wszystkie obiekty z wektora kształtów, czyści listę obiektów oraz ustawia współrzędna panelu na 0 0 1 1

commandWrite() - tworzymy plik o własnym rozszerzeniu .xyz, w którym zapisujemy informacje o każdym obiekcie z wektora m_shapes w postaci komendy wpisywanej przez użytkownika. Na początku pliku zapisujemy również komendę clear, żeby wyczyścić obszar roboczy oraz komendę range z wartościami –5000 –5000 5000 5000, aby móc zmieścić wszystkie obiekty w tym zakresie. Po zapisaniu parametrów każdego kształtu, dopisujemy komendy rotate i move z wartościami, którymi modyfikowaliśmy dany obiekt. Na koniec pliku zapisujemy komendę range z aktualnymi wartościami panelu.

commandRead() - wczytujemy kolejne linie z pliku .xyz i wywołuje odpowiednie komendy tak jakby były wpisane przez użytkownika w konsoli

commandSave() - ustawiamy wartość szerokości i wysokości bitmapy na podstawie parametrów zwróconych przez commandValidator. Działanie opiera się na stworzeniu tymczasowej bitmapy o danych wymiarach (w pikselach), naniesieniu na niej wszystkie kształty oraz zapisaniu jej do bliku graficznego.

Klasy dialogów

Każdy kształt posiada swoje własne, unikatowe dla siebie okno dialogowe. Pozwalają one na edytowanie właściwości danego kształtu (poprzez zamianę wartości w polu tekstowym i zatwierdzenie enterem) jednocześnie sprawdzając, czy dane są poprawne. Tutaj użytkownik ma większą swobodę - np. nic nie stoi na przeszkodzie, żeby przypisać obiektowi takie wartości, że jego współrzędne znajdą się poza panelem.

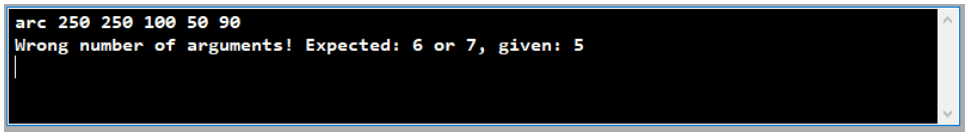
7. Testowanie

W raz z rozwojem projektu i dodawaniem nowych funkcjonalności testowaliśmy aplikację. Pierwszym etapem było sprawdzenie poprawności walidacji komend, następnie rysowanie i modyfikowanie figur. Kolejnym etapem było testowanie funkcji zapisu i odczytu z plików, a na końcu sprawdzenie działania dialogów dla każdego obiektu. Ostatecznie wszystkie testy spełniały nasze pierwotne oczekiwania.

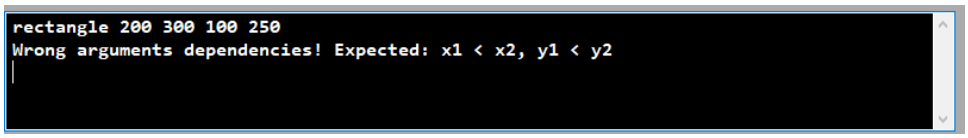
Walidację komend sprawdzaliśmy poprzez wpisywanie niepoprawnych nazw komend, złej liczby argumentów wywołania komendy, nieprawidłowych wartości argumentów wywołania, wyjścia poza panel podczas wpisania poprawnych argumentów wywołania oraz niepoprawnej nazwy koloru. Wszystkie walidacje testowaliśmy kolejno dla następujących komend:

1) *lina 100 100 300 300*

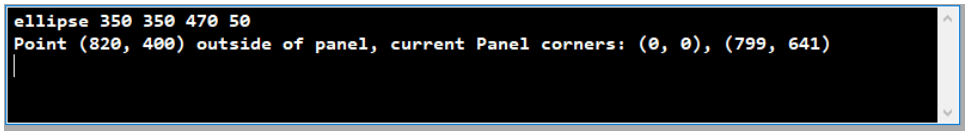
2) *arc 250 250 100 50 90*



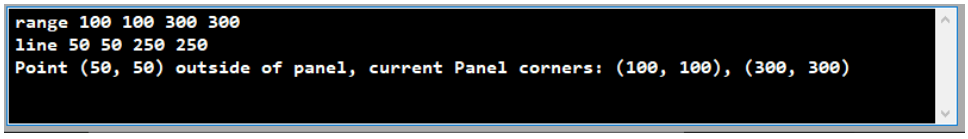
3) *rectangle 200 300 100 250*



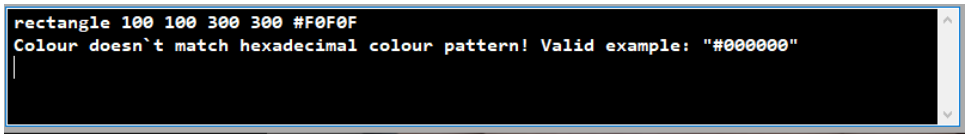
4) *ellipse 350 350 470 50*

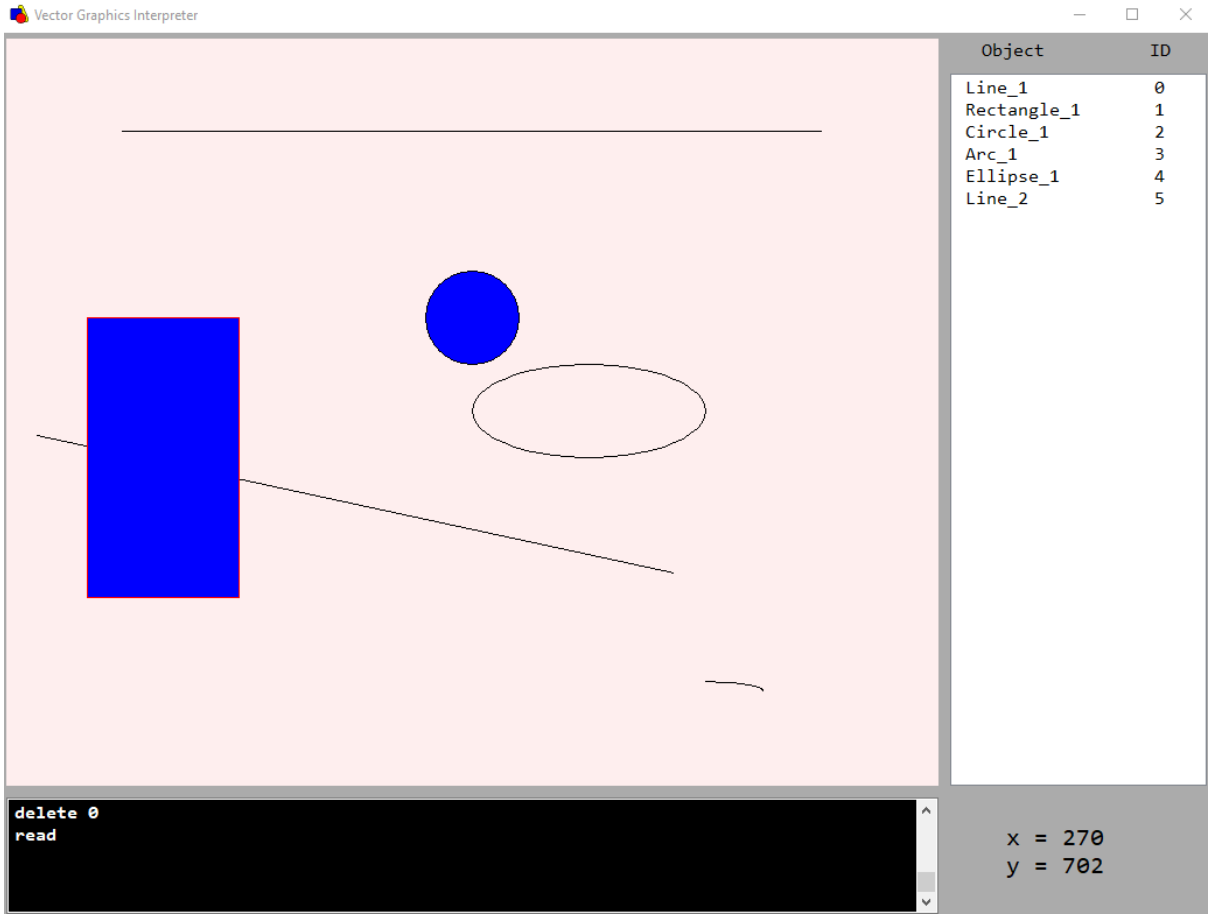


range 100 100 300 300
line 50 50 250 250

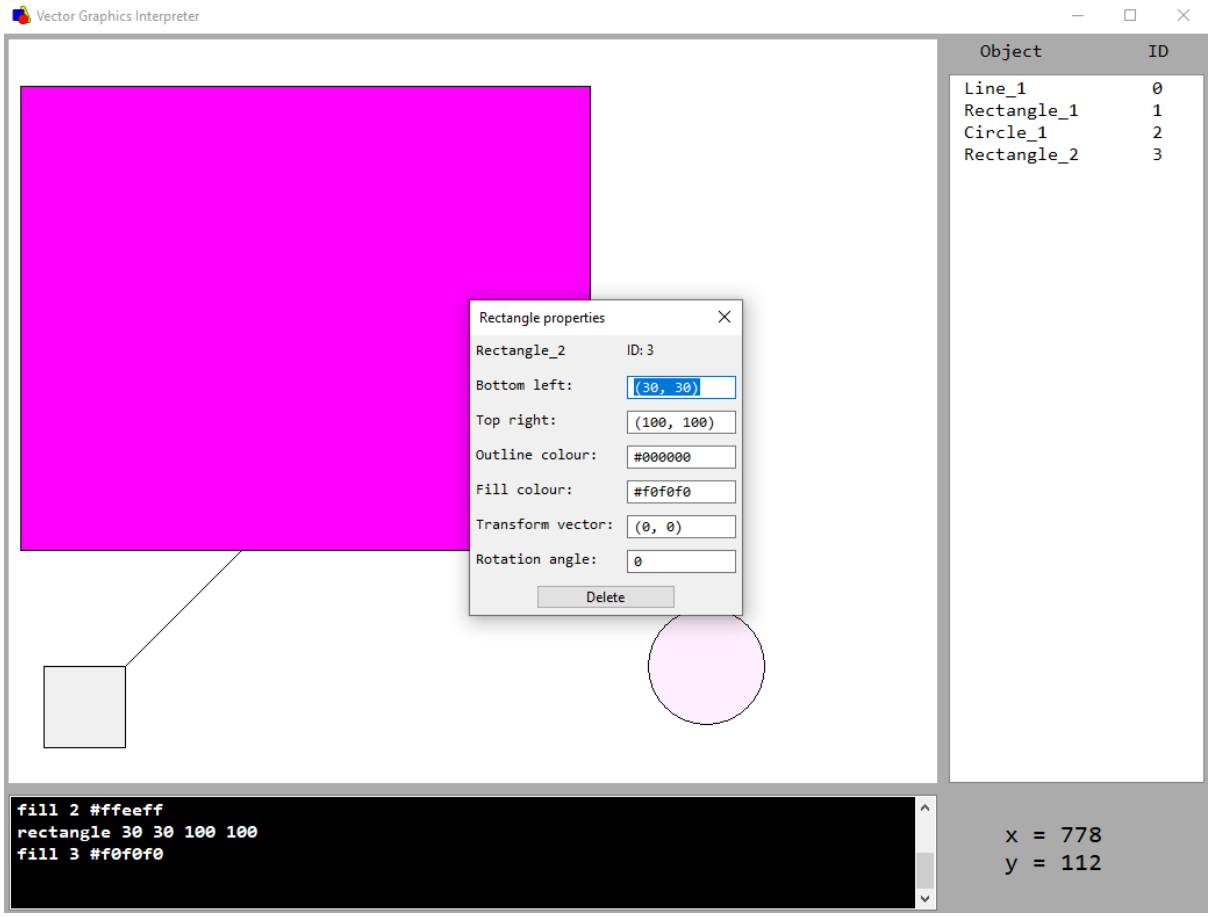


5) *rectangle 100 100 300 300 #F0F0F*





Rysunek 2. Testowanie aplikacji



Rysunek 3. Testowanie poprawności dialogu

8. Wdrożenie, raport i wnioski

Efekt końcowy działania aplikacji jest dla nas zadowalający, udało się poprawnie zrealizować wszystkie wymagania podstawowe, z drobną zmianą przy zapisie i odczycie do pliku. Działanie aplikacji jest płynne, przy zmianie rozmiaru okna figury się skalują oraz nie występuje problem migotania. Ponadto udało się zrealizować część wymagań rozszerzonych, w prawym dolnym rogu wyświetlane są aktualne współrzędne kursora myszki. Do spisu obiektów została dodana funkcjonalność pozwalająca na otworzenie okna dialogu dla poszczególnego obiektu.

Ogólny wygląd interfejsu jest prosty i czytelny, konsola przypomina wyglądem konsolę znaną z basha. Struktura programu pod względem implementacji i hierarchii klas również wydaje się być bardzo czytelna i prosta do określenia za co dana klasa jest odpowiedzialna.

Niestety nie udało się spełnić wszystkich wymagań rozszerzonych, takie jak dodatkowe instrukcje rysujące oraz komendy pozwalające powiększyć wybrany obszar i powrócić do wyświetlania w pierwotnym rozmiarze, brak tych rozszerzeń spowodowany był dużym nakładem czasowym na zrealizowanie w pełni wszystkich zadań podstawowych.